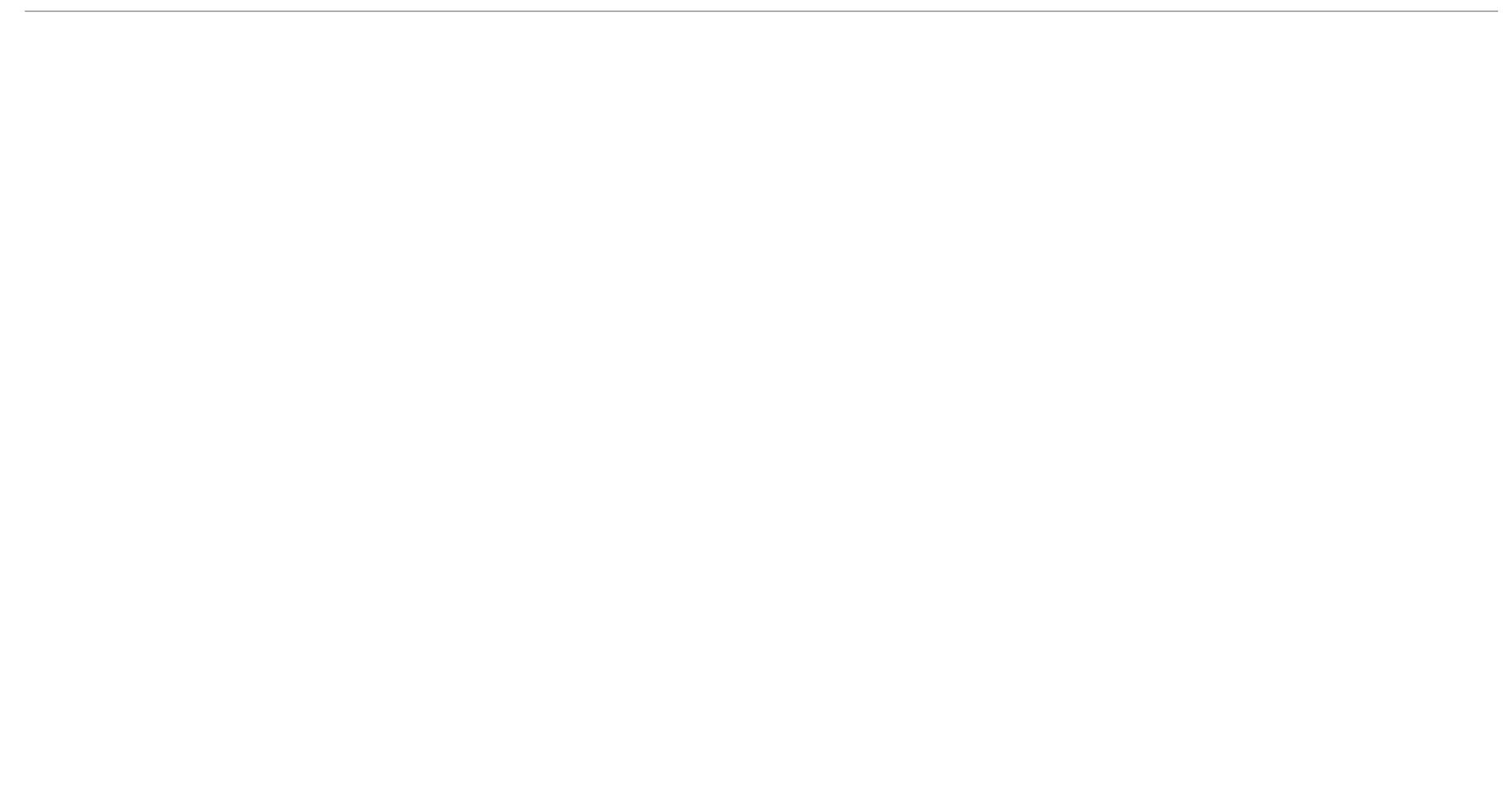


# データベース物理設計



<第1.00版 2015年1月>

お断り: 当資料は、DB2 for Linux, UNIX and Windows V10.5をベースに作成されています。



本資料掲載事項は、ある特定の環境・使用状況においての正確性がIBMによって確認されていますが、すべての環境において同様の結果が得られる保証はありません。これらの技術を自身の環境に適用する際には、自己の責任において十分な検証と確認を実施いただくことをお奨めいたします。

© Copyright IBM Japan Systems Engineering Co., Ltd. 2014



# はじめに(1)

□ 当資料は、DB2 for LUWを用いたデータベースの物理設計において実施すべきことを、設計の流れに沿って解説したものです。

当資料は、DB2 for LUW V10.5を前提としています。

データベース物理設計の流れや大まかな考慮点は、バージョンに依存しないため、異なるバージョンのDB2 for LUWをお使いの場合でも、当資料を参考にしていただけます。

一部バージョンの制約や、旧バージョンでは使用できない機能、旧バージョン固有の考慮点については、以下の資料を参照してください。

- **データベース物理設計【DB2 9.7 対応版】**

- [http://www.ibm.com/developerworks/jp/data/products/db2/pdf/db2v97\\_database.pdf](http://www.ibm.com/developerworks/jp/data/products/db2/pdf/db2v97_database.pdf)

- **データベース物理設計【DB2 9.5 対応版】**

- [http://www.ibm.com/developerworks/jp/data/products/db2/pdf/db2v95\\_database.pdf](http://www.ibm.com/developerworks/jp/data/products/db2/pdf/db2v95_database.pdf)



# はじめに(2)

- 当資料で設計対象とするDB2システムは、シングルノード構成のものとします。  
DB2 for LUWでは、DPF(Database Partitioning Feature)や、pureScale機能を使用し、分散システムを構築することが可能ですが、これらのシステム設計考慮点については、当資料では対象外です。  
分散システム設計を行う場合には、それぞれの機能についての以下の資料をご参照ください。
  - DPF
    - DWH設計ガイド – DB2 for LUW V9.5対応版
      - [http://www.ibm.com/developerworks/jp/data/library/infosphere/j\\_d-dwhdesign01/index.html](http://www.ibm.com/developerworks/jp/data/library/infosphere/j_d-dwhdesign01/index.html)
  - pureScale
    - DB2 10 for LUW 新機能ワークショップ資料(3章 可用性とスケーラビリティー)
      - [http://www.ibm.com/developerworks/jp/data/products/db2/db2\\_10-workshop/](http://www.ibm.com/developerworks/jp/data/products/db2/db2_10-workshop/)
    - 【BP様向け】DB2 10.5 for LUW 新機能テクニカル・ワークショップ資料(4章 DB2 pureScaleベストプラクティス)
      - <https://www-304.ibm.com/partnerworld/wps/servlet/mem/ContentHandler/IMO14062JPJA>
- DBサーバーの高可用性構成にてHADRを検討する場合は、以下の資料をご覧ください。
  - HADR
    - DB2 10 for LUW 新機能ワークショップ資料(3章 可用性とスケーラビリティー)
      - [http://www.ibm.com/developerworks/jp/data/products/db2/db2\\_10-workshop/](http://www.ibm.com/developerworks/jp/data/products/db2/db2_10-workshop/)



# はじめに(3)

- DB2 for LUWは、通常のリレーショナル表のほかに、XML文書をリレーショナル表のXMLカラムとして保持するpureXML機能や、データをカラム毎に保持するBLUアクセラレーション機能を提供しています。しかしながら、当資料では、データベース設計は、通常のリレーショナル表を使用することを前提としています。pureXML, BLUアクセラレーションを使用する場合には、以下の資料をご参照ください。
  - pureXML
    - XML DBデザイン・ガイド
      - 第1章: [http://www.ibm.com/developerworks/jp/data/library/dataserver/j\\_d-xmldbdesign01/](http://www.ibm.com/developerworks/jp/data/library/dataserver/j_d-xmldbdesign01/)
      - 第2章: [http://www.ibm.com/developerworks/jp/data/library/dataserver/j\\_d-xmldbdesign02/](http://www.ibm.com/developerworks/jp/data/library/dataserver/j_d-xmldbdesign02/)
      - 第3章: [http://www.ibm.com/developerworks/jp/data/library/dataserver/j\\_d-xmldbdesign03/](http://www.ibm.com/developerworks/jp/data/library/dataserver/j_d-xmldbdesign03/)
      - 第4章: [http://www.ibm.com/developerworks/jp/data/library/dataserver/j\\_d-xmldbdesign04/](http://www.ibm.com/developerworks/jp/data/library/dataserver/j_d-xmldbdesign04/)
    - BLUアクセラレーション
      - 【BP様向け】DB2 10.5 for LUW 新機能テクニカル・ワークショップ資料  
(2章 BLUアクセラレーション活用Tips)
        - <http://www.ibm.com/partnerworld/page/SWP14340JPJA>

- DB2 for LUWでは、Oracle互換機能を使用することにより、Oracle固有のデータタイプであるVARCHAR2を使用したり、PLSQLによる開発を行うことができます。当資料では、Oracle互換機能は使用しないことを前提としています。Oracle互換機能を使用する場合の考慮点については以下の資料をご参照ください。

- Oracle互換機能
  - OracleからDB2 10への移行
    - [http://www.ibm.com/developerworks/jp/data/library/db2/j\\_d-oracleikou/](http://www.ibm.com/developerworks/jp/data/library/db2/j_d-oracleikou/)



# はじめに(4)

□ 当資料では、物理設計の流れと考慮点の全体を解説することを目的としているため、設計上の選択肢となる機能の詳細や、設定方法については細かく解説していません。当資料を参考にしていただき、ある製品機能を使用しようとする場合は、それらの機能について、DB2 Knowledge Centerや、その資料について詳しく紹介された他のdeveloperWorks記事を参照してください。

## ● Knowledge Center

### ➤ V10.5

- [http://www-01.ibm.com/support/knowledgecenter/SSEPGG\\_10.5.0/com.ibm.db2.luw.kc.doc/welcome.html](http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.kc.doc/welcome.html)

### ➤ V10.1

- [http://www-01.ibm.com/support/knowledgecenter/SSEPGG\\_10.1.0/com.ibm.db2.luw.kc.doc/welcome.html](http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.1.0/com.ibm.db2.luw.kc.doc/welcome.html)

### ➤ V9.7

- [http://www-01.ibm.com/support/knowledgecenter/SSEPGG\\_9.7.0/com.ibm.db2.luw.kc.doc/welcome.html](http://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.7.0/com.ibm.db2.luw.kc.doc/welcome.html)

### ➤ V9.5

- [http://www-01.ibm.com/support/knowledgecenter/SSEPGG\\_9.5.0/com.ibm.db2.luw.kc.doc/welcome.html](http://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.5.0/com.ibm.db2.luw.kc.doc/welcome.html)

## ● developerWorks

### ➤ DB2情報ポータル

- <http://www.ibm.com/developerworks/jp/data/products/db2/developer/>



# 目次

## 1. 物理設計の流れ

物理設計の流れ

物理設計作業開始にあたって

## 2. 物理設計作業

- ①表・索引定義の作成
- ②データ容量の見積もり
- ③インスタンスの構成とデータベース分割
- ④表の分類と表スペースの構成
- ⑤表スペース容量の見積もり
- ⑥ディスク上へのオブジェクトの配置
- ⑦ユーザーと権限の設計
- ⑧構成パラメーターの設定
- ⑨シェル／コマンドの作成





ブランク・ページです

# 1. 物理設計の流れ





ブランク・ページです

# 物理設計の流れ

①表・索引定義の作成

- ・表のタイプ／参照整合や制約／属性、長さの決定
- ・主キー(1次索引)
- ・検索やジョインのパターンによって2次索引作成
- ・データ項目、長さ、索引などは既に決まっている前提
- ・表、(索引) サイズの見積もり

②データ容量の見積もり

- ・インスタンスの分割も検討
- ・バックアップ単位であるデータベースの分割を検討

③インスタンスの構成と  
データベース分割

- ・表の分類と表スペース構成の決定
- ・表スペースのタイプ、その他の属性の決定

⑤表スペース容量の見積もり

- ・表スペースの見積もり
- ・ログ領域の見積もり

⑥ディスク上への  
オブジェクトの配置

- ・物理ディスク上への表スペースの配置
- ・ログ、バックアップ用、ワークスペースの配置

⑦ ユーザーと権限の設計

- ・必要なユーザーIDと、ユーザーに付与する権限の定義

⑧構成パラメータの設定

- ・物理設計にあわせた構成パラメーターの変更

⑨シェル／コマンドの作成

- ・これまでに設計したオブジェクトを作成するコマンドおよびシェルを作成

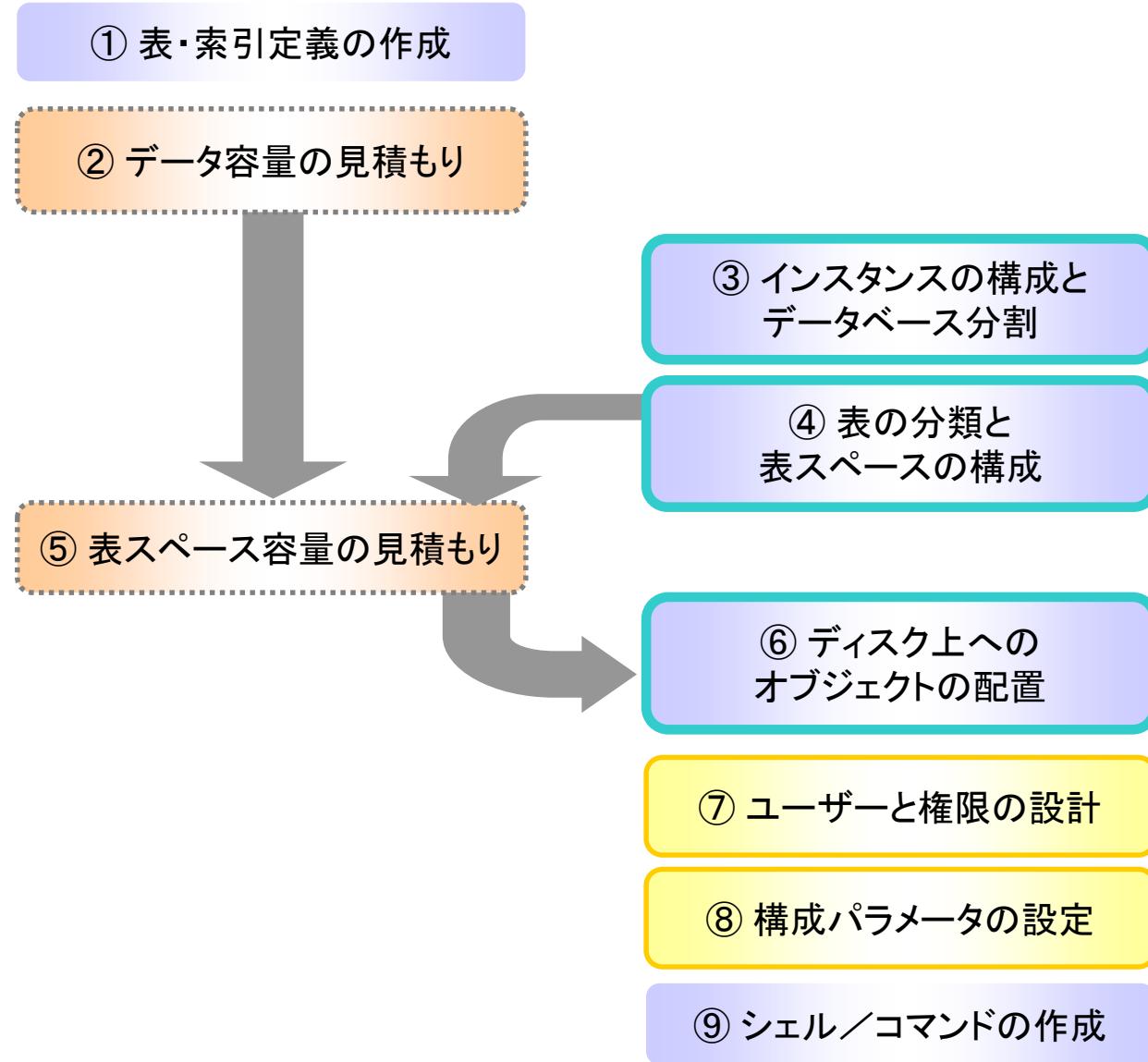


# 解説

- データベースの物理設計を行う場合、ディスクおよびサーバーを含むハードウェア構成の概要が決まっている必要があります。また、表の論理設計も既に終了していることが前提です。表の論理設計を実際のハードウェアおよび特定のDBMS実装上にどのように構築するかを決定することが、「物理設計」ということになります。
- ①DBMSに特化しない論理設計では、特定のデータベース実装に依存することなく、論理的な要件としてエンティティを識別し、データ項目のデータタイプや長さを定義します。物理設計の最初にまずこれらの定義を、特定のDBMS実装を前提とした表および列の定義に変換します。例えば、データは日付なのですが、格納方法はDATEにするかTIMESTAMP,CHARまたはVARCHARにするかなどの選択肢があります。  
表設計において、パフォーマンスを考慮して表の非正規化を行ったり、表をパーティション化するなど、表の構造を変更することもあります。また、検索要件に応じた索引を定義します。
- ②データと索引の容量を見積もります。論理設計によってできあがった表のレイアウトとレコード数から容量を見積もります。
- ③通常は、1システムで、1インスタンス1データベースが一般的ですが、格納するデータや用途、業務内容がまったく異なる場合は、データベースを分割することを検討します。分割する際には、使用される時間帯やメンテナンス、可用性、回復処理についての考慮も必要です。
- ④表を幾つかのグループに分類します。それらの表を配置する表スペースの定義を決めています。
- ⑤表スペース(システム・カタログ表スペース、ユーザー表スペース、一時表スペース)、ログ領域、等、データベース・サーバーに必要となるディスク容量を計算します。
- ⑥物理ディスクへの表スペースの配置を決定します。表スペース以外のオブジェクト(ログ領域、ワーク/バックアップ領域、等)の配置も決定します。
- ⑦DB2システムに必要なユーザーを準備し、ユーザーに対して付与する権限を決定します。
- ⑧物理設計に関連した構成パラメーターを設定します。
- ⑨最後に行うのは、これらの設計に基づいたファイルシステムや論理ボリュームなどを作成するシェル・スクリプトの作成と、これらの上に表スペースおよび表、索引を作成するDDLの作成です。
- データベースの物理設計は基本的には以上の作業を行い、ドキュメントを作成し、実際のハードウェア上に構築するところ終了します。



# 物理設計のステップ



# 解説

- 当資料では、物理設計を、「物理設計の流れ」に沿って解説します。  
しかし、実際の設計作業は、この流れのとおり単線で進むわけではありません。
- たとえば、物理設計における、容量見積もりを考えてみましょう。  
最終的には表や索引のようなデータベース・オブジェクトを物理ディスク上に配置することが必要です。そのためには、どれだけのディスク容量が必要であるか見積もる必要があります。物理設計の初期段階では、まだ、表の分類や使用方法に基づくログ量の見積もりが完全にできる情報がありません。また、索引についても設計の初期段階では、重要な索引の定義はなされているでしょうが、検索要件を満たす索引が網羅されているとは限りません。  
しかし、容量見積もりが無いと、インスタンスやデータベースの配置を決定することが難しいので、物理設計の初期段階で、大まかな容量見積もりを行います。これを元に、表の使用方法、運用要件などを考慮しながら、ディスクの割り振り、表スペースの配置を検討し、再度表スペースの見積もりを行い、容量見積もりを精緻化します。  
このように、設計作業においては、他のステップでの検討を取り込んで、スパイラル状に設計を進めていくことがあります。
- 当資料では、ステップの作業内容の関連付けを、ステップのボックスの色分けで示しています。
  - 水色①③④⑥⑨：主にオブジェクトの構造と配置について
  - ピンク②⑤：容量について
  - 黄色⑦⑧：それ以外

作業ステップでは、これらの組み合わせだけが相互に関連するわけではありませんが、以上の組み分けを意識すると、作業の流れを把握しやすくなります。



# 物理設計作業開始にあたって

## □論理データベース設計作業が完了していることが前提

- 論理データベース仕様の後からの変更は困難
  - アプリケーション開発作業に多大な手戻りを発生させる可能性がある
- 論理設計で決めた表をDB2データベースの表として定義
- 作成したデータベース仕様をシステム上の物理記憶域にマッピング
- DB2の製品仕様に依存する作業であるため、DB2の製品知識が必要
- 使用するオペレーティング・システム毎の知識も要求される

## □データベース設計の最終目標

- 時代の変化に伴う多様な要求の出現に対応できる安定したデータベースを構築する
- 各種要件(アプリケーション、性能、運用等)を最適に実装するデータベースを検討する
  - 運用設計、障害回復設計とも同期を取りながら進めて行く
- パフォーマンス・チューニングや運用の効率化のため適宜物理設計の見直しも必要
  - 充分にテストを実施して、データベース設計が最適に実装されているかを検証する必要がある



# 解説

- 物理設計作業開始にあたっては、論理データベース設計作業が完了していることが必要です。論理設計完了後の変更は、アプリケーション開発作業に多大な手戻りを発生させる可能性があります。
- 物理設計作業は、表の論理設計で決められた表を、DB2のデータベースの表として定義することです。
- 実際に、物理記憶域のどこにどのように表スペースや表、索引といった各オブジェクトを配置するかのマッピングを行ないます（論理設計で作られた論理モデルの「実装」を行なうことになります。）
- DB2の製品仕様に依存する作業であるため、DB2の製品知識が必要とされます。
- また、ファイル・システムなど使用するオペレーティング・システム毎の知識が要求されることもあります。
  
- データベースの設計の目標となるのは、自分の環境を、理解しやすくしかも将来の拡張の基礎となるよう表現したもののを作ることです。また、データの一貫性や整合性を保ちやすいデータベース設計が望ましいと言えます。そのためには、設計の段階で冗長性を少なくし、データベースの更新時に生じ得る異常をなくす必要があります。
- また、アプリケーション要件や、性能、運用要件等を最適に実装するものでなければいけません。
- 運用設計、障害回復設計とも同期を取りながら進めて行く必要があります。
- データベースの設計は、一度で終了するものではありません。多くの場合、何度かやり直すことが必要になります。パフォーマンス・チューニングや運用の効率化のため適宜物理設計の見直しも必要になるのです。



## 2. 物理設計作業



## ①表・索引定義の作成



# 物理設計の流れ

①表・索引定義の作成

②データ容量の見積もり

③インスタンスの構成と  
データベース分割

④表の分類と  
表スペースの構成

⑤表スペース容量の見積もり

⑥ディスク上への  
オブジェクトの配置

⑦ユーザーと権限の設計

⑧構成パラメータの設定

⑨シェル／コマンドの作成

## □ 列の設計

- データ・タイプと長さの決定

## □ 表の設計

- 主キーと外部キー
- 制約

- 固有(ユニーク)制約
- 参照制約
- 表検査(チェック)制約

- 表の構造

## □ 表に対するその他の要件の検討

## □ 索引の設計

## □ 圧縮の検討

## □ その他のオブジェクトの検討



# 解説

- 物理設計の第1段階として、論理設計を元に表、列および索引の設計を行います。
  - 原則として、論理設計におけるエンティティは、物理設計における表となります。同様に、データ項目は、列となります。
  - RDBMSの制限を考慮して表の分割を検討します。
  - パフォーマンスを考慮して、論理データモデルで正規化されていたエンティティを、一つの物理表にまとめていたり、コード表など論理エンティティとして存在しない表を追加することがあります。
- データ項目のデータ・タイプは、実装環境であるRDBMS製品固有のデータ・タイプに置き換えます。
- エンティティを識別するための、主キー、エンティティ間の関係を表す参照制約などを定義します。
- パフォーマンス向上のため、RDBMSの提供する表構造の採用を検討します。  
DB2では、大規模表について、主に以下の表構造を検討します。
  - パーティション表
  - MDC(多次元クラスター表)
- 以下のような、表に対する特別な要件の有無を検討します。
  - 圧縮
  - セキュリティ要件:RCAC
  - タイムトラベル検索
- 検索要件に基づいて、索引を追加します。
  - 索引の列定義順序や索引の種類についても検討します。  
DB2で使用可能な索引の種類としては、以下があります。
    - ユニーク索引
    - クラスター索引
    - INCLUDE列を使用した索引
    - 関数を使用した索引(式ベース索引)
- 表、索引以外のオブジェクトについて、作成するかどうかを検討します。  
オブジェクトとしては、以下が挙げられます。
  - View
  - MQT(マテリアライズド検索表)
  - 作成済み一時表
  - その他
- この工程によって、データベースに登録する表、索引などのオブジェクトのDDLを決定します。



# 列の設計 – データ・タイプ

## □ 列のデータ・タイプと長さの決定

- データ長や、とり得る値の制限値により、最適なデータ・タイプを選択する
  - 開発言語環境による生産性の観点での考慮なども必要
- 文字列(CHAR/VARCHAR/GRAPHIC/VARGRAPHIC)
  - 可変長 or 固定長? ⇒ 原則、固定長を使用することを推奨
    - 可変長(VARCHAR)は固定長(CHAR)に比べて4バイト分余計に必要
    - 可変長は、該当列の位置を認識するためにCPU負荷が増加
    - 固定長は定義した長さ分の領域を必ず使用する(圧縮を採用した場合はこの限りでない)
  - UNICODEデータベースではGRAPHICはNCHAR/NVARCHARも使用可
- 日付／時刻形式(DATE/TIME/TIMESTAMP)
  - CHARで保持するより、格納サイズが小さい
  - YEAR, MONTH, DAYなどの日付計算、時間計算関数が使用可能。DB2が計算・値のチェックを行える
  - 日付/時刻をCHARやINTで保持すると、DATEやTIMEなどに比べ、オプティマイザーによる値の分散推定が不正確になりやすい  
(DATEであれば、2012/12/31と2013/01/01の間にはとりうる値が無いと判断できるが、CHARであれば、2012/12/33や、2012/12/XYもとりうる。このため、レコード数の分布見積もりに必要な、とりうる可能性のある値の数が異なってしまう)
- 数値(SMALLINT/INTEGER/BIGINT/REAL/DOUBLE/DECIMAL/DECFLOAT)
  - 算術に使用するのであれば、通常は数値型。最大取り得る値によって、データ・タイプを選択する
  - CHARで保持するより、格納サイズが小さい
  - 小数点以下がある場合はDECIMALを検討
- LOB型(CLOB/DBCLOB/BLOB) ※LONG VARCHAR/LONG VARGRAPHICは9.7より非推奨
  - LOBについても、他の表データとは別の場所に保管され、各列の情報(ポインター)のみを他データと共に持つ
    - LOBをinline化して、他の列と同じページに格納することも可能
    - LOBデータの読み書きには、バッファープールが使用されない
    - データが4KB以下の場合、LOBはなるべく使用しない
- 拡張マークアップ言語(XML)
  - XML文書を格納する。
  - コード・セットUTF-8以外のデータベースでも使用可能。(ただし、XML列はUTF-8で格納される)

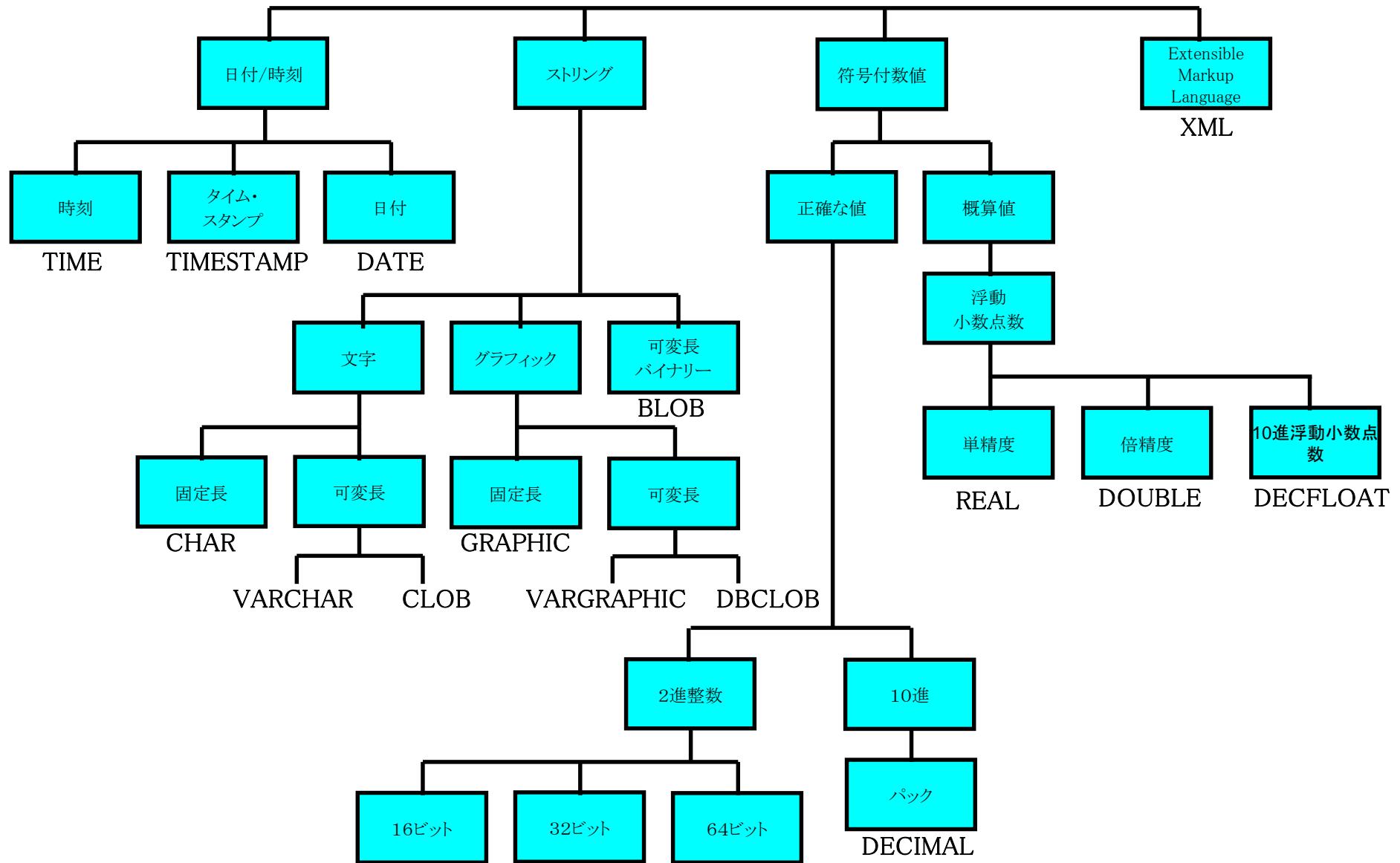


# 解説

- 数値・文字列(固定長・可変長)・日付・XMLなどのうちどれを選択するか、データ長や、とり得る値の制限値により適したデータ・タイプを選択します。
- また、開発言語環境によって扱い易いデータ・タイプであるのか、生産性の観点などからの考慮も必要です。
  
- 文字列
  - 可変長か固定長か決定する必要がありますが、まずは固定長を検討します。
  - 可変長の場合は、長さとオフセット情報を入れる領域が列あたり4バイト分余計に必要になります。
  - 可変長の場合は、該当列の位置は先頭からたどらなければならないため、CPUの負荷が該当列の位置がわかっている固定長よりも余計にかかります。
  - 列長の差が大きい(列あたり平均20バイト以上)時には可変長を採用することで、DISKスペースは削減されます。
  - UNICODEデータベースにおいては、GRAPHICをNCHARやNVARCHARとして指定することができます。
  
- 日付／時刻のデータ
  - 日付計算、時間計算、関数の使用が可能になるように、DATE/TIME/TIMESTAMPを使用してください。また、その方が、CHARデータタイプとして格納するよりもDISKスペースは軽減されます。
  
- 数値
  - 算術に使用するのであれば、通常は数値型で格納すべきです。該当の項目の最大取り得る値によって、データ・タイプを選択します。また、文字列で格納するよりもDISKスペースは軽減されます。
  
- LOB型
  - LOBタイプは表データ・ページに実際の列のデータは含まれません。別の表オブジェクトとして表スペースに格納されます。行データ中にはそれらの列の20バイトの記述子(descriptor)は含まれます。
  - LOBタイプのデータをLOB専用の表スペースに格納させることも、CREATE TABLE時の指定で可能です。
  - 4KB以下の文字データについては、上述のような特異な扱いを避けるためにもLOBタイプは使用しないようにするなど、データ長の制限値により、適したデータタイプを選択してください。
  
- XML
  - XML文書は、階層構造を持つデータとして格納されます。
  - LOB列と同様に、XML列は列の記述子であるXMLデータ指定子(XDS)のみを保持します。XMLデータ自体は、別個にXDAと呼ばれるストレージ構造保管されます。
  - XML文書のサイズの上限は、2GBです。
  - DPF/MDC/パーティション表もサポート
  - XDA/XML索引の圧縮機能サポート(XML Region Index、XML Indexのみ)



# DB2がサポートするデータ・タイプ(1)



# DB2がサポートするデータ・タイプ(2)

## ・文字タイプ

データタイプ	説明	制限値
CHAR(n)	nバイトの固定長文字列	1 <= n <= 254
VARCHAR(n)	最大nバイトの可変長文字列	1 <= n <= 32672
LONG VARCHAR(n)	長可変長文字列	最大 32700バイト
GRAPHIC(n)	n文字の固定長漢字ストリング	1 <= n <= 127
VARGRAPHIC(n)	最大n文字の可変長漢字ストリング	1 <= n <= 16336
LONG VARGRAPHIC	長可変長漢字ストリング	最大 16350文字

非推奨

非推奨

## ・数値タイプ

データタイプ	説明	制限値
SMALLINT	短精度整数	-32768 ~ +32767
INTEGER	長精度整数	-2, 147, 483, 648 ~ +2, 147, 483, 647
BIGINT	64ビット整数	-9, 223, 372, 036, 854, 775, 808 ~ +9, 223, 372, 036, 854, 775, 807
REAL	単精度浮動小数点数	-3. 402E+38 ~ -1. 175E-37 もしくは 1. 175E-37 ~ 3. 402E+38
DOUBLE、FLOAT	倍精度浮動小数点数	-1. 79769E+308 ~ -2. 225E-307 もしくは 2. 225E-307 ~ 1. 79769E+308
DECIMAL(m, n)、NUMERIC	10進数(精度桁数、小数点以下桁数)	1 <= m <= 31, 0 <= n <= m
DECFLOAT(16),DECFLOAT(34)	10進浮動小数点数	10-383 ~ 10+384, 10-6143 ~ 10+6144

## ・日付／時刻タイプ

データタイプ	説明	制限値
DATE	日付	0001-01-01 ~ 9999-12-31
TIME	時刻	00:00:00 ~ 24:00:00
TIMESTAMP	タイム・スタンプ	0001-01-01-00. 00. 00. 000000 ~ 9999-12-31-24. 00. 00. 000000

# DB2がサポートするデータ・タイプ(3)

## ・ラージ・オブジェクト

データタイプ	説明	制限値
CLOB(n K M G)	文字ラージ・オブジェクトストリング	2, 147, 483, 647バイト
DBCLOB(n K M G)	2バイト文字ラージ・オブジェクト	1, 073, 741, 823文字
BLOB(n K M G)	2進ラージ・オブジェクト	2, 147, 483, 647バイト

## ・ラージ・オブジェクトのディスクリプタ

LOBの最大長	LOB Descriptor長
1, 024	72
8, 192	96
65, 536	120
524, 000	144
4, 190, 000	168

LOBの最大長	LOB Descriptor長
134, 000, 000	200
536, 000, 000	224
1, 070, 000, 000	256
1, 470, 000, 000	280
2, 147, 483, 647	316

## ・XML

データタイプ	説明	制限値
XML	XML文書	2ギガバイト



# その他のデータ・タイプ

## □ ユーザ定義タイプ

### ● 特殊タイプ

- INTEGER、CHARなどの前もって定義されていたタイプを組み込んで、ユーザ定義タイプを定義することができる
- 既存タイプに対してビジネス的な制限を加えたいときに使用する
- 特殊タイプは組み込みデータタイプと同様に、列のデータ・タイプとして表に定義することが可能
- 作成例

```
CREATE DISTINCT TYPE IMAGE AS BLOB(10M)
CREATE DISTINCT TYPE AUDIO AS BLOB(1G)
CREATE TABLE PERSON
  (ID INTEGER NOT NULL,
  NAME CHAR(30),
  VIDEO IMAGE,
  VOICE AUDIO )
```



# LOB

## □ 長いデータを格納するためのデータ・タイプ

- LOB記述子

- 表内に他の列と同一ページに格納され、LOBデータオブジェクト格納先へのポインターとなる

- LOBデータオブジェクト

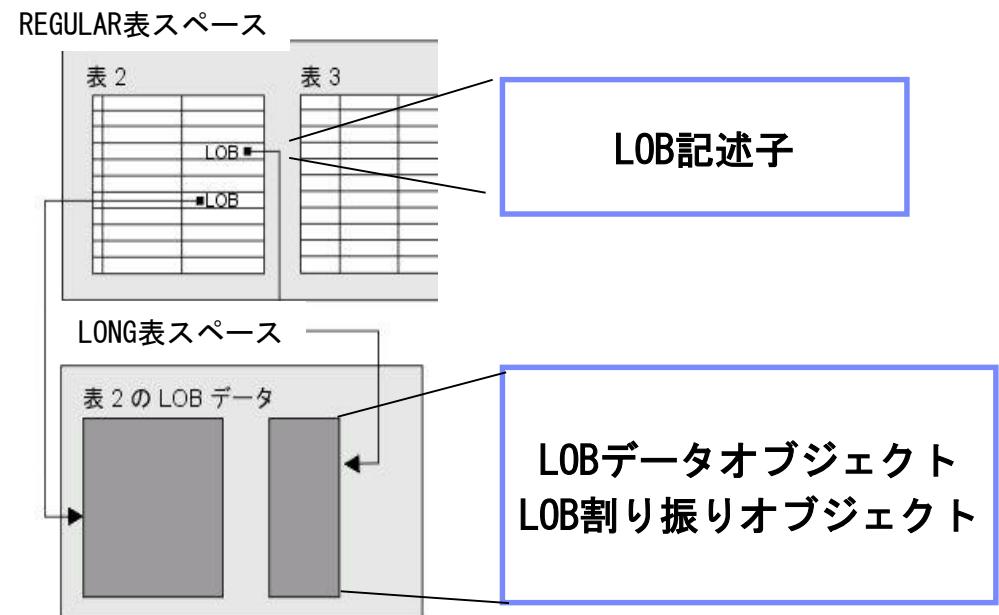
- 実際のLOBデータ
- 表の他の列とは別の場所(LOBデータ領域)に置かれる

- 自動ストレージまたはDMS表スペースの場合、CREATE TABLE文' LONG IN 表スペース名'でLOBデータの格納先の表スペースを指定可能
- CREATE TABLE時には、領域は確保しない

LOBデータが挿入されたとき、割り振られる

- LOB割り振りオブジェクト

- LOBデータ領域のスペース管理情報
- 64GBごとに4Kページ1個 + 8MBごとに4Kページ1個



# 解説

## □ LOB

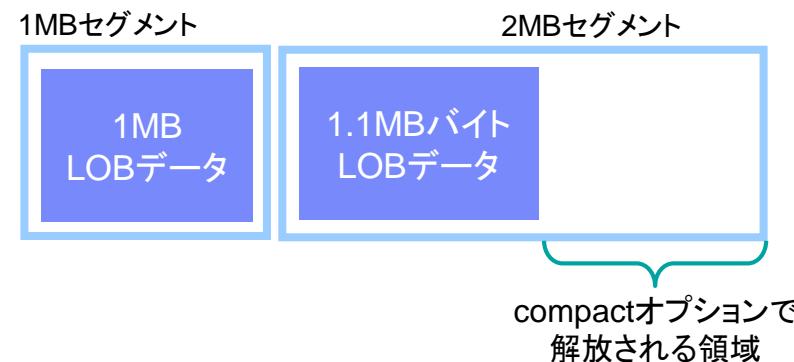
- 長いデータを格納するためのデータ・タイプです。
  - 最大長は2GB
  - 以下のタイプがあります。
    - CLOB(文字列)
    - BLOB(バイナリ)
    - DBCLOB(2バイト文字列)
- LOBデータは、表内のLOB以外の列とは別の領域に置かれます。  
表内には、LOB記述子と呼ばれるLOBデータへのポインターが置かれます。
- 自動ストレージまたはDMS表スペースの場合、LOBを含む表のCREATE TABLE文で、'LOG IN 表スペース名'を指定することにより、表の他の列と、LOBデータを異なる表スペースに格納することができます。
- SMS表スペースに保管する場合、表の通常列とは別のファイルに格納されます。
  - ファイル名は、SQLXXXX.LB, SQLXXXX.LBAです。
- SQLXXXX.LB : LOBデータオブジェクト
  - 1024\*2の累乗のセグメントずつ増えます。
    - 1024, 2048, ..., 64MB
- SQLXXXX.LBA : LOB割り振りオブジェクト
  - アロケーションとフリー・スペース情報
  - 64GB毎に4KBページ 1個 + 8MB毎に4KBページ1個。
- COMPACTパラメータ(CREATE TABLE XXXX)
  - LOBデータの将来の更新のために予めフリースペースを確保しないようにします。
    - LOBデータをより小さいセグメントに分割させます。(パフォーマンスは悪くなりますが、ディスクスペースは少なくなります。)
  - not compactはスペースを確保します(デフォルト)
    - LOBデータをひとつのセグメントの中に連続してとります。
  - 例) create table blob (col0 clob(10m) compact)
    - 2500バイト/行 × 1000 行 load
    - compact - 3MB 2048+1024バイトずつとります。
    - not compact - 4MB 4096バイトずつとります。



# 参考:LOBデータ

## □セグメント(1024)単位でデータを格納

- 1024バイト\*2の累乗<1024バイト、2048バイト、4096バイト...>



## □compactオプション<CREATE TABLEステートメントで指定>

- LOB 値が使用する最後のグループ内の余分なディスク・ページすべてを解放するため、ディスク・スペースを有効に活用できる
- LOBデータの長さを増加する操作など、パフォーマンスが低下する可能性がある
- 再編成(REORG)に注意する
  - compactオプションを使用しない場合、再編成でサイズが大きくなる可能性がある





ブランク・ページです

# 列の設計 – 考慮点

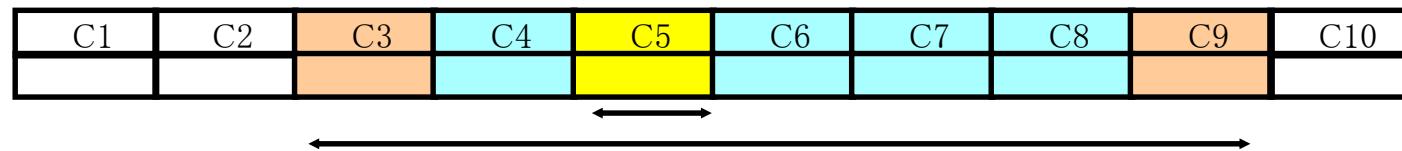
## □ 設計上の考慮点

- NOT NULLの指定
  - NULL可能にした場合、列毎に1バイトのNULLフラグが必要であり、CPU負荷が増加する
  - NULL標識変数を準備しなければならず、プログラムが煩雑になる
  - プログラムですべての列の値が与えられないNOT NULL列に対しては、DEFAULTを指定するようにする
- 比較、結合、UNIONの処理がある列はデータ・タイプを揃える
  - データ変換負荷の軽減
- パフォーマンスを考慮した列の順序
  - 可変長列は、自動的に全ての固定長列の後ろに配置される
  - 更新される列は可能であれば近くに並べる
  - 適用業務には列の指定順序は影響しない(順序にこだわる場合、視点(View)で対応も可)
  - データ行圧縮を使用する場合、複数列に跨って繰り返されるパターンのデータがあれば、それらの列は隣接して並べる(圧縮効果が高くなる)
- VARCHAR列: 適当なデータ・タイプかどうか要検討
  - 読み取りに2ステップ要: 長さ→データ
  - VARCHAR列の更新: 更新前データより、更新後のデータが長くなった場合、Tombstoneが発生しデータのフラグメンテーションを招く⇒ REORG運用が必要となる可能性が高い
- 列の自動生成
  - 生成列(GENERATED COLUMN)の利用
    - 指定されたルールに従い、列の値が動的に生成される列
    - 列の値を事前に加工して入れておくことにより、SQL実行時のパフォーマンスを向上させる
  - 識別列(IDENTITY COLUMN)の利用
    - DB2が列の値として、固有の数値を生成する列
    - 固有の値を取得するために別途表を用意したり、MAX関数を使用して取得したりする必要がない
    - 行をユニークに識別可能な、列の値を事前に入れておくことにより、識別列を使用した照会処理が可能
    - シーケンス・オブジェクトの利用も検討
  - 列変更タイムスタンプ(ROW CHANGE TIMESTAMP)
    - 列の挿入、更新の際に自動的にタイムスタンプを挿入する



# 解説

- 列にNULLデータが入る可能性がなければ、以下の理由からNOT NULLを指定してください。
  - NOT NULLでない場合
    - 列毎に1バイトのNULLフラグが必要
    - NULLか否か調べるCPU負荷の増加
    - NULLフラグによるDISKスペースの増加
    - プログラムではNULL標識変数の準備が必要
- 比較、結合、UNIONの処理がある列は、列同士のデータタイプをそろえることでデータ変換負荷を軽減させることができます。
- パフォーマンスを考慮した場合、更新される列は近くに並べて下さい。
- 更新時のログは、先頭の更新列から、最後の更新列まで取得される為、更新列が離れているとログデータが増加します。



- update testtab set c5=50 ..... ログは、c5のみ
- update testtab set c3=20, c9=90 ... ログはc3～c9まで
- ただし、可変長列でその列長が変更されるような更新がおこなわれた場合、行全体(新旧)がログとして取得されます。

- 可変長列は、全ての固定長列の後に配列されます。

- DB2は、データの保存時に固定長列、可変長列、LONG VARCHARポインター(20バイト)の順で自動的に格納しています。



固定長のinteger列を得るためにその前のVARCHAR列からたどらなければならない



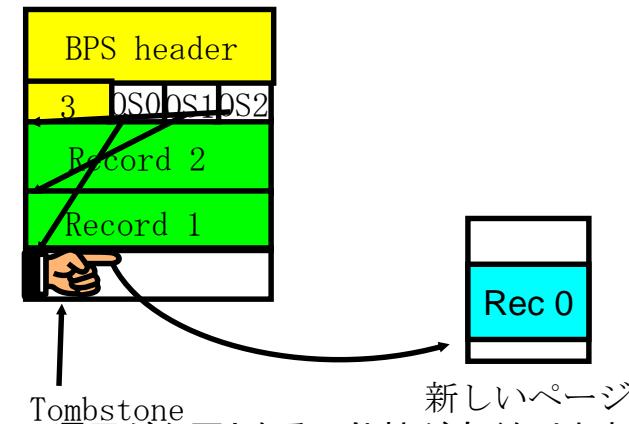
integer列は行の先頭からの相対位置が変わらないので直接得ることができる



# 解説

## □ VARCHAR/VARGRAPHIC列を使用する場合には、可変長であるメリットとデメリットを考慮した上で、使用して下さい。

- 可変長列は、2種類の情報を持っています。
  - データの長さ
  - データ
- 可変長データを読み込む場合、まずデータの長さを読み取り、次にデータをその長さ分読み取るという2段階を経るため、パフォーマンスに影響が出ます。
- また、可変長データに変更が発生した場合、元データより長くなると同じページに収まらなくなってしまう可能性があります。
- その場合、移動先の情報を持ったTombstoneが元のデータ位置に残されます。
- その為、そのレコードを取得するために、ページを2段階経なければならなくなる可能性があります。



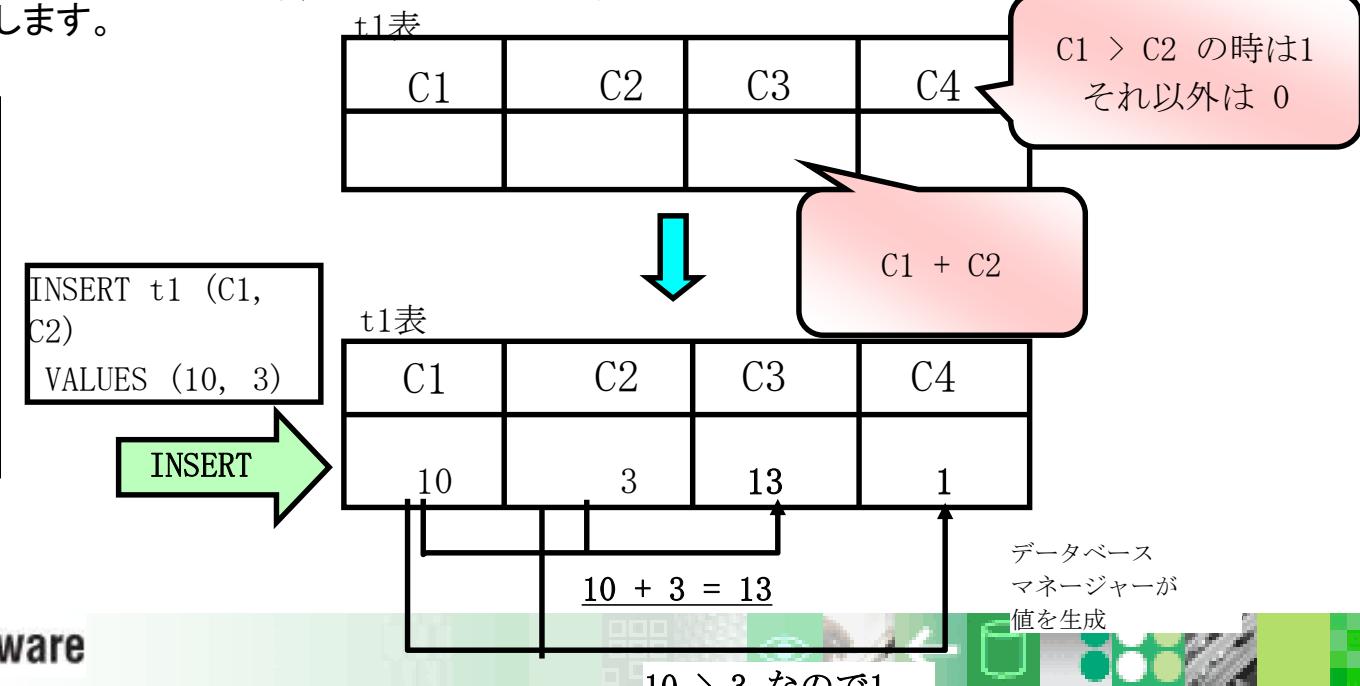
- 更新がある場合、データのフラグメンテーションを招き、REORG運用が必要となる可能性が高くなります。
- VARCHAR列を使用するのが望ましい場合：
  - データの長さの範囲がまちまちで、ほとんどの列は短い
  - データの長さの範囲がまちまちで、全ての範囲でほぼ均等に分布
- VARCHAR列を使うべきでない場合：
  - データの長さの範囲はまちまちだが、ほとんどの列は範囲の上の方にある
  - VARCHAR列にすることによって、ディスク・スペースが余計にかかり、パフォーマンスが低下するケース



# 解説(生成列)

- 生成列とは、各行の値を挿入操作または更新操作からではなく、定義した式によって定めることができる列です。更新トリガーおよび挿入トリガーを組み合わせて使用すると同様のことが行えますが、生成列を使用すると、派生した値が式と一貫したものであることを保証できます。
- 表で特定の式や述部を頻繁に使用することがわかっている場合、生成列を使用してあらかじめ値を生成しておくことにより照会の際のパフォーマンスを向上させることができます。
- ID列、列変更タイムスタンプ(ROW CHANGE TIMESTAMP)のように、DB2が自動的に値を挿入する列もあります。
- 表で生成列を作成するには、ALTER TABLEまたはCREATE TABLE時にGENERATED ALWAYS AS文節を使用して、列の値を定義する式を含んだ列を指定します。
- 生成列には検査制約やユニーク索引/基本キーが使用できない、生成列を持つ表に対して RENAME TABLE ができる等の制約事項があるので使用にあたっては考慮が必要です。
- 生成列の例
  - “c1”および“c2”という通常の2つの列と、表の通常の列から派生した“c3”および“c4”という2つの生成された列の入った表を作成します。

```
CREATE TABLE T1(c1 INT, c2 DOUBLE,
  c3 DOUBLE GENERATED ALWAYS AS (c1
+ c2),
  c4 GENERATED ALWAYS AS
    (CASE WHEN c1 > c2 THEN 1
      ELSE 0
    END)
);
```



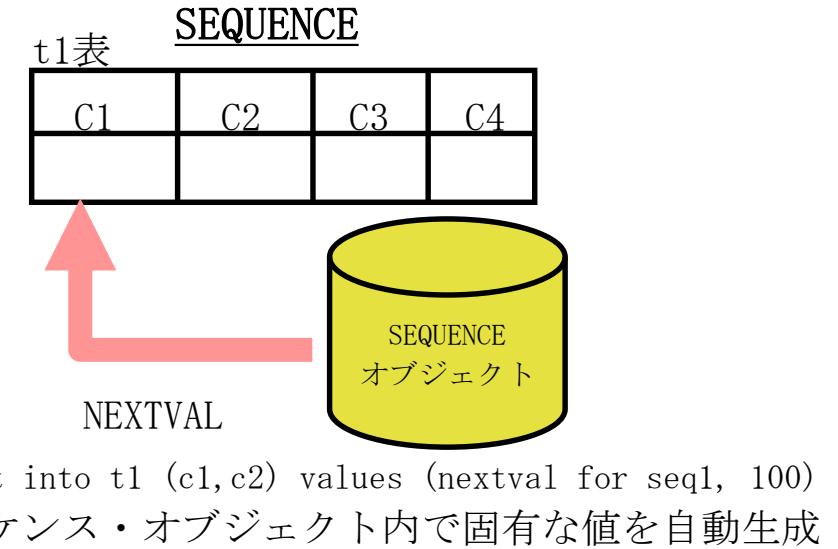
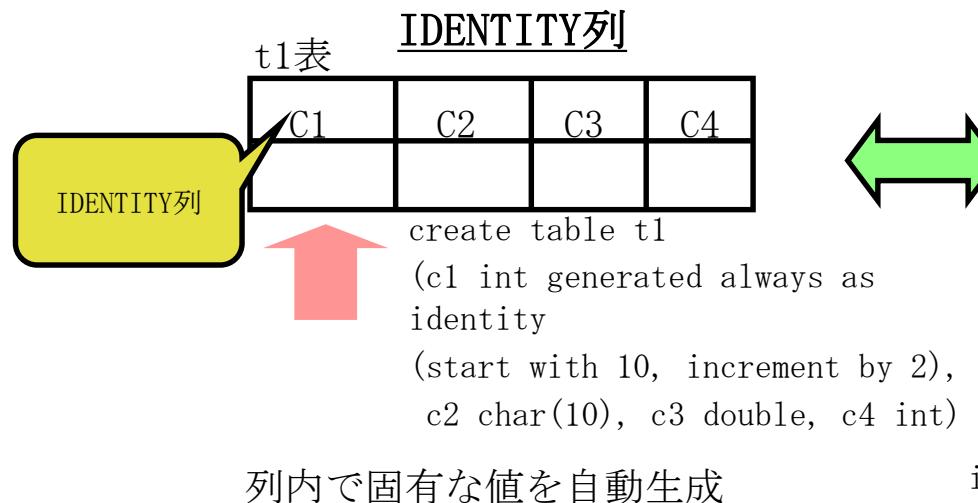
# 解説(IDENTITY列とSEQUENCE)

□ IDENTITY列(識別列)は生成列の中の一つで、表の各行に対して固有な基本キー値を自動的に生成します。

- 識別列では、アプリケーションがデータベースの外に独自のカウンターを生成する際に生じる、並行性およびパフォーマンス上の問題を回避することが可能です。
- 固有な基本キーを自動生成する際に識別列を使用しない場合には、単一行の表にカウンターを保管するのが一般的な設計方法です。各トランザクションはこの表をロックして、数を増分してからトランザクションをコミットして、カウンターのロックを解除します。しかし、残念ながら、この設計では、カウンターを増分できるのは一度に1つのトランザクションのみです。一方、識別列を使用して基本キーを自動的に生成すると、アプリケーションでより高度なレベルの並行性を実現できます。

□ SEQUENCE(シーケンス)とは、値の自動生成を可能にするデータベース・オブジェクトです。

- シーケンスを使用すると、固有キー値を生成することができます。IDENTITY列と同様アプリケーションはシーケンスを使用することで、データベースの外部に固有力カウンターを生成したことによって発生する可能性のある、並列性およびパフォーマンスの問題を回避することができます。
- 識別列属性とは異なり、シーケンスは特定の表列に関連付けられていないデータベースオブジェクトです。
- シーケンス・オブジェクトはどのアプリケーションでも使用できるため、NEXTVALおよびPREVALの二つの値を返す式が定められています。





ブランク・ページです

# 表の設計

## □RDBMSに実装する表を定義する

- 基本的に、論理データモデルのエンティティに基づいて表を定義する
  - 論理データモデルを物理的に実装可能にするための変換が必要な場合もある
    - 多対多参照の解決など
  - パフォーマンスを考慮した表の非正規化の検討
- 表の主キー、外部キーを識別する
  - 主キー：エンティティを特定する（表のレコードを特定する）
  - 外部キー：エンティティ（表）間の関係を表す
- DB2の機能/制約を考慮した表構造/表の特性の設計
  - 物理表決定のための考慮点/制限
    - 表に定義できる列数の制限
    - ページサイズ(4K, 8K, 16, 32K)ごとの行長の制限
      - ◆ inline LOB
      - ◆ 拡張行サイズ
  - 表構造の決定
    - パーティション表
    - MDC(多次元クラスター表)
    - その他
  - 特別な要件に対する考慮
    - セキュリティに対する考慮(RCAC)
    - タイムトラベル表使用の検討
  - その他の表の属性
    - ページ空き領域(PCTFREE)など

## 解説

- 表は、論理モデルのエンティティ、列はエンティティ属性にあたり、基本的に、論理モデルで設計します。物理設計では、論理モデルで識別された、エンティティ/属性を、実装環境である特定のRDBMS製品で定義できる形式にします。
- 一般的には、論理モデルのエンティティが表となります。論理データモデルを物理データモデルの表/列とするために、変更を行う場合もあります。
  - パフォーマンス向上のための、表の非正規化や、多対多関係の解決などは、物理設計における表の変更例です。
- 通常、論理設計では、エンティティ、属性を定義するとともに、エンティティを識別する主キー、エンティティ間の関係を表す外部キーも定義されています。物理設計では、これらの主キー、外部キーを実装できるように定義します。
  - 外部キーと親表のキーの値に関する参照整合性は、参照制約を定義することによって、強制できます。しかし、アプリケーションで、外部キー、親キーの値を保障する場合、参照制約を定義しない、または情報制約(NOT ENFORCED)として定義することも可能です。
- この他、表の設計で考慮すべき事項について、表定義上の制約、表の構造、その他の要件について見当します。



# 主キーと外部キー

- 表は原則として論理モデルのエンティティ  
列は、エンティティの属性

## □ 主キー(Primary Key)

- 主キーは表のレコード(エンティティのインスタンス)を一意に識別する
  - 格納されるデータの値は、ユニークでなければならない
  - NOT NULL指定必須
- 主キーを付与するとユニーク索引(1次索引)は自動的に作成される(固有制約)
  - 参照制約を使用しない場合は基本キーではなく部門番号ユニーク索引を定義しても可

部門番号	部門名	<社員表>		
社員番号	社員名	部門番号	外部キー	
100	営業部	A111	田中	100
200	電算部	B222	山田	200
300	総務部	C333	鈴木	200
主キー				

## □ 外部キー(Foreign Key)

- 親表と従属表の間の親子関係を示す働きをする
  - 別の表の主キーが、その表のデータ項目になっている時、そのキーは外部キーとなりうる
  - 結合操作の結合列になる ⇒ 索引の候補
  - 外部キーは参照の整合性を保証するために、主キーに存在しない値を持ってはいけない(参照制約)
  - 参照制約を使用しない場合、あくまでも論理的なものであって、物理定義する必要はない

## □ 定義時には、CONSTRAINTにより制約名をつける

- 管理が容易



# 解説

## □ 主キー

- 表の固有キーとは、それぞれの値が固有の行を識別する、1つの列または複数の列の順序集合のことです。たとえば、社員番号の列の各値は一人の社員だけを指すものなので、これを固有キーとして定義することができます。二人の社員が同じ社員番号を共有することはできません。
- 表の主キーは、1つの表上に定義された固有キーの1つですが、その表上で1番目に重要なキーとして選択されたものです。1つの表上には1つだけの基本キーが可能で、そのエンティティ内で行を唯一無二のものとして識別できるデータ項目です。
- 主キーと定義した場合、DB2が自動的に索引を作成します。
- 主キーはエンティティの保全性を保証するために、ユニークであり、空白値は許されません。
- V10.5からNOT ENFORCEDとすることが可能です。このとき、ユニークかどうかの検査は行われません。

## □ 外部キー

- 表の外部キーは、親表の固有キーまたは主キーを参照する、表内の1つの列または1組の列のことです。
- 外部キーは表(親表)と表(従属表)の間の参照の整合性を保証するために、基本キーに存在しない値を持ってはならず、結合操作時には結合列になります。
- 参照制約を使用しないのであれば、親表と従属表の間の親子関係を示すあくまでも論理的な意味合いのものであり、外部キーの物理的な設定は必須ではありません。
- 参照制約を定義しておきたいが、親表の値の存在チェックなどは

## □ 主キー、外部キーの定義時には、CONSTRAINTにより制約名をつけたほうが管理しやすくなります。

## □ 主キーの定義

```
CREATE TABLE 親表名
  (主キー列名 ... NOT NULL,
  ....,
  ....,
  CONSTRAINT 制約名 PRIMARY KEY(主キー列名))
```

## □ 外部キーの定義

```
CREATE TABLE 従属表名
  (....,
  外部キー列名 ... NOT NULL,
  ....,
  CONSTRAINT 制約名 FOREIGN KEY(外部キー列名)
  REFERENCES 親表名
  ON DELETE 規則)
```



# 制約

## □ データの保護や、データ間の相互関係の定義をDBMSに行わせる

- アプリケーション・ロジックとしてコードを作成する必要がない
  - 事前に制約を使用するか否かの方針決めが必要
  - 使用する場合は、制約違反エラーのハンドリング・ロジックが必要
- データ格納時にDB2により厳格にチェックされる
  - バッチによる大量の更新時や、テスト環境でのテスト・データ作成時に、格納順番やデータの値を考慮する必要あり

## □ 3種類の制約を提供

- 固有(ユニーク)制約
  - 表の1つまたは複数の列に重複する値を指定することを禁止する規則
  - V10.5から情報制約(NOT ENFORCED)とすることが可能

固有?

col1	col2
1	A1

1 INSERT

- 表検査(チェック)制約
  - 表の各行の1つ以上の列について可能な値を指定する規則

col13 < 20?

col13	col12
10	A1

20 INSERT

- 参照制約
  - 外部キーの値が親キーの値として存在する場合のみ有効とする規則
  - 情報制約(NOT ENFORCED)指定が可能

親表に存在する?

col1	col2
1	A1

'A3' INSERT

c1
A1
A2

# 解説

- 制約を利用すると、データの保護や、データ間の相互関係の定義をデータベース・システムに行わせることができます。これにより、アプリケーションでコードを作成し、これらの規則を施行する必要がなくなります。
- 事前に制約を使用するか否かの方針決めを最初に確定させる必要があり、使用するのであれば、制約違反エラーのハンドリング・ロジックは必要となります。
- データ格納時にDB2により厳格にチェックされるため、バッチによる大量の更新時や、特にテスト環境でのテスト・データ作成時に、格納順番やデータの値を考慮するが必要あります。例えば、参照制約が定義されている表については、データのINSERTは必ず親表を先にする必要があります。
- DB2は以下の種類の制約を提供しています。
  - 固有制約
    - 表の1つまたは複数の列に重複する値を指定することを禁止する規則
  - 表検査制約
    - 表の各行の1つ以上の列について可能な値を指定する規則のことです。
  - 参照制約
    - 外部キーの値が親キーの値として現れる場合、または外部キーの構成要素の一部がヌル値の場合のみ有効とする規則
  - 情報制約
    - V8から、情報制約(Informational Constraint)と呼ばれる新しいタイプの制約により、DB2による制約情報チェックの適用/非適用が設定可能になりました。
    - 制約情報を非適用とすることにより、ビジネス・アプリケーションのロジックによってチェックされ、この場合、データベース・マネージャーによってチェックされることはありません。また、オプティマイザーによる制約の利用を活動化、または非活動化させる指定も可能です。
    - 下記のオプションを CREATE または ALTER TABLE で指定します。
      - ENFORCED:DB によって更新操作時、常に制約をチェックさせる。
      - NOT ENFORCED:DB に制約をチェックさせない。また、SET INTEGRITY を使用しても、チェックされません。この場合、実際に制約に違反するデータが入る可能性あるため、アプリケーションでのチェックする必要があります。
      - ENABLE QUERY OPTIMIZATION: オプティマイザーのQuery Rewriteを活動化する。
      - DISABLE QUERY OPTIMIZATION: オプティマイザーのQuery Rewriteを非活動化する。
  - 制約は、CREATE TABLE文およびALTER TABLE文を使用して、表の列に対して定義します。



# 解説

## □ 固有制約の例

- 表t1には、列 (col1, col2, col3) があり、col1 の各データは表で固有でなくてはならない場合、例えば次のように定義します。

```
create table t1 (col1 int not null,  
                  col2 char(10) not null,  
                  col3 int,  
constraint t1_uniq UNIQUE (col1))
```

- このcol1列に重複した値をINSERTすると、SQL0803Nのエラーとなります。

## □ 表検査制約の例

- t1のcol3には、かならず20未満でなくてはならない場合は、以下のように表を定義します。

```
create table t1 (col1 int not null,  
                  col2 char(10) not null,  
                  col3 int,  
constraint col3check CHECK (col3 < 20));
```

- col3に20以上の値をINSERTしようとすると、SQL0545Nのエラーになります。

## □ 情報制約の指定

- 上記で定義した表検査制約と同等ですが、アプリケーションで保証することとし、DB2はチェックしません。

```
create table t1 (col1 int not null,  
                  col2 char(10) not null,  
                  col3 int,  
constraint col3check CHECK (col3 < 20) NOT ENFORCED ENABLE QUERY OPTIMIZATION);
```





ブランク・ページです

# 表の制限

## □ 表設計において考慮すべき制限

- 表に定義できる列数の制限

- 1行のレコードがページをまたがることはできません。

▶ DB2で選択可能な表スペースのページサイズは、4KB/8KB/16KB/32KB

— 行長が入るサイズの表スペースを選択します。

— 入りきらない行がある場合、表の分割を検討します。

- LOBデータでは、表の行長としては、LOB記述子のサイズを使用します。

▶ LOB本体は、表内ではなく、LOB領域に格納します。

— 表内には、LOB記述子と呼ばれる、LOB領域のデータへのポインタを持ちます。

▶ inline LOB

— V9.7からinline lengthを指定して、行長がページサイズ以内になる場合、LOBを表内に持つことができます。

- 拡張行サイズ

▶ V10.5からレコード中にVARCHAR, VARGRAPHIC列を含む場合、全体の行長がページサイズを超えることができる拡張行サイズがサポートされます。

## □ 表スペースのページ・サイズによる制限値

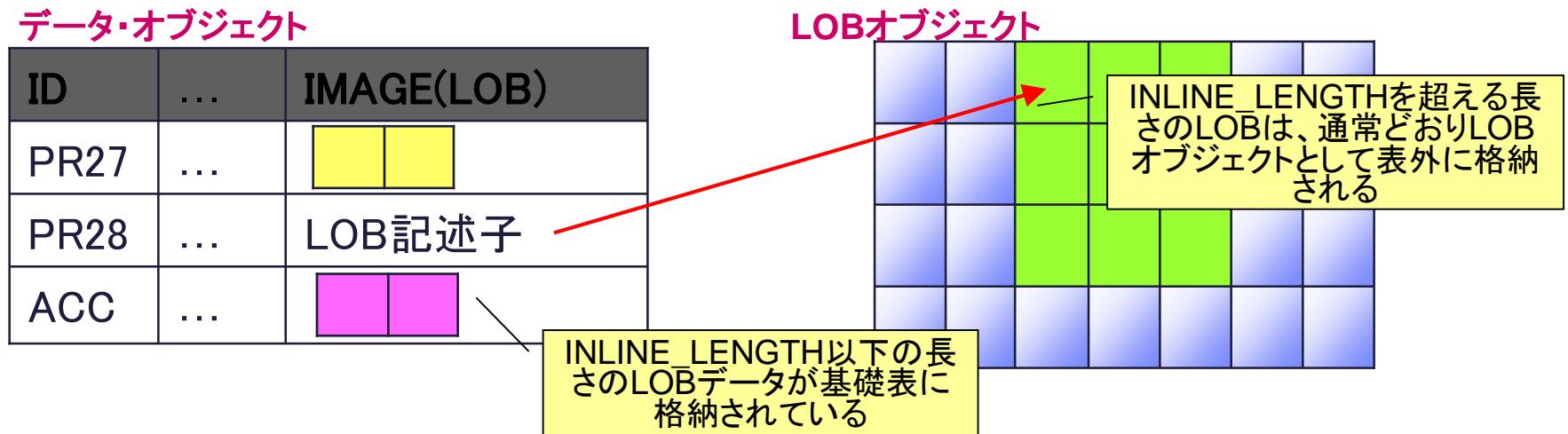
ページ・ サイズ	行長 (bytes)	列数	REGULAR表スペース SMS表スペース		V9.1, V9.5 LARGE表スペース		V9.7以降 LARGE表スペース 一時表スペース
			表容量	行数/ページ	表容量	行数/ページ	
4KB	4005	500	64GB	251	2TB	287	8TB
8KB	8101	1012	128GB	253	4TB	580	16TB
16KB	16293	1012	256GB	254	8TB	1165	32TB
32KB	32677	1012	512GB	253	16TB	2335	64TB



# LOBデータの基礎表への格納(Inline LOB)

## □ インライン格納とは

- 32KB以下のLOBデータをLOBオブジェクトではなく、基礎表へ格納する保持形態
- 表定義に、基礎表へ格納するLOBサイズの上限を指定する(INLINE LENGTHオプション)



- CREATE TABLE時もしくはALTER TABLE時に、LOB列にINLINE LENGTHキーワードを追加する

- 指定できる長さの最大長は、ページサイズごとの行の最大長以下
  - 4KBページでは4005byte
  - 他の列を含めたトータルのレコード長が、制限値以下になるよう調整する
- 表の新規作成時に指定する場合

```
create table T1 (... doc CLOB inline length 10000)
```

- 既存の表のLOB列に対して追加する場合

```
alter table T1 alter column doc set inline length 10000
```



# 参考:LOBデータの基礎表への格納のメリットと考慮点

## □ インライン格納のメリット

- バッファープールへの格納対象になるため、READ/WRITEとも飛躍的に高速化される
- レコード本体への1回のアクセスで処理が完了するため、I/O回数の削減が見込める
- インライン格納されたLOBデータは行圧縮の対象とすることが可能

## □ 考慮点

- 基礎表に入りきらないLOBデータは今までどおりLOBオブジェクトに格納される
  - 各InlineLOBには4バイトのストレージオーバーヘッドが取られる
    - InlineLOBの行最大長は32kページを使用している場合、32673バイトとなる
- LOBをINLINE化することにより、1ページ内に格納できる行数が少なくなるため、LOB列を照会結果に含めない場合は今までよりもパフォーマンスが劣化する可能性がある
- INLINE化されたLOBデータはロギングの対照となる
- INLINE化されたLOBデータを含む表のデータ再編成の処理時間はINLINE化しない場合よりも延びる傾向がある



# 拡張行サイズ

## □ 拡張行サイズ(V10.5以降)

- 行の長さがページ・サイズの最大行長を超える表の作成が可能
  - 対象は、可変長ストリング列 (VARCHAR または VARGRAPHIC) が存在する表
  - 最大行長を超える行データがある場合、可変長ストリング列のサブセットを行(基礎表行)の外に移して、LOB データとして保管する(超えた行のみ)
    - ただし、行サイズは、1048319 バイトを超えることはできない (SQL0670N, SQLSTATE 54010)

## □ メリット

- 32Kを超える行長の表を作成できる
- ほとんどのデータが小さなページに収まるが、一部の大きなデータのためにより大きなページサイズの表スペースが必要とされる場合に、その表だけを別の表スペースにするなどの考慮が不要になる
- 可変長ストリング列の追加、データ長の変更に伴い、最大レコード長を超えた場合においてもページ・サイズが大きな表スペースに表を移動する必要が無い
- ほとんどの行は小さいのに、例外的に長いデータがあるため、列長が長くなつて、結果として行長が長くなるような場合で、LOBを選択するよりもパフォーマンスが向上する

## □ DB構成パラメータの設定

- extended\_row\_sz = ENABLE(デフォルト)

構成パラメータの確認

```
[ym105usr@jonquil /home/ym105usr]$ db2 get db cfg for sample | grep EXTENDED_ROW_SZ
Extended row size support           (EXTENDED_ROW_SZ) = ENABLE
```

4KB以上の列を定義

```
[ym105usr@jonquil /home/ym105usr]$ db2 "create table t2 ( c1 int, c2 varchar(5000) ) in tbs_4k"
DB20000I  The SQL command completed successfully.
```



# 表の構造

## □ 通常表

- 一般的に使用される表で、索引を定義して使用する

## □ パーティション表

- 区分キーを指定してデータを範囲(レンジ)に分けし、ひとつの表を複数のレンジ・パーティションとして分割保存することができる
  - 表の区分(パーティション)化による性能向上
    - 特定パーティションのみアクセス
  - パーティションのアタッチ/デタッチによるデータの高速追加／削除
  - 可用性向上(表スペースレベル)

## □ MDC表

- 定義された次元でデータ配置を管理し、その次元にてクラスター化を実現する  
次元に指定したキーの値が同じであれば、物理的に連続したページに配置される
  - 表のクラスター化による性能向上
    - ロックレベルでのアクセス
  - ロック索引による効率化
  - ロック単位でのレコード削除によるデータの高速削除

## □ レンジクラスター表(RCT)

- 表定義の際にレンジキー列を指定し、すべての行がレンジキー順に並ぶように制御する
- 作成時にすべての行のページ番号とスロット番号が確定する
  - 作成時にすべての行のディスク・スペースが確保される
  - レンジキーに対するイコール条件アクセスでは、結果行を保持しているデータページ1ページのみのアクセス
  - 物理索引を持たずに位置を確定するため、高速

## □ 挿入時クラスタリング表(ITC)

- 挿入タイミングが同じ行を同じ領域に格納
- MDC表と同様に、ロック・ベースの割り振りとロック索引を使用
- スペースの有効活用が可能



□ 一般的には、通常表とし、大量データについて、パーティション表や、MDCの適用を考慮する

# 解説

- 一般的には、通常表とし、大量データについて、パーティション表や、MDCの適用を考慮します。
- 当資料では、V10.5以降でサポートされるカラム・オーガナイズ表については、扱っていません。カラム・オーガナイズ表を検討する場合には、「はじめに(3)」で紹介した「DB2 V10.5新機能テクニカル・ワークショップ資料(2章 BLUアクセラレーション活用 Tips)」を参照してください。
- 当資料では、パーティション表および、MDCについては、設計考慮点の主な項目について紹介していますが、詳しくは以下の資料を参照してください。
  - データウェアハウス(DWH)設計ガイド: 第2章 区分化によるアクセス効率の向上(2)パーティション表
    - [http://www.ibm.com/developerworks/jp/data/library/infosphere/j\\_d-dwhdesign02-2/](http://www.ibm.com/developerworks/jp/data/library/infosphere/j_d-dwhdesign02-2/)
  - データウェアハウス(DWH)設計ガイド: 第2章 区分化によるアクセス効率の向上(3)マルチディメンション・クラスタリング(MDC)表
    - [http://www.ibm.com/developerworks/jp/data/library/infosphere/j\\_d-dwhdesign02-3/index.html](http://www.ibm.com/developerworks/jp/data/library/infosphere/j_d-dwhdesign02-3/index.html)
  - RCT, ITCについては、Knowledge Centerを参照してください。



# パーティション表

□ ひとつの表を複数の区分に分割

□ 古い区分を高速にロールアウト（区分のデタッチ）

- DETACHしたパーティションは独立した表としてアクセス可能
- DELETE処理と比較して、ログ生成量が格段に少ない
- データの移動を伴わないため高速な処理(カタログ情報の更新のみ)

□ 既存データはオンライン状態で、新しい区分をロールイン（区分のアタッチ）

- LOAD済みの表をアタッチし、既存のパーティション表の一部として取り付けることができる

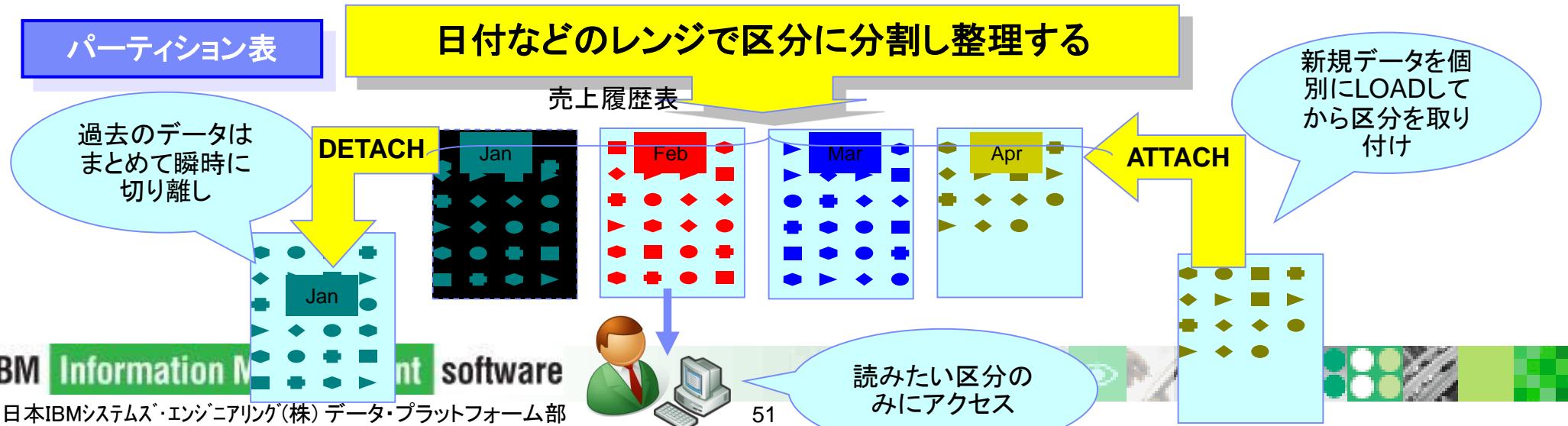
□ 区分限定スキヤンによりアクセス性能向上

- 必要なパーティションのみに限定したアクセスをおこなう

□ 各区分は異なる表スペースに配置可能

- 通常表のサイズ制限を越える巨大な表を作成することが可能

➤ 表のリカバリは、全てのパーティションを同じ時刻にロールフォワードする必要



# パーティション表の設計

## □パーティションの設計

- 特定の範囲(例えば1ヶ月単位、1年単位など)で、まとめて削除や参照を行う場合、その範囲ごとに区分化する

### ★区分キーの候補となる列

- 大規模履歴表における日付列など
  - まとめたレンジ単位の削除がある
  - まとめたレンジ単位の参照がある
  - まとめたレンジ単位のデータ投入がある

保存期間を過ぎたデータを区分ごとまとめて高速にデタッチする

1ヶ月単位などで参照を行う場合、必要な区分のみに限定した効率の良いスキャンが可能

事前に別表へのLOADを完了してから、新規パーティションとしてアタッチ可能

### ★区分化範囲(レンジ)設定のポイント

- 大量データ削除・投入の単位とパーティションの切れ目をそろえる
- 検索範囲とパーティションの範囲を併せる
- 区分化の範囲より細かい条件での検索には、MDCとの併用も考慮

ひと月分をまとめて削除する場合、その範囲をまとめて1データ・パーティションとする。

検索範囲と区分化レンジがマッチしていると効率が良い。

月ごとにパーティション化し、更に日付けごとにMDCロック化など。

# 参考: UNION ALL VIEWによる表分割

## □ UNION ALLビューとは

- 複数の結果表を組み合わせて新たな結果表として定義されたVIEW
- UNION ALLを指定すると結果は該当表のすべての行から構成される

## □ UNION ALL VIEWによる表分割

- 表サイズの制約を越える大規模表を実現するために、パーティション表ではなく、UNION ALL VIEWを使うことも可能
- 履歴データ等のレンジ・パーティションを実現
- データ削除を分割された表単位のIMPORT REPLACEなどで実行できる
- REORG, RUNSTATSなどの運用を表毎に個別に行える

## □ VIEW定義の方法

- CREATE VIEW文中のSELECT文で、WHERE文節によって値を制限
  - このVIEWに対するINSERTは不可
- 表に対するCONSTRAINTによって値を制限
  - INSERTを使うためにはCONSTRAINTが必要(V8以降)

## □ UNION ALL VIEW 使用上の注意点

- アクセス・パスを充分確認した上で使用する
- VIEWへのSELECTを表へのSELECTに変換するのはオプティマイザーである。
  - オプティマイザーが解らない場合は、表を特定できない。その場合、全ての表にアクセスしてしまう
  - 表を特定できない例
    - 照会の条件が、CHECK制約やVIEW定義のWHERE条件に合致しない
    - CHECK制約にMONTHなどの関数を使用...等
- オプティマイザーはVIEW中のSQL文を展開してから評価するので、展開後のSQL文が長くなる可能性がある
  - ステートメント・ヒープ、アプリケーション・ヒープがより多く必要
  - SQL文の長さの制限(64K)を超える可能性がある
  - 長すぎるSQL文でオプティマイザーがあきらめて、単純なアクセスパスに走る可能性がある
- UNION ALL VIEWへの更新系処理はオーバーヘッドが生じる
  - 各表へのチェック処理
  - 更新処理は、なるべく実体の表に対して直接処理を行う



# 参考：UNION ALL VIEW と パーティション表の比較

	パーティション表	UNION ALL VIEW	備考
特定レンジの除去	<ul style="list-style-type: none"> <li>特定パーティションのDETACH(パッケージはINVALIDになる)</li> </ul>	<ul style="list-style-type: none"> <li>特定レンジの表をDROP(パッケージはINVALIDになる)</li> <li>VIEWの再定義</li> <li>権限の再付与</li> </ul>	<ul style="list-style-type: none"> <li>UNION ALL VIEW: <ul style="list-style-type: none"> <li>手順が煩雑</li> <li>カタログに対するロックが多い</li> </ul> </li> </ul>
特定レンジの追加	<ul style="list-style-type: none"> <li>特定パーティションのATTACH(パッケージはINVALIDになる)</li> <li>SET INTEGRITY実施</li> </ul>	<ul style="list-style-type: none"> <li>特定レンジの表を追加</li> <li>制約の追加</li> <li>VIEWの再定義(パッケージはINVALIDになる)</li> <li>権限の再付与</li> </ul>	<ul style="list-style-type: none"> <li>UNION ALL VIEW: <ul style="list-style-type: none"> <li>手順が煩雑</li> <li>カタログに対するロックが多い</li> </ul> </li> <li>パーティション表: <ul style="list-style-type: none"> <li>SET INTEGRITY処理に時間がかかる</li> </ul> </li> </ul>
ユニーク索引	<ul style="list-style-type: none"> <li>サポートされる</li> </ul>	<ul style="list-style-type: none"> <li>表をまたがるユニーク索引はサポートされない</li> </ul>	
アクセスパス	<ul style="list-style-type: none"> <li>シンプルなアクセスパス</li> <li>必要なパーティションに限定したアクセス</li> </ul>	<ul style="list-style-type: none"> <li>稀にアクセスパスが複雑になる</li> </ul>	UNION ALL VIEW: 場合によって、コンパイルに時間がかかる
ユーティリティー	<ul style="list-style-type: none"> <li>RUNSTATS/REORGは表全体(全区分)に対して行われる。</li> <li>BACKUP/RESTOREは個別に実施可能(表スペースを分割した場合)</li> </ul>	<ul style="list-style-type: none"> <li>各レンジの表に対してそれぞれユーティリティーを実行することができる。</li> </ul>	パーティション表: 特定区分のREORGを実施したい場合は一度DETACHした後、REORG、再ATTACHが必要

# 多次元クラスタリング(MDC)表

## □ 複数属性の値でデータを分類して、自動的に格納する機能

- 単一属性のクラスターでは実現できなかった「2008年10月」の「DB2」の「東京」というような複数の属性をもつクラスター
- 次元: クラスターの切り口
  - 例の年月、地域、製品は次元列
- セル: 次元列の組み合わせが同一の値をとるもの
- 表スペースはエクステント(=ブロック)単位にアロケートする
  - 同一ブロックには、次元列の組み合わせの値が同じレコードのみ挿入される

## □ クエリー要求に対する高いパフォーマンスを提供

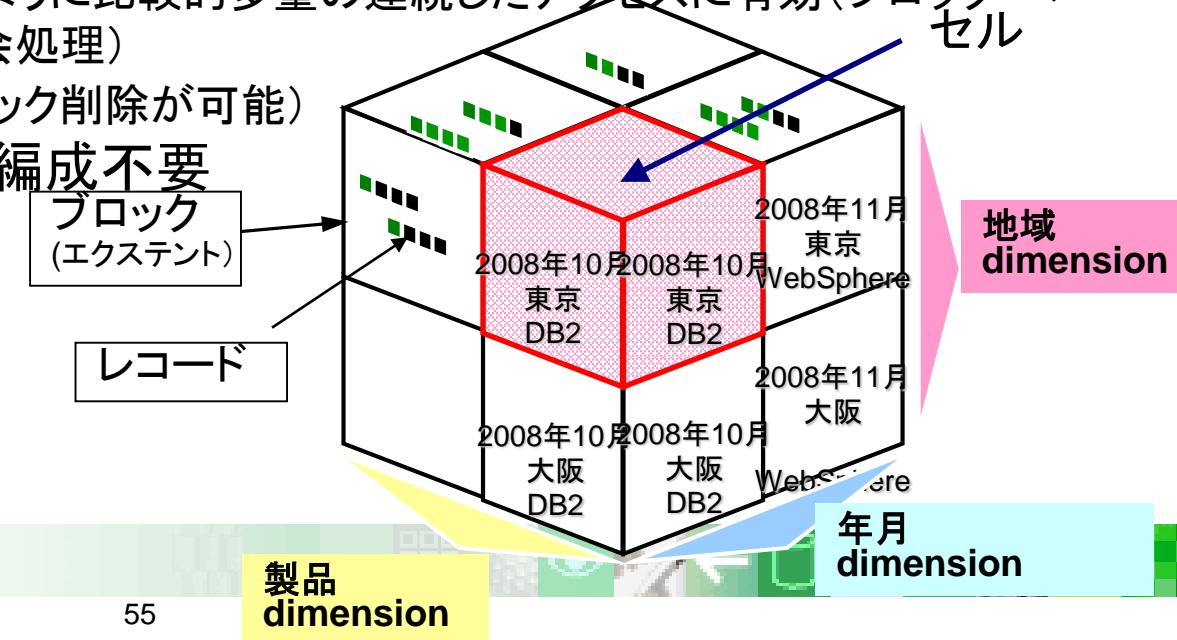
- 次元別検索のパフォーマンス向上
  - 同じ値のレコードは同じブロックに存在する
  - 範囲を指定した検索などのように比較的多量の連続したアクセスに有効(ブロック・ベース(BID)の索引を使用した照会処理)
- 削除のパフォーマンスアップ(ブロック削除が可能)

## □ データ並べ替えを目的とした再編成不要

MDC表の作成例:

```
CREATE TABLE TB_MDC (
  年月 CHAR(7),
  地域 CHAR(10),
  製品 VARCHAR(10)
)
```

```
ORGANIZE BY DIMENSIONS (年月, 地域, 製品)
```



# MDC表の設計(次元キーの選択)

## □ 次元キーの選択

- 特定の値(例えば、日付、地域、製品単位など)で、まとめて参照や削除を行う場合、その属性でのクラスタリングを行なう

### ★ 次元キーの候補となる列

- 大規模履歴表、売上明細表における日付列、顧客表の地域など
  - まとめた次元単位の参照がある
  - まとめた次元単位の削除がある

同じ値のレコードは同じセルに存在するため、範囲を指定した検索などのように比較的多量の連続した照会に有効

大規模履歴表における日付単位の消しこみなどでは、セル単位の高速削除が可能

### ★ 次元設定のポイント

- 検索述部、GROUP BYにて頻繁に指定される列は有力候補
- 1エクステントを満たすのに十分な重複値のある列を選択
- ユニーク性のあまり低くない列を次元キーに設定したい場合には、生成列との併用も検討する
- 次元キーを、レコード索引に指定する際には、索引の第一列に指定するのは避ける

ロック索引による照会は、比較的多量の連続した照会に有効

ユニーク性の高い列を選択したり、極端に多くの数の次元列を指定すると、ディスクの浪費につながる(I/Oの効率も下がる)

生成列を使用することで、ユニーク性を落とすことが可能

カーディナリティーの低い列が第一列だとアクセスプランが不安定になりやすい



# 表に対するその他の要件の確認

## □セキュリティ要件

- ユーザーやロールごとのアクセス制御
  - 従来: ユーザーに対して見せる行や列で構成されたViewを作成し、View経由でアクセスさせることで制御する
    - Viewの作成が煩雑
    - ユーザーごとにViewを切り替える必要がある
  - RCAC(V10.1以降)
    - CREATE PERMISSION: ユーザーごとにアクセス許可を条件文で定義可能
    - CREATE MASK: 列の値をマスクして表示されるように定義することが可能

## □タイムトラベル検索

- 年月日、時間を指定した検索、指定した時間断面での集計などの要件がある場合
  - 従来: アプリケーションで、レコード更新時間を持つことによって時間に関連した検索を実現するように設計する
  - テンポラル表(V10.1以降)
    - DB2の機能で、時間的要素を持つデータを格納/蓄積
- 3種類のテンポラル表を提供
  - システム期間テンポラル表
  - アプリケーション期間テンポラル表
  - バイテンポラル表



# 解説

- 表に対してセキュリティや、タイムトラベル検索の特殊要件がある場合は、DB2の機能で実現できるかどうかを検討します。
- これらの機能についての詳細は、以下の資料を参照してください。
- DB2 10 for LUW 新機能ワークショップ資料
  - [http://www.ibm.com/developerworks/jp/data/products/db2/db2\\_10-workshop/](http://www.ibm.com/developerworks/jp/data/products/db2/db2_10-workshop/)



# RCACによる行および列のアクセス制御

□ ユーザー/ロールに応じて行/列のアクセス制御を行うことが可能

- アプリケーションから実行するSQLを変更することなく、ユーザーにあわせた結果内容を返す
- Viewを切り替える必要がない

患者表: patient

SIN	USERID	ADDRESS	PHARMACY	PCP_ID
123 551 234	MAX	First St.	hypertension	LEE
123 589 812	MIKE	Long St.	diabetics	JAMES
123 119 856	SAM	Big St.	codeine	LEE
123 191 454	BOB	Good St.	influenza	JAMES
123 456 789	DOUG	123 Some St.	hypertension	LEE

行に対する参照可否権限  
(PERMISSION)を設定する

アクセス可能な  
レコードのみを返す

マスキングを  
設定する

開示したくない  
データをマスキング

`SELECT * FROM patient`

`SELECT * FROM patient`

SIN	USERID	ADDRESS	PHARMACY	PCP_ID
123 191 454	BOB	Good St.	influenza	JAMES



Dr. Lee  
Physician



Bob

Patient

SIN	USERID	ADDRESS	PHARMACY	PCP_ID
XXX XXX 234	MAX	First St.	hypertension	LEE
XXX XXX 856	SAM	Big St.	codeine	LEE
XXX XXX 789	DOUG	2 Some St.	hypertension	LEE

# アクセス制御の作成方法と手順

## □行レベルのアクセス制御(CREATE PERMISSION)

- 誰が、どの行にアクセスできるか

```
create permission cust on db2admin.sales for rows where
  ( verify_role_for_user(SESSION_USER, 'ACCOUNTANT') =1 )
OR
  ( verify_role_for_user(SESSION_USER, 'CUSTOMER') =1 )
enforced for all access enable

alter table db2admin.sales activate row access control
```

## □列レベルのアクセス制御(CREATE MASK)

- 列(データ)に対して、誰に、どのようなマスキングを行うか

```
create mask account_mask on db2admin.sales for column account return
case when verify_role_for_user(SESSION_USER, 'ACCOUNTANT')=1
then 'XXXX-XXXX'
else account
end enable

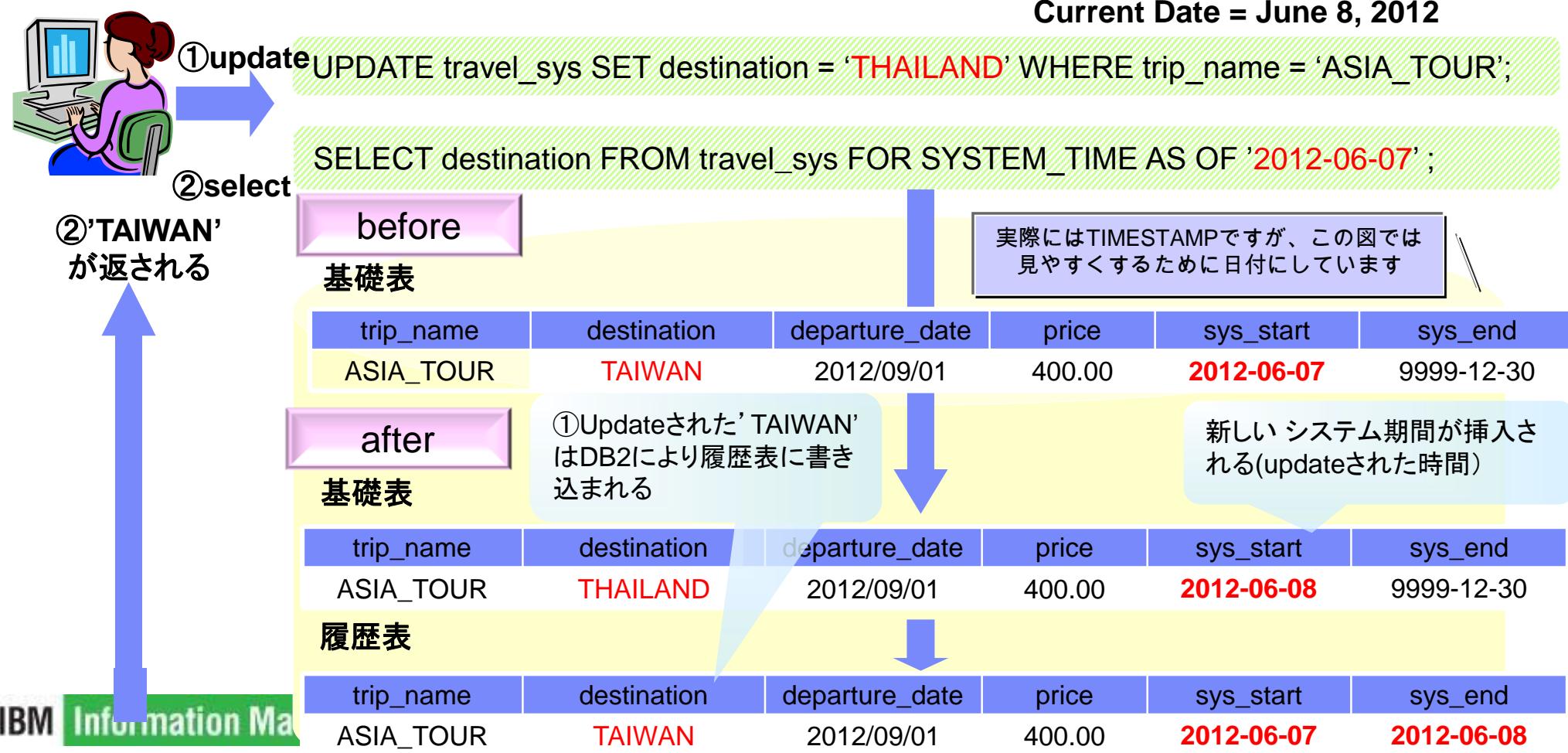
alter table db2admin.sales activate column access control
```



# システム期間テンポラル表(イメージ図)

基礎表にUpdate、DeleteのSQLが流れると、beforeイメージの情報をDB2が履歴表に挿入してくれる

- 例:destinationを' TAIWAN'から' THAILAND'に変更した場合、元の' TAIWAN'の情報は履歴表に保持される
- 例:基礎表のdestinationは' THAILAND'だが、2012年6月7日を指定してselectすると' TAIWAN'が戻される



# アプリケーション期間テンポラル表(イメージ図)

□ アプリケーション期間テンポラル表に任意の時間を指定してselectすることにより、その時間(BUSINESS\_TIME)の情報を返してくれる

- 例:任意の時間①、②を指定してselectすることで、返ってくるpriceの結果が異なる

① SELECT price FROM travel\_app FOR BUSINESS\_TIME AS OF '2012-08-15'  
WHERE trip\_name = 'JAPAN\_TOUR';

② SELECT price FROM travel\_app FOR BUSINESS\_TIME AS OF '2012-11-15'  
WHERE trip\_name = 'JAPAN\_TOUR';

Current Date = May 15, 2012



travel\_app表

trip_name	destination	departure_date	price	bus_start	bus_end
JAPAN_TOUR	KYOTO	2012-12-31	1500.00	2012-06-08	2012-08-01
JAPAN_TOUR	KYOTO	2012-12-31	1000.00	2012-08-01	2012-09-01
JAPAN_TOUR	KYOTO	2013-12-31	1500.00	2012-09-01	2013-01-01



ビジネス期間の境目をselectすると、ビジネス開始列が該当する

例:ビジネス期間 '2012-08-01'を指定して、priceを照会すると'1000.00'が返される

# その他の表属性

## □APPENDモード

- 通常の表では、レコードのINSERT時には、DELETE済みの空き領域が再利用される
- APPENDモードをONにすると、INSERT時に表内の空きページを検索することなく、最終ページにレコードを追加させることが可能
- レコードが単調増加する特性の表にINSERTする場合、大量のINSERTが行われる場合、パフォーマンスが向上する
- DELETEによる削除レコードの空き領域は再利用されないため、別途、表自体の再作成(もしくはtruncate)、または再編成などの運用により、表容量を適正に保つ仕組みの検討が必要となる
- クラスター索引のある表には適用不可

## □ページの空き領域(PCTFREE)

- 離れたページにデータが挿入されると、パフォーマンスに影響を与える為、予めページ内にフリー・スペースを残す様に設定することができる
- PCTFREEを検討する表
  - クラスター索引を持つ表
  - 可変長列の更新がある表
- ALTER TABLE 表名 PCTFREE 数値
  - PCTFREE:0~99 (デフォルト:-1)
- データのロード、およびREORG TABLE時に、指定されたサイズのフリー・スペースをページ内に残す



# 解説

## □ 行の削除ではスペースは解放されません。

- 削除してもDiskの使用率は変わりません
- 削除された領域は、表に新たな列が挿入された場合に再利用されます。
  - APPENDモードの指定により再利用させないことも可能です。
  - 削除/更新などによるスペースのフラグメンテーション解消およびレコード並びの保持したい場合は、REORG運用が必要です。
  - 定期的な大量挿入、大量削除がある表では、REORGを必要としないMDCやパーティション表を検討します。
- APPENDモード
  - APPENDモードの表に対しては、INSERT時に表内の空きページを検索することなく、最終ページにレコードを追加します。
  - レコードが単調増加していく特性の表にINSERTする場合や大量のINSERTでは、パフォーマンスが向上しますが、DELETEによる削除レコードの空き領域は再利用されないため、別途、表自体の再作成(もしくは、0件)、または再編成などの運用により、表容量を適正に保つ仕組みの検討が必要となります。
  - クラスター索引のある表には適用できません。

## □ ページの空き領域(PCTFREE)

- 離れたページにデータが挿入されると、パフォーマンスに影響を与える為、予めページ内にフリー・スペースを残す様に設定することができます。
- ALTER TABLE 表名 PCTFREE 数値
  - PCTFREE:0~99 (デフォルト:-1)
- データのロード、およびREORG TABLE時に、指定されたサイズのフリー・スペースをページ内に残します。
- PCTFREEの設定が有効な例
  - クラスター索引を持つ表(データの挿入時に、索引順とデータの並び順を同じにするようにデータを格納しようと試みる)に対して、業務上の観点から、データの追加、削除処理の頻度にも留意し、ページの空き領域(PCTFREE)の設定の検討が必要です。データの挿入が多い場合、索引順序にデータを配置しようとしても、ページ内に収まらない場合があります。
  - 可変長の属性の列項目を持つ表において、その列項目に対して更新がある場合、空き領域を設けることを検討します。ページ内に空き領域がないと、更新後の可変長列が元の領域に収まらない場合は、別のページに格納されることになり、IO効率が低下します。
  - オンライン中の更新がない、または、クラスター索引を持たない列項目の属性が固定長のみで定義されている表については、空き容量は特に必要ありません。
  - 圧縮された表に対して頻繁に更新があり、行のサイズが大きくなる場合、PCTFREEの値を増やすことを検討します。



# 索引の設計

## □ 索引の目的

- 照会処理の処理効率を高める
  - アクセス・パスにおける索引の使用による効率のよいデータへのアクセス
- 行のユニーク性を維持する
  - ユニーク索引
- データの並び順を索引順に維持することにより、データ・アクセスの効率を向上させる
  - クラスター索引

## □ 設計手順

- パフォーマンス改善を目的とし、繰り返し行う必要がある
- 索引候補の検討
- 索引数の検討
- 索引候補の取捨選択

## □ 索引の物理定義と検証

- 索引が有効に利用され最適なアクセスパスになっているか
- 意図した索引を使用しているか
- メンテナンス負荷を軽減するため、使用されていない場合にはDROP SYSCAT.PACKAGES(静的SQL)、または、EXPLAINツールで確認



# 解説

- 表に作成する索引は、本の索引と同様の機能を果たします。
- 索引の第一の目的は、データをアクセスする際の処理効率を向上させることです。余計な入出力することなく、最短の方法で目的のデータにたどりつくには、索引は非常に有効です。
- ユニーク索引を作成した際には、索引のキー列のユニーク性を保証する機能を使用可能です。
- クラスター索引
  - クラスター索引を作成すると、データの挿入時に、索引順とデータの並び順と同じにするようにデータを格納しようと試みます。
  - データを索引の列項目の値順に読み込む場合、I/O回数が軽減され、処理効率が向上します。
  - クラスター索引を作成する場合、データが格納されるページに空きスペースを準備する必要があります。
  - 索引の列項目の値が更新される(更新があった場合、索引順に再格納は行わないため、再編成の必要性を検討する必要がある)場合や、検索結果が常に1件となる照会処理が頻繁に行われる場合は、作成してもメリットはありません。
- 索引の設計手順
  - パフォーマンス改善を目的とし、内部設計から統合テストの局面まで、繰り返し行う必要があります。
  - 索引候補の検討
    - 主キーや外部キーなどは、データの意味から索引候補として決定可能であるため、外部設計後に可能な作業です。一方、その他の2次索引については、具体的なSQL文を元にアクセス・プランを検討し、候補の洗い出しを行います。
  - 索引数の検討
    - 索引数が増えると、索引のメンテナンス負荷が高くなり、処理効率が低下します。従って、トランザクションの内容により、索引数を制限して作成する必要があります。
  - 索引候補の取捨選択
    - どの列に索引を付与するか、最適なアクセス・プランを検討し、本当に必要と思われる索引を選択します。
- 索引の物理定義と検証
  - 索引が有効に活用されているかを確認し、使用されていない場合には、DROPする必要があります。索引が存在することによるメンテナンス負荷を軽減するためです。静的SQLプログラムについては、SYSCAT.PACKAGESで確認できます。また、動的SQLプログラムについては、EXPLAINツールで確認します。



# 索引候補の検討

## □ ユニーク索引が必要か

- ユニーク性の維持が必要な場合: ユニーク索引
- 参照の整合性が必要な場合: 主キー
  - CREATE TABLE実行時に、自動的に主キーに対する昇順のユニーク索引が作成される
    - 索引名 : SQL+タイムスタンプ+番号
    - 索引スキーマ: SYSIBM
    - CONSTRAINTで制約名をつけると管理が容易

## □ 外部キーに索引をつける

- 結合列になる可能性が高い列に索引があると、処理効率は良い

## □ 条件句(WHERE句に現れる述語)の中で頻繁に使用される列を検討

- 結合列
- 探索条件の列
  - ANDで結ばれた等号述語
  - 範囲指定の述語(BETWEEN, 不等号述語)
- ソート列(DISTINCT, ORDER BY, GROUP BYで指定された列)
- 式ベース索引
  - 索引作成時に式を含む索引キーを定義できる(V10.5以降)
    - 表に格納されていないデータの索引を作成可能
      - △ 式の結果が索引に格納される
      - △ 式が含まれる述部を使用するSQLでパフォーマンスが向上
        - ◆ 照会時に、すでに計算済みの索引の値を使用できる
        - ◆ 特定の計算や操作が多い場合に、対応する索引を使用すると有利
  - 索引のみのアクセスを目的とした索引
    - INCLUDE列つきのユニーク索引



# 解説

## □ 基本的な索引候補

- まず、ユニーク索引が必要かどうかを検討します。ユニーク性を維持しなければならない列が存在するのであれば、ユニーク索引が必要です。
- 主キーの必要性を検討します。他の表の列と整合性を保たなければならない、マスターとなる列が存在するのであれば、表に主キーを設定します。基本キーの設定は、表の作成(CREATE TABLE)時に指定するか、または、表の変更(ALTER TABLE)で指定します。
- 外部キーがある場合、検索条件の結合列となる可能性が高いため、索引の候補になります。

## □ さらに、その他の2次索引候補を検討します。

- 候補になる列は、条件節での登場回数が多い列です。
- また、ソートの対象となる列も候補になります。
- FOREIGN KEY(外部キー)が定義されている列項目
- レコードの探索条件として、「=」述部に指定されることの最も多い列項目、もしくは、最初のキーとしての個別の値が最も多い列
- 表を結合するときに使用するすべての列

## □ V10.5以降、式ベース索引として、索引作成時に式を含む索引を定義できます。

- 式ベース索引を使用することにより、表に格納されていないデータの索引を作成することができ、式が含まれる述部を使用するSQLでのパフォーマンスが向上します。特定の計算や操作が多い場合に、対応する索引を使用する場合に、有利となります。

## □ INCLUDE列つきのユニーク索引

- ユニーク索引の列として、ユニークではない列を含むことが可能
- 目的：索引のみのアクセスによるパフォーマンス向上
- 冗長な索引を作成しない
  - 表にアクセスすることなく、索引のみで照会処理要求を満たすことができます。これをindex-only accessといいます。
  - INCLUDE列を指定してユニーク索引を作成することにより、データページのアクセス頻度が軽減されます。
  - 索引キーの一部の列については、ユニーク性を保持する
  - ユニークではない列については、ユニーク性の検査が発生しない
- 作成方法：CREATE UNIQUE INDEX 索引名 ON 表名 (列名) INCLUDE (列名)
  - 複数列の指定が可能
  - ユニークではない列については、索引順(ASC, DESC)の指定は無効



# 参考:INCLUDE列つきのユニーク索引の使用例

## □ 处理するSQLステートメントの例:(employee\_idに主キーがある)

➢ SELECT employee\_id, mgr\_id FROM my\_employee WHERE employee\_id = 78379 ;

## □ INCLUDE列を使用しない例:2つの索引を作成・維持する必要あり

- 1. 表の作成

➢ CREATE TABLE my\_employee ( employee\_id integer not null, mgr\_id integer, phone\_no integer, hire\_date date, PRIMARY KEY (employee\_id));

— DB2は主キーには自動的にユニーク索引を作成する

- 2. 索引のみのアクセスのために、索引を作成する

➢ CREATE INDEX col\_index ON my\_employee (employee\_id, mgr\_id) ;

## □ INCLUDE列を使用する例:1つの索引だけで INDEX ONLY ACCESS

- 1. 表の作成

➢ CREATE TABLE my\_employee ( employee\_id integer not null, mgr\_id integer, phone\_no integer, hire\_date date) ;

- 2. INCLUDE列つき索引の作成

➢ CREATE UNIQUE INDEX my\_index on my\_employee (employee\_id) INCLUDE (mgr\_id) ;

- 3. 主キーの作成

➢ ALTER TABLE my\_employee add PRIMARY KEY (employee\_id);

— 既存の索引が主キーになる



## 参考：索引列の最大数と索引キーの最大長

- 索引列の最大数と索引キーの最大長は、V8以前とV9以降で異なる。

バージョン	ページ・サイズ	索引列の最大数	索引キーの最大長 (bytes)
V8まで	4/8/16/32KB	16	1024
V9以降	4KB	64	1024
	8KB	64	2048
	16KB	64	4096
	32KB	64	8192



# 索引数の検討

## □ 索引数の目安: 表あたり5個以下が望ましい

- むやみに索引を作成することは、ディスクを無駄に消費し、負荷を増やすことになる
  - オンラインでの更新処理環境: 1~2個
  - 照会のみの環境: 5個以上作成してもよい
  - オンラインの更新処理と照会処理の混在した環境: 2~5個

## □ 更新処理時には索引のメンテナンスが必要となり、負荷が発生する

- INSERT処理では、索引列の追加処理が発生
- DELETE処理では、索引列の削除処理が発生
- 索引列に対するUPDATE処理では、索引列の変更処理
- 索引のスプリット処理
  - 行のランダムな追加を予想し、事前にページにフリースペースを確保しておく(PCTFREE)

## □ 他の負荷

- クラスター索引がある表へのINSERTは、索引順を極力保持するようにデータを格納する
- ディスク・スペース使用量の増加
- LOAD、REORG時の索引の再作成の負荷
- 索引の追加によりプログラムのPREPARE時間が増加
  - 静的SQLではBIND時、動的SQLでは実行時の時間が増加する
  - 検討すべきアクセスパスの組み合わせが増加する

## □ 前方スキャン、逆方向スキャンの両方が必要な場合、ALLOW REVERSE SCANSオプションを指定して索引を作成する(二つの異なる索引を作成する必要はない)

- V9以降、新規で作成される主キー、ユニーク・キー、または索引(拡張索引は除く)は、デフォルトでALLOW REVERSE SCANS設定になる。



# 解説

- 索引数が増えると、照会のパフォーマンスは向上する可能性がある一方で、索引のメンテナンス負荷が高くなり、更新の際の処理効率が低下します。従って、トランザクションの内容により、索引数を制限して作成する必要があります。
- 複数の索引を作成する前には、ディスク・スペースや処理時間への影響も留意し検討する必要があります。
- 索引のスプリット処理
  - 索引のリーフ・ページが満杯になった時点で、さらにそのページにデータが格納される必要があったとき、索引ページは分割され、2つのリーフ・ページになります。分割されるデータの割合は、満杯になった索引ページが索引構造内でどの場所にあつたかにより異なります。
- 索引ページのフリースペース(PCTFREE)
  - 表にランダムにデータをINSERTするようなアプリケーションにより、索引のリーフページが頻繁にスプリットされてしまうのを防ぐために有効です。
  - CREATE INDEX ステートメントのオプションです。また、LOAD時にMODIFIED BY INDEXFREESPACE=xにより、再指定することも可能です。
  - フリースペースが確保されるタイミングはLOADおよびREORG時です。
- 以下の場合には、フリースペースは必要ありません。
  - INSERT,DELETEがない
  - 索引キーの更新がなく、固定長列のみなので更新による行のネスティングが発生しない
  - 照会のみである
- LOADに関する考慮点
  - LOADの前に索引作成を行っておいたほうが、LOAD後に索引を作成した場合と比較すると、合計時間が短くてすみます。また、事前にユニーク索引を作成しておいた場合には、LOAD時にユニーク性の検査が行われます。
  - LOADの前に索引を作成しておく場合には、索引を作成するための一時領域を確保しておく必要があります。メモリー領域としては、DB構成パラメータSORTHEAPIに設定された容量のソート領域が使用されます。また、ディスク領域としては、一時表スペースが使用されます。



# 索引候補の取捨選択

## □ 索引の作成を避けた方がよい列

- 可変長列
  - 索引のメンテナンスの負荷が高い
- 統計情報のCOLCARDの値が小さい(重複値の多い)列
  - (例) フラグ(0 or 1)や区分など
  - SYSCA.COLUMNSのCOLCARD列: ユニークな値の数
  - オプティマイザ-が索引を選択しない
- サイズがごく小さい表の列
  - アクセス・パスの決定時に索引が有効とみなされず、表スキヤンになる可能性が高い

## □ 複合列索引の考慮点

- 複合列索引の全ての列が等号で使用されるものは有効
- 索引列は、最も頻繁に等号で指定される列か、最もユニーク性の高い列から順に指定する
  - 最初の索引列で結果行を大幅に絞り込める索引は使用されやすい
- 完全にマッチングする索引を優先する
  - (例) 索引1(col1,col2,col3) と 索引2(col1,col2) がある場合で、条件がcol1=x and col2=x であれば索引2を優先
    - 索引2はFULLKEYCARDが有効であり、かつ、キー長が短いのでバッファーヒット率が高い
- 統計情報でFULLKEYCARDが大きいものは有効
  - SYSCAT.INDEXES(FULLKEYCARD): 列全体でユニークな値の数





ブランク・ページです

# パーティション表の索引

## □ パーティション表の索引

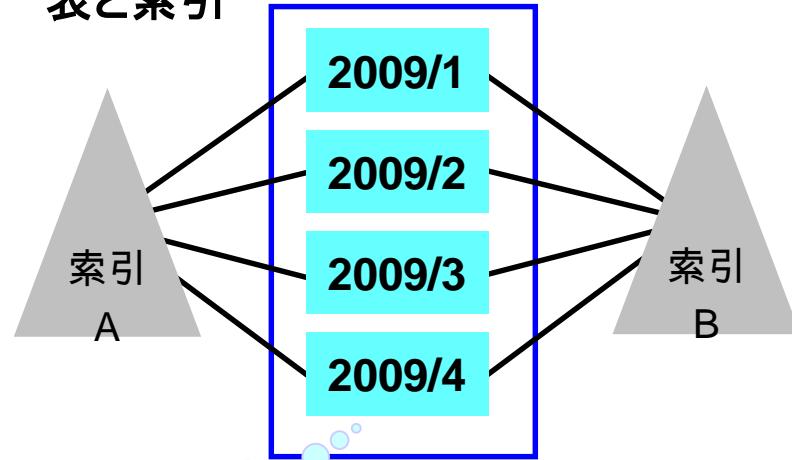
- グローバル索引：表全体で一つの索引オブジェクト
- ローカル索引：V9.7より登場。パーティションごとに分割された索引オブジェクト

## □ 索引をどちらで作成するかは、索引作成時に指定

## □ ローカル索引により

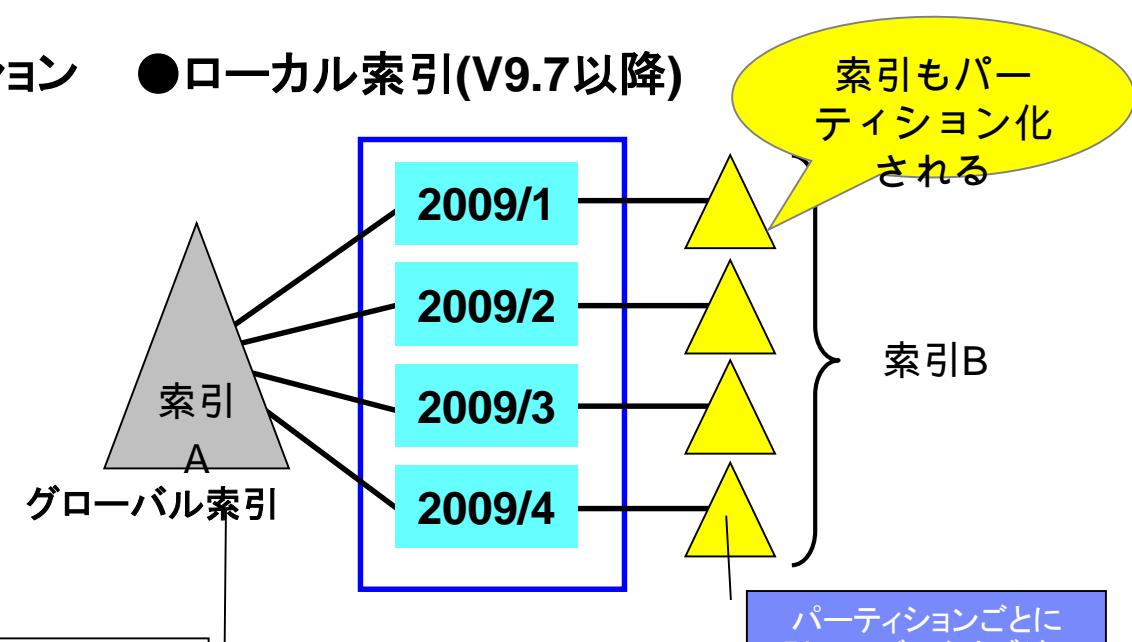
- 表データだけでなく、索引もパーティションごとに別々の表スペースに格納できる
- アタッチ時の(既存の索引に対する)索引メンテナンスが不要となり、アタッチしたパーティションにアクセスできるようになるまでの時間が短縮
- デタッチ時の索引メンテナンスが不要
  - ▶ 注: 索引メンテナンスはバックグラウンドで非同期に行われるためグローバル索引でも表へのアクセスは可能

### ● グローバル索引(V9.5までのパーティション表と索引)



区分化されるのは  
データ部分のみ

### ● ローカル索引(V9.7以降)



# パーティション表への索引の作成

## □ローカル索引の作成（索引作成時のデフォルト動作※）

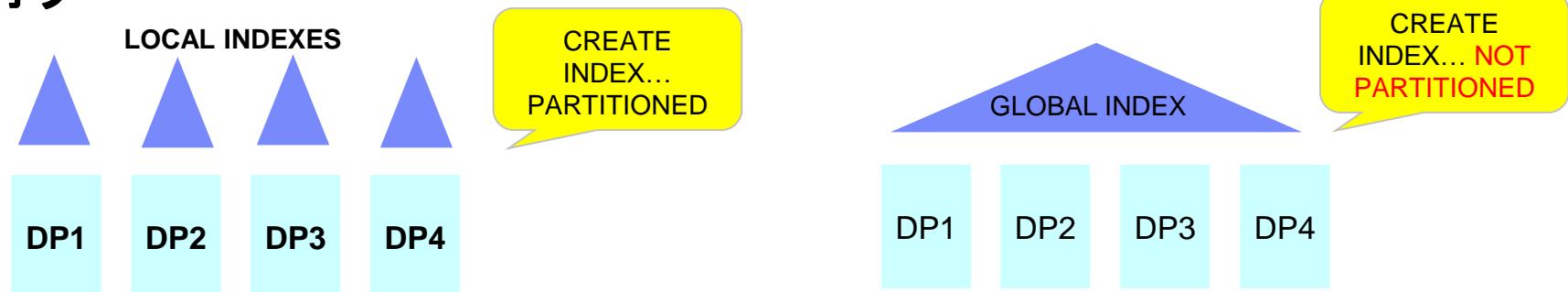
- ユニーク索引の場合、ローカル索引とするためには、すべてのパーティションキーを含む必要がある

**CREATE INDEX <索引名> ON <表名> ... PARTITIONED**

## □グローバル索引の作成（V9.7以前と同様のグローバル索引）

**CREATE INDEX <索引名> ON <表名> ... NOT PARTITIONED**

## □ローカル索引の確認はSYSIBM.SYSINDEXPARTITIONSカタログ表で行う



- ※ 下記の場合以外、デフォルトでローカル索引が作成される。
  - ユニーク索引でかつ、全てのパーティション列を索引に含んでいない場合
  - XML索引の場合



# MDC表の索引(1)

## □ MDC表には、2種類のブロック索引が、自動作成される

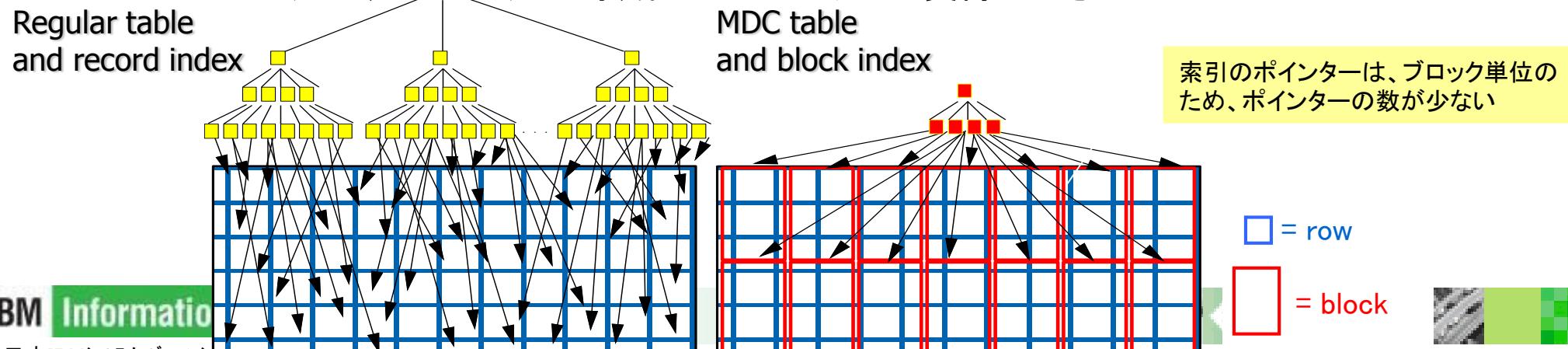
- 次元ブロック索引
  - MDC表の各次元に対して1つずつ作成される
- 複合ブロック索引
  - 1つのMDC表に対して1つ作成され、各次元に含まれる全ての列をキーに持つ

```
CREATE TABLE TB_MDC (年月 CHAR(7), 地域 CHAR(10), 製品 VARCHAR(10))
ORGANIZE BY DIMENSIONS (年月, 地域, 製品)
```

年月の次元ブロック索引、地域の次元ブロック索引、製品の次元ブロック索引、年月 & 地域 & 製品の複合ブロック索引という計 4つのブロック索引が作成される。

## □ ブロック索引は、各ブロックへのポインター(BID)を保持する索引

- レコード索引(RIDを保持する)と比べて、索引のサイズがとても小さい
- 照会処理時の索引スキャンの負荷が小さい
- INSERT処理、DELETE処理時、索引のメンテナンス負荷が小さい



## MDC表の索引(2)

### □ MDC表に、レコード索引を定義することも可能

- ブロック索引と通常のレコード索引のANDing, ORingも可能

### □ 索引のメンテナンスは、2つのクリーンアップのタイプから選択

- 即時(Immediate)索引クリーンアップ・ロールアウト(デフォルト)
  - レコード索引のメンテナンスがすべて完了してからDELETE完了通知を戻す
  - 各索引のキー削除のために、行をスキヤンする
  - 索引のロギングには、通常のDELETE処理と違いはない
  - 一時点で処理される索引は、1つ
- 非同期索引クリーンアップ(AIC) 据え置き(Deferred)索引クリーンアップ・ロールアウト
  - レコード索引のメンテナンスは、レコードのDELETE完了通知後にバックグラウンドにて実行する(V9.5~)
  - 索引のロギングは、索引ページ単位のため、ログ量が減少する
  - メンテナンスのため、索引ごとに非同期バックグラウンド・プロセスが起動されるため、クリーンアップの処理時間が削減される



# 索引に関するその他の検討事項

## □ 索引ページの分割およびデフラグ

- PCTFREE: 索引リーフページに確保する空き領域の割合
- LEVEL2PDTFREE: 索引のレベル2ページに確保する空き領域
- PAGE SPLIT: 索引ページを分割する方法
- MINPCTFREE: 動的デフラグを実行するかどうか、動的デフラグを実行する基準の設定

## □ 索引用の表スペース

- 索引用の表スペースを作成する(DMSのみ)
  - 表データとは別の物理ディスクに配置することによる並列I/Oが期待できる
  - 索引だけ早いディスクに格納することができる
- 索引用の表スペースに対するバッファープールを作成する
  - 索引をメモリー上に保持し、バッファーヒットさせたい場合

## □ 索引が有効利用され、最適なアクセスパスを得るために

- 索引順を維持する
  - クラスター索引により索引順を維持する
  - REORGにより定期的に索引順を維持する
  - ユニーク索引の列による1件検索の場合には、クラスター率が低くても問題はない
- RUNSTATSの実行による統計情報更新とBIND実行
  - 実行タイミング
    - 表のデータがLOADされ、適切な索引が作成された時
    - 表のデータがREORGされた時やアプリケーションをBINDする時
    - 表および索引のデータの10%–20%がUPDATE/DELETE/INSERTされた時
  - 現時点の統計情報に更新することにより、現状で最適なアクセス・パスが選択される(動的SQLの場合)
  - RUNSTATS実行後、BINDを実行する(静的SQLの場合)



# 解説

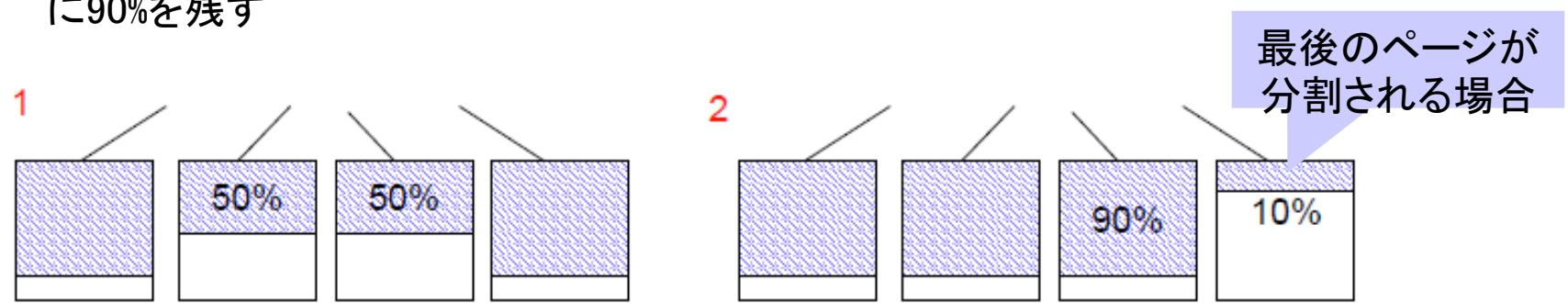
- 表のレコードが追加されると索引のリーフページにも索引エントリが追加されますが、リーフページが満杯になれば、ページの分割が必要となります。頻繁な索引ページの分割を避けるために、空き領域を確保しておくことができます。
  - PCTFREE: キーのランダウな追加に備えてリーフページ内に確保しておく空き領域の割合(%)
  - LEVEL2PCTFREE: 索引のレベル2ページに確保しておく空き領域の割合(%)
  - ページの分割方法はPAGE SPLITで指定できます。
- また、表からレコードが削除されると、索引リーフページからは索引エントリが削除されます。DB2では、一定以上の索引エントリが削除された場合に、動的に索引リーフページをマージすることができます。
  - MINPCTFREE: 索引のリーフページの使用率がこの%以下となった場合に、隣のページと索引ページをマージし、空いたページを開放することができます。頻繁なマージを避けるために50以上には設定しないこと。
- 索引データを索引用の表スペースに格納するためには、表の作成(CREATE TABLE)時に、その表に対して作成された索引のデータをどの表スペースに格納するかを明示的に指定します。
  - CREATE TABLE 表名 (COL1 INTEGER, .......) IN 表の表スペース名 INDEX IN 索引の表スペース名
- 索引用の表スペースと、表用の表スペースを分け、表スペース単位のバックアップ/リストアを行う場合、表と索引の両方の表スペースを同じ特定の時点まで同時にロールフォワードする必要があります。
  - 索引用表スペースが一つで、DB内のすべての表の索引を持つような構成では、索引をリストアし手ロールフォワードする場合、すべての表の表スペースも必要となります。
- RUNSTATSは、表のデータが大きく変化した場合に、統計情報を最新の状態に更新するために実行します。動的SQLは動的にBINDを実行するため、RUNSTATSで統計情報を変更することによりアクセス・パスが変わること可能性があります。
- 静的SQLの場合は、RUNSTATS実行後、より最適なアクセス・パスを得るには、BINDを実行する必要があります。
- RUNSTATSはAND INDEXESオプションつきで実行します。また、必要に応じて、WITH DISTRIBUTIONオプションつきで実行し、非一様分布統計情報を収集してみます。



# 参考: 索引ページの分割

## □ PAGE SLIT: SYMMETRIC (デフォルト)

1. 中間のリーフページを分割する場合は、分割されたページが元の索引の半分づつを引き継ぐように分割する
2. 先頭または最終ページを分割する場合は、新しいページが10%を引き継ぎ、元のページに90%を残す



- SYMMETRICの分割は、キー挿入パターンがランダムで、各ページに対する挿入が一定の割合で起こることを前提としている
  - データが特定のパターンで挿入されるケースでは、分割後に、分割された一方のページに挿入が行われない場合がある
    - 索引ページに無駄なスペースができる
    - 余分なページのREADが必要
    - 索引ページの分割回数が増える

## □ PAGE SPLIT HIGH

- 各レンジ内のキー値が次第に増加する場合に最適な非対称分割を行う

## □ PAGE SPLIT LOW

- 各レンジ内のキー値が次第に減少する場合に最適な非対称分割を行う



ブランク・ページです

# 表/索引の圧縮

## □ 表/索引を圧縮可能

- 表圧縮(V9.5以降)
- 索引圧縮(V9.7以降)

## □ 圧縮のメリット/デメリットを考慮して圧縮する/しないを選択

- 一般的には、IOバウンドとなるウェアハウス系は圧縮向き  
CPUバウンドとなるオンライントランザクション系は圧縮は必要としない場合が多い
  - メリット: ディスク容量の削減  
IO量の減少によるパフォーマンスの向上  
バッファープールヒット率の向上
  - デメリット: 圧縮/解凍にCPU負荷がかかる

## □ 圧縮の方式ごとの圧縮指定方法

- 表
  - クラシック圧縮: 辞書を使用した行圧縮
    - V10.1以降 CREATE TABLE/ALTER TABLEに COMPRESS YES STATICを指定
      - ◆ V9.5/V9.7では、CREATE TABLE/ALTER TABLEに COMPRESS YESを指定
  - アダプティブ圧縮: クラシック圧縮に加えて、動的にページ単位圧縮を実行する
    - V10.1以降: CREATE TABLEに COMPRESS YESを追加
- 索引
  - V9.7以降
    - 圧縮表に作成される索引は自動的に圧縮
      - ◆ 圧縮しない場合は、CREATE INDEX/ALTER INDEXでCOMPRESS NOを指定
    - 非圧縮表に作成される表は自動的に非圧縮
      - ◆ 圧縮する場合は、CREATE INDEX/ALTER INDEXでCOMPRESS YESを指定



# 圧縮されるデータ

## □表

- CREATE TABLE/ALTER TABLEで圧縮指定することにより圧縮される
  - LOB: inline格納されれば圧縮される
  - XML: inline, XDAどちらも圧縮対象

## □一時表

- DB2のオプティマイザーが圧縮するかどうかを自動的に判断し、辞書作成と圧縮を行う
  - システム一時表
  - ユーザー一時表

## □索引

- CREATE INDEX/ALTER INDEXでCOMPRESSを指定、または圧縮表に作成された索引が自動的に圧縮対象となる
  - 一時表に作成された索引も圧縮対象となる
- 圧縮されないもの
  - MDCブロック索引、カタログ表の索引、specification onlyの索引、XMLメタ索引、XMLパス索引

## □ログ

- 上記の圧縮されたデータ



# クラシック圧縮(辞書による静的行圧縮)

## □クラシック圧縮

- 辞書を使って行内の連続パターンを圧縮する(V9.1から)
  - 列単位ではなく、行単位で圧縮
- 辞書作成のタイミングと方法
  - ADC(自動辞書作成): 圧縮が指定されており辞書が存在しない表に対して以下のような方法で一定量のレコードが挿入されると、自動的に辞書が作成され以降に挿入されるレコードが圧縮される
    - INSERT/INSERTモードのLOAD, IMPORT
  - LOAD REPLACE
  - クラシックREORG
  - INSPECT
- 考慮点
  - 辞書が作成された時点でのデータに基づき圧縮されるため、追加されるデータのパターンが変わると圧縮率が低くなる
    - 辞書の再作成と再圧縮が必要になることがある

Name	Dept	Salary	City	State	ZipCode
Fred	500	10000	Plano	TX	24355
John	500	20000	Plano	TX	24355

圧縮前

Fred	500	10000	Plano	TX	24355	John	500	20000	Plano	TX	24355	...
------	-----	-------	-------	----	-------	------	-----	-------	-------	----	-------	-----

Dictionary

01	500
02	Plano, TX, 24355
...	...

圧縮後

Fred	(01)	10000	(02)	John	(01)	20000	(02)	...
------	------	-------	------	------	------	-------	------	-----



# アダプティブ圧縮

## □アダプティブ圧縮はクラシック行圧縮に対して追加可能な拡張機能

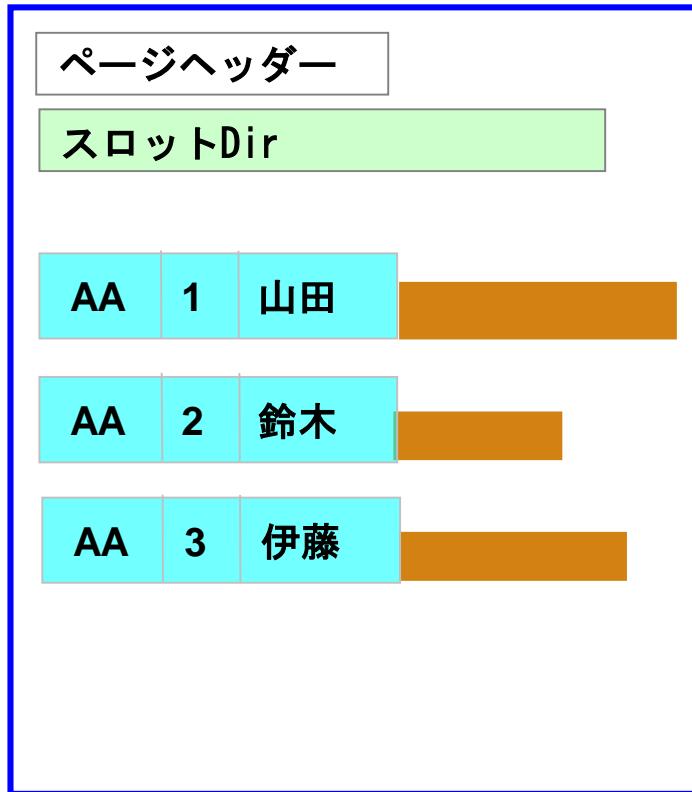
- 2種類の辞書の組み合わせで行を圧縮
  - クラシック圧縮辞書
  - ページ・レベルの辞書
- メリット
  - 辞書の作成から時間を経て、既存辞書によって圧縮効率が悪いデータパートンが多くなった場合でも高い圧縮率を保つ
    - 高圧縮率を維持するために再編成の必要性がない
  - 行圧縮のみの場合より、さらに圧縮できる
- 考慮点
  - 行圧縮で圧縮済みのデータに対して圧縮を行うため、行圧縮だけのときよりもCPU負荷が高くなる
  - データがそれほど変化しない場合、時間がたっても圧縮率がそれほど低下していない場合、運用で圧縮辞書の再作成を行う場合には、指定の必要は無い



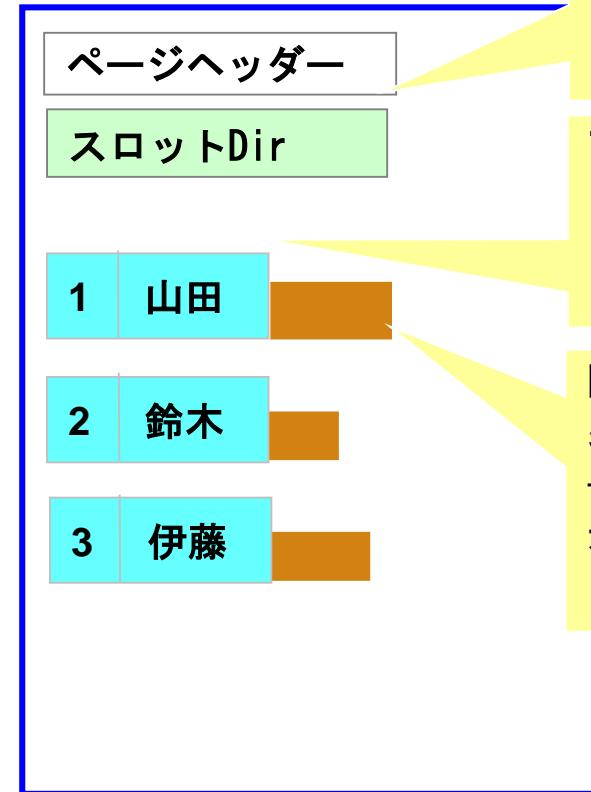
# 索引の圧縮

## □ 索引を3つの手法を組み合わせて圧縮

索引ページ



圧縮された索引ページ



**可変長のスロット・ディレクトリー**  
ページに格納されるキーの数によってスロット・ディレクトリーの長さを変動できるようにした。

**前方一致部分の圧縮**  
各キーで共通な前方一致部分を圧縮

**RIDリストの圧縮**  
各エントリーのRID(ページ番号+Slot番号)を保管する代わりに差分情報を保管するように変更



ブランク・ページです

# その他のオブジェクトの検討

## □ビュー

- 実体の表は持たず、SELECTの結果表に名前を付けて指定したもの

## □MQT(マテリアライズ照会表)

- 照会処理の応答時間短縮を目的に、1つまたは複数の表を参照して得ることのできる集計データを事前に実行し、その結果を蓄積しておくための表

## □グローバル一時表

- 宣言済み一時表(declared global temporary table)
- 作成済み一時表(create global temporary table)

## □SEQUENCE

- 連番を作成するためのオブジェクト

## □アプリケーション的なオブジェクト

- トリガー
- UDF
- ストアード・プロシージャー



# 解説

□ 表、索引以外にDBに作成するオブジェクトがあるかどうか、検討します。以下のようなオブジェクトがあります。

□ ビュー

- 実体の表は持たず、SELECTの結果表に名前を付けて指定したもの
  - 1つ以上の表にあるデータを、元の表定義とは異なる形式で見せるための方法
  - データを保守せずに様々な検索要件に対応することができる
  - ストレージを必要としない

□ MQT(マテリアライズ照会表)

- 照会処理の応答時間短縮を目的に、1つまたは複数の表を参照して得ることのできる集計データを事前に実行し、その結果を蓄積しておくための表
  - ストレージを永続的に使用
  - 以下を検討する必要がある。
    - 基礎表更新時のデータの反映(リフレッシュ)のタイミング
    - データ保守を誰が行うか
- MQTについては、以下の資料を参照

□ グローバル一時表

- アプリケーション中で一時的なデータを保管するための表。
- 永続的なストレージは必要としない。
- ユーザー一時表スペースを使用する。
- 以下の種類がある。
  - 宣言済み一時表(declared global temporary table): アプリケーション中で宣言し、一時的なデータを保管するシステムカタログには登録されない
  - 作成済み一時表(create global temporary table): あらかじめ作成しておき、アプリケーション中で参照することによって、その接続用の専用インスタンスが作成されるシステム・カタログに登録される

□ SEQUENCE

- 連番を作成するためのオブジェクト
- 「解説(IDENTITY列とSEQUENCE)」ページを参照。

□ アプリケーションの要件にしたがって、以下のようにロジックを持つオブジェクトを作成することができます。

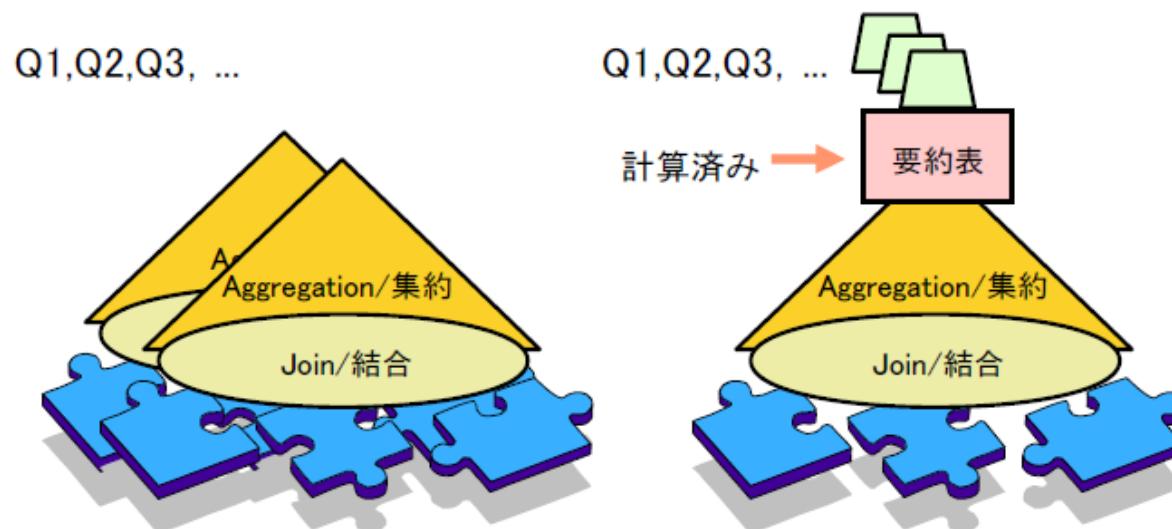
- トリガー
- UDF
- ストアード・プロシージャー



# MQT(マテリアライズ照会表)

## MQT

- 定義が照会の結果に基づく表
- MQT表の定義の基礎となる表(複数も可)に存在するデータに基づいた、事前に計算された結果を格納する
  - 例
    - JOIN済み
    - 合計値の集計済み
- SQL コンパイラーが、基礎表に対するよりも、MQT表に対する照会のほうが効果的に実行できると判断する場合は、照会はMQT表に対して実行される
- 基礎表に対する照会よりも速く結果を得られる



# MQTの作成

## □ CREATE TABLE 文にて指定

- materialized-query-definition で基礎表に対するSELECT文を定義

## □ DATA INITIALLY DEFERRED

- SELECTもとの表からMQT表への初期データは手動で反映させる

## □ REFRESH <DEFERRED/IMMEDIATE>

- 基礎表への変更がMQT表へ即時反映されるかどうか

- DEFERRED 変更は即時に反映されない
  - ステージング表が使われる場合には、ステージング表に反映が即時に反映され、ステージング表からMQTへの反映は非同期

- IMMEDIATE 即時に反映

## □ <ENABLE/DISABLE> QUERY OPTIMIZATION

- オプティマイザーが最適化の手段としてMQTを選択するかどうか

- ENABLE 適切な状況下で照会はMQTを使用するようにREWRITEされる

- DISABLE 適切な状況下で照会はMQTを使用しないが、MQT自身は照会可能

## □ MAINTAINED BY <SYSTEM/USER>

- データがDB2システム、またはユーザーのどちらによって保守されるか

- SYSTEM MQT: データはDB2システムによって保守される

- USER MQT: データはユーザーによって保守される

- ユーザーはMQTに対して更新、削除、挿入が可能

- SYSTEM MQTで使用可能なREFRESH TABLEステートメントやSET INTEGRITY IMMEDIATE CHECKEDは呼び出せない

- REFRESH DEFERRED MQTにのみ指定可能



## ②データ容量の見積もり



# 物理設計の流れ

①表・索引定義の作成

②データ容量の見積もり

③インスタンスの構成と  
データベース分割

④表の分類と  
表スペースの構成

⑤表スペース容量の見積もり

⑥ディスク上への  
オブジェクトの配置

⑦ユーザーと権限の設計

⑧構成パラメータの設定

⑨シェル／コマンドの作成

- 表の見積り
- 索引の見積り
- 表圧縮(行圧縮)/索引圧縮の考慮



# 表・索引の見積もり

## □前提:

- 物理設計手順「①表・索引定義の作成」において、作成すべき表、索引が洗い出されている
- 見積もりの基礎となる前提の数値、算定根拠を明確にする
  - 各表のレコード件数を見積もる上で必要な数値の収集
    - コード類、マスター類の件数、1日あたりの処理件数
    - データの保持期間、データ増加率
    - ... etc.
  - お客様にAuthorizeされた前提の数値であることが重要
    - 前提に変更があれば、都度再見積もり可能なようにワークシートにまとめておく

## □手順

- 表の容量を見積もる
  - MQTは、表と同様
- 索引の容量を見積もる
- 圧縮率を加味する

## □あくまでも見積もりであり、テスト環境のDBでデータを格納し確認



# 解説

- 物理設計手順「①表・索引定義の作成」において、作成すべき表、索引が洗い出され、表の列定義、索引キー定義が行われていることを前提とします。表定義(カラムの属性・長さ)および、索引のキーが決定した後、どれだけのディスク容量が必要となるかを見積もる必要があります。
- この容量見積もりは、表に含まれるデータだけではなく、索引のデータも含まれます。後段階でパフォーマンスチューニングを行う際に索引が追加されることも考慮し、索引を作成する表スペースの容量には余裕を持たせる必要があります。
- 見積もりの基礎となる前提の数値(コード類、マスター類の件数、1日あたりの処理件数、データの保持期間、データ増加率等や算定根拠を明確にする必要があります)。
- 各表のレコード件数を見積もる上で必要な数値の収集は、お客様にAuthorizeされた前提の数値であることが重要です。
- 前提に変更があれば、都度再見積もりが必要となりますので、再見積もりが容易になるようにワークシートにまとめておきましょう。
- 手順としては、全ての表、索引の容量を見積もり、圧縮を使用する場合は、圧縮率を加味します。
- MQTはViewの一種ですが、永続的にストレージを使用するため、表と同様に容量を算出します。
- データベース・オブジェクトのサイズは、正確に見積もることができません。サイズの見積もりを難しくする原因是、ディスクのフラグメント化によって発生するオーバーヘッド、フリー・スペース、および可変長列の使用などです。これは、列タイプや行の長さが広い範囲で異なる可能性があるためです。まずデータベースのサイズを見積もってから、テスト・データベースを作成し、それに標準的なデータを入れてみてください。



# 表の見積もり

## □ 表容量概算： (論理レコード長 + 10) × レコード件数 × 安全率

- 論理レコード長
  - 各データ項目のデータ・タイプ、桁数から各項目の物理サイズを算出。
  - 可変長の場合は、平均長を採用。
  - NULL値を許す場合1バイト、可変長の場合4バイトを各該当の列項目あたり加算
  - すべてを合計したのが1行あたりの論理レコード長
- レコード件数
  - 保存期間、データ増加率も加味
- 安全率(余裕率)
  - PCTFREEの割合やAPPENDモードであるかも考慮する
  - オーバーヘッド分(フラグメンテーションやオーバー・フロー・レコードの有無)
- Long列データは他の表データとは別のところに保管
  - 表内の他の列と同じページには、実際のLONG, LOBデータへのポインター情報のみを持つ
    - LONG VARCHAR, LONG VARGRAPHICは24バイト
    - LOB記述子(LOBデータへのポインター)はLOB最大サイズによって長さが異なる
  - V9.7から、指定サイズ以下のLOBは、他のデータと同じ場所に保存可能(インライン格納)
- 必要なディスク容量
  - 表スペースのページ・サイズを決定し、1ページあたりの行数から必要ページ数を導く(後述)



# 解説

- ここで、レコード長を計算する場合、ヌルが可の場合1バイト、可変長の場合4バイトを加える必要があります。
- 論理レコード長は、各データ項目のデータ・タイプ、桁数から各項目の物理サイズを算出して、可変長の場合は、平均長を採用します。NULL値を許す場合1バイト、可変長の場合4バイトを各該当の列項目あたり加算し、これらをすべて合計したのが1行のレコード長となります。
- レコード件数は、保存期間、データ増加率も加味します。また、安全率としてPCTFREEの割合やAPPENDモードであるかの考慮や、オーバーヘッド分(フラグメンテーションやオーバー・フロー・レコードの有無)も考慮します。



# 索引の見積もり

## □ 索引容量概算:

(平均索引キー・サイズ + 索引キーのオーバーヘッド) × 行数 × 安全率

- 平均索引キー・サイズ

- 基本的な算出方法の考え方は表と同等

- 索引キーのオーバーヘッド

- 索引を保管する表スペースのタイプ、索引が作成される表のタイプ、に依存

表スペースの タイプ	表タイプ	索引タイプ	索引キーの オーバーヘッド (バイト)
通常 (REGULAR)	非パーティション	任意	9
	パーティション	パーティション	9
	パーティション	非パーティション	11
大きい (LARGE)	パーティション	パーティション	11
	パーティション	非パーティション	13

- 安全率

- ノンリーフ・ページやフリー・スペースなどのオーバーヘッドのため、少なくとも2倍は必要
  - 索引の作成時のソートに必要な一時スペースの領域は、上記の式の安全率を3.2として見積もる

- 後にパフォーマンスチューニングで索引が追加されることも考慮し余裕を見ておく



# 解説

- 索引のキー長も算出方法は基本的には表と同等です。
- 行数については、MDC表のブロック索引(MDC表に対して、内部的に作成される)は、「行数」を「ブロック数」で置き換えます。
- LARGE表スペース(RID6バイト)は、通常のREGULAR表スペース(RID4バイト)より、2バイト拡張されています。また、索引タイプを非パーティション(NOT PARTITIONED)にすると、パーティション表のグローバル索引となり、索引キーにパーティションIDが含まれるため、その分の2バイトが追加で必要となります。
- 後段階でパフォーマンスチューニングを行う際に索引が追加されることも考慮し、索引を作成する表スペースの容量には余裕を持たせる必要があります。
- 最終的に必要なディスク容量は、表スペースのページ・サイズを決定し、1行の長さから1ページあたりの行数を算出し、全データ件数をその1ページあたりの行数で割ることにより、必要ページ数を導きます。(後述)



# 解説

## ■各データ・タイプ毎のサイズ

データ・タイプ	バイト
INTEGER	4
SMALLINT	2
DOUBLE	8
DECIMAL(n,m)	(n/2 + 1)
CHARACTER(n)	n
VARCHAR(n)	n+4
LONG VARCHAR	24
GRAPHIC	n*2
VARGRAPHIC	(n*2)+4
LONG VARGRAPHIC	24
DATE	4
TIME	3
TIMESTAMP	10
XML	96

注) Unicodeデータベースの場合、文字列のバイト数(n)に注意してください。

■UTF-8でデータを格納したい場合

全角、半角カタカナ1文字 3バイト  
英数半角1文字 1バイト

■UTF-16でデータを格納したい場合

すべての文字が2バイト固定

ただし、サロゲート・ペアになる文字(Unicode(UCS)でU+10000以上の文字)は、1文字でUTF-16で4バイト、UTF-8でも4バイトになります。

GRAPHIC, VARGRAPHICのデータタイプはUTF-16の形式でしか、データを格納することはできません。UTF-8で格納したい場合は、CHARACTER, VARCHARのデータタイプを使用してください。



# LOBデータの表スペース計算

## □ LOB記述子

- 表の行長計算時に、LOB記述子分を行長に加える

## □ LOBデータオブジェクト

- 実際に格納されるデータのセグメント \* レコード件数

## □ LOB割り振りオブジェクト

- $\text{ROUNDUP}(\text{LOBデータオブジェクトサイズ} / 64\text{GB}) * 4\text{KB}$   
+  
 $\text{ROUNDUP}(\text{LOBデータオブジェクトサイズ} / 8\text{MB}) * 4\text{KB}$

LOB最大サイズ (バイト)	LOB記述子サイズ (バイト)
1K	72
8K	96
64K	120
512K	144
4M	168
128M	200
512M	224
1G	256
1.5G	280
2G	316



# 圧縮率の見積もり

□表、索引を圧縮する場合は、圧縮率を加味する

□圧縮率の予測は困難

- 旧バージョンからの移行で、すでに圧縮済みであれば同程度
- 類似の他システムの実績を参考にする
- 一般論として
  - 数値は圧縮されにくい
  - 文字列は圧縮されやすい
- 圧縮率予測ツールを使用する
  - ADMIN\_GET\_TAB\_COMPRESS\_INFO表関数
  - ADMIN\_GET\_INDEX\_COMPRESS\_INFO表関数



# 解説

- DB2は、データを圧縮するために、特定のフィールドだけではなく、行全体を検査して、反復する項目やパターンを検出します。反復する項目が収集された後、データベース・マネージャーは圧縮辞書を作成して、短い数値キーをこれらの項目に割り当てます。数値データの圧縮では、ある数値が別の数値に置き換えられます。置換される数値のサイズによっては、文字列を圧縮する場合ほどのストレージの節約にはならない可能性があるため、一般的に、文字列は、数値データよりも圧縮されやすいと言えます。
- 圧縮率を見積ることは難しいため、典型的なデータを準備し、ADMIN\_GET\_TAB\_COMPRESS\_INFO表関数やADMIN\_GET\_INDEX\_COMPRESS\_INFO表関数を使って、ストレージの節約量を推定してください。これらの表関数の見積もりには、統計情報が使用されます。表関数を実行する前に、RUNSTATSコマンドを実行して、最新の統計情報を取得してください。



# 表圧縮(行圧縮)の見積り

## □ 圧縮の設定確認

- SYSCAT.TABLESのROWCOMPMode列

A = ADAPTIVE

S = STATIC

ブランク = 行圧縮は有効ではない

## □ 圧縮率の見積もり

- ADMIN\_GET\_TAB\_COMPRESS\_INFO表関数
  - PCTPAGESAVED\_STATIC列: クラシック行圧縮により節約されるページの見積もり(%)
  - PCTPAGESAVED\_ADAPTIVE列: アダプティブ行圧縮により節約されるページの見積もり(%)

```
>>-ADMIN_GET_TAB_COMPRESS_INFO--(--tabschema--, --tabname--)----><
```

## □ 圧縮後のサイズの確認

- ADMIN\_GET\_TAB\_INFO表関数
  - データ、索引、LONG、LOB、XMLそれぞれの容量(KB)を取得することが可能

```
>>-ADMIN_GET_TAB_INFO--(--tabschema--, --tabname--)-----><
```



# 索引圧縮の見積り

## □ 圧縮の設定確認

- SYSCAT.INDEXESのCOMPRESSION列

N = 索引圧縮がアクティブにされていない

Y = 索引圧縮がアクティブにされている

## □ 圧縮率の見積もり

- ADMIN\_GET\_INDEX\_COMPRESS\_INFO表関数

➢ PCTPAGESAVED: 索引圧縮により節約されるページの見積もり(%)

```
>>-ADMIN_GET_INDEX_COMPRESS_INFO--('I', --objectschema--, --objectname--, -->>--  
member--, --datapartitionid--)-----><
```

## □ 圧縮後の索引サイズの確認

- ADMIN\_GET\_INDEX\_INFO表関数

➢ INDEX\_OBJECT\_P\_SIZE: 索引オブジェクトの物理サイズ(KB)

```
>>-ADMIN_GET_INDEX_INFO--(--objecttype--, ----->  
>--objectschema--, --objectname--)-----><
```



### ③インスタンスの構成と データベース分割



# 物理設計の流れ

①表・索引定義の作成

②データ容量の見積もり

③インスタンスの構成と  
データベース分割

④表の分類と  
表スペースの構成

⑤表スペース容量の見積もり

⑥ディスク上への  
オブジェクトの配置

⑦ユーザーと権限の設計

⑧構成パラメータの設定

⑨シェル／コマンドの作成

- インスタンスの分け方
- データベースの分け方
- データベース作成のために決定すること



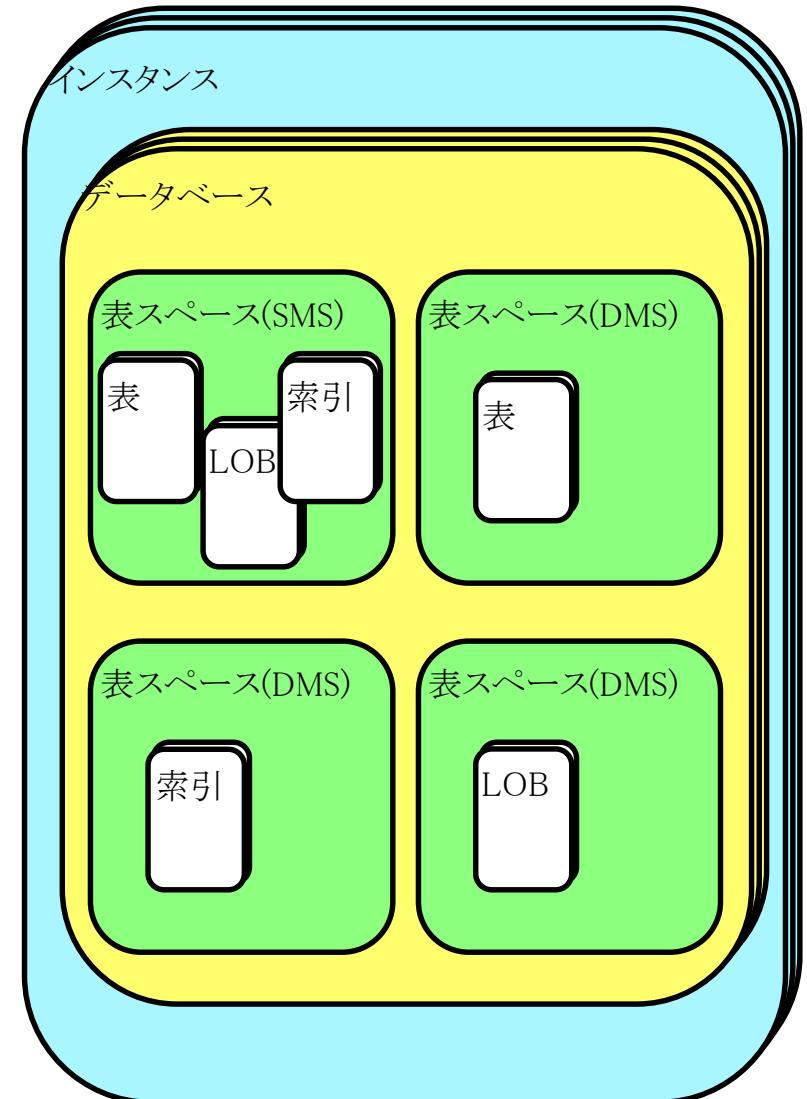
# インスタンスとデータベース

## □ インスタンス

- DB2のオブジェクトの最も大きな単位
- 論理的なデータベース・マネージャー環境
  - Unix環境
    - db2syscプロセスを中心としたスレッド群
  - Windows環境
    - 「DB2 - インスタンス名」サービス
    - 「DB2 - DB2コピー名 - インスタンス名 - ノード番号」サービス
- 全てのオブジェクトが含まれる
- DB2の起動／停止の単位
  - db2startコマンド／db2stopコマンド
- 1マシンに複数インスタンスの作成が可能
  - db2icrtコマンドで作成

## □ データベース

- 有機的にまとめられた表スペース、表や索引の集まり
- 一つのインスタンスに複数のデータベースを作成することが可能
  - create database コマンド
- 接続(CONNECT)の対象となる
- バックアップ・リストアの最大単位



# 解説

## □ インスタンス

- 一つのデータベース・マネージャー構成ファイルを使用して稼動する、データベース・マネージャー環境のことです。
- インスタンスは、DB2を構成するオブジェクト群の中で、最も大きな単位です。
- db2startで起動し、db2stopで停止するのは、このインスタンスです。(管理サーバーの場合、db2admin start/db2admin stop)
- データベースのエンジンともいえる、db2syscプロセスが立ち上がる単位ということもできます。
- インスタンスは、一台のマシン上に複数持たせることができます。しかし、一つのアプリケーション環境から扱えるのは、一つのインスタンス環境のみです。
- db2startで起動されるインスタンス、およびアプリケーション環境で使用するインスタンスは、以下で決まります。
  - 環境変数 DB2INSTANCE に指定されているインスタンスが現行インスタンスとして使用されます。
  - PC環境では、環境変数「DB2INSTANCE」が設定されていない場合があります。
  - その場合、レジストリー変数「DB2INSTDEF」に設定されているインスタンス(デフォルトではDB2)が現行インスタンスとして使用されます。
  - DB2INSTDEFは、グローバル・レベルのレジストリー変数です。
  - これを変更するには以下を実行します。  
db2set db2instdef=新インスタンス名 -g

## □ データベース

- DB2のデータベースには、様々なオブジェクトが含まれますが、ユーザーから見るときには、表や索引の集合体と言えるでしょう。
- 一つのインスタンス上に、複数のデータベースを作成することができます。
- データベースは、接続・運用上および許可の単位です。
- connectコマンドで接続するデータベースの名前を省略した場合、デフォルトでは、DB2DBDFTレジストリー変数に設定されているデータベース名が使用されます。
- connect toでデータベースに最初に接続する時には、ログ・ファイルやバッファープールのアロケーションなど初期作業が行われますので、それ以降の接続に比べて時間がかかります。
- 最初の接続に時間をかけたくない場合には、db2start後にactivate databaseコマンドで初期作業を行なわせて下さい。
- データベースは、バックアップ取得や、CONNECT特権の単位でもあります。



# インスタンスの分け方と考慮点

## □ インスタンス分割の目安

- 運用管理面
  - 管理者の権限(SYSADM、SYSCTRL、SYSMAINT、SYSMON)を持つユーザーを分けたいとき
  - インスタンスごとにデータベース・マネージャーの構成を最適化する
  - 起動/停止のタイミングを分けたい(サービス時間、運用時間の違い)
- 可用性
  - エンジン動作部分に影響を与える様なトラブル発生時に、影響範囲を小さくしたい
- 開発環境用と本番環境用

## □ その他考慮点

- 1つのマシンに複数インスタンスを作成することが可能だが、追加のシステム・リソースが必要
  - 各インスタンスが使用するメモリー
  - インスタンス・ホーム、DB2診断情報ディレクトリー(DIAGPATH)の容量



# 解説

## □ インスタンスを分ける例としては以下の様な場合があります。

- 開発用と本番用
- 管理者の権限(SYSADM、SYSCTRL、SYSMAINT、SYSMON)を持つユーザーを分けたいとき
- DBMSのエンジン動作をコントロールしたい場合(DBM構成パラメーター、レジストリー変数の設定を最適化する)
- 起動/停止のタイミングを分けたい(運用、サービス時間)
- 可用性の観点で、エンジン動作部分に影響を与える様なトラブル発生時に、その影響をできるだけ受けさせたくない

## □ 1つのマシンに複数インスタンスを作成することが可能ですが、インスタンスごとに、追加のシステム・リソース(仮想メモリーとディスク・スペース)が必要になります。また、追加インスタンスを管理するためにさらに管理が必要になります。

- INSTANCE\_MEMORY=AUTOMATIC(デフォルト)の場合、DB2を起動したタイミングで、1つのマシン(LPAR)に搭載されている物理メモリーの75~95%の間でメモリー使用量が計算されます(メモリーをアロケーションしてしまうわけではありません)。1つのマシンに複数インスタンスが存在しても、別インスタンスのメモリー使用状況を考慮して計算することはない点に注意してください。INSTANCE\_MEMORYをAUTOMATICにせず、固定の数値を設定すれば、そのサイズの範囲以上にメモリーを使用することができないので、1つのマシンに複数インスタンスで構成する場合には、数値指定することを検討してください。
- インスタンス・ホームの容量は、DB2のバージョン、FixPackによって異なる可能性があるため、テスト機で確認してください。
- インスタンス単位で、db2diagログやコア/ダンプ・ファイルがDIAGPATH(DBM構成パラメーター)に出力されます。
- 複数インスタンスにすると、プロセス監視(db2sysc)の対象が増えます。
- インスタンスに障害が発生した際の運用方法を検討します(複数インスタンスのうち、1つのインスタンスに障害が発生した場合、待機系に切り替えるのか否か、etc)。



# データベースの分け方

## □データベースの分割の目安

- アプリケーション構成(業務内容)
- 同時に処理する要件があるか
  - パフォーマンスの観点から一つにすることも検討
  - 分割されていると2フェーズ・コミット必要
  - JOIN不可(フェデレーションやレプリケーションを使用する場合は可能)
- 運用管理(バックアップ／リストア)の面から検討
  - 処理時間
  - 運用時間帯
- 可用性
- テスト環境ではフェーズやチーム単位で分割
- コード・セット別



# 解説

## □ データベースの分け方

- まずアプリケーションの構成から決定します。格納するデータや用途、業務内容がまったく異なる場合は、データベースを分割することを検討します。それぞれに属する表同士に何らかの関連があつて同時に処理する必要がある場合は、パフォーマンスの観点からも同じデータベースにし、プロジェクトや使用する部門単位にスキーマを分けるといったことも考えられます。
- 運用管理の面で、バックアップする頻度や回復処理の要件が異なる場合には、データベースを分割することも検討します。
  - 1つのデータベースのサイズが大きいとバックアップやリストアにかかる時間が長くなります。
- 複数に分けた場合、1つがダウンしても別のデータベースは使用可能ですので、可用性という点では優位です。
- フェデレーションは、他のDBに存在する表を、別のデータベースにあるように見せる機能です。
- レプリケーションは、他のDBに存在する表を、別のデータベースにも定義し、更新内容を別DBにも定期的に反映する機能です。
- このほかに、データベースを分ける例としては以下の様な場合があります。
  - 開発用と本番用
  - 共用する表を持たない異なるシステムの場合(表をJOINしたい場合は通常分けない)
  - データベース毎に構成パラメーターの設定を変えたい
  - Unicodeデータベースなど、コードセットの異なるDBが必要…など



# データベース作成のために決定すること

## □ データベースの作成時に検討すべき項目

- コード・セット(USING CODESET)
  - V9.5以降、デフォルトで、Unicodeデータベース(UTF-8)として作成される
- 照合順序(COLLATE USING)
  - SORT(ORDER BY)を指定したときの文字の並び順、文字の大小比較演算の結果にも影響する
    - Unicode データベース以外のデフォルトはSYSTEM(データベースのテリトリーに基づいた照合順序)
      - ◆ 日本語環境でデータを五十音順に照合したい場合はIDENTITY を指定
      - Unicode データベース(LANGがUTF-8) のデフォルトはIDENTITY(バイナリーコード順)
        - ◆ 同じIDENTITYでもSJISデータベースと照合順序が異なる点に注意
  - デフォルトのページ・サイズ(PAGESIZE)
    - デフォルトは、4096
  - 自動ストレージ・データベース(AUTOMATIC STORAGE)
    - 表スペースを作成する際に、そのコンテナーおよび表スペースタイプ(SMS,DMS)が完全にDB2によって決定されるデータベース(V10.1以降、YES(デフォルト)推奨)
  - ストレージ・パス(ON)
    - 自動ストレージ(AMS)表スペース用のコンテナーが保管されるディレクトリーを指定する
  - データベース・ディレクトリー(DBPATH ON)
    - データベースに関連する情報(回復履歴ファイル、ログ制御ファイル、など)が保管されるディレクトリー
  - 構成アドバイザーの使用有無(AUTOCONFIGURE)
    - デフォルトは、構成アドバイザーが推奨した値を適用した形でデータベースが作成される
      - 構成アドバイザーの推奨値を参照するだけであれば、AUTOCONFIGURE APPLY NONEと指定して、データベースを作成する

(括弧)内は、CREATE DATABASEコマンドで指定するオプション句となっています。



# 解説

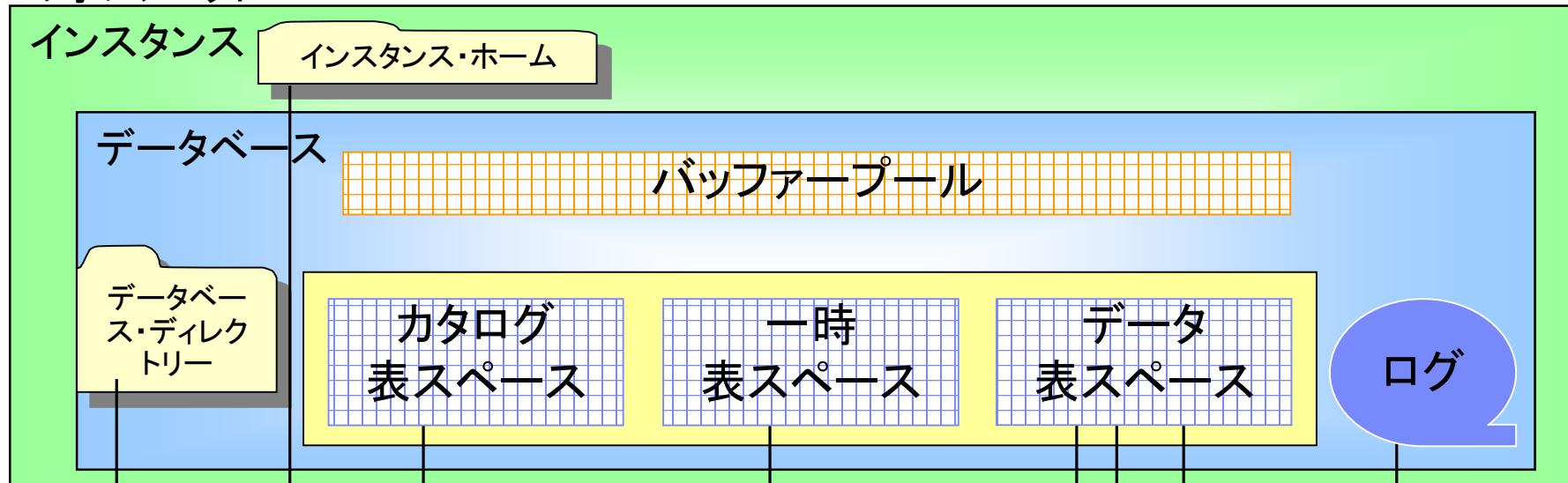
- CODESETに指定できる値は、以下のリンク先を参考にしてください。
  - [サポートされているテリトリー・コードおよびコード・ページ]日本、テリトリー ID: JP
    - > [http://www-01.ibm.com/support/knowledgecenter/SSEPGG\\_10.5.0/com.ibm.db2.luw.admin.nls.doc/doc/r0004565.html](http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.admin.nls.doc/doc/r0004565.html)
    - 例) AIXで、SJISデータベースを作成する場合は、CODESET:IBM-943、TERRITORY:JPと指定します。
- SJISからUnicodeデータベースに変更する場合、データ容量の見積もりに注意してください。
  - Unicodeでは、半角カナ/全角文字については、CHAR型は3バイト、GRAPHIC型は2バイトとなります。
- CHAR、VARCHAR、LONG VARCHARの照合(ソート、比較など)においては、データベースの照合順序に考慮する必要があります。
- 照合順序は、CREATE DATABASEのCOLLATE USING句で指定することができます。
  - SYSTEM : 現行のテリトリーに基づいた照合順序 (Unicode 以外のデータベースのデフォルト)
    - > 例:a(X'61')、A(X'41')、b(X'62')、B(X'42')、c(X'63')、C(X'43')、…
  - IDENTITY : バイナリーコード順 (Unicode データベース(LANGがUTF-8) のデフォルト)
    - > 例:A、B、C、…、a、b、c…
  - Unicodeデータベースで、LANGがSJIS環境の場合のデフォルトは、SYSTEM\_943
  - その他、COMPATIBILITY(DB2 V2の照合順序と同じ)や、Unicodeデータベース用のUCAに基づいた照合順の指定も可能
- 日本語などのダブルバイトの文字については、1バイトずつに分割して照合が行われます。
  - SJISのデータベース SYSTEMの場合の日本語例:
    - > チ(X'8361')、ア(X'8341')、ツ(X'8362')、イ(X'8342')、ツ(X'8363')、イ(X'8343')、…
  - SJISのデータベース IDENTITYの場合の日本語例:
    - > ア(X'8340')、ア(X'8341')、イ(X'8342')、イ(X'8343')、…、チ(X'8360')、チ(X'8361')、ツ(X'8362')、ツ(X'8363')、ツ(X'8364')、…
- 日本語環境では五十音順にデータを照合したい場合、あるいは順序の分かりやすさや他者DBとの互換性が求められる場合は、COLLATE USING句にIDENTITYを指定してデータベースを作成する事をお薦めします。GRAPHICタイプのデータについては、COLLATE USING句の指定に関わらず、バイナリーコード順に照合が行われます。
- COLLATE USINGに指定した値は、一旦データベースを作成した後に変更する事はできませんので注意して下さい。
- データベースのコード・セット/照合順序は、DB作成後、GET DB CFGコマンドで確認することができます。
  - 照合順序SYSTEMで作成されたDBは、「データベース照合順序 = UNIQUE」と表示されます。
- V8.2.2以降は、データベース作成時にデフォルトのページ・サイズを指定することができます(4KB以外のページ・サイズを使用する場合でも、ページ・サイズを1種類に統一することができます)。
  - 初期表スペースSYSCATSPACE、TEMPSPACE1、USERSPACE1、デフォルト・バッファーピールIBMDEFAULTTBPは、データベース作成時に指定されたページ・サイズで作成されます。
- ストレージ・パスは、自動ストレージ表スペースのコンテナーが作成されるパスとなります。複数のパスをカンマで区切ってリストすることができます。表、索引、表スペース容量で見積もったサイズが格納できるように、ディレクトリーを指定します。



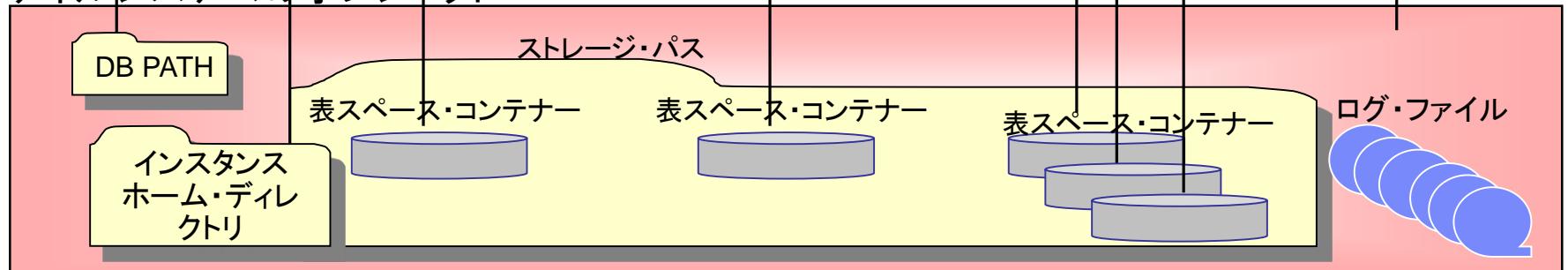
# データベースの構成要素

## □インスタンス/データベースを構成するオブジェクト

DB2のオブジェクト



OSファイルシステムのオブジェクト



# 解説

## □create databaseで自動的に作成されるオブジェクト

- データベース・ディレクトリー(create databaseのdbpath onで指定)
- デフォルトのストレージ・グループ/ストレージ・パス\*
  - IBMSTOGROUP
  - ストレージ・パス(create databaseのonで指定)
- 表スペース
  - SYSCATSPACE: カタログ表を保管するための表スペース
  - USERSPACE1: ユーザー定義の表や索引を保管するための表スペース
  - TEMPSPACE1: システム一時表スペース
  - SYSTOOLSPACE/SYSTOOLSTMPSPACE: DB2管理ツール(自動保守)用の表スペース
- バッファープール
  - IBMDEFAULTBPP
- ログパス
  - データベース・ディレクトリー配下に作成される
    - newlogpath(DB構成パラメーター)で変更可

\*automatic storage noと明示的に指定した場合は、ストレージ・グループとストレージ・パスは作成されません。デフォルトは、automatic storage yesです。



## ④表の分類と 表スペースの構成



# 物理設計の流れ

①表・索引定義の作成

②データ容量の見積もり

③インスタンスの構成と  
データベース分割

④表の分類と  
表スペースの構成

⑤表スペース容量の見積もり

⑥ディスク上への  
オブジェクトの配置

⑦ユーザーと権限の設計

⑧構成パラメータの設定

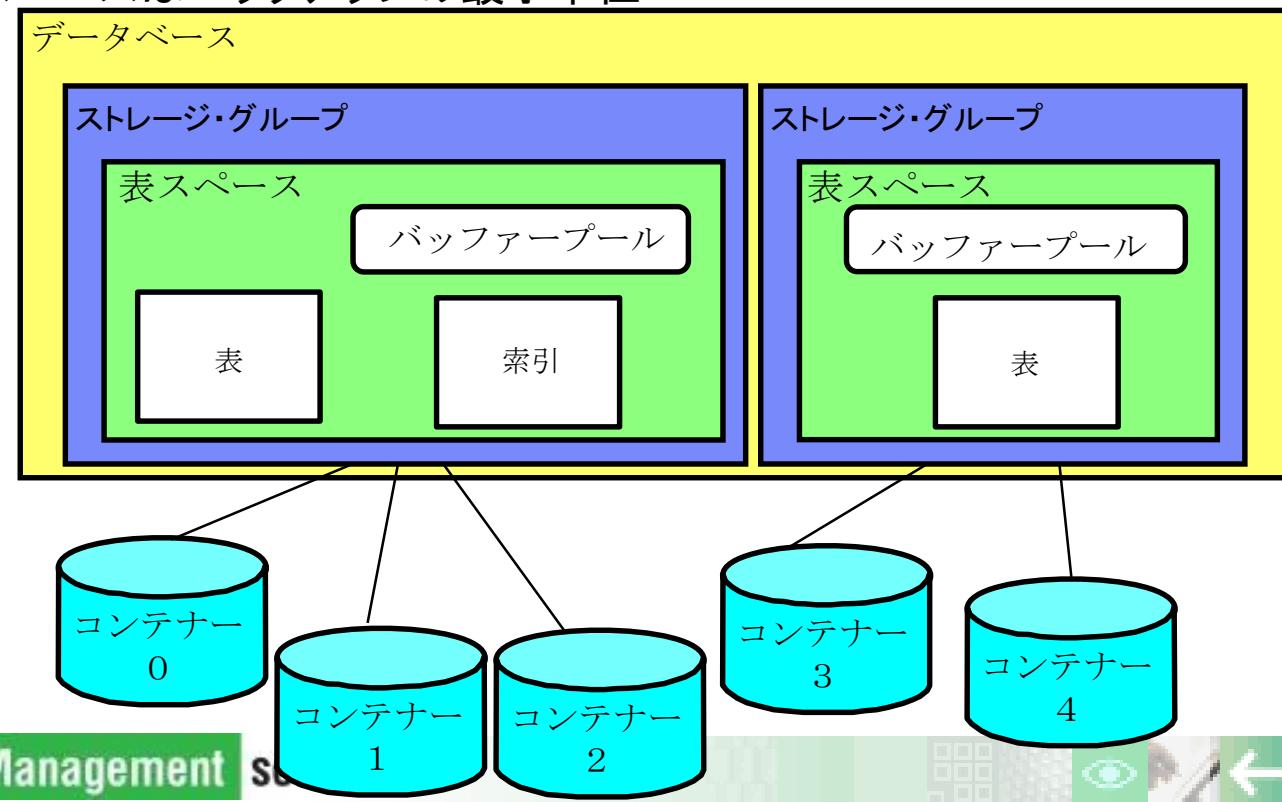
⑨シェル／コマンドの作成

- 表スペースとは
- 複数表スペースの検討
- 表スペース作成のために  
決定すること

# 表スペースとは

## □ 表スペース

- 表のデータを格納するための論理的な領域媒体
- 実際にデータを格納する物理的な媒体をコンテナーという
  - コンテナーの実体はファイルシステムのディレクトリ、ファイル、または論理ボリュームなどのローデバイス
- 一つの表スペースを1つ以上のコンテナーで構成
- データベース内に作成される表スペース
  - システム・カタログ表スペース、システム/ユーザー時表スペース、ユーザーデータ用表スペース
- 表スペースはバックアップの最小単位



# 解説

- 表スペースは、表のデータを記憶するための論理的な領域媒体です。
- 実際に表のデータが物理的に格納されるのは、表スペースに紐付けされたコンテナーになります。
- コンテナーは、表スペースのタイプにより、ディレクトリーやファイル、デバイスといった形態を取ります。
- 一つの表スペースに複数のコンテナーを割り当てることができます。
  - データはコンテナー間で平均的に割り振られます。
- 一つのコンテナーは、一つの表スペースにしか属することはできません。
- 表スペースへのデータの書き出しあは、エクステントと呼ばれる割り当て単位に行われます。(デフォルトは32ページ)(詳細は後述)
- `create database`を実行すると、デフォルトでは3つの表スペースが自動的に生成されます。
  - SYSCATSPACE:システム・カタログ表スペース
    - システム・カタログ表、およびその索引が入っている表スペースです。
    - データベース作成時に作られ、一旦作られた後は、変更や削除することができません。
  - TEMPSPACE1:システム一時表スペース
    - JoinやSort時に一時的にデータが入る表スペースです。
  - USERSPACE1:ユーザー・データ用表スペース
    - ユーザーのデータや索引が入れられる表スペースです。
    - 表スペースを指定せずに`create table`を実行すると、他のユーザー・データ用表スペースが無い場合は、デフォルトで、この表スペースが使われます。
    - 既に他のユーザー・データ用表スペースが作られている場合には、最初に作られた表スペースが使われます。
- 表スペース単位でバックアップ/リストアを行うことも可能です。
- ロールフォワード回復不可能な場合(循環式ロギング)は、データベース単位のバックアップのみ可能です。
  - 循環ロギングの設定
    - LOGRETAIN(DB構成パラメーター)=NO
    - USEREXIT(DB構成パラメーター)= NO
    - LOGARCHMETH1/LOGARCHMETH2 (DB構成パラメーター)=OFF
- ロールフォワード回復可能な場合(アーカイブ式ロギング)は、表スペース単位のバックアップ/リストアが可能です。
  - アーカイブ・ロギングの設定
    - LOGRETAIN=ON またはRECOVERY(V6.1より)
    - USEREXIT=YES
    - LOGARCHMETH1/LOGARCHMETH2でアーカイブ方法を指定



# ユーザーデータ用表スペースの検討

## □ 表スペースの作成の目安

- CREATE DATABASEでユーザーデータ用表スペースとしてUSERSPACE1が作成されている
  - USERSPACE1はバッファープールとしてIBMDEFAULTTBPを使用する
- 追加のユーザーデータ表スペースを作成するかどうかを検討する
  - 表のレコードサイズに応じた表スペース
  - パフォーマンスを考慮し、バッファープールの割り当てを検討
  - I/Oの並列処理を考慮し、データの物理配置に合わせて表スペースを構成
  - バックアップの単位、等の運用面で検討

## □ 表をグルーピングし、表スペースを分ける

- 1. できる限りバッファープールに読み込むことが望ましい表
  - 頻繁に読み書きされるトランザクション系データ
- 2. バッファープールには読み込む必要が無いデータ
  - アプリケーション活動の履歴的な、一度書いたらほとんど変更／参照されないデータ
- 3. 1と2の中間
- 4. 小さな表はある程度の単位で一つの表スペースにまとめる
- 5. バックアップ取得の頻度で別の表スペースにする
  - 大量の更新がある表⇒頻繁なバックアップ取得必要
  - 変更が非常に少ない表⇒頻繁なバックアップ取得不要

## □ 索引・LOBデータを別の表スペースに格納することができる

- 自動ストレージ表スペース、または、DMS表スペースに表を作成する場合、索引・LOBのデータを別の表スペースに格納することが可能。
- 表と索引のバッファープールを分け、特定の表や索引のヒット率を確保したい場合には、表と索引を別の表スペースに格納することを検討する
- LOBデータにはバッファープールは有効ではない
  - 32KB以内のLOBデータであれば、インラインで格納することにより、バッファープールを使用可能
- LOBを別のファイルDMS表スペースに配置することによって、ファイルキャッシュを有効にすることができる
  - LOBとして格納するデータは、通常それほど頻繁に読み書きが発生しないので、キャッシュが効かなくても影響が少ないケースも多い
- 索引・LOBデータを別の表スペースに分けた場合で、表スペースのロールフォワードをする場合、依存関係のあるデータ用、索引用、LOB用の表スペースを同じ時点にする必要がある



# 解説

□ CREATE DATABASEによってデフォルトのユーザーデータ用表スペースとしてUSERSPACE1が作成されますが、ページサイズ、パフォーマンス要件などにより、追加のユーザーデータ表スペースの作成を検討します。

- 表の1レコードはページをまたがることができないため、レコード長以上のページサイズを持つ表スペースが必要
- バッファープールを分けたい場合、表スペースを分割する必要がある
  - 全体としては長い

□ 典型的な分け方は以下のようにになります。

- 1. 頻繁に読み書きされるトランザクション系データ
- 2. アプリケーション活動の履歴的な、一度書いたらほとんど変更されないデータ
- 3. 1と2の中間
- 分類1の表は、アプリケーションのパフォーマンスに大きな影響を与えます。ディスク資源とメモリ資源を十分に割り当てる必要があります。
- 分類2の表は、容量的には大きくなりますが、一度書いた情報をあまり読まないため、メモリ資源は低く抑えることができます。このタイプの表を全件検索するようなSQLが実行された場合、ディスクのアクセスが非常に多くなるので、パフォーマンスが常時必要な分類1の表とは異なる物理ディスクを割り当てることが理想的です。できればディスクへつながるSCSIやファイバーチャネルなどのインターフェースも別であることが理想的と言えます。
- 分類3は、メモリ資源をそれなりに与える必要がありますが、優先順位では分類1より下になります。
  - 頻繁に使用される表、それもアクセスされるデータがほぼ決まっている場合には、大きなバッファープールを持った表スペースを割り当てます。
  - このときには、データの表スペースと、索引の表スペースとを分けるのもひとつ的方法です。(DMSの場合)
  - 大きな表に、ランダムにデータ・アクセスする場合には、小さいバッファープールを持った表スペースを割り当てます。
  - たまたましか使用されないアプリケーションからアクセスされる表には、小さいバッファープールを持った表スペースを割り当てます。このような表は、まとめて一つの表スペースに入れるのもひとつの方法です。
  - 参照制約、トリガーなど関連のある表同士は一つの表スペースにまとめます。
  - 小さな表は全て同じ表スペースにまとめます。
  - バックアップを取得する単位で表スペースを分けます。

□ 表スペース単位のバックアップは時間とリソースの節約になります。

- 大量の変更がある表スペースは頻繁にバックアップを取り、変更が非常に少ない表スペースは時折バックアップを取ります。
- V9以降は、データベースの再ビルト機能によりデータベース全体のバックアップを取得していくなくても、表スペース・バックアップからデータベースを回復することができます。

□ 索引とLOBデータについては、必ずしも別の表スペースに分ける必要はありませんが、例えば、索引のデータは必ずバッファープール上にキャッシュさせたいという理由で、索引用の表スペースを分けるという考え方があります。

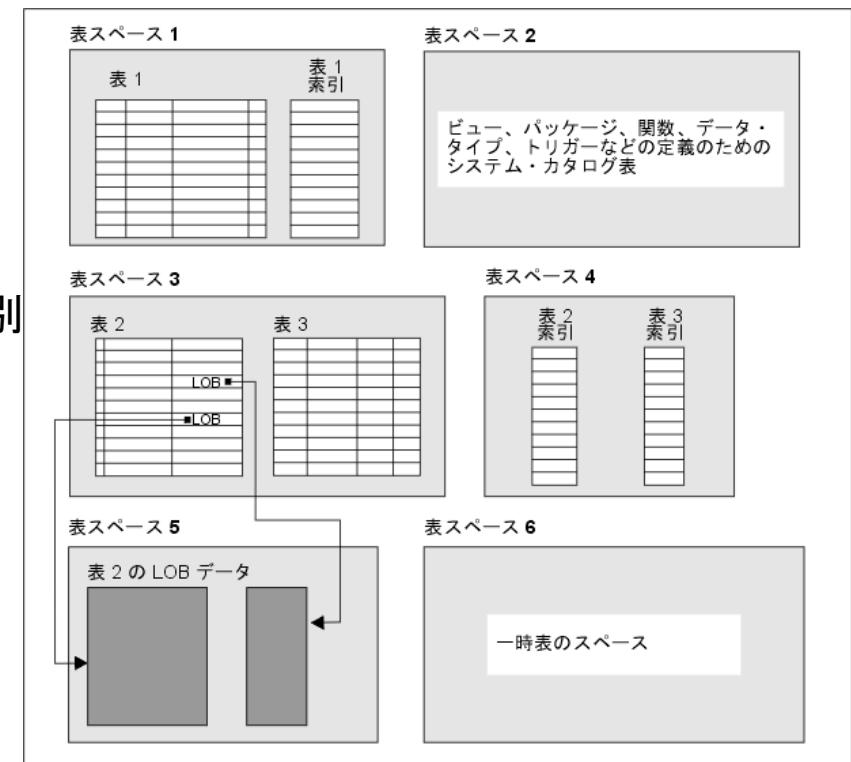


# 表スペース作成のために決定すること

## □ データベースの作成時に検討すべき項目

1. 保管データのタイプによる表スペースの種別
  - REGULAR }  
■ LARGE } ユーザー表スペース
  - SYSTEM TEMPORARY }  
■ USER TEMPORARY } 一時表スペース
2. ページサイズの決定
  - 4K
  - 8K
  - 16K
  - 32K
3. ストレージ管理のタイプによる表スペースの種別
 

■ AUTOMATIC STORAGE	→	AMS
■ DATABASE	→	DMS
■ SYSTEM	→	SMS
4. エクステント・サイズ
5. プリフェッヂ・サイズ
6. バッファープール
7. ダイレクトI/O
8. DROPPED TABLE RECOVERY
9. ストレージ・グループ





ブランク・ページです



# 1. 保管データのタイプによる表スペースの種別

□ 保管データのタイプにより、以下のいずれかのタイプを指定する

- LARGE(V9以降のDMS表スペースのデフォルト)
  - V8以前:LOB、LONGデータ、索引用
  - V9以降:通常のデータ、LOB、LONGデータ、索引用
- REGULAR(V9以降のSMS表スペースのデフォルト、V8以前のDMS/SMS表スペースのデフォルト)
  - 通常のデータ、索引用
- TEMPORARY
  - 一時表用
    - システム一時表スペース
    - ユーザー一時表スペース



# 解説

## □ 保管データのタイプにより、以下のいずれかのタイプを指定します。

- **LARGE表スペース**
  - すべての永続データを保管します。このタイプは、データベース管理スペース (DMS) 表スペースでのみ使用できます。また、タイプを指定しない場合の、DMS 表スペースのデフォルト・タイプでもあります。LARGE 表スペースに表を配置すると、以下のようにになります。
    - REGULAR 表スペースに配置する表よりもサイズを大きくできます。
    - 表のデータ・ページ当たり、255 を超える行数をサポートできるので、データ・ページのスペース使用効率が向上します。
    - REGULAR 表スペースに配置した表に索引を定義する場合に比べ、索引の 1 つの行項目当たり 2 バイトが追加で必要になります。
- **REGULAR表スペース**
  - すべての永続データを保管します。このタイプは、DMS 表スペースと SMS 表スペースのいずれも指定可能です (SMS 表スペースでのデフォルト・タイプです)。
- **TEMPORARY表スペース**
  - システム一時表スペース
    - Join や Sort で一時的にデータが入る表スペースです。
    - システム一時表スペースは、データベースに最低一つは必要です。一つしかない時には、削除しようとするとエラーとなります。
    - 4KB 以外のページを持つ表スペースがある場合、そのページに合せた一時表スペースも作っておいて下さい。
  - ユーザー一時表スペース
    - 省略時にはデータベース作成の時点で作成されません。Global Temporary Table が使用する表スペースです。
    - 参考: Global Temporary Table(ユーザー定義一時表)
      - ◆ アプリケーションは、データベースに接続している間、一時的な表を作成して使用できます。
      - ◆ 一時表を定義するには、DECLARE GLOBAL TEMPORARY TABLE ステートメントを使用します。
      - ◆ ユーザー定義一時表が保持されるのは、アプリケーションがデータベースから切断されるまでの間だけです。アプリケーションが終了したりデータベースから切断されたりすると、表の中のデータはすべて削除され、表は暗黙的に除去されます。
      - ◆ この表の記述は、システム・カタログには現れません。したがって、この表を他のアプリケーションのために保持したり、他のアプリケーションと共用したりすることはできません。
      - ◆ ユーザー定義一時表は、ロックとログ記録を回避するので、一時表を活用するアプリケーションは大幅なパフォーマンス改善を見込めます。



# 一時表スペースの定義

## □ システム一時表スペースとユーザー一時表スペース

- システム一時表スペース
  - ソート、JOIN(表同士の結合)、再編成、索引作成時(LOADを含む)などで一時的にデータが入る表スペース
  - データベース作成時に、デフォルトで作成されるTEMPSPACE1は、システム一時表スペース(データベースに最低一つは必要)
- ユーザー一時表スペース
  - ユーザー定義一時表の一時データが入る表スペース
  - データベース作成時に、デフォルトで作成されるものではなく、作成は任意

## □ システム一時表スペースは、ページサイズ毎に1つ作成

- 一時表スペースを使用した再編成の場合、表の存在する表スペースのページサイズと一時表スペースのページサイズは同じである必要がある。
- ソート時に使用する一時表スペースは、格納できるページサイズを選択する。
  - 異なるページサイズを持つ別々の一時表は、SMSとして同じファイルシステムに配置するとスペースの有効利用ができる
- 同じページ・サイズの複数のシステム一時表スペースを持つことは推奨されない

## □ 表スペースの管理タイプ

- 一時表スペースとしてはSMSがお勧め(一般)
  - スペースの有効利用
  - 多くのクライアントアプリケーションからのソート要求を処理する場合、DMSよりパフォーマンスが良い
  - 複数のコンテナーを作成して、パフォーマンス向上を計る
  - ファイルシステム上に作成するため、ファイルシステムの制限(ラージイネーブル、ulimit)に注意が必要
- 一時表スペースとしてのDMS(例外)
  - Solarisでは一時表スペースを使用した再編成、ロードや索引作成時などの大量データの処理の場合パフォーマンス的に有利な場合がある(ケース・バイ・ケースなので、ベンチマーク・テストが必要)
  - HAのティクオーバー時間を短くために、fsckの必要になるファイルシステムの代わりにローデバイスDMSを選択する場合がある

## □ 後からでも比較的簡単に作成し直せる表スペース



# 解説

- 一時表スペースには、システム一時表スペースと、ユーザー一時表スペースがあります。
  - システム一時表スペース
  - ユーザー一時表スペース
- システム一時表スペースは、データベースに最低限1つ作成する必要がありますが、ユーザー一時表スペースは、ユーザー定義一時表(Global Temporary Table)を使う場合にのみ作成します。データベース作成時に、デフォルトで作成されるTEMPSPACE1は、システム一時表スペースです。TEMPSPACE1のページ・サイズは、データベース作成時に指定したPAGESIZE(デフォルト4KB)になります。
- データベース中に複数のページサイズが存在する場合、システム一時表スペースをページサイズ毎に作成されることをお勧めします。これは一時表スペースを使用した再編成を行う場合には必須です。再編成を行う表が含まれる表スペースのページサイズと同じページサイズを持つ一時表スペースがないと一時表スペースを使用した再編成は行えません。
- 一時表スペースに関しては、DMSよりSMSの方が勧められています。一時表スペースの使われ方は、ソートやジョインに必要なときに一時的にシステム一時表が作成され、データが格納されます。つまり、表の作成・削除が内部で行われています。表作成時にSMSではファイルが作成され、使用後に削除されます。DMSでは表スペース内の空き領域を管理する「スペースマップ」や、各表に対してディスクのどの部分を使用しているのかを管理する「エクステントマップ」をメンテナンスする必要があります。多くのユーザーがそれほど大きくないソートやジョイン用の一時表を作成する場合、この2つのマップのメンテナンス作業が競合し、パフォーマンスが劣化する場合があるため、DMSは一時表スペースには勧められていません。
- 例外として、Solarisで比較的大量なデータを一時表スペースに書き出すような場合、索引の作成やロードなどの場合、DMSの一時表の方がパフォーマンス的に有利な場合があります。
- ただし、Solarisでは必ず一時表スペースはDMSにしなければならないという訳ではありません。使用方法によってはSMSが適当なケースも存在します。
- 一時表スペースでは通常の表スペースとは、データを書き込む単位が異なります。通常の表スペースにおいて、DMSではデータはエクステント単位に書き込まれ、SMSではページ単位に書き込まれます。SMSではページ単位で書き込む上に、ファイルシステム上で、ファイルのサイズを大きくしていく必要があり、これがDMSに比べてオーバーヘッドになり得ます。db2empfaというコマンドがあり、SMSでもエクステント単位でディスクに書き出すことが可能になるため、通常の表スペースでは効果がありますが、一時表スペースでは、必要な容量を一度にアロケーションするため、db2empfaの効果は期待できません。
- 複数の一時表スペースを定義する場合、一時表スペースがSMSであれば、同じファイルシステムに配置することが可能になります。これによって使用率が低いディスクエリアを共有し、ディスクスペースの有効利用が可能になります。
- もちろん、複数の再編成を並行で実行する必要があるような場合は、別々のディスクに配置する方が、ディスクI/Oが衝突せずパフォーマンス的に有利です。
- 一時表スペースは比較的簡単に作成し直せる唯一の表スペースです。本番稼働を始めた後でも、データのExport/Loadなどの移行などが必要ではないためです。ただし、一時表スペースを作成し直す場合、データベースに最低限1つのシステム一時表スペースを残さないとエラーになります。新しいシステム一時表スペースを作成後に、古いシステム一時表スペースを削除すればこのエラーは回避できます。



## 2. ページ・サイズの決定

### □ 表を格納するページサイズを決定する

- データの行長・カラム数の制限
  - V10.5から、拡張行サイズにより、行長を超える表の作成が可能
- 表容量の制限
  - LARGE RID(V9)が使用可能な表スペース(LARGE DMS表スペース、一時 DMS表スペース)の場合、制限値が異なる(下表参照)
- 格納効率も考慮する必要がある
  - LARGE RIDを使用しない表スペースでは、1ページに格納できる行数は最大254行まで
  - LARGE RIDを使用する表スペースでは、1ページに255行以上格納可能(下表参照)
- アプリケーションの特性(OLTP or DSS)や管理面も考慮する

### □ ページ・サイズによる制限値

ページ・ サイズ	行長 (bytes)	列数	REGULAR表スペース SMS表スペース		V9.1, V9.5 LARGE表スペース		V9.7以降 LARGE表スペース 一時表スペース
			表容量	行数/ページ	表容量	行数/ページ	
4KB	4005	500	64GB	251	2TB	287	8TB
8KB	8101	1012	128GB	253	4TB	580	16TB
16KB	16293	1012	256GB	254	8TB	1165	32TB
32KB	32677	1012	512GB	253	16TB	2335	64TB

- V9.7からLARGE表スペースと一時表スペースの上限が拡張された
- 大きな表は、パーティション表やUNION ALL VIEWによる表の分割も検討する



# 解説

## □ ページ・サイズ

- 表に必要となるディスク容量を計算する時に、どのページサイズを使用するかが非常に重要な要素になります。
- DB2では、4K/8K/16K/32Kバイトの4種類のページサイズをサポートします。
- 1行のレコードが複数のページにまたがることはできません。1レコードの長さがページ内に収まらない場合は、ページサイズを大きくして下さい。
  - V10.5から、拡張行サイズの使用により、行の長さがページ・サイズの最大行長を超える表の作成が可能になりました(後述)。
- 表の最大容量の制限
  - V8までは、表の最大容量は64/128/256/512GB(それぞれのページサイズは4/8/16/32KB)でしたが、V9以降では、LARGE RIDを使用することにより、最大容量が拡張されました。
- 格納効率
  - LARGE RIDを使用しない表スペースの場合、1ページに格納できるのは最大255行までという制限があります。
  - 1行のサイズが100バイトの表があった場合、32KBページでは最大約  $(32000 \div 100)$  320行を1ページに格納できるはずですが、この制限によって(320 - 250)約70行分のデータ領域にはデータが格納されず使われない無駄な領域になります。
  - 例えば、レコードサイズがLOB(Large Object:BLOB,CLOB,DBCLOB,LONG VARCHAR)を含まない5000バイトの表があった場合、4KBページでは表を作成することができません。5000バイトの長さを持つ表を作成するためには、少なくとも8Kバイトのページサイズを使用する必要があります。しかしこの場合、8Kバイトページのうち5000バイトにしかデータが書かれないので、残りの3000バイトは使用されない無駄な領域になります。このようなケースの場合、16KBページという選択肢もあります。16KBページには、5000バイト長のレコードは3レコード入ります。残りは1000バイトとなり、使用されないスペースを抑えることができます。
  - LARGE RIDを使用する表スペースでは、この行数の上限値が大きくなりますので、レコードが格納されない無駄な領域を減らすことができます。(1ページ当たりに格納できる行数の上限値は、ページサイズ毎に異なります。)
- 行のランダム読み取り および 書き込みを実行するOLTPアプリケーションは、不必要的行に使用するバッファーページを少なくするために小さいページサイズを使用するようにしてください。
- 一度に多くの連続した行にアクセスするDSSアプリケーションは、指定された数の行を読み取るのに必要な入出力要求の数を減らすように大きなページサイズを使用するようにしてください。

## □ 異なるページ・サイズによる考慮点

- バッファープール、一時表スペースはページサイズ毎に必要
- USEオプションを使用した再編成では、一時表スペースは表スペースと同じページサイズである必要がある
- バックアップを異なるページサイズに復元することは出来ない



### 3. ストレージ管理タイプによる表スペースの種別

□ストレージ管理のタイプにより、以下のいずれかのタイプを指定する

- 自動ストレージ(AMS)
  - データベース・マネージャーが必要に応じてコンテナーの作成を制御(V10.1以降、推奨)
  - コンテナー
    - ストレージ・パス
- DMS
  - データベース・マネージャーによるファイル管理(V10.1FP1以降、ユーザー表スペースとしては、非推奨)
  - コンテナー
    - ファイル
    - ロー・デバイス
- SMS
  - オペレーティング・システムのファイル・マネージャーによるファイル管理(V10.1以降、ユーザー表スペースとしては、非推奨)
  - コンテナー
    - ディレクトリー



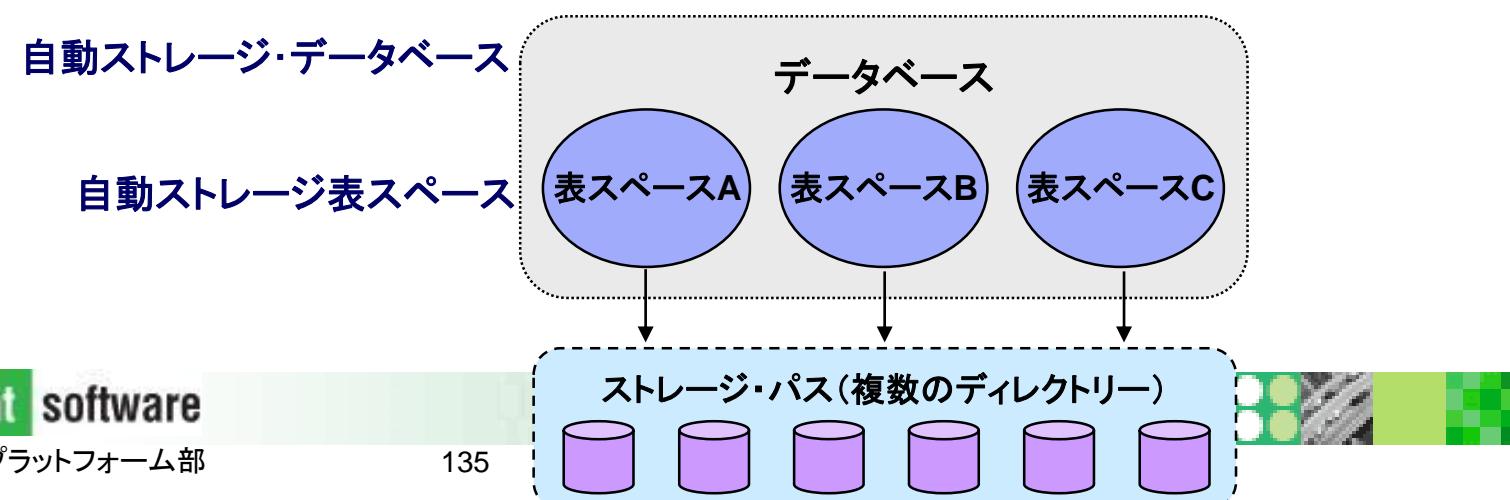
# 解説

- ストレージ管理のタイプにより、以下のいずれかのタイプを指定します。
  - 自動ストレージ(AMS: AutoMatic Storage)
    - データベース・マネージャーがストレージを管理します。
    - コンテナーとしてストレージパスを指定。指定しない場合、自動的にデータベースに関連付けられたパスに作成される。
    - V10.1以降、自動ストレージが推奨されており、デフォルトの表スペースのタイプとなります。
  - DMS(Database Managed Storage)
    - データベース・マネージャーがファイルを管理します。
    - コンテナーとしては、ファイル、またはデバイスを設定します。
    - V10.1FP1以降、ユーザー表スペースとして、DMSを使用するのは非推奨となっています。カタログ表スペースや一時表スペースでの使用を検討します。
  - SMS(System Managed Storage)
    - 各オペレーティング・システムのファイル・システムがファイルを管理します。
    - コンテナーとしては、ディレクトリーを設定します。
    - V10.1以降、ユーザー表スペースとして、SMSを使用するのは非推奨となっています。カタログ表スペースや一時表スペースでの使用を検討します。
- 自動ストレージ(AMS)は、コンテナーを定義する必要はありません。AMSで作成した、REGULARあるいはLARGE表スペースは、DMS、一時表スペースは、SMSとして作成されます。
- DMS・AMSは、SMSに比べて、表用、索引用、長形式のデータ用の表スペースを別々に作成できるため、それぞれのディスクI/Oを分散させることができます。これによってディスクI/Oを効率化し、パフォーマンスを向上することができます。
- DMSでは、データ挿入時にエクステント単位でデータが書き込まれます。また、そのエクステントは、既に容量を確保して作成されたDMSファイルまたはローデバイスの中に書き込みます。この時、DMSファイルやローデバイス自身の大きさは変わりません。
- ローデバイスとファイルDMSの差はそれほど大きくありませんが、DMSとSMSではパフォーマンスに大きな差が出る場合があります。
- 管理の容易さでは、SMSは、同じファイルシステムに複数表スペースを作成でき、表スペースごとに空きスペースの監視を行う必要がありません。
- SMSでは、表のスペースは要求時に割り振られますので、DMSに比べて特に大量データの挿入に関してパフォーマンスが劣ります。一度に割り振られるスペースの量は、`multipage_alloc` データベース構成パラメーターの設定によって左右されます。この構成パラメーターが YES に設定されている場合、スペースが必要なときにはエクステント全体（通常は複数のページで構成される）が割り振られます（マルチページ・ファイル割り振り）。それ以外の場合は、一度に 1 ページのスペースが割り振られます。V8.2以降では、マルチページ・ファイル割り振りはデフォルトで使用可能です。



# 参考: 自動ストレージ(AMS)表スペース

- V8.2.2以降、自動ストレージ・データベース、自動ストレージ表スペースが使用可能
- V9以降、データベースを作成すると、デフォルトで自動ストレージ・データベースとして作成される。
  - 自動ストレージ・データベースには、自動ストレージ表スペースを作成可能(通常の自動ストレージでない表スペースを作成することも可能)
- **自動ストレージ表スペース**
  - 表スペースのコンテナーおよびスペース管理の特性(SMS/DMS)はDB2によって自動的に(以下のように)決定されるため、指定しなくてもよい。
    - REGULARまたはLARGE表スペースの場合、DMS(ファイル・コンテナー)
    - TEMPORARY表スペースの場合、SMS
  - データベース作成時に定義された(1つ以上の)ストレージ・パスを使用してDB2がコンテナーを自動的に割り振るため、表スペース作成時にコンテナーの明示的なリストを指定する必要が無い。
    - 各ストレージ・パスに作成されるコンテナはひとつのみ
- **自動ストレージ表スペースを作成するための指定(CREATE TABLESPACE実行時)**
  - MANAGED BY AUTOMATIC STORAGE 文節を指定する、または、MANAGED BY 文節を指定しない



# 解説

## □ AMSの特徴

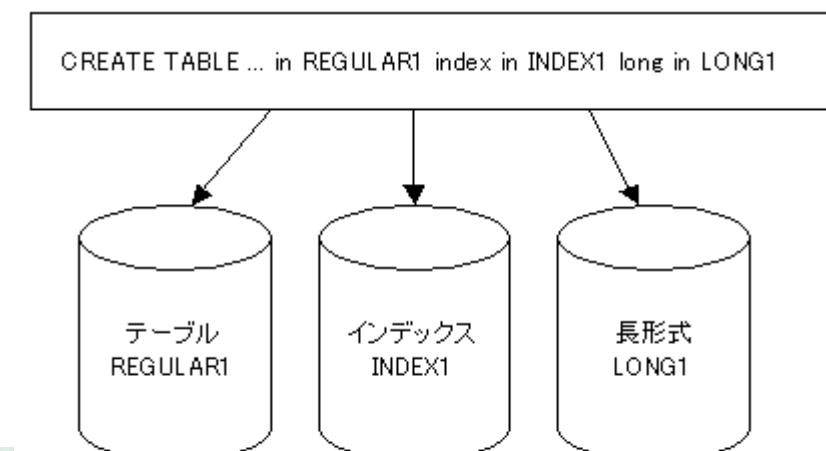
- AMSでは、コンテナーを指定する必要はありません。DB2がコンテナーの作成と拡張を自動的に行います。
- 表スペース作成時に指定するストレージ・グループに紐づくストレージ・パスに、コンテナーが作成されます。
- 各ストレージ・パスに作成されるコンテナーは1つです。
- ストレージ・パスを追加したり、削除したりすることができます。また、ALTER TABLESPACEステートメントのREBALANCEオプションで、AMS表スペースのリバランスが可能です。
- ストレージ・パスに指定したファイルシステムのサイズが、AMS表スペースのサイズの上限となります。
- コンテナーあたりの最大サイズを設定するDB2\_SET\_MAX\_CONTAINER\_SIZE レジストリー変数があります。自動ストレージ表スペース用のコンテナー 1つあたりの最大サイズを制限するレジストリー変数となります。
  - オペレーティング・システム: すべて
  - デフォルトは、-1(設定しない)。コンテナーのサイズに対する制限はなく、従来と同一の動作を行う。
  - 値を設定する場合は、64 MB より大きい正の整数とする。
- 注意事項
  - 表スペースの最大サイズを指定するレジストリー変数ではない。
    - ◆レジストリー変数の設定値まで、コンテナー・サイズが達した場合には、新規にコンテナーが追加され、表スペースは拡張される。
    - ◆このレジストリー変数を使用した場合には、1つの自動ストレージ・パスに、1表スペースあたり複数のコンテナーが作成される場合がある。



# 参考: DMS表スペース

## □ DMS(Database Managed Storage)表スペース

- データベース・マネージャーが記憶スペースを管理
- 作成時にスペースを割り当てる
- コンテナーはファイル、デバイス
  - ファイルの操作に対してはファイル・システムI/Oを使用
  - ロード・デバイスの操作に対しては直接I/Oを使用
- 柔軟なデータ配置
  - 表用、索引用、および長形式のデータ用の表スペースを別々に作成することが可能
  - ディスクのIOを複数の物理ディスクに分散させることが可能
- コンテナの追加/削除/拡張/縮小が可能
  - データは自動的に再バランス(オプションにより再バランスさせないことも可)
- 高パフォーマンス



データ、インデックス、長形式を別々に配置する DMSの例

# 解説

## □ DMSの特徴

- DMSでは、表スペースは大きな1つのファイルまたはローデバイスに相当します。中をどのように使われているのかはOSからは確認できません。表スペースを作成する際に、あらかじめ必要な大きさを定義する必要があります。DMSではさらに2つのタイプがあり、1つはDMSファイルともう一つはローデバイスです。
- DMSファイルの場合は、ファイルシステム上に1つの大きなファイルが作成されますが、ローデバイスの場合はファイルシステムを経由せずにDB2が直接IOを行います。
- DMSでは、表用、索引用、および長形式のデータ用の表スペースを別々に作成することができます。それによって、ディスクのIOを複数の物理ディスクに分散させることができます。
- DMSでも複数のコンテナーを1つの表スペースに対して定義でき、さらに動的に追加することができます。表スペースを作成後にコンテナーの大きさを変更することもでき、DB2 UDB V8以降では小さくすることも可能です。
- 複数コンテナーからDMS表スペースを構成する場合、どれかのコンテナーが一杯になつても空いているコンテナーを探して書き込もうとします。全てのコンテナーが一杯になった場合、それ以上データの追加はできません。  
➤V8.2.2からは、ファイルDMS表スペースの自動サイズ変更が可能です。

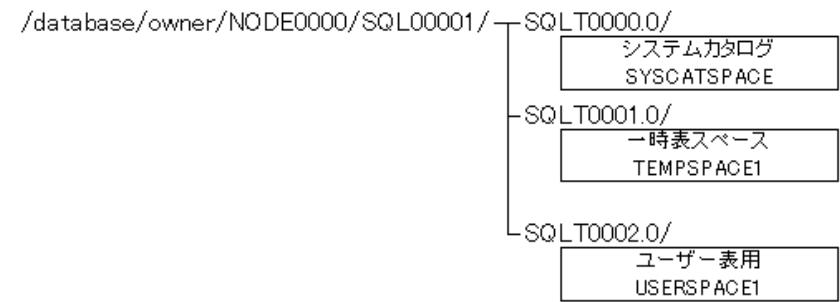


# 参考: SMS表スペース

## □SMS(System Managed Storage)表スペース

- オペレーティング・システムのファイル・システム・マネージャーが管理
- 表データと索引、Longデータは全て同じ表スペースを共有
- 管理が容易
- コンテナーはディレクトリー
- ファイルは動的に拡張し、サイズの上限は以下によって決まる
  - コンテナーの数
  - ファイル・システム/ドライブ/ファイルのOSの限界サイズ
- コンテナーは動的に追加不可
  - ファイル・システム/ドライブのサイズは増加可能
  - 再定義は表スペース復元時に可能(表スペースのリダイレクション)
  - 各コンテナーサイズは同じ大きさに
- 一時表スペースに推奨

CREATE DB DB1 on /database (データベース作成例)



# 解説

## □ SMSの特徴

- SMSでは、表スペースはファイルシステムのディレクトリに相当し、表や索引はそれぞれ別々のファイルになります。SMS表スペース(ディレクトリ)を作成したファイルシステムが許す限り、表に対してデータを追加することができます。つまり、表スペースの大きさはファイルシステムの大きさに依存します。
- SMSでは表や索引およびLOBやLONG VARCHARなどの長形式のデータを含むロング形式のデータを全て同じ表スペースに格納する必要があります。
- 複数のディスクにIOを分散させる為に、1つの表スペースに対してコンテナーを複数定義することがSMSでも可能ですが、コンテナーの追加や削除を行うことはできません。表スペースを作成するときのコンテナー定義を変更する為には、一度削除して再作成が必要があります。
- 複数コンテナーによってSMS表スペースを作成した場合、どれかのファイルシステムが一杯になると表スペースはそれ以上のデータを追加することができなくなります。



## 4. エクステント・サイズ

### □ エクステントとは

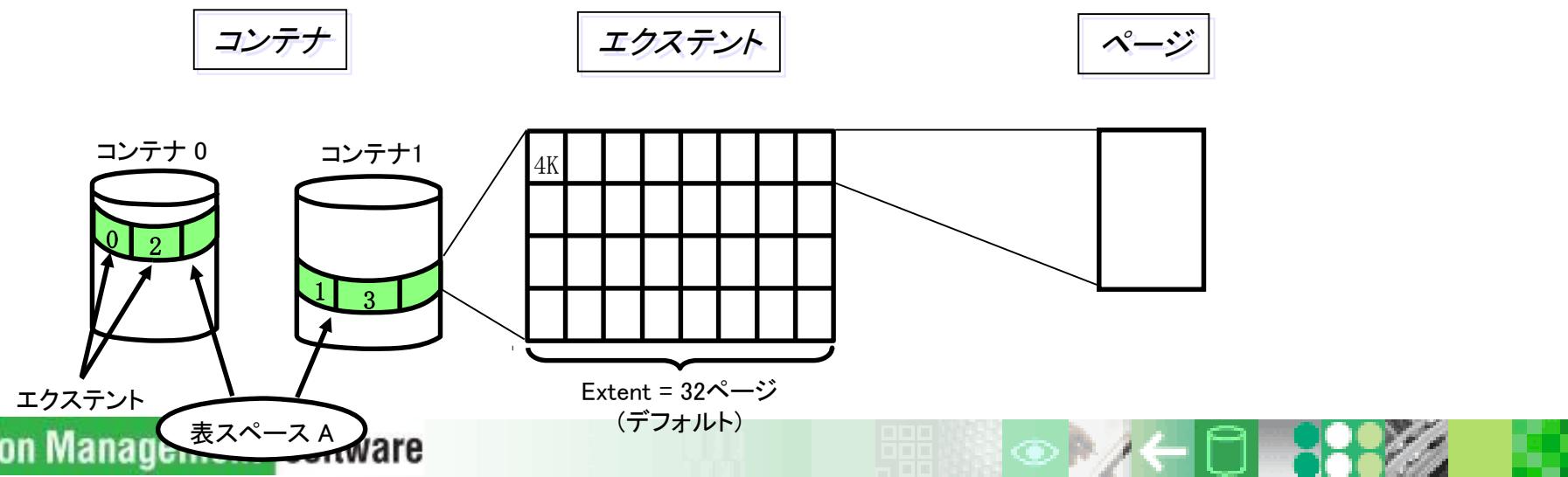
- 表スペースにおけるコンテナ内の領域の割り振り単位(ページ)
  - 一度にエクステント全体をアロケートし、その後でエクステント内のページを使用

### □ データはラウンド・ロビン方式でコンテナに書き込む

### □ DMSのコンテナーで最小限必要なスペース(最初のオブジェクトを作成した時) = 5エクステント + 1ページ

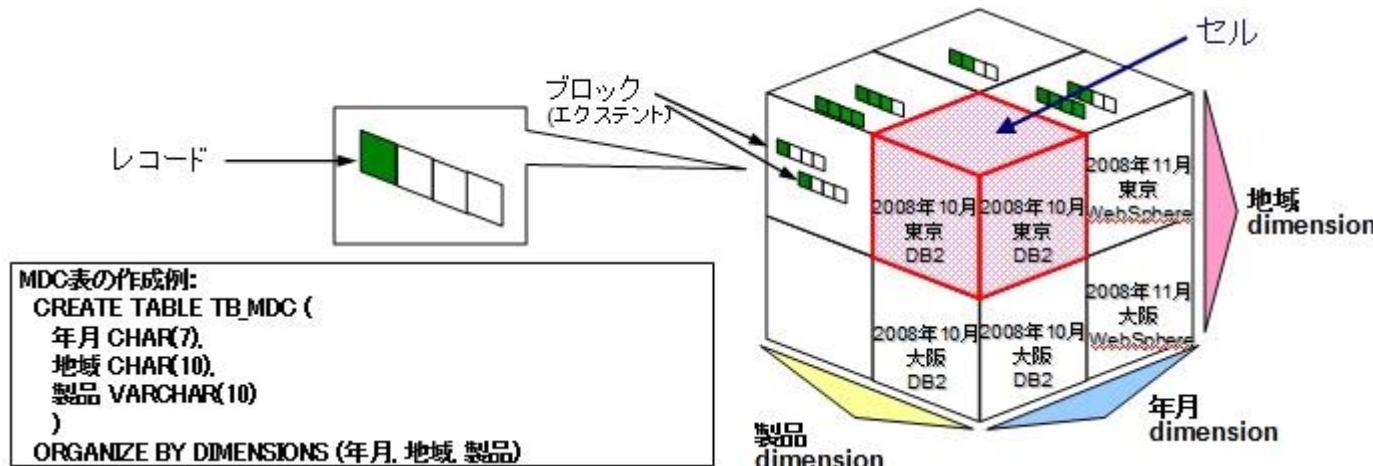
### □ エクステント・サイズのデフォルトは、DFT\_EXTENT\_SZ(DB構成パラメーター)の値で決まる(デフォルトは32ページ)

- 多数の小さい表からなる表スペースは、サイズを減らすことも検討
- 大量の結果行を返す照会を行う表や急激に拡張される表からなる表スペースは、サイズを増やすことも検討



## 参考: MDC表とエクステント・サイズ

- MDC表は、次元に対する表スペースの割り振り単位がエクステント(ブロック)となります。つまり、次元キーの値1種類につき、必ず1エクステントが割り振られます。
- ユニーク性の高い列を次元に設定してしまうと、1エクステントに入る行数が少なくなり無駄なスペースが発生します。そのため、通常表に比べて、表のサイズが急激に増え、また、I/O、バッファーポールの効率も低下します。
- ユニーク性が比較的高い列を次元に設定したい場合には、エクステント・サイズを小さめにすることも考えられますが、バックアップやロック索引によるスキャン時のスループットが低下する弊害があります。
- バックアップやロック検索によるScan時にはエクステント単位でI/O要求が出されるため、エクステントの単位が小さい場合、I/Oのブロック・サイズもまた小さくなってしまう。
  - たとえば、最小のエクステントである4KBページ×2をエクステント・サイズとした場合、その表スペースに対するBackupのブロック・サイズは8KBとなり、バックアップのスループットが低下しやすい。
  - エクステント・サイズの決定においては、セル密度および上記スループットのトレードオフとなる。
- I/O効率の観点からは、エクステント・サイズはRAIDストライプ・サイズと同等か整数倍であることが望ましいといえます。



## 5. プリフェッヂ・サイズ

### □ プリフェッヂとは

- いくつかのページがアプリケーションで必要になることを想定してディスクからバッファー・プールへ読み出すこと
  - 各プリフェッヂャーは、1エクステントずつ読み取る

### □ プリフェッヂ・サイズとは、コンテナーからの入出力の並列度を決める

□ db2\_parallel\_io(レジストリー変数)の設定によっても、並列処理の計算方法が異なる

□ プリフェッヂ・サイズを指定しない場合、エクステント・サイズ、コンテナ数、およびコンテナあたりの物理ディスク数に基づいて、表スペースに適したプリフェッヂ・サイズをDB2が自動計算する

- ①がデフォルトの設定となるため、1表スペース1コンテナーの場合、並列I/Oとはならない
  - ストレージ・パスに、複数のディレクトリーを設定すると、1表スペース複数コンテナーとなる

並列処理の計算方法	表スペースのプリフェッヂ・サイズ	DB2_PARALLEL_IO 設定	並列処理は以下に等しくなる
	① AUTOMATIC	設定されない	コンテナーの数
②	AUTOMATIC	表スペース ID or *	コンテナー数 * 6
③	AUTOMATIC	表スペース ID:n	コンテナー数 * n
④	AUTOMATICではない	設定されない	コンテナーの数
⑤	AUTOMATICではない	表スペースID or *	プリフェッヂ・サイズ/エクステント・サイズ
⑥	AUTOMATICではない	表スペース ID:n	プリフェッヂ・サイズ/エクステント・サイズ



# 解説

- エクステント・サイズ、コンテナ数、およびコンテナあたりの物理ディスク数に基づいて、表スペースに適したプリフェッチ・サイズをDB2が自動計算する(V8.2以降)
  - 利点: コンテナの追加・削除に応じ、プリフェッチ・サイズを変更する必要がない
  - 考慮点: コンテナ数、物理スピンドル数が多いほど、多く設定される
- 設定方法
  - データベース構成パラメーター DFT\_PREFETCH\_SZ の値を AUTOMATIC に設定する
  - CREATE TABLESPACE ステートメントで、PREFETCH サイズを指定しない
- 算出式
  - プリフェッチ・サイズ = (コンテナ数) \* (コンテナあたりの物理ディスク数) \* エクステント・サイズ
- DB2\_PARALLEL\_IO の値との関連
  - DB2\_PARALLEL\_IO の値は、コンテナあたりの物理ディスク数として使用される
    - (例1) DB2\_PARALLEL\_IO=\*<br/>> 全ての表スペースは、デフォルトで、各コンテナあたり6個の物理ディスクをもつと仮定して計算される<br/>> 全ての表スペースに対して、パラレルI/Oが有効となる。プリフェッチ要求は、プリフェッチサイズをエクステント・サイズで割った数に分割されて、パラレルに実行される
    - (例2) DB2\_PARALLEL\_IO=\*:3<br/>> 全ての表スペースは、コンテナあたり3個の物理ディスクを持つと仮定して計算される<br/>> 全ての表スペースに対して、パラレルI/Oが有効となる。
    - (例3) DB2\_PARALLEL\_IO=\*:3,1:1<br/>> 全ての表スペースは、コンテナあたり3個の物理ディスクを持つと仮定して計算される。ただし、表スペース 1 だけは、物理ディスク数は1となる。<br/>> 全ての表スペースに対して、パラレルI/Oが有効となる。
- DB2\_PARALLEL\_IO が設定されていない場合には、物理ディスク数は1



# 6. バッファープール

## □ バッファープールとは

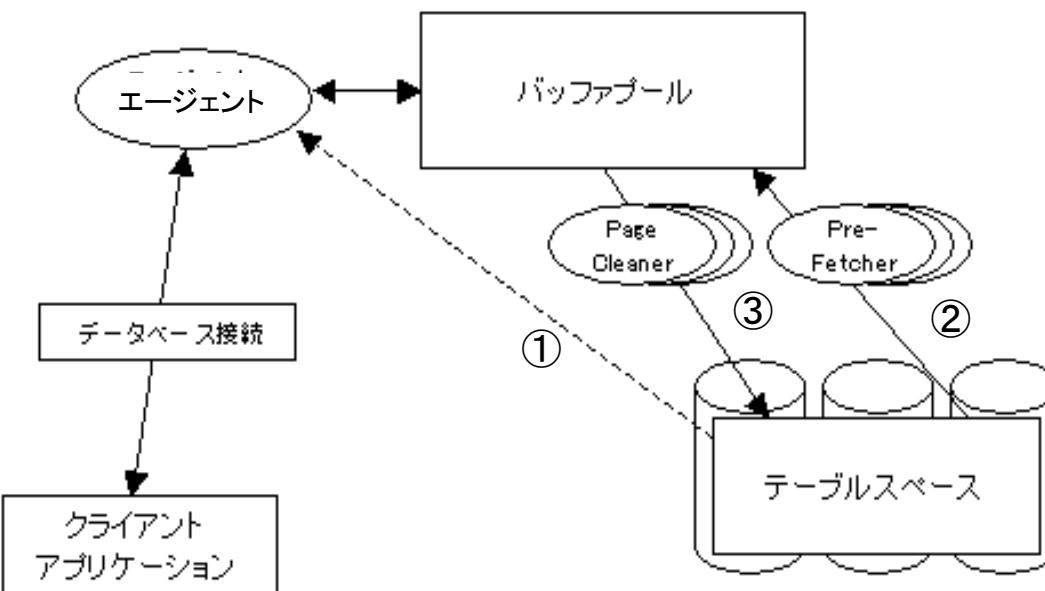
- データ・ページや索引ページを一時的に読み込んだり、変更したりするためのキャッシュ領域
- LOB、ラージ・データは、バッファープールを使用しない
  - LOBをインライン格納した場合は、バッファープールを使用する

## □ バッファープールは、各ページ・サイズ毎に少なくとも1つ作成する

- データベース作成時に、デフォルトのバッファープール(IBMDEFAULTBP)が作成される

## □ 各表スペースは、それぞれ特定の1つのバッファープールに関連付けられる

- 紐付ける表スペースとバッファープールは、同じページ・サイズである必要がある
- 複数の表スペースが同じバッファープールを使用することもできる
- 一時表スペース用のバッファープールとデータ用のバッファープールを分ける、データ用のバッファープールと索引用のバッファープールを分けることもできる



- ① 小さなデータを読み込むときは、エージェントが直接読み出しバッファープールに書き込む
- ② 連続したデータを読み込むときは、DB2が判断してプリフェッチャーは先読みを行う
- ③ バッファープールから表スペースへデータを書き出すプロセスをページ・クリーナーが行う



# 解説

- バッファープールは、各ページ・サイズ毎に少なくとも1つ作成する必要があります。
- データベースの表スペースのページ・サイズが1種類に統一されているケースで、バッファープールを割り当てる場合、一つの大きなバッファープールを使用し、すべての表スペースは、同じバッファープールを共用する設計が可能です。この場合、複数のバッファープールがないので、各バッファープールのヒット率をモニタリングし、バッファープール間でサイズ調整を図る必要がないため、運用管理が容易な構成と言えます。
- データベースの表スペースで、ページ・サイズが2種類あるケースで、バッファープールを割り当てる場合、データ量やアクセス頻度に応じて、割り当てるメモリー・サイズを決定し、バッファープールを2つ以上用意することになります。
- さらに、データやアプリケーションの性格が解っている場合、同じページ・サイズのバッファープールを分割しチューニングすることによって最適化を図ることもできます。一時表スペース用のバッファープールとデータ用のバッファープールを分ける、データ用のバッファープールと索引用のバッファープールを分ける、アクセス頻度により、バッファープールのヒット率を上げたいマスター系の表のバッファープールを別にするなどが考えられます。



# 参考: バッファーポールのサイズ

- 経験則として、データの5%をバッファーポールに割り当てると言われていた
- サーバーに搭載されたメモリーのうち、OSや他のソフトウェアが使うメモリーを引いた残りのメモリーをDBのメモリー領域としてアサインする
- 目安としては、物理メモリーの1/2~3/4程度
- トランザクション系のシステムでは、物理メモリーの75%、分析系のシステムでは、物理メモリーの50%をバッファーポールに割り当てるとのガイドがある
  - Buffer pools should make use of about 75% (OLTP) or 50% (OLAP) of available memory (Buffer Pools).
    - 『Best practices for tuning DB2 UDB v8.1 and its databases』
      - <http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0404mcARTHUR/>
- 実際には、バッファーポールのヒット率を確認しながらサイズを調整するのが一般的
  - db2pd -bufferpoolsでモニタリング
  - データであれば、80-85%以上、索引であれば、90-95%以上のヒット率をめざす
- 頻繁にアクセスする表や索引をバッファーポールにキャッシングさせるにはどのくらいのサイズが必要かを検討する
- バッファーポールのサイズは、オンラインで変更可能
- バッファーポールは、STMM(自己チューニング・メモリー)の対象にできるので、サイズ調整をDB2に管理してもらう構成も可能





ブランク・ページです



## 7. ダイレクトI/O

### □ ダイレクトI/Oとは

- ファイル・キヤッショとバッファープールでの二重バッファリングを回避
  - 二重バッファリングを行うCPU負荷を軽減
  - ファイル・キヤッショに余分なデータをのせないことによる、本来のファイル・データのヒット率向上
  - 大量にファイル・キヤッショを使用することによって引き起こされるページングを回避(システム全体の性能が不安定になることを回避)

### □ 表スペース作成時に、NO FILE SYSTEM CACHINGと指定する

- ユーザー表スペースで指定しない場合のデフォルトは、環境やファイル・システムのタイプ等により異なる
  - FILE SYSTEM CACHING:
    - JFS on AIX
    - Linux System z
    - Solaris 上のすべての非 VxFS ファイル・システム
    - HP-UX
    - すべてのプラットフォーム上の SMS TEMPORARY 表スペース・ファイル、すべての LOB および ラージ・データの場合
  - NO FILE SYSTEM CACHING:
    - その他のすべてのプラットフォームおよびファイル・システム・タイプの場合

### □ SMS表スペースで、ダイレクトI/Oを設定すると、フラッシュ・コピー対象とすることができる



# 解説

- コンテナーにファイルを使ったDMS表スペースやSMS表スペースの場合、表や索引のデータはOSが提供するファイル・システムのファイルに格納されます。ダイレクトI/Oの使用により、ファイルシステムを使うことによるオーバーヘッドを削減することができ、CPUコストの低下が期待できます。
- また、ダイレクトI/Oを使用することで、ファイルキャッシュに余分なデータが乗らないことになり、本来ファイルキャッシュを十分に活用すべきファイルデータについてキャッシュヒット率の向上が期待できます。
- ダイレクトI/Oを使用しない場合のデメリットとして、ファイル・キャッシュを大量に消費し、その結果引き起こされるページングによってシステム全体の性能が不安定になることが挙げられます。ダイレクトI/Oを使用すると、この問題を回避することができます。
- V8.2.2以降、SMS、およびDMS (File)の一時表スペースに対しても、ダイレクトI/O(およびコンカレントI/O)を行うことができるようになりました。これにより、従来は避けることのできなかった一時表スペースに読み書きされるページの二重バッファリングを回避することができるようになりました。また、大量データを処理する際などに、一時表がファイルキャッシュを大量に使用することによりページングが発生し、システム全体の性能が劣化するという問題を避けることができます。
- ダイレクトI/Oが一時表スペースにも使用できるようになったことで、FlashCopyを使ってDBのバックアップ取得を行っていたり、スタンバイDB作成を行う場合に発生していた考慮点が一部緩和されます。
- こちらの考慮点については、テクニカルフラッシュ「DM05-018 AIXプラットフォームにおけるStorageのFlashCopy機能を用いたDB2のバックアップに関するガイド」に詳しい記述があります。
- SMSの一時表スペースにダイレクトI/Oが使用されるように設定する場合には、SMSの一時表スペースをフラッシュコピー対象とすることができます。



# 8. DROPPED TABLE RECOVERY

## □ 表スペースのDROPPED TABLE RECOVERY属性

- ドロップされた表をリカバリー可能にするための属性
  - 表スペース作成時、または変更時に、DROPPED TABLE RECOVERY属性を指定する(REGULAR表スペースにのみ指定可能)
  - V8以降のデフォルトはON
- DROP TABLE ステートメント実行時
  - DROPされた表を識別するための項目がログ・ファイルとリカバリー履歴ファイルに書き込まれる
- 表のDROP時に書き込まれる内容
  - ログ・ファイルに書き込まれる情報
    - 表の名前
    - タイム・スタンプ
    - トランザクションID
    - 表ID
  - 回復履歴ファイルに書き込まれる情報
    - 表の名前
    - タイム・スタンプ
    - トランザクションID
    - 表ID
    - DDL

## □ 必要がなければOFFにする

- 影響範囲
  - 誤ってDROPした表の回復を行うしくみを利用できなくなる
- 定期的に大量に表を作成/削除を繰り返すような業務がない場合には、デフォルトのままで業務パフォーマンスへの支障はない



# 解説

## □ 表スペースのDROPPED TABLE RECOVERY属性

- 表スペース作成時、または変更時に、DROPPED TABLE RECOVERY属性を指定することで、ドロップされた表がリカバリー可能になります（REGULAR表スペースにのみ指定可能）。
- DROPPED TABLE RECOVERY属性を持つ表スペースに対してDROP TABLEステートメントが実行されると、DROPされた表を識別するための項目がログ・ファイル内に作成されます。また、この項目はリカバリー履歴ファイルにも追加され、表を再作成する際に使用されます。

## □ 必要がなければOFFにする

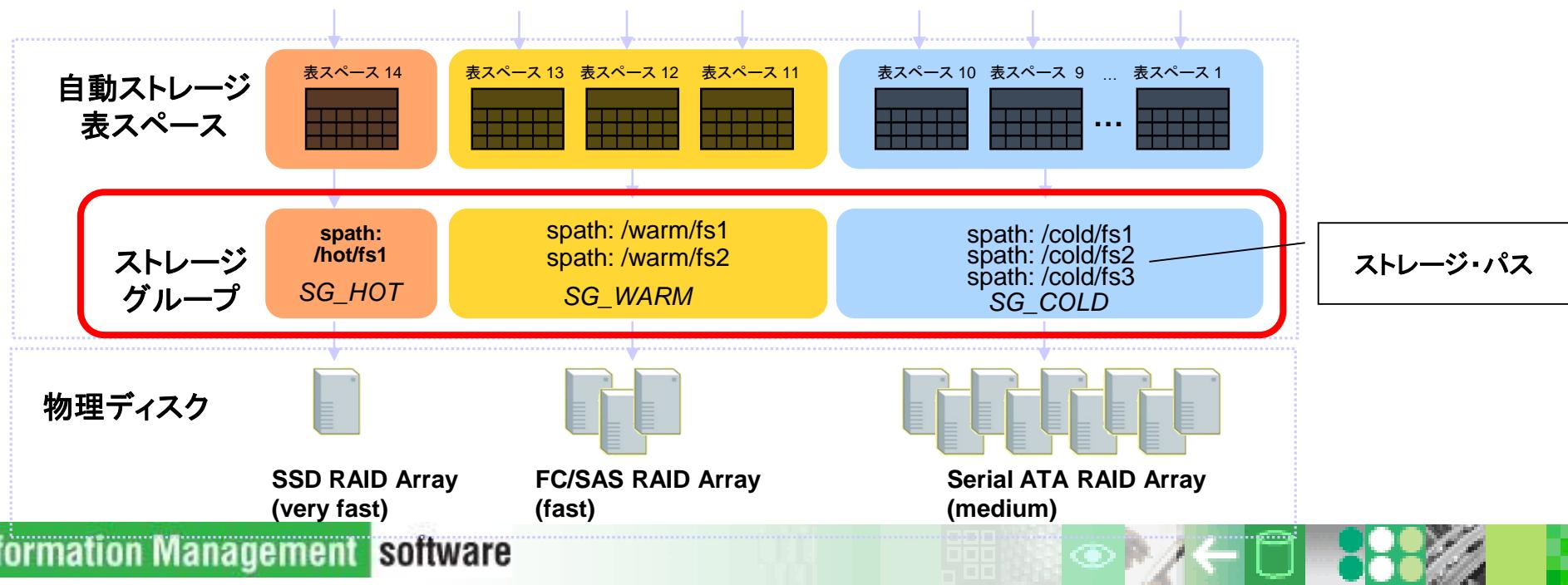
- V8以降では、CREATE TABLESPACE時のDROPPED TABLE RECOVERY属性はデフォルトでONになっています。これにより、表のDROPに関する情報の記録が行われることになるため、大量に表をDROPした際に、DROP処理のパフォーマンスが劣化する可能性があります。これを回避するため、必要がなければOFFにするようにしてください。
- OFFにする方法
  - 1. CREATE TABLESPACE文によって表スペースを作成する際、DROPPED TABLE RECOVERYオプションを明示的にOFFにします。
  - 2. 既に作成されている表スペースについては、ALTER TABLESPACE文により、DROPPED TABLE RECOVERYオプションをOFFにします。
- ONのままにする場合は、PRUNE HISTORYコマンドによって回復履歴ファイルを定期的に削除するようにしてください（容量が増大し続けるため）。



# 9. ストレージ・グループ(V10.1以降)

## □ ストレージ・グループとは

- データを保管できるストレージ・パスの集合名
  - ストレージ・グループを使用するのは、自動ストレージ(AMS)表スペースのみ
  - 自動ストレージ・データベース(データベース作成時に、AUTOMATIC STORAGE YESを指定(デフォルト)であれば、IBMSTOGROUPというデフォルト・ストレージ・グループが自動的に作成される
    - IBMSTOGROUPに紐づくストレージ・パスは、データベース作成時にONオプションで指定したディレクトリーとなる
    - ストレージ・グループを指定しない場合には、デフォルト・ストレージ・グループであるIBMSTOGROUPになる
- 高速アクセスが必要な新しいデータ、アクセス頻度の低いデータを異なるストレージに配置したい場合に、デフォルト以外のストレージ・グループを定義する(Multi-temperature Storage)



# デフォルトで作成されるストレージ・グループ

```
$ db2 "create db sample on /dbland1"
$ db2 get snapshot for db on sample
```

データベース・スナップショット

〈略〉

自動ストレージ・パスの数

= 1

自動ストレージ・パス

= /dbland1

ノード番号

= 0

状態

= 使用中

```
$ db2pd -d sample -storagegroup
```

Database Member 0 -- Database SAMPLE -- Active -- Up 0 days 00:02:15 -- Date 2012-05-28 18:16:00

Storage Group Configuration:

Address	SGID	Default	DataTag	Name
---------	------	---------	---------	------

0x0700000050BC0000	0	Yes	0	IBMSTOGROUP
--------------------	---	-----	---	-------------

明示的に「AUTOMATIC STORAGE NO」をつけない限り、自動ストレージ・データベースが作成される。  
なお、「AUTOMATIC STORAGE NO」の指定はV10.1では非推奨

データベース・スナップショットを取得すれば、自動ストレージ・データベースかどうかが分かる

Storage Group Statistics:

Address	SGID	State	NumPaths	NumDropPen
---------	------	-------	----------	------------

0x0700000050BC0000	0	0x00000000	1	0
--------------------	---	------------	---	---

デフォルトで「IBMSTOGROUP」というストレージ・グループが作成される。

Storage Group Paths:

Address	SGID	PathID	PathState	PathName
---------	------	--------	-----------	----------

0x0700000050BC0120	0	0	InUse	/dbland1
--------------------	---	---	-------	----------

# ストレージ・グループの使用方法

- ストレージ・グループを作成する (alter/drop stogroup も可能)

```
CREATE STOGROUP SG_FAST ON '/ssd/path1', '/ssd/path2'
CREATE STOGROUP SG_SLOW ON '/hdd/path1', '/hdd/path2'
```

- 特定のストレージ・グループを指定して、自動ストレージ表スペースを作成する

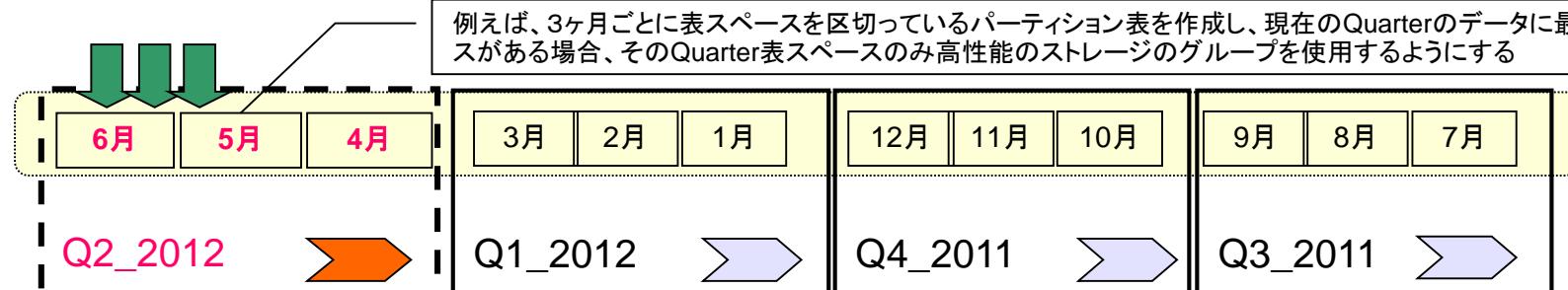
```
CREATE TABLESPACE Q3_2011 USING STOGROUP SG_SLOW
CREATE TABLESPACE Q4_2011 USING STOGROUP SG_SLOW
CREATE TABLESPACE Q1_2012 USING STOGROUP SG_SLOW
CREATE TABLESPACE Q2_2012 USING STOGROUP SG_FAST
```

- パーティション表を作成する

```
CREATE TABLE TAB1(C1 INT NOT NULL, C2 DATE) PARTITION BY RANGE(C2) (
PARTITION Q3_2011 STARTING '2011-07-01' ENDING '2011-09-30' IN Q3_2011,
PARTITION Q4_2011 ENDING '2011-12-31' IN Q4_2011,
PARTITION Q1_2012 ENDING '2012-03-31' IN Q1_2012,
PARTITION Q2_2012 ENDING '2012-06-30' IN Q2_2012)
```

例えば、3ヶ月ごとに表スペースを区切っているパーティション表を作成し、現在のQuarterのデータに最も頻繁にアクセスがある場合、そのQuarter表スペースのみ高性能のストレージのグループを使用するようにする

Tab1

STORAGE GROUP **SG\_FAST**STORAGE GROUP **SG\_SLOW**

/hdd/path2



ブランク・ページです

## ⑤表スペース容量の見積もり



# 物理設計の流れ

①表・索引定義の作成

②データ容量の見積もり

③インスタンスの構成と  
データベース分割

④表の分類と  
表スペースの構成

⑤表スペース容量の見積もり

⑥ディスク上への  
オブジェクトの配置

⑦ユーザーと権限の設計

⑧構成パラメータの設定

⑨シェル／コマンドの作成

- 表スペースの見積もり
- ログ領域の見積もり
- ワーク/バックアップ領域  
の見積もり
- 製品インストール・ディレ  
クトリーの見積もり



# 表スペースの見積もり

## □ システム・カタログ表スペース

- システム・カタログ表が使用する容量は、表スペースのタイプ(SMS or DMS)とエクステント・サイズによって異なる。
  - DMSの場合、各表オブジェクトについて最低 2つのエクステントが割り当てられる。この未使用的スペースを削減するために、カタログ表スペースをDMSで作成する場合は、エクステント・サイズを小さくすることを(2から4ページ)を検討する。
- データベース作成時、デフォルトでは自動サイズ拡張が使用可能なDMS(ファイル)表スペースとして作成される(初期サイズはDB2が自動決定)ため、表スペース容量が不足する前に自動的にサイズが拡張される。
  - ファイル・システムがいっぱいになると自動拡張が失敗するため、余裕をもって作成しておく。(通常、数百MB ~1GB程度あれば問題なし。)
  - 自動ストレージ表スペースであるため、ALTER TABLESPACEによる拡張はできない。

## □ ユーザー表スペース

- 既に決まっているページ・サイズと平均行サイズから、1ページに格納できる行数を算出
- 予想される行数を格納するために必要となるページ数を算出
  - ROUND DOWN(ページ・サイズ/(論理レコード長)) = 1ページあたりの行数
  - (レコード件数/1ページあたりの行数) \* 安全率 = 必要ページ数
    - 安全率: オーバーヘッド分、PCTFREEの分や、同じ表スペース内でREORGする場合なども考慮

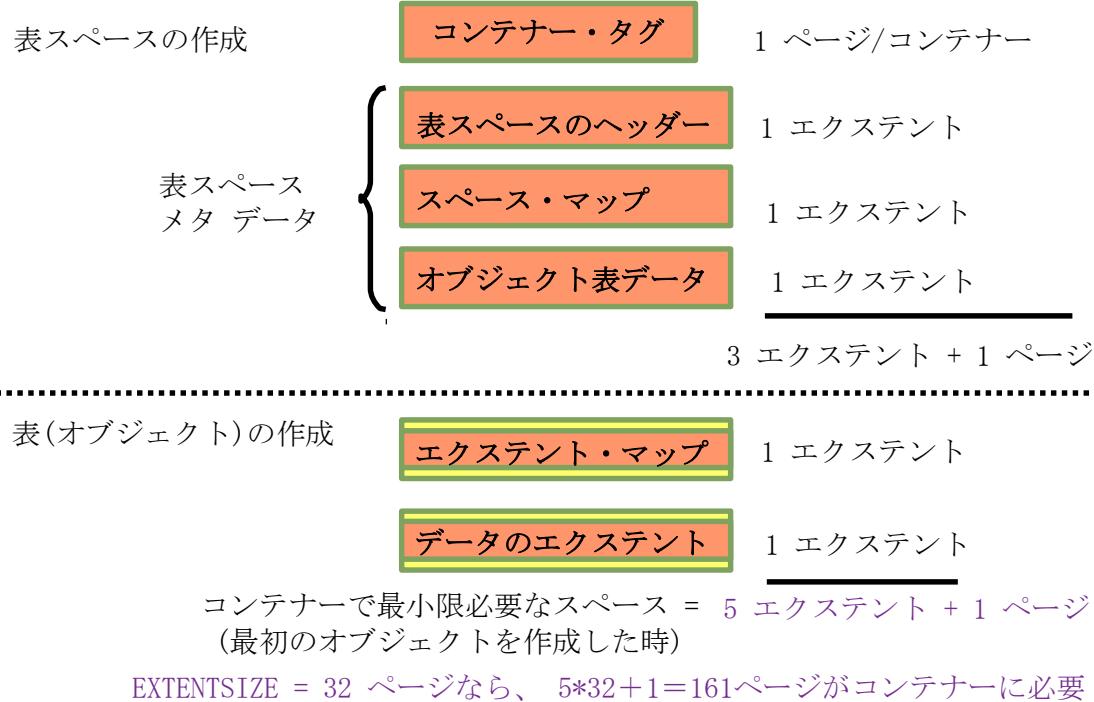
## □ 一時表スペース

- 一番大きく使用すると思われるケースで検討する
- システム一時表スペース
  - JoinやSortを行うアプリケーションやREORG、LOADなどの運用次第で大きく変わる
  - SMS表スペースの使用が望ましい
    - DMSでは一時表の作成時に多くのオーバーヘッドがある
    - SMSではディスク・スペースを動的に割り当てるが、DMSでは事前に割り当てられてしまうため、領域を有効活用できない
- ユーザー一時表スペース
  - 宣言済み一時表/作成済み一時表を使用する場合
  - デフォルトのユーザー一時表スペースはないため、作成する必要がある
- 一時表スペースの見積もりは最終的には事前に入念なテストを行って確認する



# 解説

## □ DMS表スペース - 必要な最小ページ値



- 表の再編成実行時に表スペースの名前を指定しない場合、再編成しようとする表を含む表スペースにその表の作業コピーを保管します。
- 一時表スペースを使って表を再編成する場合、一時表スペースのページ・サイズは表のページ・サイズと一致しなければなりません。
- 宣言済みグローバル一時表/作成済みグローバル一時表は、ユーザー一時表スペースの中に作成されます。デフォルトのユーザー一時表スペースはありませんので、これらの一時表を使う場合には、ユーザー一時表スペースを作成しておく必要があります。
- 一時表スペースの必要なスペースの量は照会および戻される表のサイズや照会アプリケーション、REORG、LOADなどの運用に依存するため、正確に見積もることは難しく、事前にテストで確認することが必要です。



# 自動保守用表スペースの見積もり

## □ 自動保守用表スペース(SYSTOOLSPACE)

- V8.2の新機能、データベースの自動保守(自動統計収集および再編成)を使用する場合に自動作成される表スペース
- 自動統計収集および再編成の作業データを、SYSTOOLSPACE表スペース中の表に格納する
- SYSTOOLSPACE表スペース作成時、約700KB
- 通常、SYSTOOLSPACEに必要な容量は、データベース中の表の数に比例し、1つの表に対して約1KBとして計算する

## □ SYSTOOLSTMPSPACE表スペース

- REORGCHK\_TB\_STATS、REORGCHK\_IX\_STATS、ADMIN\_CMD プロシージャによって一時データの保管に使用されるユーザー一時表スペース

## □ 自動保守用表スペースは、デフォルトでは、自動ストレージ(AMS)表スペースとして作成されるため、ストレージ・パスの容量に余裕があれば、精緻に容量を見積もる必要はない





ブランク・ページです

# ログ領域の見積もり

## □ アクティブ・ログ領域の見積もり

- 最低限必要なサイズ:(logprimary + logsecond) \* (logfilsz + 2 ) \* 4096
  - 循環ロギングの場合
    - 上記の式で求めた値に余裕を持たせたサイズを用意する
  - アーカイブ・ロギングの場合
    - アーカイブ処理の失敗によりログ・ファイルが再利用されない場合や、ロールフォワード時にログがリトリーブされる領域を考慮し、余裕を持たせる(上記の式で求めた値の2倍程度)
- アクティブ・ログの二重化を行う場合(MIRRORLOGPATHを使用)、2倍の容量が必要
- 最大のトランザクション(通常大量更新を行うバッチジョブなど)と並列で実行されるすべてのトランザクションを実行し、使用されるログ容量をスナップショット等により測定する、その数値を元に(データ件数などの増加率などを含めて)見積もり、パラメータ(LOGFILSZ/LOGPRIMARY/LOGSECOND)を調整する
- LOGSECONDは作成時にオーバーヘッドがあるため、原則として、アクティブルogの書き込みはLOGPRIMARYのみで行われるようにLOGPRIMARY数を決定する

## □ アーカイブ・ログ領域の見積もり

- 一日にアーカイブされるログ容量と保持期間を掛け合わせて必要な容量を算出する  
アーカイブ・ログの削除や退避が行われなかつた事態に備え、余裕を持たせて用意する
- 複数パスへのアーカイブを行う場合(LOGARCHMETH1とLOGARCHMETH2を使用)、2倍の容量が必要 FAILARCHPATHを使用する場合は、その分の容量も必要)
- V10.1以降、アーカイブ・ログ領域の圧縮が可能(LOGARCHCOMPR1とLOGARCHCOMPR2を使用)



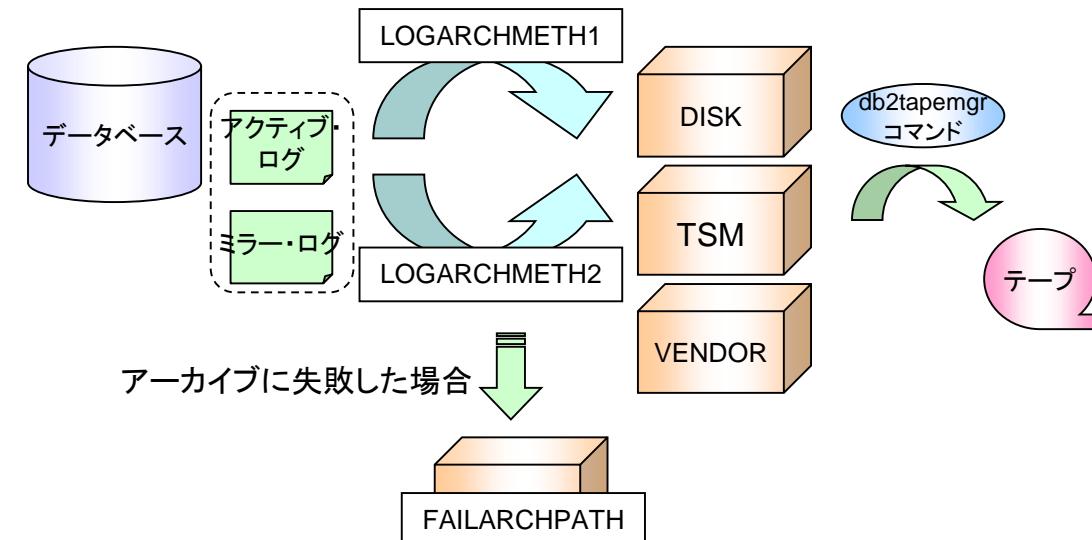
# 参考: ログ・アーカイブ機能

## □ 内容

- アクティブ・ログのアーカイブ機能をDB2の標準機能として提供する
  - V8.1までのログのUSER EXITの機能の代替機能を提供する
- アーカイブ方法を2つまで指定することができる
  - 最大で2つの別個のロケーションにアーカイブ・ログ・ファイルを保管できる
- アーカイブが失敗した場合の代替ディレクトリーを指定できる
  - アーカイブ先が使用可能になると、ログ・ファイルは自動的に移動される

## □ 方法

- 次のデータベース構成パラメーターに、アーカイブ方法を指定する
  - LOGARCHMETH1
  - LOGARCHMETH2
- 次のデータベース構成パラメーターに、アーカイブ・ログ・ファイル用の代替ディレクトリーを指定する
  - FAILARCHPATH
- 次のデータベース構成パラメーターに、アーカイブ・ログ・ファイルの圧縮を使用する場合は設定する(V10.1以降)
  - LOGARCHCOMPR1
  - LOGARCHCOMPR2



# ワーク／バックアップ領域の見積もり

- ワーク領域とは、LOADやIMPORTの入力ファイルを格納するファイルシステムを想定
  - 日次/週次/月次で、一度にデータ投入する量を見積もる
  - データ増加率を考慮する
- LOADをCOPY YESで実行すると、ロードのコピー・ファイルが出力される(出力先は、LOADコマンド実行時に指定する)
  - サイズを見積もる式はないため、テスト環境で実測する
  - サイズは、索引の再作成の量にも依存する
- バックアップ領域とは、データベースをBACKUPコマンドで取得した際のバックアップ・ファイルを保管するファイルシステムを想定
  - バックアップ・ファイルのサイズは、表スペース情報の“最高水準点”(DMSの場合)もしくは、“使用したページ”(SMSの場合)の値に、各表スペースのページ・サイズをかけた結果を合計することで、大まかな値を算出することができる
  - ディスク上に保管する世代数を検討する



# 製品インストール・ディレクトリーの見積もり

- DB2のインストールに必要なディスク容量は、選択するインストールのタイプ、使用するファイル・システムのタイプに応じて異なる
  - DB2セットアップ・ウィザードを使用すると、事前に見積もり可能
    - インストール・タイプ(標準、コンパクト、カスタム)ごとに選択されるコンポーネントに基づいて、動的にサイズの見積もりを行う



## ⑥ディスク上への オブジェクトの配置



# 物理設計の流れ

①表・索引定義の作成

②データ容量の見積もり

③インスタンスの構成と  
データベース分割

④表の分類と  
表スペースの構成

⑤表スペース容量の見積もり

⑥ディスク上への  
オブジェクトの配置

⑦ユーザーと権限の設計

⑧構成パラメータの設定

⑨シェル／コマンドの作成

- ユーザー表スペースの配置
- その他の表スペースの配置
- ログの配置
- ワーク/バックアップ領域の配置
- その他の領域



# ユーザー表スペースの配置

- 複数の物理ディスクに表スペースを配置し、DISK I/Oを分散させる
  - 1つの表スペースを複数の物理ディスクに配置
  - 索引は別の表スペースに配置
  - 長形式は別の表スペースに配置
- 複数のディスクをまとめてLUNにしている場合には、1つのLUNに、1つのストレージ・パスを作成してよい
  - ファイルシステムを細かく切っても、あて先のディスクの分散状況が変わらないため
- 複数の物理ディスクをまとめてLUNとして論理的に分割しているため、LUNの先のディスク数を意識する
  - DB2が自動で計算する並列処理数が妥当であるか確認する
- 基本的には、1つのストレージ・パスに1つのコンテナーが作成されるため、1表スペースに紐づくコンテナーの数は、ストレージ・グループに割り当てるストレージ・パスの数に依存する
- ストレージ・グループに割り当てるストレージ・パスによって、表スペースを配置する物理ディスクを制御する
  - ストレージ・グループに割り当てるパスはすべて、同様のメディア特性(待ち時間、デバイス読み取り速度、サイズなど)を持っている必要がある
  - メディア特性が異なる場合、一貫したパフォーマンスが発揮されない可能性があるため、別のストレージ・グループとして定義する



# 解説

- パフォーマンスのためには、各表スペースに対して、6個～10個以上の物理ディスク上にコンテナーを作成し、I/Oを分散することが理想的であると言われています。
- データはラウンドロビンで書き込まれている為、各コンテナーのサイズは同じであることが理想的です。それによって、データの分散度合いのばらつきを無くし、ディスクI/Oが特定のコンテナーにできるだけ集中しないようにします。
- DB2\_PARALLEL\_IOを指定した場合は、プリフェッチ・サイズが重要になります。RAIDディスクの場合、コンテナーが1つでも実際、内部では複数のディスクが使われています。このような状態では、コンテナー数は先読みに対しては意味を持たず、DB2はプリフェッチ・サイズがエクステント・サイズの何倍になっているかによって、プリフェッチャーを起動し先読みを行います。



# その他の表スペースの配置

## □ 一時表スペース

- ソート、結合、再編成、索引作成時(LOADを含む)等に使用
- バッファープールで処理できる場合は、実際のディスクI/Oは発生しない
- 一時表のI/Oが多く、データ／索引用の表スペースのI/Oと衝突する場合は、別ディスクに配置する
- コンテナーに複数のディレクトリーを指定することが推奨(SMSの場合)

## □ カタログ表スペース

- 通常、カタログ表のデータは殆どキャッシュ上に読み込まれているため、特にI/O効率を考慮したディスク上への配置は不要(データベース作成時のデフォルトのままでも良い)

## □ SYSTOOLSPACE/SYSTOOLSTMPSPACE表スペース

- V8.2以降でデータベースの自動保守(自動統計収集および再編成)を使用する場合に自動生成され、自動保守が実行される際に使用される。デフォルトの配置のままでも特に問題はない。

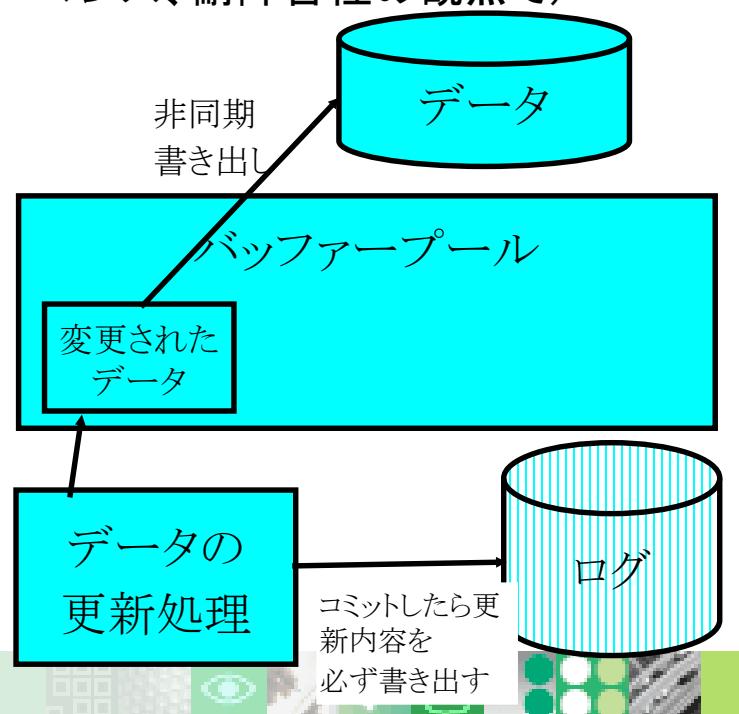




ブランク・ページです

# ログの配置

- アクティブログに対する書き込み速度はアプリケーションのレスポンスに直接影響する
- アクティブログは安全性を考えると、二重化した方がよい
  - (アクティブログの障害=データベースの障害) である
  - DB2の二重ログ(MIRRORLOGPATHの設定)は、両方に書き出して終了なので、どちらか1つのディスクが遅い場合、ログへの書き込みは遅くなる
  - DBにおける設定ではなく、OSやディスクレベルで二重化することも検討する
- ディスク容量が許すのであれば
  - アクティブログ専用のディスク上に配置する(パフォーマンス、耐障害性の観点で)
    - 別ディスク、別パスにできればより良い
  - ストライピングによって書き込みを速くする
    - RAID10など



# 解説

- ログはDB2にとって重要なオブジェクトです。特にアクティブログは非常に重要で、アクティブログが損傷したことはデータベース自身が損傷したことを意味します。
- つまり、アクティブログが壊れたら、データベースが壊れたのと同じです。
- 何故、アクティブログがそれほど重要なのでしょうか。更新処理を行う時のDB2の動作を考えてみます。
- データベースが更新された場合、バッファープールと呼ばれるキャッシュのデータが更新されますが、ディスクにはすぐには書き出されません(このようなデータが含まれるページをデータページと呼びます)。その更新処理がコミットされると、必ずログに書き出されます。
- この状態でもしDBサーバーがクラッシュした場合、次にデータベースが使われる前に、クラッシュリカバリーという処理を行なう必要があります。クラッシュリカバリーによって、更新されたはずなのにディスクに反映されていない更新を、ログを読みながらディスクへ反映します。
- ここで、重要な点はログがなければクラッシュ前にどのような更新処理が行われていたのかを知ることができないということです。DB2はこのクラッシュリカバリーによって、突然のDB2の異常停止などの時でもデータベースに格納されているデータの論理的な整合性を保証しています。つまり、ログが壊れるということはDB2がデータの整合性を保証できないということを意味しています。
- ログの重要性を理解したところで、そのI/Oパフォーマンスが非常に重要であることも気が付かれたと思います。更新処理がコミットされたら必ずログに書き出される訳ですから、ログへの書き出しが終わらないと次の処理へ進むことができません。
- 更新処理のボトルネックがログのI/Oにならないようにする為には、他のI/Oとの衝突をできる限り避ける必要があります。他のI/Oとは表や索引のデータが含まれるディスクへのI/Oが含まれます。つまり、ログは表や索引などとは全く別のディスクに配置することが推奨されます。ESSのような1つのアレイが大きな場合でも、できる限り全く別のディスク上に配置し、SCSIやファイバーチャネルなどのインターフェースも別にすることが推奨されます。
- また、書き込みに対して多少不利なRAID5よりもできればミラーリング(RAID1)、ミラーリングとストライピングの組み合わせ(RAID01, RAID10)などを選択するようにしてください。



# ログの配置(続き)

## □ アーカイブログとアクティブログは別のディスクに配置する

- アーカイブログ・ディレクトリ(コピー先)とアクティブログ・ディレクトリ(コピー元)は別にした方がパフォーマンス的に有利
- 障害時の危険分散を考えても別の方がよい
  - アクティブログとアーカイブログが壊れた場合、バックアップからのリストアはできても、最新状態へのロールフォワードができない。

## □ 大量更新を行うバッチ系処理の場合(Importなど)

- ログに対するキャッシュに相当するログバッファー(LOGBUFSZ)を大きくする

## □ ローデバイスロギングは、V9.1以降は推奨されない

- ローデバイスロギングを使用すると、二重ログが使えない
- 何らかの問題があって、ログのコピーに失敗した場合、手動によるコピーが不可

## □ ログ・アーカイブ機能(V8.2以降)を使用する場合は、アーカイブ先ディレクトリの容量と配置に注意する

- 二つのアーカイブ先ディレクトリ(LOGARCHMETH1, LOGARCHMETH2)を使用する場合
  - 領域は2倍必要
    - 代替ディレクトリ FAILARCHPATH を使用する場合は、更に領域が必要
  - それぞれ別のディスクに配置する

## □ FlashCopyによるデータベースのオンライン・バックアップを取得する場合、アーカイブログとアクティブログは別LUNに配置する

- アーカイブログは、FlashCopyの対象外とすべき領域のため、FlashCopy対象のLUNとは別に配置する



## 解説

- アクティブログの重要性を説明してきましたが、DB2にはもう一つアーカイブログというログがあります。通常、ログ・アーカイブ機能によって、アクティブログがアーカイブされた時に、そのアーカイブ・ログをアクティブログとは別のディレクトリ(またはテープのような別のメディア)にコピーして保存します。このログファイルをコピーする処理も重要です。
- コピー先のログファイルは、コピー元やデータベース自身の障害発生時に復元の為に必要になるので、コピー元とは全く別のディスクやメディアに保存する必要があります。また、ログ・アーカイブ機能はデータベースが使われている時に動作しますので、もし同じディスクにコピー先を配置してしまうと、コピー先の書き込みI/Oの為にアクティブログ自身のI/Oへ影響を与える可能性があります。データベースのパフォーマンスを維持するためには、できる限りこのような影響は排除する必要があります。
- 1つの作業単位(UOW)によって大量のデータを更新するような場合、コミットとコミットの間隔が非常に長くなります。このような場合、データベース構成パラメータのログバッファー(LOGBUFSZ)を大きくするとパフォーマンスに好影響を与えます。このような更新処理の場合、ログバッファが小さいと(デフォルトでは32KB)、更新処理がコミットされないとしても、ログを頻繁にディスクに書き出さなければならなくなる為に、パフォーマンスに悪影響があります。
- ログをローデバイスに配置する機能をDB2はサポートしていますが、V9.1以降では推奨されません。ローデバイスロギングを使用する場合、何らかの問題があつてログ・アーカイブ機能によるログファイルのコピーに失敗し、アクティブログのデバイス中に特定のログファイルが残ってしまったようなケースでは、手動でコピーすることができません。障害時の復旧作業は、通常時とは異なり、多種の問題が発生している可能性があり、手動で復旧できないということは、安全性が低いとも言えます。
- また、DB2の二重ログの機能はローデバイスで使用することはできません。



# ワーク／バックアップ領域の配置

## □ バックアップ時間の短縮が必要な場合、バックアップ領域のパフォーマンスを考慮する

- ディスクへのバックアップの場合、バックアップ元のあるディスクとは別にする
- 1つのディスクでは要求が満たせない場合、複数のディスクへのバックアップを検討
- テープへのバックアップは、TSMがサポートしていれば、複数テープへの同時書き込みによって高速化が可能

## □ ロードの処理時間短縮が必要な場合

- ロード元のデータを格納するファイルシステムのパフォーマンスを考慮する
- AIXでは、以下のパラメータがパフォーマンスに影響する場合がある
  - vmtune or iooコマンドのminpgahead, maxpgahead(ファイルシステムの先読み)

## □ 一時表スペースと同居できるか

- ロード時の入力ファイルとロードのCOPY YESの際に出力されるコピー・ファイル、ソート用の一時表スペースのI/Oが競合するとロードのパフォーマンスに悪影響が考えられる
- 通常は、再編成時にはロードやバックアップなどの処理は行わない。



# その他の領域

## □ DB2診断情報格納ディレクトリー

- インスタンスホーム・ディレクトリー(デフォルト)、または、DIAGPATH(DBM構成パラメーター)にて指定した場所に格納される
- 管理通知ログ、db2diag.log、ダンプ・ファイル、トラップ・ファイル、コア・ファイル(UNIXのみ)を格納する
  - V9.7以降のLinux, UNIX環境であれば、DIAGSIZE(DBM構成パラメーター)を設定することで、管理通知ログとdb2diag.logの最大サイズを制御することができる
  - ダンプ・ファイル、トラップ・ファイル、コア・ファイルは、インスタンスのクラッシュなど、問題が起きた場合に生成される
    - コア・ファイルは非常に大きくなる可能性があるため、DIAGPATHの容量が逼迫しないように注意する

## □ データベース・ディレクトリー

- バッファーポール情報、表スペース情報、データベース構成情報、リカバリ履歴ファイル、ログ制御ファイル

## □ インスタンスホーム・ディレクトリー

- インスタンスオーナーのホーム・ディレクトリー

## □ DB2の製品インストール・ディレクトリー



# 参考: 回復オブジェクトの自動削除

## □ 回復処理に不要となった回復オブジェクトをDB2が自動的に削除する

- 新しく提供される、AUTO\_DEL\_REC\_OBJ(DB CFG)を設定することにより、pruneされた回復履歴ファイルに紐づいているバックアップ、ロードコピー、アーカイブログを物理的に削除する
- TSMへのバックアップ取得、自動保守バックアップ機能も対象となる

## □ 対象回復オブジェクト(物理ファイル)

- バックアップファイル
- ロードコピーファイル
- アーカイブログファイル

## □ メリット

- DB2が自動的に回復オブジェクトの管理を行う
  - 回復処理に必要となる、バックアップ、アーカイブログ、ロードコピーファイルを一括して管理を行なうため、誤って一部の回復オブジェクトだけを削除してしまうというオペレーションミスを避けることが可能となる
- メンテナンス用のシェルが不要となる
  - DBAは、事前に運用設計を行い、適切な構成パラメーターを設定することで、回復オブジェクトのメンテナンスをDB2が自動的に行なってくれるため、世代管理を行なうためのシェルなどを作成する必要がない

## □ 考慮点

- Q - replicationを使用して、ログを別DBへapplyするケース、また、HADRでアーカイブログをスタンバイ機へログシッピングするケースでは、適用オブジェクトへログが適切に適用されてから消し込みを実施するという運用にする必要がある
  - ログの保管世代を要件を満たすように設定する必要ある





ブランク・ページです

## ⑦ユーザーと権限の設計



# 物理設計の流れ

①表・索引定義の作成

②データ容量の見積もり

③インスタンスの構成と  
データベース分割

④表の分類と  
表スペースの構成

⑤表スペース容量の見積もり

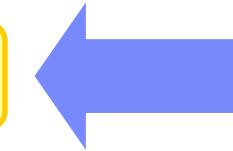
⑥ディスク上への  
オブジェクトの配置

- DB2のユーザー
- 権限
- DB特権の制限

⑦ユーザーと権限の設計

⑧構成パラメータの設定

⑨シェル／コマンドの作成



# ユーザーと権限の設計

## □ DB2システムに必要なユーザーを識別する

- UNIX/Linuxシステムでは、少なくとも以下のユーザーが必要
  - インスタンスオーナー
  - fencedユーザー

## □ DB権限に応じたユーザーIDを用意するかどうかを決定する

- インスタンスオーナーはDB2インスタンスに対してSYSADM権限を持つ
- すべての作業をインスタンスオーナーのユーザーIDで実施することはセキュリティ上好ましくない
- 必要なユーザーとそれらのユーザーに与える権限を定義する
  - アプリケーション開発者
  - アプリケーション実行ユーザー
  - 運用管理者など
- セキュリティを強化するためには、権限を分け、ユーザーの役割に対して必要十分な権限のみ付与するようとする

## □ DBに対してPUBLICユーザーの特権を制限するかどうかを決定する

- デフォルトでは、OSで認証されたユーザーは、すべてPUBLICユーザーとしてDBに接続したり、表を作成したり、カタログ表を参照することができる
- CREATE DATABASEでRESTRICTIVEオプションを指定することにより、PUBLICユーザーの特権を制限することができる



# DB2に必要なユーザー

## □導入するために必要なユーザー(UNIX, Linux)

- インスタンスオーナーのユーザー名がDB2のインスタンス名になる
- インスタンスオーナーのグループは、自動的にSYSADMグループになる

	ユーザー デフォルト	グループ デフォルト
インスタンスオーナー	db2inst1	db2iadm1
Fencedユーザー (fencedモードのUDFやストアドプロシージャの実行ユーザー)	db2fenc1	db2fadm1

## □Windowsでは、コンピュータのAdministratorグループに属するユーザーで導入



# DB2に対する権限の概要

新規DB環境を構築すると...

データベース

SECADM

セキュリティ管理

DBADM

データベース管理権限

DATAACCESS

ACCESSCTRL

WLMADM

SQLADM

EXPLAIN

インスタンス

SYSADM

システム管理

SYSCTRL

システム制御

SYSMAINT

システム保守

SYSMON

システム・モニター



SYSADMはSECADM、DBADM権限を付与できない

SYSADMには暗黙的にDBADM権限が付与されない(SYSADMが作成した場合はDBADMが付与される)

# インスタンスレベルの権限

## SYSADM…システム管理権限

### ・最高位の権限

- sysadm\_group DBM構成パラメーターによって指定されたグループに割り当てられる
- DB2インスタンスの構成(DBM CFGパラメータの設定)を行うことができる  
データベースのアップグレード/データベースのリストアが可能
- 他ユーザー やグループへの権限付与／取り消しを行うことができる(SECADM以外)
- DBを作成した場合は、自動的にDBADM、SECADM権限を付与されるが、SYSADMに対して、暗黙のDBADM,SECADM権限は付与されない

## SYSCTRL…システム制御権限

- sysctrl\_group DBM構成パラメーターによって指定されたグループに割り当てられる
- インスタンスの開始／停止
- データベースや表スペースの作成・削除
- 既存データベースのバックアップ／リストアを行うことができる
- データベース内のデータへの直接アクセスはできない

## SYSMAINT…システム保守権限

- データベースの構成(DB CFGパラメータの設定)を行うことができる
- データベースや表スペースのバックアップ／リストアを行うことができる
- データベースの作成／削除、データへの直接アクセスはできない

## SYSMON …システム・モニター権限

- インスタンスやデータベースのモニター・データを取得するための権限
- 各種スナップショットを取得することができる



# データベースレベルの権限

## SECADM…セキュリティ管理者権限

- ・セキュリティ関連のデータベース・オブジェクトの作成および管理
- ・すべてのデータベース権限と特権の付与および取り消し
- ・カタログ表とカタログ・ビューからの SELECT はできるが、ユーザー表データにアクセスできない

## DBADM…データベース管理権限

- ・データベースに対する管理権限
- ・非セキュリティ関連のデータベース・オブジェクトの作成、変更、およびドロップ
- ・ログ・ファイルの読み取り
- ・イベント・モニターの作成、活動化、およびドロップ
- ・表スペースの状態の照会
- ・ログ履歴ファイルの更新
- ・表スペースの静止
- ・表の再編成
- ・RUNSTATS の実行

### ACCESSCTRL

Grant/revoke権限

### DATAACCESS

データへのアクセス権限

### WLMADM

Workload manager用の権限

### SQLADM

SQLチューニング用の権限

### EXPLAIN

EXPLAIN実行用の権限



# CREATE DATABASEのRESTRICTIVE オプション

## □ データベース作成時に、RESTRICTIVEオプションを指定

- (例) CREATE DATABASE database名 **RESTRICTIVE**
- RESTRICT ACCESS データベース構成パラメーターが、YES に設定される
  - 表示のみの構成パラメーターのため、変更は不可

## □ 以下の特権に関し、PUBLICにGRANTされていない状態となる

- CREATETAB
- BINDADD
- CONNECT
- IMPLICIT\_SCHEMA
- スキーマ SQLJ 内のすべてのプロシージャーに対する EXECUTE WITH GRANT
- スキーマ SYSPROC 内のすべての関数およびプロシージャーに対する EXECUTE WITH GRANT
- NULLID スキーマ内で作成されたすべてのパッケージに対する BIND
- NULLID スキーマ内で作成されたすべてのパッケージに対する EXECUTE
- スキーマ SQLJ に対する CREATEIN
- スキーマ NULLID に対する CREATEIN
- 表スペース USERSPACE1 に対する USE
- SYSIBM カタログ表に対する SELECT アクセス
- SYSCAT カタログ・ビューに対する SELECT アクセス
- SYSSTAT カタログ・ビューに対する SELECT アクセス
- SYSSTAT カタログ・ビューに対する UPDATE アクセス



# 解説

- セキュリティ強化のため、PUBLICの権限を制限したDBを作成することができます。
- CREATE TABLE RESTRICTIVEを指定して作成したデータベースでは、上記の権限がPUBLICに許可されていないため、必要な権限を持ったユーザーを作成する(またはPUBLICに対して許可したい権限を選んで付与する)必要があります。





ブランク・ページです

## ⑦構成パラメーターの設定



# 物理設計の流れ

①表・索引定義の作成

②データ容量の見積もり

③インスタンスの構成と  
データベース分割

④表の分類と  
表スペースの構成

⑤表スペース容量の見積もり

⑥ディスク上への  
オブジェクトの配置

⑦ユーザーと権限の設計

⑧構成パラメータの設定

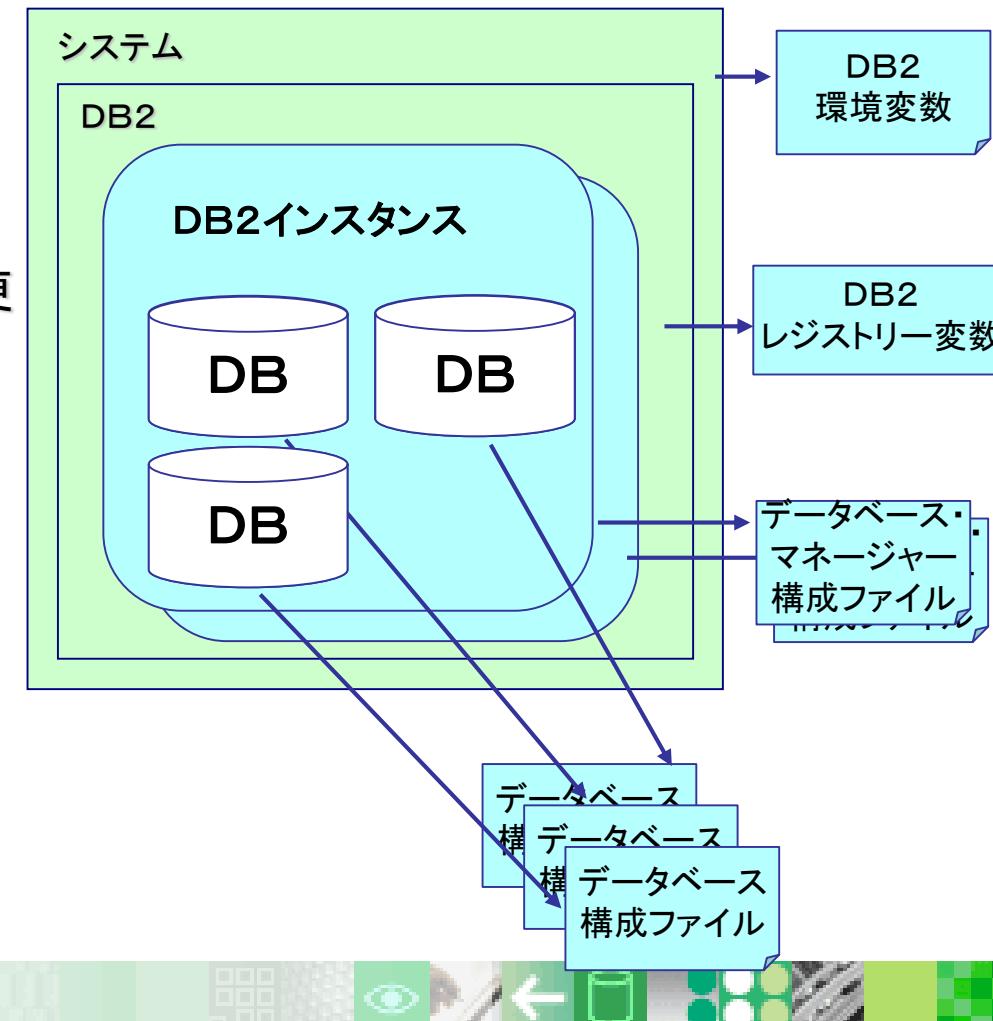
⑨シェル／コマンドの作成

- DB2のプロセスモデル
- メモリーモデル
- レジストリー変数と構成パラメータ

# DB2のレジストリー変数と構成パラメータ

## □ DB2で設定されるパラメーター類

- DB2レジストリー変数
  - DB2インターフェイスや通信パラメーターを規定
  - db2setコマンドで設定
  - ex) DB2COMMなど
- データベース・マネージャー構成パラメータ
  - インスタンスレベルでの設定項目
  - get dbm cfgで確認/update dbm cfgで変更
  - ex) DIAGLEVELやSVCENAME
- データベース構成パラメータ
  - データベースレベルでの設定項目
  - get db cfgで確認/update db cfgで変更
  - ex) MAXLOCKSやLOCKTIMEOUT



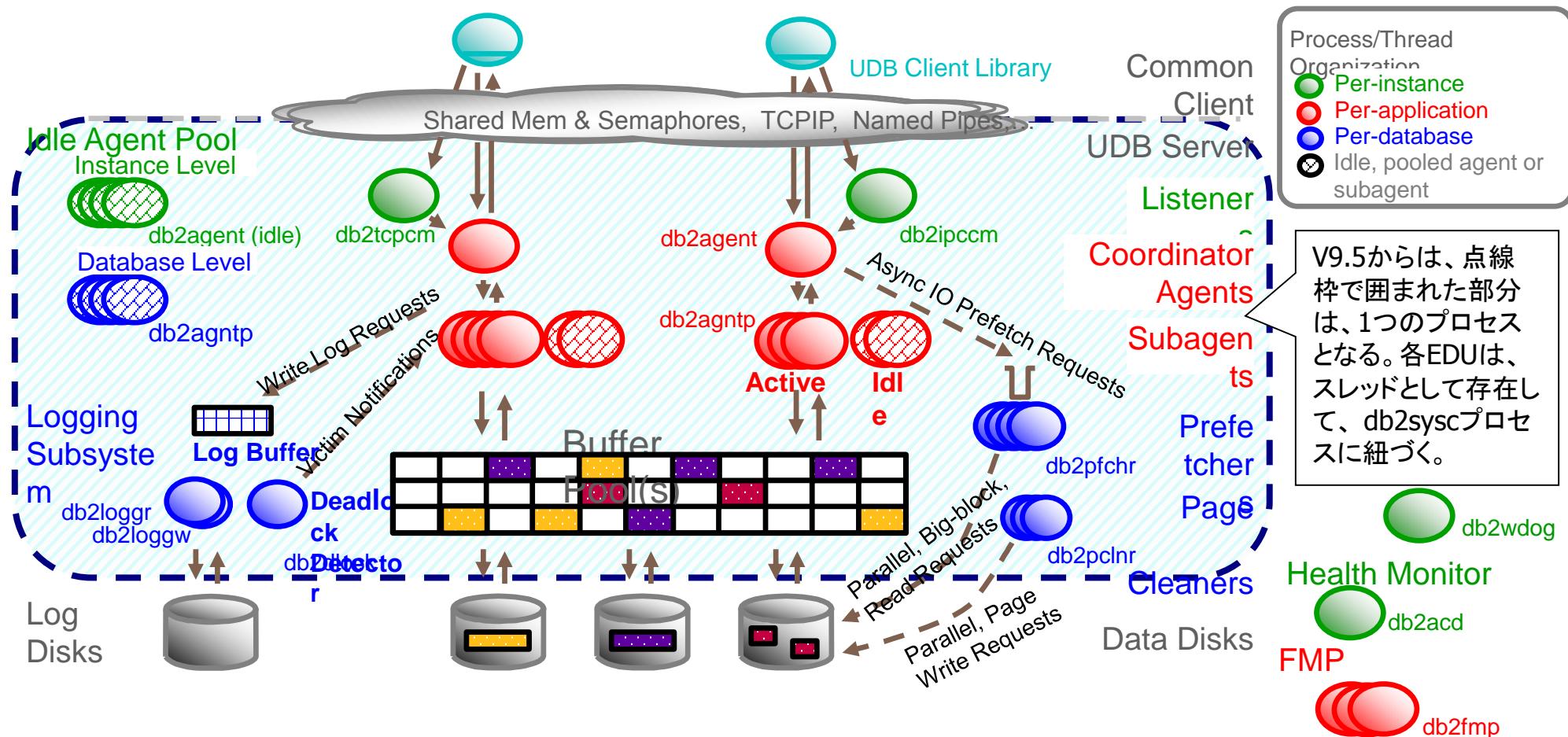
# 解説

- ここでは、DB2の環境を設定するものとして、DB2レジストリー変数と、構成パラメータを設計します。
- レジストリー変数では、環境、通信、パフォーマンス動作などについての設定を行います。
- 構成パラメータは、インスタンスレベルのデータベース・マネージャー(DBM)構成パラメータと、データベースレベルのデータベース(DB)構成パラメータがあります。
- 構成パラメータでは、DB2の作業を行うEDU(ディスパッチ可能作業単位)の数や、メモリ構成、その他のDB構成を設定します。



# DB2のプロセスモデル

- db2syscプロセスが1つ存在し、残りのEDU(エンジンディスパッチ可能単位)はdb2syscプロセス配下にスレッドとして存在する



# 解説

- db2syscプロセスが 1つ存在し、残りのほとんどEDU(エンジンディスパッチ可能単位)は db2syscプロセス配下にスレッドとして存在します。
- 主なEDUには以下のものがあります。
  - db2wdog: 異常終了をハンドルする UNIX および Linux オペレーティング・システムのウォッチドッグプロセス
  - db2acd: ヘルス・モニター、自動保守ユーティリティー、管理用タスク・スケジューラー用デーモンプロセス
  - db2fmd: 障害モニター・デーモン
  - db2agent: クライアントアプリケーションからの接続要求に対応するコーディネーター・エージェント
  - db2ipccm: クライアント接続用リスナー
  - db2tcpccm: TCP/IP 接続用リスナー
  - db2pfchr: バッファー・プール・プリフェッチャー
  - db2pclnr: バッファー・プール・ページ・クリーナー
  - db2lfr: 個別のログ・ファイルを処理するログ・ファイル・リーダー
  - db2loggr: トランザクション処理およびリカバリーをハンドルするログ・ファイルの取扱
  - db2loggw: ログ・ファイルへのログ・レコードの書き込み
  - db2logmgr:: ログ・
  - db2logts: どのログ・ファイルでどの表スペースがログ・レコードを持つかのトラッキング
  - db2lused: オブジェクト使用の更新
  - db2stmm: セルフチューニング・メモリー管理フィーチャー
  - db2wlmd: ワークロード管理統計の自動収集



# DB2のメモリーモデル

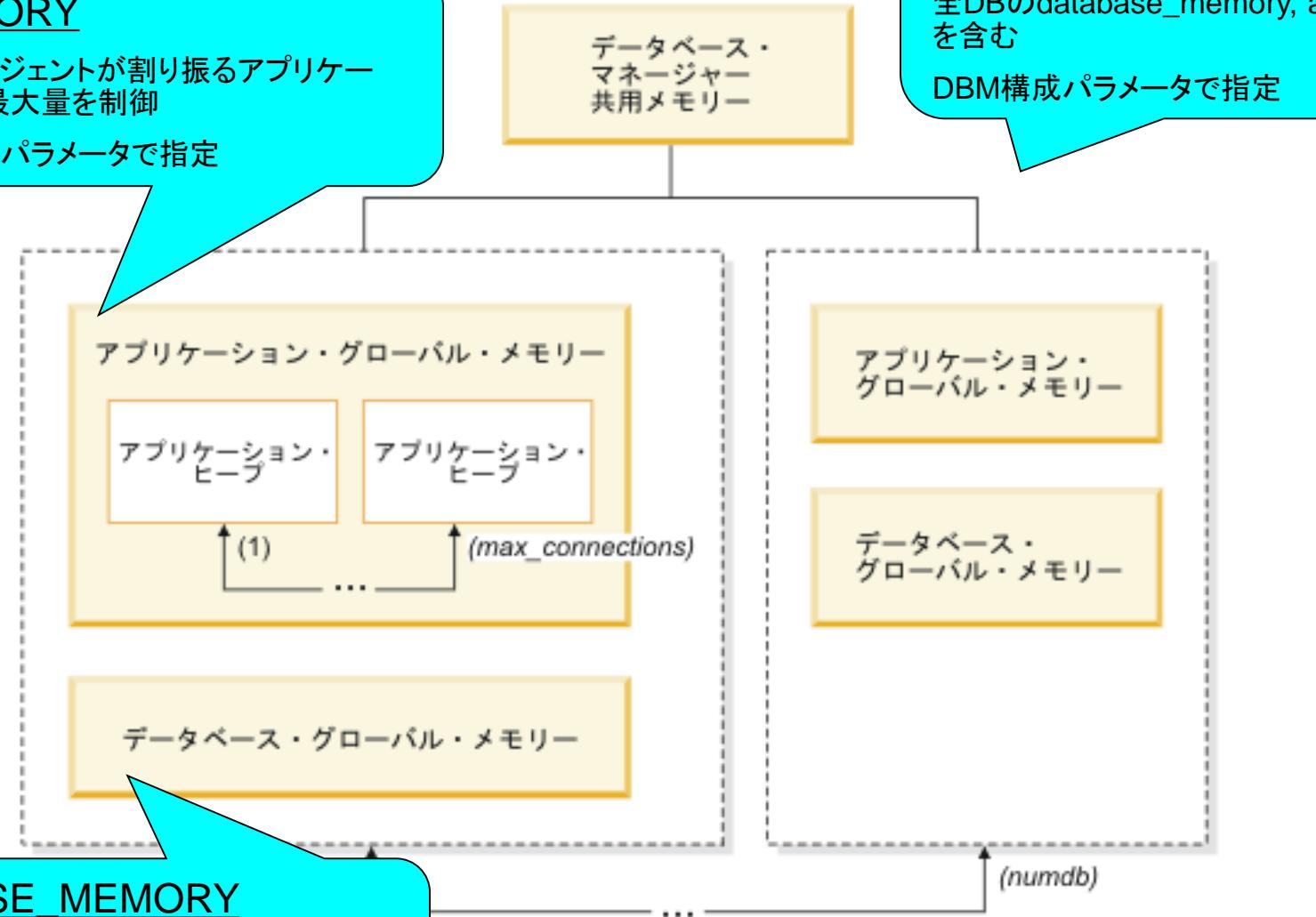
## APPL MEMORY

データベースエージェントが割り振るアプリケーションメモリーの最大量を制御

DBごとにDB構成パラメータで指定

## INSTANCE MEMORY

DB2インスタンスの使用するメモリの全体  
全DBのdatabase\_memory, appl\_memory  
を含む  
DBM構成パラメータで指定



# DB2のメモリーの構成要素(1)

## データベース・マネージャーの共用メモリー (FCM を含む)

モニター・ヒープ  
(mon\_heap\_sz)

監査バッファー・サイズ  
(audit\_buf\_sz)

## データベース・グローバル・メモリー (database\_memory)

ユーティリティ・ヒープ  
(util\_heap\_sz)

バックアップ・バッファー

パッケージ・キャッシュ  
(pckcachesz)

共有ソート・  
メモリー・コンシューマーの  
ソート・ヒープしきい値  
(sheapthres\_shr)

ソート・ヒープ (sortheap)

ロック・リスト (locklist)

カタログ・キャッシュ  
(catalogcache\_sz)

データベース・ヒープ  
(dbheap)

ログ・バッファー  
(logbufsz)

バッファー・プール

## アプリケーション・グローバル・メモリー (appl\_memory)

共用アプリケーション・メモリー

アプリケーション固有メモリー

アプリケーション・ヒープ  
(applheapsz)

統計ヒープ  
(stat\_heap\_sz)

ステートメント・ヒープ  
(stmtheadp)



# DB2のメモリー構成要素(2)



注:カッコ内はそれぞれのサイズを制御する構成パラメータ名



# 主なレジストリー変数

## □ DB2COMM

- データベース・マネージャーを開始したときに開始されるコミュニケーション・マネージャー
- TCPIP接続を持つデータベースでは、以下の設定が必須
  - db2comm=tcpip
  - SSLを使用する場合は、db2comm=ssl

## □ DB2\_PARALLEL\_IO

- 表、索引のスキャンを実行時の、並列実行数を制御する
  - db2\_parallel\_ioを指定しない場合の、表スペースの並列処理の多重度は表スペースのコンテナーの数となるため、コンテナが一つしかない表スペースでは並列入力は行われない
  - 並列処理数は、コンテナ数、表スペースに明示的に設定されるプリフェッチサイズ、db2\_parallel\_ioの組み合わせで決定される
  - 物理設計の流れ「④表の分類と表スペースの構成」-「5.プリフェッチサイズ」参照
  - db2\_parallel\_io=\*
    - \*を設定すると、デフォルトの6並列と解釈される

## □ 集約レジストリー変数

- DB2\_WORKLOADにより、特定の環境に適合するよう複数のレジストリー変数を同時に設定する
  - db2\_workload=SAP : SAP用のデータベースとして適切な設定とする
  - db2\_workload=ANALYTICS : BLUアクセラレーションの使用に適した設定とする

## □ DB2の他DB互換機能を使用する

- DB2\_COMPATIBILITY\_VECTOR
  - db2\_compatibility\_vector=Oracle : Oracle互換機能を使用可能にする



# DBM構成パラメータ – インスタンスマモリの構成

## □ INSTANCE\_MEMORY

- 1つのデータベースパーティションに割り当てることが可能な最大メモリーサイズを指定する
  - デフォルト値: AUTOMATIC
  - automaticの場合、db2start時に稼動サーバーの実メモリーの75% – 95%の間で割り当てられる
  - 実メモリーが割り当たられるわけではなく構成パラメーター値に上限値がセットされる
  - 以下の場合は、メモリの制限が必要
    - 複数インスタンスがある場合
    - WASなど、DB2以外のアプリケーションがあり、デフォルトメモリ量をDB2に割りあてることができない場合

## □ DBM共用メモリの構成

- MON\_HEAP\_SZ: モニターヒープのサイズ
- AUDIT\_BUF\_SZ: 監査(db2audit)バッファーのサイズ
- FCM(DPF使用時の高速通信マネジャー)バッファーの設定

## □ エージェント専用メモリの構成

- AGENT\_STACK\_SZ: エージェントスレッドのスタックサイズ
- JAVA\_HEAP\_SZ: Java UDF またはストアード・プロシージャーを実行するエージェントが使用するヒープのサイズ  
db2fmpプロセスあたり1個
- RQRIOBLK: ブロック・カーソル用のメモリ

## □ エージェント/アプリケーション共用メモリ

- ASLHEAPSZ: ローカル・アプリケーションとその関連エージェントの間の通信バッファー



# その他の主なDBM構成パラメータ

## □ 診断パス、診断ログに関する設定

- DIAGPATH
  - 診断情報用のディレクトリを指定する
  - デフォルトの出力先については、リンク先を参照
    - [DB2 LUW] db2diag.log や管理通知ログの出力先 (IM-10-0AC)
      - ◆ <http://www.ibm.com/support/docview.wss?uid=jpn1J1006159>
  - DIAGLEVEL  
NOTIFYLEVEL
    - db2diag.log、管理通知ログ(インスタンス名.nfy)に書き込むメッセージのレベル
  - DIAGSIZE
    - db2diag.logおよび管理通知ログのファイルの最大サイズを制御する
      - デフォルト値:0(ログファイルのサイズを制限せず、ファイルを循環使用しない)

## □ 権限ユーザーに割り当て設定

- SYSADM\_GROUP
- SYSCTRL\_GROUP
- SYSMAINT\_GROUP
- SYSMON\_GROUP
  - インスタンスレベルの権限ユーザーの設定

## □ イントラパラレルの設定

- INTRA\_PARALLEL
  - パーティション内照会並列処理をデフォルトで使用するかどうかを指定
    - デフォルト値: no
- MAX\_QUERYDEGREE
  - INTRA\_PARALLELをyesにする場合の最大の実行多重度も設定
  - デフォルト値: any

## □ エージェント数に関する設定

- MAX\_CONNECTIONS:クライアント接続最大数
- MAX\_COORDAGENTS:コーディネーター・エージェントの数

# DB構成パラメータ – データベース共用メモリの構成

## □DATABASE\_MEMORY

- データベース共用メモリ(データベース・グローバル・メモリ)全体のサイズを指定する
- 以下の要素が含まれる
  - ()はそれらのサイズを設定するDB構成パラメータ名
  - > バッファープール
  - > データベース・ヒープ(dbheap)
  - > ロックリスト(locklist):
  - > ユーティリティー・ヒープ(util\_heap\_sz)
  - > パッケージ・キャッシュ(pckcachesz)
  - > カタログ・キャッシュ(catalogcache\_sz)
  - > 共有ソート・ヒープ(sheaptres\_shr)
  - > オーバーフロー域
- デフォルト値:automatic
  - > 上記の構成要素のサイズに基づいて初期データベース・メモリー・サイズが計算される
  - > STMM=ONの場合は、database\_memoryサイズ自体もinstance\_memoryの範囲内で動的にチューニングされる
- 固定値を設定した場合
  - > STMM=ONであれば、与えられたメモリ範囲内で、構成要素のうちAUTOMATICであるものおよびオーバーフロー域が調整される
  - > database\_memoryサイズを固定して、かつ、STMMをONにする場合、database\_memory の固定構成の半分以上を、STMM によるチューニング用に残しておくことを推奨
    - バッファープールサイズを、database\_memoryの半分以下にする



# STMM(Self Tuning Memory Manager) 自己チューニングメモリ

## □ DB2が自動的にメモリーチューニングを実施

- 管理者がメモリーチューニングを実施する必要なし
- 予期しないワークロードを検知し、必要に応じてチューニング
  - メモリーの再分配を必要とするワークロードの変化に対応
- メモリーチューニングに労する時間を節約可能
  - 情報収集→分析→設定→テスト→情報収集……

## □ 対象

- データベース構成パラメータ
  - SORTHEAP
  - SHEAPTHRES\_SHR
  - LOCKLIST(MAXLOCKS)
  - PCKCACHESZ
  - DATABASE\_MEMORY
- バッファープール

## □ データベース構成パラメーターSELF\_TUNING\_MEMにて設定

- デフォルト値はON(=STMM稼動)
- STMMによるチューニング対象とするメモリ要素の構成パラメータをAUTOMATICに設定する(デフォルトAUTOMATIC)
  - 例:
    - LOCKLIST=AUTOMATIC
  - SELF\_TUNING\_MEM=OFFに設定することで、各メモリ構成要素のパラメータがAUTOMATICに設定されてもSTMMは停止する

## □ 検証段階でSTMMを稼動させ、最適な設定値になった時点で、STMMを停止する、という使い方も可能



# STMM対象のメモリ要素

- 以下のパラメータはすべてデフォルトAutomaticで、STMM=ONのときSTMMでチューニングされる
- SORTHEAP
  - ORDER BY, GROUP BY, JOINなどのソート処理用のメモリ領域
  - 一つのソート処理に対する割り当てサイズ
  - 不足した場合は、オーバーフローとなり一時表スペースが使用される
- SHEAPTHRES\_SHR
  - ソート処理全体で使用するデータベース共用メモリのサイズ
    - ▶ SORTHEAP × 同時実行ソート処理数
  - 領域不足の場合はオーバーフローとなり、一時表スペースが使用される
  - 共用ソートとする場合は、SHEAPTHRES(DBM構成パラメータ)を0とする
  - STMMでチューニングするためには、共用ソートであることが必要(SHEAPTHRES=0)
- LOCKLIST
  - ロックレコードを保持するためのメモリ容量の上限
  - LOCKLISTサイズが不足するとロックエスカレーションが起こり、トランザクションの並列実行が妨げられる
- MAXLOCKS
  - 1アプリケーションに許されるLOCKLISTの使用パーセント
  - 1アプリケーションで取得するロックの量がMAXLOCKSを超えると、ロックエスカレーションが起こる
- PCKCACHESZ
  - コンパイル済み動的SQLパッケージセクション、静的SQLパッケージを保存するキャッシュ
  - パッケージキャッシュはフラグメントしやすく、サイズ固定した場合でもキャッシュオーバーフローのため、指定したサイズより大きくなるため、database\_memoryも固定する場合には、オーバーフロー一分を見込むこと



# 参考:STMM使用時のバッファーポールサイズ設定方法

## □作成

- バッファーポールサイズを指定し作成、その値からSTMMによるチューニング開始
  - db2 create bufferpool BPNAME (immediate/deferred) size yy automatic
    - yyは初期サイズ
- バッファーポールを作成し、STMMによるチューニング開始  
サイズ未指定でバッファーポールを作成すると、1000ページで作成
  - db2 create bufferpool BPNAME (immediate/deferred)
  - db2 create bufferpool BPNAME (immediate/deferred) size automatic

## □変更

- 設定値をyyに変更し、その値からチューニングを開始
  - db2 alter bufferpool BPNAME (immediate/deferred) size yy automatic
- 現在の設定値からチューニングを開始
  - db2 alter bufferpool BPNAME (immediate/deferred) size automatic
- 設定値をyyに変更し、その値でチューニングを停止
  - db2 alter bufferpool BPNAME size yy



# その他のメモリ関連DB構成パラメータ

## □ DBHEAP

- DBの活動で一時的に必要となるワーキングメモリをアロケートする
- ログバッファー(LOGBUFSZ)を含む
- デフォルト値: AUTOMATIC
  - DBで必要なメモリのキャッシュオールメモリ的に使用されるので、AUTOMATICに設定し、必要に応じて割り振らせることを推奨

## □ APPL\_MEMORY

- アプリケーションを実行するために割り当てることが可能な最大メモリーサイズ
  - DBで消費されるアプリケーションメモリーの合計量をこのパラメーターで制限することができる
- 以下のメモリが含まれる
  - APPLHEAPSZ: アプリケーション全体で消費可能なアプリケーションヒープ
  - STAT\_HEAP\_SZ: RUNSTATSによる統計の収集の際に使用される、ヒープの サイズ
  - STMTHEAPSZ: SQLコンパイルに使用されるヒープのサイズ
- デフォルト値: AUTOMATIC
  - DB活動化の時点で、最少のメモリーが割り当てられ、instance\_memoryの範囲内で増減する



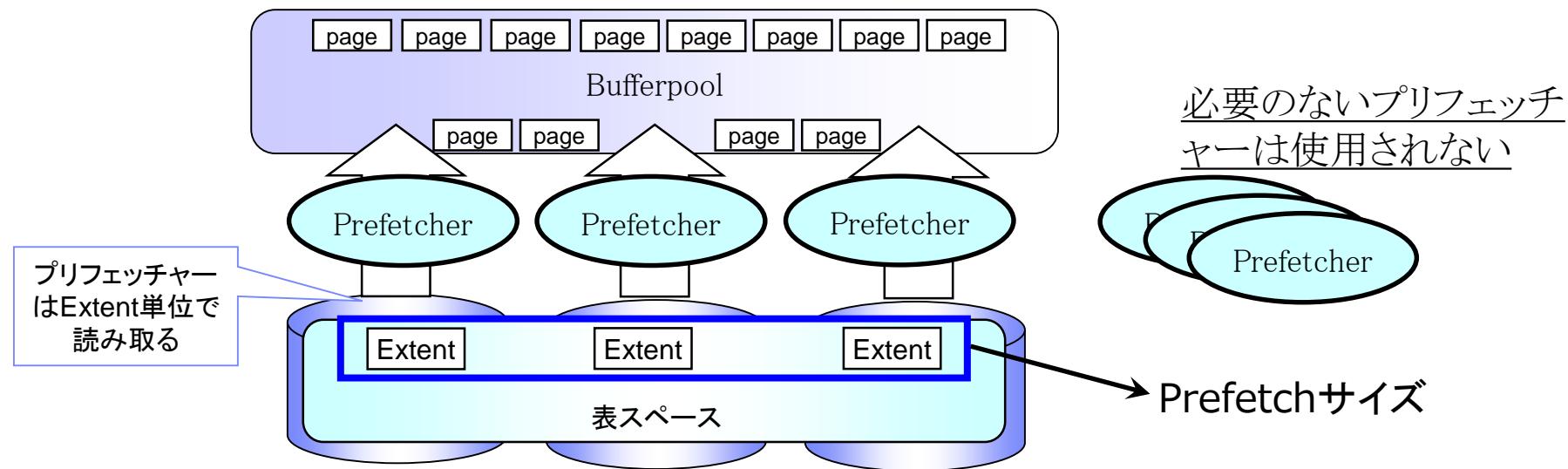
# EDU関連DB構成パラメータ

## □ MAXAPPLS

- データベースに（ローカルとリモートの両方で）接続できる並行アプリケーションの最大数
- デフォルト値: AUTOMATIC

## □ NUM\_IOSERVERS

- 表・索引の先読み用に起動するPrefetcher数
- デフォルト値: AUTOMATIC
  - AUTOMATICの場合、データベース活動化時点で、適正值を計算
  - 固定する場合は、表スペースのPrefetcherサイズ/エクステントサイズ以上
    - 必要のないPrefetcherは呼び出されないため、特に大きなオーバーヘッドはない



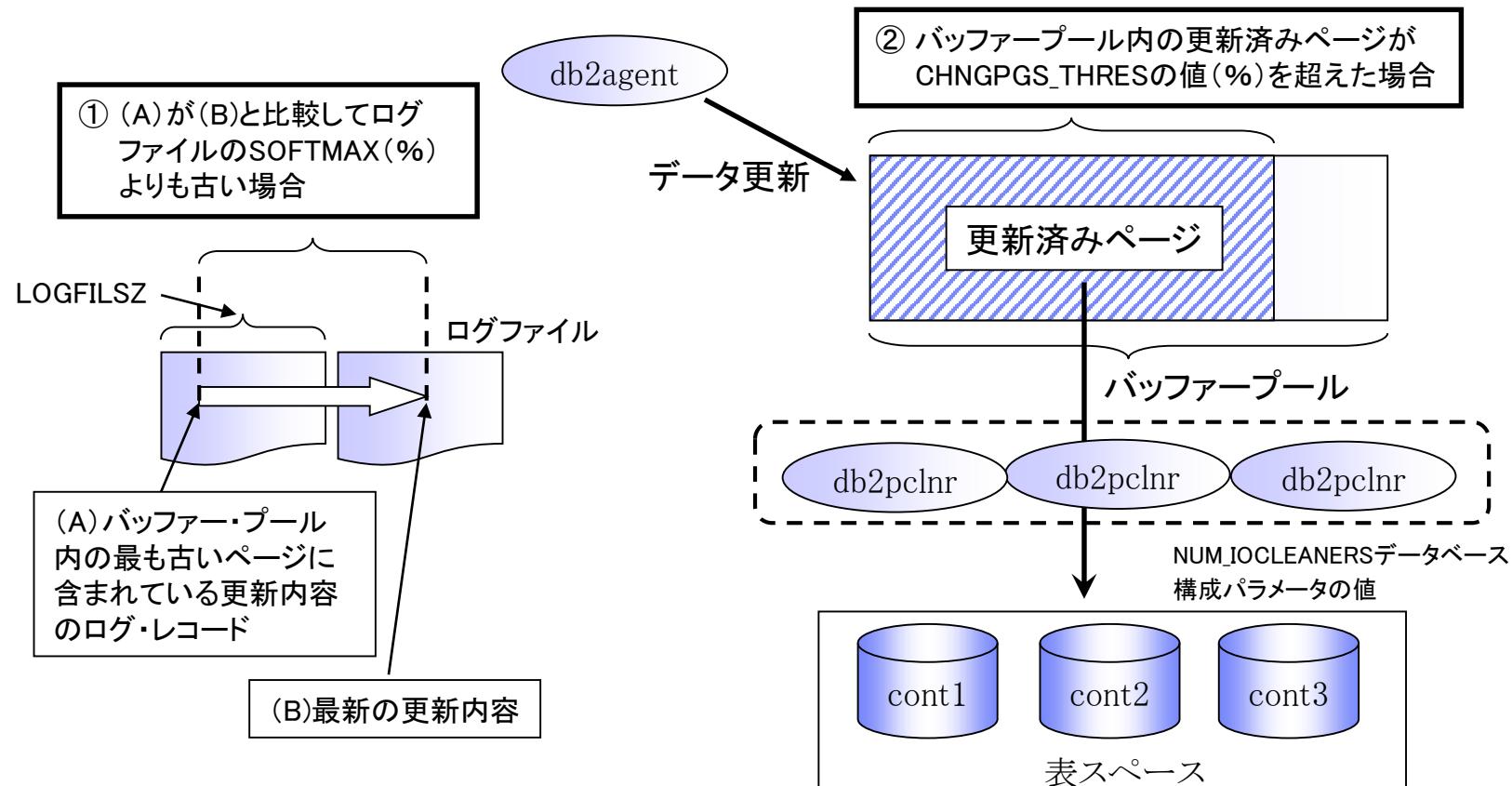
## □ NUM\_IOCLEANERS

- バッファープール内の更新されたページをディスクに書き出すページクリーナーの数
- デフォルト値: AUTOMATIC
  - CPUコア数に基づいて設定される
  - 固定する場合、大きく構成し過ぎると、必要以上にCPUの負荷が大きくなる可能性がある

# ページクリーニングの制御(1)

## □ ページクリーナーは以下の時に呼び出される

- ソフトチェックポイント(SOFTMAXで設定)
- ページ変更しきい値(CHNGPGS\_THRES)に達したとき
- Agentがページを読み込もうとしたときに、書き出し済みのページが無くダーティページスチールが発生したとき



## □ 上記のイベントドリブンのページクリーニングではなく、DB2にあらかじめページクリーニングをさせておくこともできる

- レジストリー変数で、DB2\_ALTERNATE\_PAGE\_CLEANING=ONに設定する

# ページクリーニングの制御(2)

## □ PAGE\_AGE\_TRGT\_MCR

- 更新ページをバッファーポール上に保持しておく期間(秒)を設定し、ページ・クリーニングの頻度を制御
- V10.5からサポート
- デフォルト値: 240(秒)
- SOFTMAX=0のときに有効



# ログ関連DB構成パラメータ

## □ ログファイル関連

- ログ容量の見積もりに応じて、以下の構成パラメータを設定する
- LOGFILSZ
  - 一つのログファイルのサイズ
  - 大きくするとログ・ファイルの切り替えの際にディスクI/Oの負荷が大きくなる
  - 小さくすると頻繁にログ・ファイルが切り替えられるオーバーヘッドがある
  - デフォルト値: 1000(ページ)
    - 一般的には、デフォルトでは小さいので、調整する
    - 50-100MB程度が一般的
- LOGPRIMARY
  - アクティブログの一次ログファイル数
  - アクティブログ容量=LOGFILSZ × (LOGPRIMARY+LOGSECOND) × 4096(バイト)
  - 通常は、LOGPRIMARYだけを割り振った状態で、トランザクションが実行できるように構成する
  - LOGPRIMARYは、DBをactivateする際にすべてアロケートされる
  - デフォルト値: 3
- LOGSECOND
  - LOGPRIMARYを使い切った場合にのみ一つづつアロケートされる
  - LOGSECONDを使い切ると、トランザクションログフルとなる
    - 通常はアロケートされていない状態にしておくと、LOGSECONDがアロケートされたことによりトランザクションログでの問題(未コミットのアプリケーションがあったなど)を発見できる
  - デフォルト値: 10

## □ ログパス関連

- 「ログの配置」の設計に基づき、以下の構成パラメータを設定する
- NEWLOGPATH
  - アクティブログパスを指定したいとき設定
- MIRRORLOGPATH
  - アクティブログを二重化する場合設定する
- LOGARCHMETH1
  - アーカイブ先
- LOGARCHMETH2
  - アーカイブを二重化する際設定する
- FAILARCHPATH
  - LOGARCHMETH1, LOGARCHMETH2のどちらにもアーカイブできないときに、ログをアーカイブする先を指定する



# リカバリオブジェクトの保守用DB構成パラメータ

## □ NUM\_DB\_BACKUPS

- 履歴情報を保持するデータベース・バックアップの数
- 指定数に達すると、回復履歴ファイルで古いバックアップが有効期限切れとマークされる
  - デフォルト値: 12

## □ REC\_HIS\_RETENTIN

- バックアップで履歴情報が保持される日数を指定
- 指定日数以前に取得されたバックアップの情報は、回復履歴ファイルで効期限切れとなる
- デフォルト値: 366(日)
  - 大きすぎるので、必要なバックアップ保存期間を指定する
  - NUM\_DB\_BACKUPSの値と、REC\_HIS\_RETENTINの値は整合性が取れているように設定する
    - たとえば、毎日バックアップを取得し、リカバリ用に3世代分のバックアップを保存したいのであれば、NUM\_DB\_BACKUPS=3, REC\_HIS\_RETENTIN=3
- NUM\_DB\_BACKUPSとREC\_HIS\_RETENTINを超えた回復履歴ファイル内のバックアップおよびログアーカイブ情報は、prune historyの実行および、backup dbコマンドで次のフルバックアップが取得されたタイミングで削除される
- 回復履歴ファイルには、データベースのバックアップ、ログのアーカイブ、表に対するLOADの実行などが記録される
- リカバリ履歴ファイルが大きくなると、回復履歴ファイルに変更を行う操作(LOAD, ログアーカイブなど)に時間がかかるようになるため、回復履歴ファイルサイズは、あまり大きくならないように保つことが望ましい

## □ AUTO\_DEL\_REC\_OBJ

- データベース・ログ・ファイル、バックアップ・イメージ、およびロード・コピー・イメージを、それらに関連する回復履歴ファイルの項目が整理されるときに、削除するかどうかを指定
- デフォルト値: OFF
  - ONに設定した場合、履歴ファイルを整理するときに、対応するアーカイログ・ファイル、バックアップ・イメージ、およびロード・コピー・イメージも削除する



# ロック待ち関連のDB構成パラメータ

## □ CUR\_COMMIT

- Currently Committed動作を使用するかどうか
- デフォルト値: ON
  - CURC\_COMMIT=ONの場合、CSの読み取り処理は、Commit済みデータにロックを取らず、書き込みロックを待たない
  - ONを推奨

## □ LOCKTIMEOUT

- ロックを獲得するために待機する秒数を指定
- デフォルト値:-1
  - デフォルトでは無限にロック待ちしてしまうので、適切な値に設定すること

## □ MON\_LCK\_MEG\_LVL

- ロック・タイムアウト、デッドロック、およびロック・エスカレーションなどのイベントが発生した際、管理通知ログへメッセージを出力するように指定する
- デフォルト値: 1
  - 0: ロック・エスカレーション、デッドロック、およびロック・タイムアウトのメッセージは出力されない
  - 1: ロック・エスカレーションの通知
  - 2: ロック・エスカレーションおよびデッドロックの通知
  - 3: ロック・エスカレーション、デッドロック、およびロック・タイムアウトの通知



# Statement Concentratorの構成パラメータ

## □ STMT\_CONC

- デフォルト値: OFF
  - LITERALS: Statement Concentratorが有効
    - パラメータマークを使用せず条件節にリテラルが指定されている動的SQL文も、再コンパイルせずに既存のパッケージを使用できる
      - SQLはリテラル値以外はすべて同一であること
      - リテラルの長さが異なっていると、同じSQLと見なされない
    - コンパイル済みステートメント再利用するため、リテラルを使った動的SQLコンパイルのオーバーヘッドを削減することができる
    - 統計上、値によって、最適アクセスパスが異なるケースでも再コンパイルしないため、値に応じたパスにはならないことに注意



# 自動保守関連DB構成パラメータ

## □ AUTO\_MAINT

- 自動保守全体の設定を行なうためのデータベース構成パラメーター
- 以下の個々の自動保守を行う場合に必要
  - OFF の場合、全ての保守の自動化設定がOFFとなる
  - デフォルト値: ON
  - 自動保守を行うためには、パラメータをONに
- AUTO\_DB\_BACKUP: データベースの自動バックアップ操作
  - 自動バックアップを実施するためには、パラメータONかつポリシーの設定が必要
  - デフォルト値: OFF
- AUTO\_TBL\_MAINT: 表保守パラメーター (auto\_runstats および auto\_reorg) の親パラメーター
  - AUTO\_REORG: 自動REORGを行うか
    - デフォルト値: OFF
    - 自動REORGを実施するには、パラメータONかつポリシーの設定が必要
  - AUTO\_RUNSTATS: 自動RUNSTATSを実行する
    - デフォルト値: ON
    - 以下の子パラメータがある
      - ◆ AUTO\_STMT\_STATS: リアルタイム統計の収集を有効/無効
      - ◆ AUTO\_STATS\_VIEW: 統計ビューでの自動統計収集を使用可能
      - ◆ AUTO\_SAMPLING: 自動統計収集で大規模な表の統計を収集するときにサンプリングを使用するかどうか





ブランク・ページです

# 構成パラメーターの設定

## □autoconfigureコマンドによる設定

- 構成パラメータ、バッファープール、WLMの主要な値の推奨値を計算する
  - 推奨値をDBに適用するか、参考として表示するのみか選択できる
- データベース作成時に、システムリソース(メモリー、CPU、等)を考慮し、構成パラメーターをデフォルトで自動設定することができる
  - CREATE DATABASEコマンドでは、デフォルトでautoconfigureが実行され、推奨値が設定される
  - 構成パラメータの値として推奨値と異なる設定したいときは、必ず  
`create database autoconfigure apply none`(推奨値を適用しない)を指定すること
    - autoconfigureの推奨値は、各構成パラメータのデフォルト値とは異なるため、構成パラメータをデフォルト値にしようとしていた場合もapply noneを指定すること



# autoconfigureによる推奨例(1)

## □DBM構成パラメータ

Current and Recommended Values for Database Manager Configuration

Description	Parameter	Current Value	Recommended Value
Application support layer heap size (4KB)	(ASLHEAPSZ)= 15	15	15
No. of int. communication buffers(4KB)	(FCM_NUM_BUFFERS)= AUTOMATIC	AUTOMATIC	AUTOMATIC
Enable intra-partition parallelism	(INTRA_PARALLEL) = YES	YES	YES
Maximum query degree of parallelism	(MAX_QUERYDEGREE) = ANY	ANY	ANY
Agent pool size	(NUM_POOLAGENTS) = AUTOMATIC(100)	AUTOMATIC(100)	AUTOMATIC(100)
Initial number of agents in pool	(NUM_INITAGENTS) = 0	0	0
Max requester I/O block size (bytes)	(RQRIOBLK) = 65535	65535	65535
Sort heap threshold (4KB)	(SHEAPTHRES) = 0	0	0



# autoconfigureによる推奨例(2)

## □DB構成パラメータ

Current and Recommended Values for Database Configuration

Description	Parameter	Current Value	Recommended Value
Default application heap (4KB)	(APPLHEAPSZ)	= 256	256
Catalog cache size (4KB)	(CATALOGCACHE_SZ)	= 360	360
Changed pages threshold	(CHNGPGS_THRESH)	= 80	80
Database heap (4KB)	(DBHEAP)	= 6610	6617
Degree of parallelism	(DFT_DEGREE)	= ANY	ANY
Default tablespace extentsize (pages)	(DFT_EXTENT_SZ)	= 4	4
Default prefetch size (pages)	(DFT_PREFETCH_SZ)	= AUTOMATIC	AUTOMATIC
Default query optimization class	(DFT_QUERYOPT)	= 5	5
Max storage for lock list (4KB)	(LOCKLIST)	= AUTOMATIC	AUTOMATIC
Log file size (4KB)	(LOGFILSZ)	= 100	1024
Number of primary log files	(LOGPRIMARY)	= 5	30
Number of secondary log files	(LOGSECOND)	= 19	19
Max number of active applications	(MAXAPPLS)	= AUTOMATIC	AUTOMATIC
Percent. of lock lists per application	(MAXLOCKS)	= AUTOMATIC	AUTOMATIC
Number of asynchronous page cleaners	(NUM_IOCLEANERS)	= 8	1
Number of I/O servers	(NUM_IOSERVERS)	= 36	8
Package cache size (4KB)	(PCKCACHESZ)	= AUTOMATIC	AUTOMATIC
Sort list heap (4KB)	(SORTHEAP)	= 64713	64720



# autoconfigureによる推奨例

## □DB構成パラメータ(つづき)

Description	Parameter	Current Value	Recommended Value
SQL statement heap (4KB)	(STMTHEAP) = 16384	16384	16384
Statistics heap size (4KB)	(STAT_HEAP_SZ) = 4384	4384	4384
Utilities heap size (4KB)	(UTIL_HEAP_SZ) = AUTOMATIC	AUTOMATIC	AUTOMATIC
Self tuning memory	(SELF_TUNING_MEM) = ON	ON	ON
Automatic runstats	(AUTO_RUNSTATS) = ON	ON	ON
Sort heap thres for shared sorts (4KB)	(SHEAPTHRES_SHR) = 1294274	1294411	1294411
Log buffer size (4KB)	(LOGBUFSZ) = 2153	2153	2153
Default table organization	(DFT_TABLE_ORG) = ROW	ROW	COLUMN
Database memory threshold	(DB_MEM_THRESH) = 100	100	100



## ⑧シェル／コマンドの作成



# 物理設計の流れ

①表・索引定義の作成

②データ容量の見積もり

③インスタンスの構成と  
データベース分割

④表の分類と  
表スペースの構成

⑤表スペース容量の見積もり

⑥ディスク上への  
オブジェクトの配置

⑦ユーザーと権限の設計

⑧構成パラメータの設定

□ データベース構築に必要なシェル／コマンド

⑨シェル／コマンドの作成



# データベース構築に必要なシェル／コマンド

## □ 以下のような操作を行うシェルやコマンドを用意する

- 論理ボリューム、ファイルシステムの作成、権限の変更
  - OS上の操作でDB2が使用する資源を準備する
- データベース作成
  - データベースのホームディレクトリー
  - レジストリー変数、構成パラメータの設定
- バッファープール、表スペース作成
  - prefetchsize, extentsizeを正しく設定(extentsizeは後から変更できないので特に注意)
- 表・索引作成(DDL)
- その他

## □ 構築時に使用したシェルは、事前にテスト環境で十分に検証を行う

- また、再構築などで使用する場合に備えて管理しておく



# OS特有の設定

## □ AIX

- IOCPが必須(V9.7からデフォルトで使用)

## □ Linux

- Kernel parameterの設定
  - DB2が必要なカーネル・パラメータを自動設定する
- ASLR(Address Space Layout Randomization)
  - DB2 9.5, 9.7では、ASLRはOFFにすること
    - Various DB2 failures may occur on Linux with Address Space Layout Randomization (ASLR)
      - ◆ <http://www-01.ibm.com/support/docview.wss?uid=swg21365583>

## □ UNIX/Linux

- rootユーザーによる導入時には、ulimitは適切に設定される
- 非root導入の際には、以下を設定
  - data, fsize: unlimited
  - nofiles: 65536

## □ Windows

- Windows上の拡張セキュリティー
  - Windows上に以下の二つのユーザーグループが追加される
    - DB2ADMNS DB2製品の管理者グループ
      - ◆ すべての DB2 オブジェクトに対するフル・コントロールを持つ
      - ◆ Windows7以降で、DB2ADMNS グループのメンバーは、Windowsでツールを起動する際「管理者として実行 (Run as administrator)」を選択して、管理者特権を使用してツールを起動する必要がある
    - DB2USERS DB2の使用者グループ
      - ◆ インストール・ディレクトリーとインスタンス・ディレクトリーに配置されたすべての DB2 オブジェクトに対する読み取りおよび実行アクセスを持つ

