
Vectorwise
Developer Guide
Technical white paper

目次

はじめに

Vectorwise 2.5 開発者ガイドにようこそ

Vectorwiseは、データベース処理にはいままで使用されてこなかったCPUの機能を（ユーザーが意識することなく）活用してレポート処理やデータ解析を高速に処理し、ビッグ・データを使った効果的なアクションを可能にするリレーショナル・データベース管理システムです。このドキュメントは、Vectorwiseを最大限に活用するための実用的な情報、ガイドライン、ベスト・プラクティスを記載しています。Vectorwiseを評価・適用しようとするなら、このドキュメントをぜひ読んでください。

この開発者ガイドは、以下のトピックをカバーしています。

- システムのサイジング：Vectorwise用ハードウェア構成をどのように選択するか
- データベース構成：データベース設定に関する検討事項
- スキーマ設計：データベースのテーブルの設計とインデックス設定についてのベスト・プラクティス
- データのロード：もっとも効率の良いロード方法とVectorwise中のデータの管理
- 可用性：Vectorwiseについてダウン・タイムを最小限にする方法
- 管理とモニタリング：Vectorwiseを管理、モニタリング用のユーティリティ

このガイドは、Vectorwiseユーザー・ガイドを置き換えるものではなく、それを補足するものです。

システムのサイジング

Vectorwiseをインストールし稼働させるのに三つの選択肢があります。

1) 専用サーバー（と専用のストレッジ）を用意し、Vectorwiseを実OS上で動作させる。

2) 仮想マシンを作成し、Vectorwiseを仮想環境の中で実行する。

3) クラウドのリソースを利用し、クラウドでVectorwiseを実行する。詳しくは、
http://community.actian.com/w/index.php/Ingres_VectorWise_Amazon_EC2_Images.

このセクションでは、選択肢1）専用サーバー（と専用のストレッジ）を用意、に焦点をあわせます。これは、最高の結果を得られます。以下の議論は、選択肢2）と選択肢3）についても大部分が関連があります。仮想環境は、特にストレッジのスループットに関連する部分について、専用ハードウェア環境ほどは理想的ではありません。クラウド環境でVectorwiseを使用する場合、物理的なストレッジのスループットについてコントロールできません。もし、データのほとんど、もしくは全てが仮想環境中やクラウド中のメモリにフィットするならば、これについて懸念する必要はないかもしれません。

Vectorwiseが達成可能な最大のデータ処理スループットは、問合せに使用されるデータ型、データの値、計算の型や問合せの複雑さなど、さまざまな要因に左右されます。Vectorwiseでは、データ解析処理においてCPUコアあたり最高で1.5GB/秒という信じられないくらいのスループットを実現できます。

コア数の決定

CPUコア数は以下の二つの要因の組み合わせで決まります。

1) 満足のできる問合せの応答時間を得るために並列度をいくつにするか

2) 同時にいくつの操作を実行したいか（最大の性能のままで）

理論的には、システムのどこにもボトルネックがないのであれば、並列度を、係数Xに設定すれば、問合せの実行時間は、おおよそ係数Xに比例して減ります。現実には、完全に比例するスケーラビリティになることはありません。

並列度は、変更可能な設定なのでなんどか試すことができます。設定すべき並列度（DOP）を見積る最初のステップは、以下の計算式で求めます。

DOP = 検索されるカラムデータ量 GB単位（非圧縮）/ 目標応答時間 / 1.5

注意１：この計算式は、Vectorwiseの最大データ処理率をコアあたり毎秒1.5GBと仮定しています。この値は、実際の問合せにおいては大きすぎることや小さすぎることがあります。必要に応じて計算式中の1.5を調整してください。

注意２：問合せにおいて実際に使用されるカラムのデータ量のみを使用してください。Vectorwiseは、カラム指向の格納方法を使用しているので、問合せで使用されないカラムは、処理されません。

例えば、使用するシステムの最大の問合せが10億件の単一の非正規化されたテーブルに対するものであり、その問合せは、以下のような製品コード毎の売上の合計を求めるものだとします。

- 製品コードの平均の長さは、20バイト
- 売上の平均の長さは、10バイト
- 目標とする応答時間は、5秒

設定すべき並列度（DOP）は

DOP = (20+10) / 1000 (GB) * 1,000,000,000 / 5 / 1.5 = 4

DOPに4を設定すると、すべての問合せにおいて、問合せの間、4 CPUコアを使用します。最大の性能のまま、同時に実行できる問合せ数は、以下の計算式で求められます。

同時実行問合せ数 = CPUコア数 / DOP

DOPは、`max_parallelism_level`構成パラメータで設定します（詳細は、Vectorwiseユーザー・ガイドを参照してください）。

Vectorwiseの並列実行は、システムの過負荷を防止するよう適応しています。Vectorwiseデータベースが他の処理でビジーになると自動的に並列度を下げます。問合せの性能を最大限にしたいと考え、また、問合せの応答時間がおおきく変化することを許容するなら、高めの並列度を設定することができます。一貫した応答時間が必要なら、低めの並列度を設定します。並列問合せを行わないようにするには、`max_parallelism_level`パラメータを１に設定して下さい。

使用するシステムの理想的な CPU コア数を求めるためには、目標とする応答時間（並列度を設定することで達成）と同時に最大限の性能で実行したい問合せの数とのバランスをとる必要があります。多くのアプリケーション、とりわけ、アドホックな問合せを行うなど、ユーザ入力が必要なアプリケーションでは、問合せが連続的には実行されません。ユーザは、あるデータを問合せし、その結果を吟味して、さらに追加の問合せを行います。その結果、ある時点で同時に実行できる問合せ数より多くの同時実行ユーザ数をサポートすることができます。

ただし、ユーザは、一人で同時に何個も問合せを開始することがあり、これらは、同時実行問合せとして、それぞれを追加でカウントする必要があります。

複数のデータベースを同時に実行させることを計画する場合は、それぞれのデータベースが CPU 資源を使用することを考慮して下さい。現時点では、データベース間の資源の割り当てについて制御で

きず、OS が優先度を決定します。

ハイパースレッド

最新のCPUの多くが、ハイパースレッドをサポートしていて、物理的なCPUコアがただ１つでもOSには、複数のCPUコアが存在するように見せていることがあります。ハイパースレッドで、単一のコア上で同時に複数のタスクを実行することが可能になり、CPUコアは複数のタスクの間でマルチタスクで処理をおこないます。

Vectorwise は、CPU コアの資源を最大限に活用していて、一般的にハイパースレッドを利用しても利点がありません。事実、ハイパースレッドに伴うオーバーヘッドのため、Vectorwise の性能がダウンすることがあり、こういった場合は、BIOS でハイパースレッドを無効にして下さい。

メモリサイズの決定

Vectorwiseへの投資から最大限の価値を得るには、Vectorwiseの運用において、適切にメモリをサイジングすることは非常に重要です。Vectorwiseに割り当てられたメモリが不十分であれば、いろいろな操作に関してディスクへの書き出しを有効に設定する必要があり、ディスクへの書き出しが発生した問合せは、そうでない場合に比較して非常に性能が悪くなります。

使用するサーバーには、オペレーティング・システム（OS）、Vectorwise、それにサーバーで動作している他のアプリケーションをサポートする十分なメモリを搭載してください。OSは、メモリ不足の状況になるとメモリの内容をディスクにスワップすることで対処しますが、これは、システムのスローダウンやひどいパフォーマンス、予測不能なパフォーマンスを引き起こすため、絶対に避けなければなりません。

Vectorwiseは、２つのメモリプールを使用します。

- 実行メモリ：実行メモリは、同時に実行される最大のテーブルに対するハッシュ結合と集計とソートをサポートするのに十分大きい必要があります。
- カラム・バッファー・マネージャ（CBM）メモリ： メイン・メモリの一部で、圧縮された格納データのためのメモリです。

これらのメモリ・プールは、データベースごとに割り当てられ、データベース間で共有することはありません。複数のデータベースを同時に動作させようとするなら、同時に動作するデータベース間で利用可能な資源（この場合はメモリ）を分割しなければなりません。それぞれの使用するメモリの合計が、システムのメモリ・サイズを超えないようにして下さい。複数のデータベースの代わりに、複数のスキーマを使用することを検討して下さい。そうすることで、複数のアプリケーション間でメモリ資源を有効活用できます。

どれだけのメモリが必要かを特定するには、以下に示すように、各メモリ・プールにどれだけのメモリが必要かを特定します。もし複数のデータベースを動作させるなら各データベースごとに特定して下さい。それから、2 GB をOSのために加え、同一サーバーで実行させる他のアプリケーションのためのメモリを加えます。

システムのサイジングを簡単にし、予測可能なベストのVectorwiseの性能を達成するには、Vectorwiseを専用のサーバーで動作させるべきです。

現実的には、Vectorwise を非常に基本的なテスト以上で動作させるには、すくなくとも 8GB のメモリで構成する必要があります。

実行メモリのサイジング

問合せで必要となる実行メモリの量を決定する簡単な計算式はありません。使用する最大のテーブルに対する問合せ演算をサポートできるように十分なサイズの実行メモリが必要です。また、同時に実行される問合せについても考慮する必要があります。テーブル結合、集計、ソートは、暗黙か（つまり、問合せにキーワード`DISTINCT`を使った場合など）または明示的か関係なく、大きなデータ・セットに対して実行されると非常に多くのメモリを使用します。

大きなハッシュ結合については、その結合の一番小さいほうがメモリに収まる必要があります。たとえば、その結合の一番小さいカラムが 1 行あたり平均30バイト検索し、問合せ全体で1億行を検索すると、ハッシュ結合のためにこの問合せでは、以下のメモリを使用します。

30 * 100,000,000 = 3,000,000,000 bytes = 3 GB

フィルター条件を適用した後、結合のどちらの側が小さいのかをVectorwiseが正しく認識するためには、テーブル統計が信頼できるものであることが重要です。統計データの収集とメンテナンスについて、詳しくは、Vectorwise ユーザー・ガイドを参照してください。

集計については、結果（または中間）セット全体がメモリに収まる必要があります。たとえば、10億行のテーブルからフィルター条件なしで、1行あたり50バイトを検索し、20行ごとに集計を行うとすると、この問合せでは以下のメモリを使用します。

50 * 1,000,000,000/20 = 2,500,000,000 bytes = 2.5 GB

応答時間をなるべく短くするために、Vectorwiseは実行メモリ中では圧縮を行いません。

同時に実行されている問合せがあれば、それぞれの問合せでメモリを消費します。実行メモリに必要とされる合計の量を決める際には、同時に実行される問合せと負荷について考慮してください。

問合せにおいて、Vectorwiseはできるだけ多くのメモリを使用します。必要とするメモリが利用可能

なメモリより多い場合は、ディスクへの書き出し操作を有効にしていない場合、メモリ不足のエラーを返します。ハッシュ結合と集計についてのディスクへの書き出しは、デフォルトでは無効になっています。vectorwise.confパラメータ・ファイル中で有効に設定変更できます。ソートについてのディスクへの書き出しはいつも有効で、無効にすることはできません。構成を変更しない限り、デフォルトではVectorwiseは、データベースごとに物理メモリの50%を実行メモリに割り当てます。

カラム・バッファ・マネージャ（CMB）メモリのサイジング

必要とするCMBメモリのサイズを決定するにあたり、最初の質問は、データセットをメモリ内に収められるようにする必要があるのか、です。入出力の性能は、問合せ性能を最大にするにあたって非常に重要です。そして、データをメモリに格納することで、ストレッジがボトルネックになる可能性を避けることができます（高い可用性とリカバリのためにディスクにも格納します）。

デフォルトでは、VectorwiseはCBMメモリ中ではデータを圧縮した状態で保持します。メモリ中で、データを圧縮しない状態で保持すれば最大のパフォーマンスが得られます（以下のデータベースの構成のセクションを参照）が、必要となるメモリ・サイズが増大します。

データセットが、計画中のCBMのメモリ・サイズより大幅に大きいならば、CBMのメモリ・サイズとしては、頻繁にアクセスするデータのサイズ（理想的にはCBMメモリに収まる必要があります）を考慮してください。Vectorwiseは、スマートなバッファ管理を行い、データをディスクから読み出すことが必要になった時のディスク・アクセスのスループットを最大にします。

構成を変更しない限り、デフォルトではVectorwiseは、データベースごとに物理メモリの25%をCBMメモリに割り当てます。

OSについての検討

Vectorwiseは、64ビットのLinuxと64ビットのWindowsで動作します。他の関係データベースと比較して、同じハードウェアを使っても、どちらのOSでもデータ分析やレポートといた処理で非常によい問合せ性能の向上が見られます。

単一の操作において大量のメモリ（10GB 以上）を使用するなら、同じシステムでも Windows より Linux のほうで性能がよくなることがよくあります。これは、効率のよいメモリ管理機能のためです。

ディスク装置の決定

Vectorwiseデータベースの主要なストレッジは、ハードディスクです。ストレッジの容量についての要件を別にして、ストレッジの性能がパフォーマンスの要件を満たしていることが非常に重要です。注意すべき点は、ストレッジを構成する各コンポーネントのいずれもがボトルネックになる可能性が

あることです。個々のドライブ、ディスク・コントローラ、HBA、スイッチ、などです。

入出力性能の要件

問合せと実際のデータによりますが、Vectorwiseは、CPUコア当たり1.5GB/秒のデータを処理できます。このデータ処理率を達成するためには、CPUコアを十分働かせるだけの十分なデータを供給する必要があります。データが圧縮されていれば（デフォルトでは圧縮されています）、解凍のコストは、CPUコアの処理能力の約25%です。

必要となる最大帯域は以下の計算式を使用してください。

コアあたりの最大帯域 = コア処理スピード / 圧縮率 * 0.75

この数にシステムのコア数を掛けた値で、CPUコア全体で最大帯域幅を駆動するために必要なディスク全体のスループットを計算することができます。

注意：このような非常に高い入出力性能の要件は、単一のテーブルに対する集計がついたテーブルスキャンといった比較的単純な問合せで見られます。より複雑な問合せ（つまり、小さなメモリ常駐のテーブル間の多数の結合など）では、入出力性能の要件は、大幅に小さくなることがあります。

注意：データの圧縮率は、テーブル中のデータ型と実際のデータに依存しています。共通しているのは3倍から5倍の圧縮率ですが、より高い圧縮率とより低い圧縮率の両方とも見られます。

全てのコアが完全に占有され、同時に実行されているすべての問合せが比較的単純な場合だけに、最大のデータ処理スループットが達成されます。システムが最大パフォーマンスを達成できない構成で、比較的単純な問合せの実行で完全にビジーとなった場合、そのシステムは、入出力の性能限界に達しています。

回転するディスク

過去数年の回転するディスクについての開発の主眼は、単一ディスクの容量の増大でした。データ転送率は、ほんのわずかしか向上していません。事実、回転する大きなディスクでは、データをディスクの外側に格納するか、内側に格納するかでデータ転送率に大きな差があります。このため、一貫した良いパフォーマンスを得るためには、大きなディスクではなく、小さいディスクを選択する必要があります。つまり、146 GBディスクを選ぶべきで、500 GBやそれより大きなディスクは選ぶべきではありません。

ディスクの回転数が15k RPM と速いものは、10k RPMや7.2k RPMの遅いディスクと比べて、高い転送率があります。理想的な状況では、単一の15k RPMのディスクは、最大120MB/sのデータ転送率があります。

SSD

単一のSSDドライブは、サイズによらず最大250 MB/sの転送率です。SSDの価格は大幅にさがってきていますが、ストレッジ・ユニット当たりでは、回転ディスクより依然としてSSDのほうが高価で、また、一般的に容量はより小さくなっています。最大のパフォーマンス構成のために（もっとも頻繁にアクセスするデータがメモリ上に常駐しない限り）、SSDはVectorwiseのデータ処理のための内蔵ストレージとして使用するドライブとしては、最良のものです。

特別のタイプのSSDとしては、たとえばFusion-IO から提供されているPCIカードがあります。これらのカードのスループットはPCIチャネルによって制限されますが、高いものではカードごとに1.5GBです。非常に高い入出力スループットが必要で費用が問題のない範囲ならば、このようなカードを検討して下さい。

RAID構成

使用しているデータ全体が、メモリ上に収まらない限り、Vectorwiseのデータ処理能力を十分発揮するためには、並列に動作する複数のドライブが必要です。これを行うには、RAID（Redundant Array of Inexpensive/Independent Disks）ボリュームを介して複数のドライブにまたがってストライピングを使用することです。512KB、または512KBの整数倍の大きなストライピングのサイズを使用して、ディスクの入出力を最適化します。

ハードウェア・コントローラが限られたスループットしかない場合は、Vectorwiseで必要とする入出力スループット・レベル以下に制限されることがあります。そのため、複数の非RAIDコントローラを使って、ソフトウェア・ベースのRAIDを検討してください。RAID構成に多数の高速ドライブ（たとえば8台以上のSSD）が含まれるなら、そうすることで単一のハードウェアRAIDコントローラのボトルネックを防ぐことができるかもしれません。

ドライブの故障に備えてRAID構成を選ぶことも（高い可用性をもつ他のアプローチを採用していないかぎり）必要です。RAID5またはRAID6設定が典型的なRAID構成で、これは、ストレッジのオーバーヘッド、パフォーマンス、可用性の間で良いトレードオフを提供します。

ストレッジの要件

最小のディスク数は、必要とされるストレッジの全容量と入出力のスループットで決まります。また、Vectorwiseにデータをロードするためディスク上にステージングのためのストレッジ領域が必要になります。さらに、将来のデータベースの成長と、ファイル・システムの拡張（データベースの領域が追加で必要になったとき）時の作業についても考慮に入れる必要があります。

Vectorwise はデータを複数の位置に格納することができます。これにより、データ領域の拡張が行え、

データの種類にあわせて異なる種類のストレッジにデータを格納できます。たとえば、一時領域は、通常のデータより速いストレッジに割り当てることができます。

ファイル・システム

使用するファイル・システムには特に制限はありません。Linuxでは、ext2、ext3、ext4、xfsが使用できます。WindowsではNTFSを選択して下さい。以下に注意して下さい。

- ファイルシステムによっては、最大のファイルサイズに制限があります。たとえば、ext3のファイル・サイズの制限は最大2TBで、非常に大きなテーブルでは、単一のカラムのための単一のファイルが非常に大きくなることがあります。
- Linuxでファイル・サイズについて最大限の柔軟性が必要ならxfsファイル・システムを使用してください。

ガイドライン

すべてのCPUコアがフルに動作し最大のパフォーマンスを得るための完全にバランスのとれたシステムを構成するには、以下を検討してください。

- CPUコアあたり4台以上の15k RPMディスク
- CPUコアあたり2台以上のSSD（PCI SSDカードを使用する場合を除く）

データベースの構成

Vectorwiseの構成は、データベースごとに行います。このセクションでは、通常よく使用・変更されるいくつかのデータベース設定について議論します。

Vectorwise ソフトウェアのインストールプロセスで、初期状態の vectorwise.conf 構成ファイルが生成されます。そのファイルには、使用しているシステムのサイズにあわせたデフォルトの設定の構成パラメータがいくつか含まれています。構成ファイルに明示的には含まれていない構成パラメータは、デフォルト値が仮定されます。それらのデフォルト値を変更すべきかどうかについては、詳しくは、Vectorwise ユーザー・ガイドを参照してください。

デフォルトでは、vectorwise.conf設定は、作成されるすべてのデータベースに適用されます。（これを変更することもできます。詳しくは、Vectorwise ユーザー・ガイドを参照してください。）Vectorwiseは、1データベースごとに1プロセスを占有します。このため、各データベースで必要とされる資源の合計を考慮する必要があります。たとえば、Vectorwiseはデフォルトで50%の物理メモリを実行メモリとして割り当てます。これは、データベースごとです。もし、3つのデータベースを頻繁に使用するなら、この仮定では150%のメモリとなって、これは適切ではありません。このようなケースでは、各データベースの実行メモリのサイズを十分小さく設定し、すべてを同時に使用しても、適切なサイズのメモリを使用するようにします。複数のデータベースを使用するのではなく、一つのデータベースでいくつかのスキーマを使用することですべてのユーザーが消費する資源を最適化することを検討して下さい。

データのアクセス

データがアクセスされ、それが、CBMメモリにない場合、そのデータは、解凍されずにデータベースから圧縮されたままメモリ（RAM）に読み込まれ、CPUキャッシュがあたかも処理メモリであるかのようにして処理されます。Vectorwiseは、スマートなバッファ管理を行い、スループットを最大化します。

大きなブロック入出力

Vectorwiseは、ディスクからデータを読むために常に大きなブロック入出力（large block IO）を行います。データが一緒に格納されていて一回の入出力で行えるなら、大きなブロック入出力は、非常に効率的です。vectorwise.conf構成ファイル中のblock_sizeパラメータは、このブロック（ページ）のサイズを指定します。デフォルトの値は、512 KBです。

典型的なデータウェアハウス問合せは、データのとても大きなボリュームをスキャンします。次のデ

ータブロックをアクセスしなければならないなら、もっとも効率のよいデータの取り出し方法は、ディスク上でそれらのブロックを一緒に格納することです。これは特に、回転するディスクにとって重要です。なぜなら、ディスクヘッドの移動のコストは高いので、シーケンシャル入出力は、ランダム入出力の少なくとも 2 倍のスループットがあります。Vectorwise は group_size パラメートを使って、一緒に格納されるブロック（ページ）の数をコントロールします。デフォルト値は 8 です。

block_size と group_size パラメータは、データベースが作成された後では変更できません。データベース作成後にこれらのパラメータを変更するなら、そのデータベースをアンロード/削除/再作成/再ロードする必要があります。

データの割り当て

デフォルトでは、Vectorwiseは、ほとんどのケースでカラム毎のデータの格納を行うので 、データは、カラムごとに割り当てられます。カラム毎のデータの格納の場合、ディスク上でそのテーブルが占める最低限の領域は、データの量に関係なく、以下の値になります。

```
number of columns * block_size * group_size
```

テーブルのカラムの数がとても多い（テーブル数が多いかテーブル当たりのカラム数が多い）場合は、これを考慮に入れる必要があります。最小の割り当てサイズを減らすには、`group_size`と`block_size`パラメータのどちらかまたは両方の値を小さくしますが、大きなテーブルのスキャンのパフォーマンスが犠牲になります。

Vectorwise は、1 行のデータ全体を単一のデータブロックに格納するメカニズムをサポートしています。データブロック内ではデータ圧縮を最適化するためにカラム毎に格納されます。テーブルに非常にカラムが多くあり、比較的データ行が少ない場合（ストレッジへの割り当てを削減するため）や、テーブルに比較的少数のカラムがあり、それらのカラムの大部分をいつでも問合せする場合に、この行毎の格納方式を検討してください。行毎の格納方式は、create table 文の最後に WITH STRUCTURE = VECTORWISE_ROW を指定することで使用することができます。

Vectorwiseはいつでも入出力を、ブロック単位で行います。ブロックにフルにデータがない（例えば、圧縮したあとブロックをフルにするだけのデータがない）場合や、問合せの条件を満たさない不要なデータを読み出す（例えば、行毎の格納方式が使用されているのに、問合せで必要とされているカラムは少数）場合には、不必要な入出力が行われます。入出力のパフォーマンスが最大限となるようなデータ割り当てを選択してください。

CBMメモリ

カラム・バッファ・マネージャ（CBM）メモリは、サーバーのメモリ（RAM）の一部で、ディスクから読み込んだデータを格納するために使用されます。大きなCBMメモリは、問合せ性能を向上させ

ます。問合せの条件を満たすためにCBMメモリに存在しないデータだけをディスクから読み出せば良いからです。Vectorwiseは、すべてのデータがメモリに収まらない場合のスループットを最大化するために、スマートなバッファ管理を行います。

vectorwise.conf の初期化パラメータ bufferpool_size を使って、CBM メモリのサイズを指定することができます。デフォルトでは、データベースごとにサーバーのメモリの 25%が CBM メモリとして割り当てられます。

CBM メモリ中のデータは、ディスク上のデータのミラーです。たとえば、データがディスク上で圧縮されているなら、CBM メモリ上でも圧縮されたままで、メモリバッファの使用を最適にしています。データは、処理されるために CBM メモリから取り出されるときだけ解凍されます。解凍のためのコストを避けたいならば、テーブルを作成する前に、set trace point qe82 コマンドを実行してデータを圧縮しないで格納することができます。圧縮しないテーブルは、より多くのディスク領域が必要になることと、ディスクからのデータの読み込みに時間がかかるようになるので、小さいテーブル対してのみ使用するようにしてください。

ストレッジの割当

Vectorwiseは、カラムグループごとにストレッジを割り当てます。カラムグループとは、一緒に格納されるカラムのことで、複数カラムのインデックスの場合とVECTORWISE_ROW格納構造の場合以外は、通常は単一のカラムです。一時領域は、別個の一時ファイルに格納されます。
Vectorwiseファイルは、ストレッジ領域が不要になった時点で削除されます。一時用として使われるファイルは、処理が終了した時点で削除され、データファイルはテーブルやカラムが削除されたときに削除されます。
詳しい情報は、Vectorwiseユーザーガイドを参照してください。

スキーマ設計

Vectorwise を使用するにあたって特別なスキーマを設計する必要ありません。Vectorwiseは、問合せ性能をあげるためのマテリアライズド・ビュー（実体化ビュー）やプロジェクション（射影）を必要としないため、これらを設計する必要もありません。スター・スキーマやスノー・フレーク・スキーマだけでなく、以下のように他の関係データベースではほとんど使用できない極端なものも使用することができます。

- すべてのカラムを非正規化した単一の幅広いテーブル
- 現在使用しているスキーマをそのまま使用。データをロードしてそのまま問合せを実行。

Vectorwiseのベクトル処理によるキャッシュ内での処理は、計算処理とテーブルの結合に恩恵をもたらします。そのほか、Vectorwiseが透過的に提供しているデータアクセスの最適化には以下があります。

- カラム毎の格納：問合せでは、関連のあるテーブルのカラムだけにアクセスします。
- 最小/最大インデックス：問合せのフィルターに明示されている条件と、明示されずにテーブル結合や副問合せで指定されている条件の両方で、Vectorwiseは候補のデータ・ブロックを特定します。追加されたデータのデータベース中の位置とデータの値の間に、相関関係があるならば（たとえば時間毎のデータは一つの例です）、最小/最大インデックスをつかって非常に効率よく大量のデータブロックをフィルタリングできることがあります。極端なケースでは、最小/最大インデックスは他のデータベースで提供されているパーティショニングによる利点と同様のパフォーマンスを提供します。
- 圧縮：デフォルトでデータは圧縮されて格納されます。このためディスクからのデータの読み込み時間が少なくなります。

カラム毎の格納方式と最小/最大インデックスの組み合わせによって非正規化した単一の大きなファイルに対する問合せは、場合によって非常に効率的です。効率的な圧縮により非正規化のためのストレッジのオーバー・ヘッドはデータによっては最小化できます。

インデックス

Vectorwiseは、よい性能を達成するためにインデックスに依存することはありません。事実、ほとんどのレポートやデータ分析作業において、明示的にテーブルにインデックスを作成しても役にたちません。このような作業ではインデックスを作成すべきではありません。

Vectorwiseでは、テーブルにインデックスを作成できます。インデックスが作成されたテーブルでは、

インデックスによって物理的に順序付けされて格納されます。これは、索引構成表やクラスター・インデックスと呼ばれています。Vectorwise では、1テーブルにつき1つのインデックスだけを作成できます。内部的な最小/最大インデックスは、いつでもカラム毎に作成され、テーブル上に明示的に作成されるインデックスのある・なしには関係がありません。そのテーブルの大部分のアクセスが、インデックスを付けた（単一または複数の）カラム上のフィルターを使用するか、インデックスを付けた（単一または複数の）カラムで結合する場合には、性能向上を得られるかもしれません。

インデックスは、問合せ実行プランにも影響を与えます。結合される両方のテーブルの結合キー上にインデックスがある場合、ハッシュ結合ではなくソート・マージ結合を選択します。ソート・マージ結合は、ハッシュ結合よりメモリ使用量が少なく、より効率的です。なぜなら、データがすでにソートされているからです。

テーブルに対してインデックスを作成すると、更新処理について考慮が必要になります。また、データのロードやデータの更新でメモリ使用量が増加します。詳しくは、以下の『データのロード』セクションを参照してください。

制約

データ分析に使用するデータベースは、典型的にはコントロールされたプロセスを通してデータがロードされます。そのプロセスにはデータの検証も含まれるでしょう。その結果、データの整合性の観点からみると、主キー、ユニーク・キー、外部キーといったデータベースレベルの制約を定義する必要はないはずです。一方、制約は、問合せの最適化の際に追加の情報として使用することができます。これにより、より良い問合せ実行プランを作成できることがあります。データの制約が、データのロード戦略に影響を与えないのであれば、制約を定義するほうが定義しないより良い結果となります。

Vectorwise における制約は、常に強制され、主キー、ユニーク・キーについてもインデックスは必要ありません。制約の検査は、ユニークの検査と外部キーの制約の検査についてカラム（複数）をスキャンすることで実現しています。このスキャンは、インデックスと最小/最大インデックスをつかって効率化できることがあります。しかし、大きなテーブルに対して頻繁に少量の変更を加え、そのテーブルに制約を強制すると、制約を検査するためのスキャンによって非常に大きな性能上のインパクトがあることに注意して下さい。

データのロード

レポートやデータ分析に使用するデータベースは、典型的にはコントロールされたプロセスを通してデータがロードされます。従来、大量の増分データ・セットは、１日ごと、１週間ごと、場合によっては１月ごとに追加され、その増分のデータ・セットは、挿入されるだけで、更新や削除されることがありませんでした。今日では、データのロードはより頻繁に行われるようになり、場合によっては、データがコンスタントに投入され続け、データが来ない時間帯がどこにもないという状況もあります。データの投入頻度が増え続けるだけでなく、ロードするデータのボリュームも増え続けています。

Vectorwiseは、大量の増分データ（大部分が挿入であるもの）について、効率よく処理します。投入され続けるデータは、２，３秒ごと（問合せに対する要件による）に集められ増分バッチとして処理されることがよくあり、増分バッチの分量が比較的多ければ効率よく処理されます。複数のデータ・ブロックが各回の増分データ・ロードでいっぱいになるのが理想的です。

初期データのロード

Vectorwise に初期データを効率的にロードするには、必ず一括ロードのアプローチをとる必要があります。このセクションは、初期データのロードの方法のいくつかを議論します。

vwload

Vectorwiseに一括ロードでデータをロードするのに最も高速なアプローチは、vwloadユーティリティを使用する方法です。vwloadは、SQL層をバイパスして、ダイレクトにデータ・ブロックをデータベースに追加します。セキュリティの観点から、このユーティリティは、Vectorwiseが動作しているデータベース・サーバーでのみ実行が可能です。そのため、vwloadで処理するためにはデータ・ファイルがデータベース・サーバーで直接アクセスできることが必要です。また、Vectorwiseにデータをロードするには、データ・ファイルの文字のエンコードが正しくなければなりません。データ・ファイルのエンコードがUTF8でなければ、vwloadのパラメータとしてエンコードを指定できます。（現状では、Shift_JISの扱いには問題があるので、UTF-8に変換してロードしてください。）

以下にvwloadの使用例を示します。

test というデータベースに、region というテーブルを作成します。

```
CREATE TABLE region
( r_regionkey INTEGER not null,
r_name CHAR(25) not null,
r_comment VARCHAR(152) not null
);
```

以下のデータをロードするためのデータ・ファイルregion.tblを用意します。

```
0|AFRICA|Continent of the elephants
1|AMERICA|Both North and South America
2|ASIA|Where tigers live
3|EUROPE|Many languages are spoken here
4|MIDDLE EAST|Sunny and very warm
5|AUSTRALIA|For goodness sake, please don’t leave us out
```

以下のコマンドでデータをロードします。

```
vwload -t region test region.tbl
```

vwloadユーティリティには、エラー処理、日付のフォーマット、引用符、エスケープ文字などをオプションとして指定することができます。詳細についてはVectorwiseユーザー・ガイドを参照して下さい。現時点では、time と timestamp フィールドは、ANSIデータ表現でなければなりません。

- hh24:mi:ss が time フィールド
- yyyy-mm-dd hh24:mi:ss.ffffff が timestamp フィールド

copy文

copy文は、データ・ファイルからVectorwiseにデータを一括ロードする方法として柔軟で効率的な方法です。 copy 文は、リモートまたは、データベース・サーバー上からsql端末を使用して実行できます。

copyコマンドの構文は、その他の一括ローダーの構文とは少し異なっています。以下は、copy文を使用する例で、vwloadのセクションで説明したテーブル定義、データ・ファイルと同じものを使用します。

sql端末セッションから以下のcopy文を使用します。

```
copy table region (
      r_regionkey = 'c0|',
      r_name = 'c0|',
      r_comment = 'c0nl'
)
from 'region.tbl' ¥g
```

上の例で、‘c0|’はフリー・フォーマット（固定長ではなく）の文字列で、縦棒の区切りが続き、‘c0¥n’は、フリー・フォーマットの文字列で、行の最後に改行がつくことを表しています。

copy文は、良く使われるコンマやセミコロンで区切られたファイルを（二重引用符による囲いとバックスラッシュによるエスケープ文字も含めて）サポートします。例題については、Vectorwiseユーザー・ガイドを参照してください。またcopy文の詳細の構文とオプションについては、Ingres SQL Reference Guideを参照してください。

現時点で、date、time、timestampフィールドについては、ANSI規格を使用したフォーマットのみをサポートしています。

- yyyy-mm-dd が date フィールド（II_DATE_FORMATを指定していないとき）
- hh24:mi:ss が time フィールド
- yyyy-mm-dd hh24:mi:ss.ffffff が timestamp フィールド

データのロードが何らかの原因で失敗したとき、たとえば、レコードが正しくない場合、デフォルトの動作は、データのロードを停止し、文全体をロールバックすることです。データのロード時の問題点を調べるには、copy 文にWITH句を指定して、オプションを追加する必要があります。ON_ERROR = CONTINUE と LOG = 'filename' を使用することで、問題が発生してもデータのロードを続け、問題のあるレコードを'filename'という名前のファイルにロギングします。ERROR_COUNT を含めると、エラーが何件か発生した時点で処理を停止することができます。

以下に例を示します。

```
copy table region (
    r_regionkey = 'c0|',
    r_name = 'c0|',
    r_comment = 'c0nl'
)
from 'region.tbl'
with on_error = continue
, log = 'region_bad_records.log' ¥g
```

INSERT/SELECT

Vectorwise は、従来のIngresのテーブルとVectorwiseテーブルの間（両方向）でINSERT/SELECTをサポートします。従来のIngresテーブルにすでにデータがあるなら、INSERT/SELECT (または CREATE TABLE AS SELECT) は、Vectorwiseデータベースにデータを投入する良い方法となります。

バッチ挿入

Vectorwise はODBC、JDBC、.NETでのバッチ文をサポートします。バッチインタフェースを通して、INSERTは内部的に効率的なデータ追加に変換されます。データ・ストレッジの観点からは、大きなトランザクション、とりわけ同一テーブルへの多数のINSERTからなるトランザクションを適用することは、バッチインタフェースを通した場合、vwloadやcopy文を使用した一括ロードにつづいてファイルシステム上の中間ステージングのステップを通した場合と同じくらい効率的です。

サード・パーティのツール

データをアンロードし、vwloadやcopy文を使用してデータをVectorwiseテーブルにデータをロードするためには、プログラムやスクリプトを書くことが必要ですが、そうしない方法としては、Vectorwiseの一括ロードをサポートするサード・パーティのツールを利用できます。たとえばIngres High

Volume Replicator (HVR)とPentaho は一括ロードをサポートしています。また、サード・パーティんのツールがVectorwiseへの効率的なデータのロードを行うODBC、JDBC、.NETのバッチ処理をサポートしているか確認してください。

増分データのロード

Vectorwiseへのもっとも効果的な増分（インクリメンタル）データのロード方法は、vwload、copy文またはODBC、JDBC、.NETのバッチインタフェースといった一括データ操作を通して行うことです。通常のinsert/update/delete を使って単一行のDML文を適用することができますが、これらの"バッチ"処理は、"一括"処理に比べると効率はあまり良くありません。バッチ処理と一括処理によるデータのロードについては、Vectorwiseユーザー・ガイドの第１１章 データを更新する方法 を参照してください。

データを効率的に追加するには以下の方法から選択してください。

- `vwload`ユーティリティまたは`copy`文を使ってデータ・ファイルからロードし、テーブルにデータを追加します。対象のテーブルがインデックスつきでデータがすでに存在する場合を除き、この方法は効率的にデータを追加します。
- バッチ・インタフェースを使用してODBC、JDBC、.NETのアプリケーションから直接データをロードします。対象のテーブルがインデックスつきでデータがすでに存在する場合を除き、この方法は効率的にデータを追加します。
- `insert into <table> as select <columns> from <other table>`を使用してデータをロードします。対象のテーブルがインデックスつきでデータがすでに存在する場合を除き、この方法は効率的にデータを追加します。
- COMBINE文を使用してテーブルにデータを追加します。

COMBINE文の使用とinsert as selectの使用時は、両方とも、ステージングテーブルを作る必要があります。insert as select を使用する時は、ステージングテーブルは、従来のIngresテーブルでもVectorwiseテーブルでも構いません。COMBINE文を使用する時は、使用するすべてのテーブルは、Vectorwiseテーブルでなければなりません。

COMBINE文

COMBINE文は、一つのテーブルに対して処理が行われ、以下の処理をおこないます。

- そのテーブルについて完了しキャッシュされているメモリ上のinsert/update/delete文のデータをディスクに書き込み、ディスク上のテーブルのレイアウトを最適化します。
- COMBINE文の引数の中で指定されたトランザクションを、一括操作として実行します。

COMBINE文の使用は、テーブル中の大きなパーセントのデータの更新と削除を行うには、もっとも

効率のよい方法です。

テーブル上の変更でまだ処理していない量と、そのテーブルにインデックスがついているかに依存しますが、COMBINE文は、長時間になることがあります。

COMBINE文の使用について、実際的な例も含め、詳しくは、Vectorwiseユーザー・ガイドを参照ください。

DMLとメモリの使用

データを追加しないDMLはすべてテーブルごとに最適化されたPositional Delta Tree（PDT）と呼ばれる構造でメモリ上に格納されます。これには、すでにデータが存在するインデックスつきのテーブルに対するCOMBINEによる実行以外のすべてのDMLによる更新処理も含まれます。

Vectorwiseは複数バージョン読み取り一貫性をサポートします。PDT中のコミットされたデータは、ディスクから読み込まれたデータと効果的にマージされて問合せ結果となります。また、PDT中のコミットされたデータは、コミットの完了前にトランザクションログに書き込まれ、リカバリ目的のためにディスク上で保持されます。

Vectorwiseのトランザクションは、最初の操作で、開始することに注意して下さい。その操作が読み込みであっても、そのトランザクションをとおして、読み取り一貫性が保持されることに注意して下さい。あるセッションが読み込みだけの問合わせを行い、別のセッションがDMLを実行している場合、最初の読み込みだけのセッションがDMLセッションの変更を見ることができるのは以下の時です。

› DMLセッション中で変更をコミットします。（トランザクションが完了します。）.

› 読み込みだけのセッションをコミットかロールバックを使って実行中のトランザクションを完了します。

読み込みだけのセッションについての読み取り一貫性を維持するために、別のセッションがDMLを実行しトランザクションを完了させると、多くのメモリを使用します。このため、大量のDMLを実行するようなシステムでは、周期的ににコミットまたはロールバックを実行するか、セッションを終了し再度開始して下さい。

.すべてのVectorwiseデータベースは、PDTを保存するためにサーバーのメモリの一部を予約します。予約されるメモリの量は、max_global_update_memory パラメータでコントロールされます。これは、PDTとして使用できる問合せ実行メモリの上限です。デフォルトは、25%です。

PDTによって使用されるメモリ量が指定された最大値を超えた時は、Vectorwiseは自動的に未処理のPDTトランザクションに含まれる全てのテーブルに対して自動的なプロパゲートを起動します。この

プロセスの間、トランザクションは実行され続けますが、更新情報を保持するため、`vectorwise.conf` で指定した最大値を超えてメモリが消費されることがあります。

Vectorwise への増分データのロードのベストプラクティスは以下のとおりです。

- 可能なかぎりデータは、追加で処理します。
- 大量のデータの削除または更新を行うときは、COMBINE文を使用します。ガイドラインとしては、テーブル中の10%以上のデータを変更するなら、COMBINE文を使用して下さい。
- 読み込みのみのセッションでも定期的にコミットかロールバックを実行し、(1) コミットされた更新データを参照し、(2) トランザクションの一貫性を保持するための必要だった古くなったメモリを開放します。
- 未処理のDMLに含まれるテーブルに対して、明示的なCOMBINE文の呼び出しを検討して下さい。これを行ってPDT中のメモリを解放するのは、システムの資源に余剰があり、この操作を行うことができる時です。

制約

Vectorwiseは、制約を指定したカラム（複数）に対してSELECT文を使用して、主キー、一意キー、外部キーの制約を実現しています。宣言された制約は、いつでも強制されます。その結果、制約の検査は、頻繁な少量の増分ロードに関して資源の消費に著しい影響を与えることがあります。スキーマを設計する際には、これを考慮にいれ、制約を宣言するか、データのロード処理中に整合性を検査するかを検討して下さい。

データのウィンドウの移動

データ・ウェアハウスやデータ・マートにおいてファクト・テーブルのデータのウィンドウを移動することがよくあります。たとえば、オンラインで保持するデータを最新の36カ月分にするなどです。Vectorwiseでデータのウィンドウを移動させながらデータをロードするには3つのアプローチがあります。

1) データが到着するごと、Vectorwise中のファクトテーブルに直接データを追加します。毎日データを追加するか、1日に何度かデータを追加します。このアプローチでは、ファクトテーブルにインデックスはついていないことを仮定しています。

   一か月に一度、その月についてファクトテーブルから削除したい行（の主キー）を選択し、ステージング・テーブルを作成します（create table … as select …）。そして、COMBINE文を使用して、元のテーブルからデータを削除します。

   ```
   CALL VECTORWISE (COMBINE 'basetable - staging')
   ```

   次にそのステージング・テーブルを削除します。

2) データが到着したらファクト・テーブルに直接追加するのではなく、ステージング・テーブルSTAGING1にデータを追加します。毎日データを追加するか、1日に何度かデータを追加します。このアプローチでは、STAGING1テーブルにインデックスはついていないことを仮定しています。ステージング・テーブルSTAGING1とベースのファクト・テーブルをunion（all）したビューを作成します。

   一か月に一度、その月についてファクト・テーブルから削除したい行（の主キー）を選択し、2番目のステージング・テーブルSTAGING2を作成します（create table … as select …）。そして、COMBINE文を使用して、ステージング・テーブルSTAGING1のデータを追加し、ステージング・テーブルSTAGING2のデータで元のテーブルからデータを削除します。

   ```
   CALL VECTORWISE(COMBINE 'basetable + staging1 - staging2')
   ```

   この操作で、ベース・テーブルにはインデックスがついていてもかまいません。

   ステージング・テーブルSTAGING2を削除します。次にステージング・テーブルSTAGING1を空にします。

   ```
   CALL VECTORWISE (COMBINE 'staging1 - staging1')
   ```

3) 複数のテーブルを使用し、すべてのテーブルを同じ構造にしたファクトテーブルにします（つまり、一か月に一つのテーブル - 36カ月のデータを36のテーブル）。

   すべてのテーブルの結果をunion allするビューを作成し、それをアプリケーションには単一のファクトテーブルとして見せます。データが到着するごと、最新の月のデータに追加します。テーブルにインデックスがついている場合は、データの追加にCOMBINEを検討して下さい。一か月に一度、一番古いテーブルを削除するか、空にします。

使用するアプローチが異なると、問合せの実行プランが異なったものになり、問合せの性能にも影響がでることに注意して下さい。

高可用性

どんなデータベースでも、さまざまな計画的または非計画的な機能停止によって利用できなることがあります。そのため高い可用性を追求する必要があります。

計画的な機能停止には、事前に計画されているシステムやソフトウェアのアップグレードなどがあり、この場合は、ユーザへの影響は最小限にすべきです。

このセクションでは、非計画的な機能停止によるダウンタイムを最小限にするための戦略について以下の議論をします。

- サーバーやストレッジの故障を含むハードウェアの問題
- ハードウェアやソフトウェアの問題によるデータ破損の問題
- 電源またはネットワークの故障によるシステムの機能停止
- データまたは操作の間違いによるデータの問題

高可用性の主なゴールは、機能停止を防ぐことです。二番目のゴールは、機能停止のあと、システムの復旧を確実に行うことです。三番目と最終的なゴールは、いかなる機能停止でも停止時間をなるべく最短にすることです。以下に議論する戦略は、防ぐことができない非計画的な機能停止による影響を最小にすることです。

防止計画と、復旧計画の両方についてテストし、それが機能することを確かめ、非計画的な機能停止が実際に発生するときに備えておく必要があります。

ハードウェア保護

ハードウェアの非計画的な機能停止（通常、コンポーネントの一つで発生）からシステムを保護するための複数の技術があります。例としては以下があります。

- 電源故障からサーバーを保護するためにバックアップ電源を使用します。
- 複数のネットワークポートを結合するか、ソフトウェアベースのソリューションを使って、ネットワーク接続の故障に対応します。
- RAID5やRAID6といったRAID構成を使って、ディスクの故障に対応します。
- 別個のスタンバイシステムを（理想的には、離れた箇所に）用意します。このドキュメントのスタンバイ構成を参照してください。

バックアップと復元

データベースのバックアップはある時点のデータベースに復元することを可能にします。バックアップを行うことで、最後のバックアップ以降に発生するデータの問題やデータの破損からデータを保護することができます。さらに、異なるハードウェア上にデータベースを復元することもでき、現在使用中のハードウェアが完全な回復不可能なシステムクラッシュを被った場合にも対応できます。データベースの復元中は、データベースは使用できず、どれくらいの時間がかかるかは、データのサイズと使用しているシステムの性能によります。

バックアップ戦略の設計は、機能停止のリスクの分析と、どれくらいの間、システムが利用できなくても大丈夫かを分析するところから始めます。さらに、バックアップ・ソリューションのコストも検討します。

Vectorwise は、フル・バックアップアップをサポートしますが、バックアップと同時にDMLやDDLを実行することはできません。Vectorwiseは、インクリメンタル（増分）バックアップをサポートしていません。

以下のセクションでいろいろなバックアップ戦略について述べます。場合によっては、最適な高可用性にとって、これらのアプローチの組み合わせを使うことがよいこともあります。

Ingresデータベース・バックアップ・ユーティリティ（ckpdb と rollforwarddb)

Ingres `ckpdb` ユーティリティを使ってバックアップを行い、`rollforwarddb` を使ってデータベースを復元したり、復旧したりします。バックアップは、自動的にデータベースを静止モードへ移行し、バックアップを始める前に、未完了のトランザクションが完了するのを待ちます。これらのユーティリティの使用法については、Vectorwiseユーザー・ガイドを参照してください。

バックアップ・ファイルは圧縮されていません。出力を圧縮ユーティリティへパイプすることで、バックアップ・ファイルを圧縮することができます。

OSレベルのデータベースのフル・バックアップ

OSレベルのフル・バックアップはデータベース・ファイルのバックアップを作成します。フル・バックアップの利点は、必要なものを全て含んでおり、理解しやすいことです。フル・バックアップの欠点は、バックアップを保管するのに必要なストレッジの領域の量と、データベースの一部分だけを復元するオプションがないことです。

データベースをバックアップするには

- そのデータベースをアクセスするアプリケーションを停止します。

- データベースをシャットダウンします。
- データベース・ファイルと構成ファイルを、tarコマンドやcpioコマンドなどで安全な場所にコピーします。
- データベースを開始します。
- データベースをアクセスするアプリケーションを再開します。

データベースを復元するには

- そのデータベースをアクセスするアプリケーションを停止します。
- データベースをシャットダウンします。
- バックアップしてあったデータベース・ファイルと構成ファイルを、tarコマンドやcpioコマンドなどで戻します。
- データベースを開始します。
- データベースをアクセスするアプリケーションを再開します。

サーバーが復旧ができず完全にクラッシュしたなら、データベースを復元する前に、Vectorwiseソフトウェアをインストールする必要があるかもしれません。

Ingres copydb/unloaddbデータベース・バックアップ

Ingres copydbコマンドとunloaddbコマンドは、データベース全体または、指定された個々のテーブルについて、データベース・オブジェクトの定義とアンロードのためのコマンドのスクリプトを作成します。このアプローチの利点は、必要なものを全て含んでおり、理解しやすいことです。それに加え、データベースの一部だけをバックアップできます。このバックアップメカニズムの欠点は、結果のデータ・ファイルを保管するのに必要なストレッジの領域の量と、データベースのアンロードと再ロードに時間がかかる点です。

データベースをバックアップするには

- そのデータベースをアクセスするアプリケーションを停止します。
- そのデータベースに対してcopydbまたはunloaddbコマンドを実行し、関連するスクリプトを実行します。
- データベースをアクセスするアプリケーションを再開します。

データベースを復元するには

- そのデータベースをアクセスするアプリケーションを停止します。
- 対象のデータベース・オブジェクトを削除するか、データベースを破棄/作成します。
- データベースに再ロードするためにスクリプトを実行します。

› データベースをアクセスするアプリケーションを再開します。

詳しくは、copydbとunloaddbについてのIngresのドキュメントを参照してください。

一部分のデータベースのバックアップ

データベースの一部分だけを復元したいなら、データベースの一部分のみをバックアップできる必要があります。たとえば、あるテーブルでデータの保守を行うとき、変更する前に、テーブルのデータを変更前の状態に戻すことができるようにしておく必要があります。そのためには、そのテーブルのデータ・セットのバックアップを取る必要があります。create table as selectを使ってそのデータのコピーを別のテーブルにコピーするか、copyコマンドを使ってテーブルのデータをファイルにエクスポートします。テーブルを復元するには、COMBINEコマンドを使用して、そのテーブルのデータを全て削除します。

```
CALL VECTORWISE (COMBINE 'basetable - basetable')
```

次に、insert as selectを使い、データを再度挿入するか、copy文を使ってファイルからデータを読み、テーブルに戻します。

警告：COMBINE文を使わずに、テーブルを削除（drop）すると、依存するすべてのオブジェクト（ビュー、権限、制約など）が削除されます。

データベースの再ロード

バックアップ戦略には、もとのデータソースからデータを再ロードすることも含めて考えます。このシナリオでは、使用しているデータベース全体についてソース・データ・セットが利用可能な必要があります。

ハイブリッドのアプローチ：フル・バックアップと増分再ロード

データベースのフル・バックアップは、Vectorwise をバックアップするもっとも簡単な方法です。しかし、データベースのサイズが数TBあるかもしれません。その結果、使用しているハードウェアにもよりますが、バックアップとその復旧は、かなり時間がかかります。また、データベースのフルバックアップは、ストレッジのかなりの領域を占めることになります。

データウェアハウスで良く使われている戦略は、週毎や月毎の定期的なデータベースのフル・バックアップと、次回のフル・バックアップまでの増分データ・ロード（データ・ファイル）を保管することです。データベースの復元と復旧は、最新のデータベースのフル・バックアップを復元し、最新のフル・バックアップ以降のデータ・ロードについて再度ロードを行います。

スタンバイ構成

さまざまな計画的、非計画的な機能停止に対するもっとも良い手段は、スタンバイ構成をとることで

す。スタンバイ構成は、メインのシステムとは異なるデータセンター、理想的にはできるだけ離れた場所に配置すべきです。これによって洪水、地震、大規模な停電やネットワーク障害といった災害からシステムを守ることができます。

スタンバイ構成を実装する一番シンプルな方法はメインのシステムと同じものを用意し、同じデータのロード作業を行ってスタンバイにデータをロードすることです。この実装方法であれば、レプリケーション技術によるデータ破損のリスクや、事故のリスクを最小限にします。たとえば、DBAが、テーブルを空にする作業で、テスト環境に接続したと思いこみ、実は、運用環境に接続したといった事故があります。レプリケーションやミラーリングがこのケースで使用されていると、データの削除作業がスタンバイに複製/コピーされ、災害復旧戦略の価値がなくなります。

Vectorwise の監視

Vectorwiseを監視するには大きく分けて２つの方法があります。Actian Directorを使用する方法と、サーバー上でコマンド行ツールを使う方法です。

Actian Director

Vectorwiseには、Actian Directorが含まれています。Director をデータベースサーバー上でダイレクトに動作させてもよいし、DirectorをリモートのWindows、Linux、Mac OS/2上にインストールしてもかまいません。

vwinfo ユーティリティ

vwinfoユーティリティは、コマンド行のユーティリティでデータベースサーバー上で動作させます。以下にこのユーティリティで得られる情報について示します。詳しくは、Vectorwiseユーザーガイドを参照して下さい。

統計

vwinfoのデフォルトのモード（または、-sオプションを指定）は、統計情報を出力することです。メモリの使用状況と、ストレッジの使用量、セッション数が表示されます。たとえば

```
/u01/vw/ingres/bin/vwinfo
Using settings:
database   : 'dbt3'
table      : ''
Connecting to VW server at port '48896'
Executing query
+------------------------------+------------------------------------------+
|stat                          |value                                     |
|varchar(32)                   |varchar(44)                               |
+------------------------------+------------------------------------------+
|memory.query_allocated        |18205360                                  |
|memory.query_maximum          |4294967296                                |
|memory.query_peak             |302183296                                 |
|memory.query_virtual_allocated|18199312                                  |
|memory.query_virtual_maximum  |70368744177664                            |
|memory.query_virtual_peak     |141964520976                              |
|memory.update_allocated       |0                                         |
|memory.update_maximum         |1073741824                                |
|memory.committed_transactions |0                                         |
|memory.bufferpool_allocated   |0                                         |
|memory.bufferpool_maximum     |2147483648                                |
|bm.block_size                 |524288                                    |
|bm.group_size                 |8                                         |
|bm.columnspace_total_blocks   |262144                                    |
|bm.columnspace_free_blocks    |260865                                    |
|bm.bufferpool_total_blocks    |4096                                      |
|bm.bufferpool_free_blocks     |4096                                      |
|bm.bufferpool_used_blocks     |0                                         |
|bm.bufferpool_cached_blocks   |0                                         |
|bm.columnspace_location       |/u01/vw2.0/ingres/data/vectorwise/dbt3/CBM/default/0|
|system.active_sessions        |0                                         |
|system.log_file_size          |1357420                                   |
|system.threshold_log_condense |33554432                                  |
```

```
+----------------------------+----------------------------------------+
(23 rows)
vw@vanma01-vw-t61:~$
```

データベースの構成

vwinfo -c <db name> を実行することで、データベースの構成情報を表示できます。以下はその例です。

```
vw@vanma01-vw-t61:~$ vwinfo -c dbt3

/u01/vw/ingres/bin/vwinfo
Using settings:
database   : 'dbt3'
table      : ''
Connecting to VW server at port '48896'
Executing query
+-----------------------------------------------+------------------------------------------------------------+
|config                                         |value                                                       |
|varchar(53)                                    |varchar(60)                                                 |
+-----------------------------------------------+------------------------------------------------------------+
|cbm.append_fill_threshold                      |0.500000                                                    |
|cbm.append_max_tuples_to_recompress            |1048576                                                     |
|cbm.append_reuse_last_block                    |true                                                        |
|cbm.auto_pax_storage                           |false                                                       |
|cbm.block_size                                 |524288                                                      |
|cbm.bufferpool_blocks                          |0                                                           |
|cbm.bufferpool_memuse                          |0.250000                                                    |
|cbm.bufferpool_size                            |2147483648                                                  |
|cbm.columnspace_grow                           |true                                                        |
|cbm.columnspace_size                           |137438953472                                                |
|cbm.compression_max_compressible_string        |-1                                                          |
|cbm.compression_nulls_enabled                  |true                                                        |
|cbm.group_size                                 |8                                                           |
|cbm.io_prefetch_blocks                         |16                                                          |
|cbm.max_database_size                          |0                                                           |
|cbm.max_num_columns_per_pax_group              |64                                                          |
|cbm.max_num_pax_groups                         |0                                                           |
|cbm.max_sum_column_width_per_pax_group         |262144                                                      |
|cbm.minmax_maxsize                             |1024                                                        |
```

テーブルブロックの使用量

-tbu オプションを使用することで、ディスク上のテーブルのサイズについて情報が出力されます。表示されたblock_countに、ブロックサイズを掛けることでストレッジの使用量が求められます。たとえば

```
/u01/vw/ingres/bin/vwinfo
Using settings:
database   : 'dbt3'
table      : ''
Connecting to VW server at port '48896'
Executing query
+----------------------+--------------------+
|table_name            |block_count         |
|str                   |slng                |
+----------------------+--------------------+
|vwSregion             |                   3|
|vwSnation             |                   4|
|vwSsupplier           |                   8|
```

```
|vwSpart                |                  31|
|vwSpartsupp            |                 210|
|vwScustomer            |                  50|
|vwSorders              |                 217|
|vwSlineitem            |                 719|
+-----------------------+--------------------+
(8 rows)
vw@vanma01-vw-t61:~$
```