



**データ・ウェアハウスのポイントと
IBM Red Brick Warehouse™**

目次

はじめに	1
1 データ・ウェアハウスのアーキテクチャー	1
1-1 基幹系データベースと情報系データベース	1
1-2 OLTP系データベースとDWH系データベース	2
2 IBM Red Brick Warehouseの優れた点	2
2-1 Red Brick®の更新処理	2
2-2 Red Brickクエリー処理(参照処理)	3
3 IBM Red Brick Warehouseのコスト・パフォーマンス	3
4 IBM Red Brick Warehouse技術の概要	4
4-1 STARjoin™およびインデックス	4
4-2 TARGETindex™、TARGETjoin™	5
4-3 インデックスの採用	5
4-4 IBM RISQL™	6
4-5 クエリー式	6
4-6 参照整合性	7
4-7 Table Management Utility(テーブル管理ユーティリティ)	7
4-8 Warehouse Administrator	8
4-9 IBM Red Brick Vista™	8
4-10 DST/ECC	9
4-11 接続性、セキュリティ	9
5 IBM Red Brick Warehouseの新機能	10
6 参考資料	11
6-1 スター・インデックス(STARindex)	11
6-2 ターゲット・インデックス(TARGETindex)	13
6-3 ターゲット・ジョイン(TARGETjoin)	13
6-4 並列処理とセグメント化	14
6-5 有用な集約データ(IBM Red Brick Vista)	15

はじめに

ウィリアム・イモン氏がデータ・ウェアハウスの提唱者であるならば、Red Brickの発表と共に1991年にスター型スキーマという方法論を唱えたラルフ・キンボール氏は伝道師といわれています。そのラルフ・キンボール氏が創業し、開発したデータベースがIBM Red Brick Warehouse(以降Red Brickと表記)です。現在ではIBMに引き継がれ、多くのお客様の高い評価をいただいています。

Red Brickはデータ・ウェアハウス(以降DWHと表記)用のデータベースとして代名詞的な取り上げ方をされてきました。現在のDWHデータベースであれば必ずと言っていいほど導入されている技術、スタースキーマの元祖であり、データ・ローディングの重要性を指摘し、業界最高クラスの処理速度を実現するローディング・ツールを製品に取り入れたのもラルフ・キンボール氏です。

この説明書では、前半(1章 - 3章)に、DWHを包括する情報システム、基幹システムの説明、そしてOLTP系処理とDWH用のデータベースの紹介に続き、Red Brickの紹介と投資効果をご説明します。後半(4章 - 5章)はRed Brickの技術的な説明をしています。

1 データ・ウェアハウスのアーキテクチャー

1-1 基幹系データベースと情報系データベース

< 1-1-1 基幹系システム >

日々の業務伝票の処理に代表されるような、製品の出入庫データ、在庫データ、入出金の情報、そして、Web上でのオンライン・ショッピングを営業している会社では、そこからの受発注データ、Webアクセス情報などがオンラインでOn-Line Transaction Processing(以降OLTPと表記)データベースに流れ込みます。このようなオンライン・ベースの多くの更新処理を速やか、かつ正確にこなすことが基幹システムの重要な役割です。そしてその技術的中核を成すOLTP系データベースはこのような要件を満たす必要があり、また高い応答速度が要求されます。そしてデータの規模はDWH用データベースに比べ容量が少なく、瞬間データベースと呼ばれることもあります。

< 1-1-2 情報系システム >

これに対して情報系システムはデータの参照が中心となり、その中核を成すデータベースはDWH系のデータベースになります。

基幹系システム	データの更新が中心
情報系システム	データの参照が中心

情報系の参照は、基幹系の参照と比較すると、検索時に指定される期間の長さが違うといわれています。DWHデータベースには、半年から数年のデータが蓄積されることが多く、1年間に発生する全データへの分析には大量のデータが対象になることとなります。そしてこのような分析は一般的に行われています。

例えば、あるコンビニエンス・ストアの売上分析において、人気のある商品分析を去年と今年のベストテンで比較する場合、参照対象となるデータの規模は非常に大きくなることが、簡単に想像できます。

このように情報系の中核を成すDWHデータベースには大規模な参照処理を高速に処理する能力が求められているのです。

情報系の更新は一般的に、深夜に当日発生分のデータをバッチで取り込むときに行われます。

バッチ処理の場合は、多重処理(複数の処理が同時に行われるためにシステムの処理能力が低下する傾向)への配慮が不要なため、効率的に大量データの更新が可能になります。夜間はオンライン・ユーザーが少なく、システム負荷の配慮もそれほど必要ないことから、日中ではなく夜間に自動起動するケースが一般的です。

またDWH系データベースは数年間のデータを保存することが多いことから、基幹系に比べデータ容量が多いことも特徴の一つです。

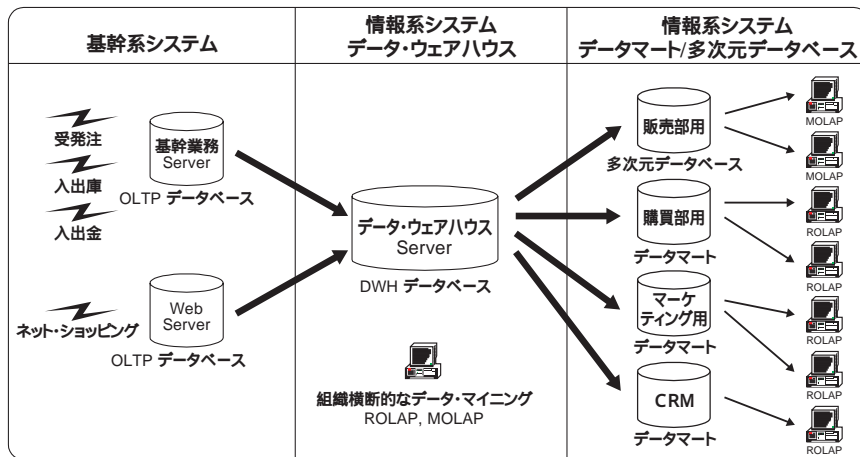


図1-1: データ・ウェアハウスのアーキテクチャー

基幹システム(OLTPデータベース)

役割：オンライン・ベースの多くの更新処理を速やかかつ正確にこなす
特徴：データの更新が中心、データ量は少ない
拡張：更新処理の能力向上を目的に実施することが多い

情報システム(DWHデータベース)

役割：大量のデータへの問い合わせを速やかに処理する
特徴：データの参照が中心
拡張：データ参照能力向上を目的に実施することが多い

1-2 OLTP系データベースとDWH系データベース

基幹系、情報系との比較でその特徴を説明してきました。それぞれのシステムで中核となるデータベース技術は、OLTP系とDWH系に分けられます。特徴は、OLTP系は更新処理を、DWH系は参照処理を重視する特徴を持っています。

このように、DWHを構築する際、基幹システム用のデータベース(OLTPデータベース)とは別に、DWH専用のデータベースを配置することで、更新処理、参照処理それぞれを得意としたデータベースの配置が可能になります。

今日、他社製の汎用データベースとして、OLTP処理と、DWH処理の双方をサポートする製品がでており、人気を集めています。これはそれぞれの目的を持った配置に対応できるということで、物理的な1つのデータベースが2つの目的に利用できるわけではありません。あくまでもどちらかの仕様を採用し、開発者がその最適化を図ることで目的に特化したデータベースの配置が可能となるのです。

Red Brickは元々DWH専用のデータベースですので、出荷時点からDWH用にデータベースが最適化されています。これにより比較的容易にシステムを構築することができ、他社製の汎用データベースよりも優れたパフォーマンスを発揮できます。

また、OLTP処理を得意とする汎用データベースを基幹システムに配置して連携することで、ユーザーにとって使いやすい環境が構築できます。

2 IBM Red Brick Warehouseの優れた点

レッドブリック社は1985年にシステム・コンサルティングの会社としてスタートし、多くのお客様の意見を取り入れた製品作りを続けてきました。DWH専用のデータベースの技術開発は、15年を過ぎた現在も続けられており、より多くのお客様の満足を得られるよう絶え間ない努力を続けています。

2-1 Red Brickの更新処理

データ・ウェアハウスの提唱者ウィリアム・イモン氏が高く評価するラルフ・キンボール氏はDWHのデータ更新を「プロダクション・データロード」と呼び、その技術の重要性を強調しています。

一般的に解釈されているデータ・ローディングとは単純にアプリケーション(データベース)にデータを取り込むことを言いますが、Red Brickのローディングでは通常のローディングに加えて多くの重要な作業がなされています。

1. データ変換
2. データの読み込み
3. 参照整合性のチェック
4. インデックスの生成
5. サマリー・データの生成

Red Brick Warehouseのローディング・パイプライン処理

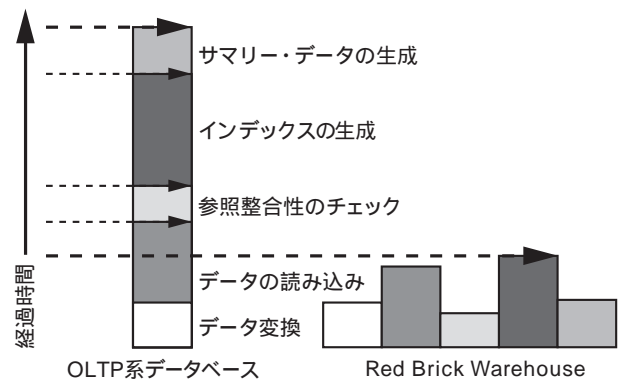


図2-1-1: ローディングのパイプライン・パラレル比較図

Red Brickは上記の5つの処理を並列処理技術で高速に処理し、そのスピードは業界トップクラスの評価を獲得しています。その高速化を実現しているのは、パイプライン・パラレル技術で、図2-1-1の左側のグラフのように他社データベースが順次処理をするところを、並列に処理するという技術です。

1件ずつのデータがパイプの中を流れるように順次処理が流れ、すべてのパイプを通過した時点で処理が終わることから、この名称が使われています。

このようにRed BrickはDWH専用のデータベースとして、他社製データベースにはない、柔軟な更新機能を積極的に取り込み、優位性を持っているといえます。またDWHには必要の無い機能を積極的に外すことでシンプルな構造を目指し、高い信頼性を獲得しています。

例えば、汎用データベースが得意とする、OLTP処理への対応は最小限に抑えられています。

一般的にDWHへの更新処理はデータのインテグリティを高めるために、データ・ローディング以外のオンライン更新を控える傾向にあるからです。このシンプルな構造によって、他社製の汎用データベースよりもはるかに安定したサイトを構築できるといえます。DWH専用データベースであるRed Brickであればこそできること、そこにこのエンジンの強みがあるといえます。

他社DBMSとのパフォーマンス比較 (イメージ)

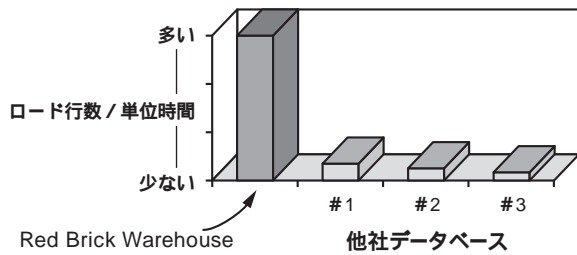


図2-1-2: ローディング性能の実績比較

過去の導入事例などによる調査データをもとにRed Brickのパフォーマンスを視覚イメージ化したもので、システム環境、諸条件などによりパフォーマンスは異なります。また、将来にわたって上記のパフォーマンスを保証するものではありません。

2-2 Red Brickクエリー処理 (参照処理)

Red Brickのクエリー・スピードも業界トップクラスの高速を誇っています。そのスピードを実現する3つの技術的特徴を紹介します。

1. STARindex/TARGETindexによる高速クエリー
2. 集計テーブルの自動計算機能とクエリー・リライト
3. 豊富な分析用関数によって複雑な処理を高速処理

この3つの技術によって次のようなことが可能になります。

1. 大量明細データへの検索を高速に処理する。
2. 大量データの検索・集計を求める計算時にあらかじめ用意した集計テーブルを自動的に採用し、高速に処理する。
3. 大量データのカテゴリからトップテンを出力する。

「特徴1. STARindex/TARGETindexによる高速クエリー」については4章にてご説明いたします。

「特徴2. 集計テーブルの自動計算機能とクエリー・リライト」の技術はVistaと呼ぶ技術で、自動集計テーブルの自動集計機能のほか、集計テーブルの作成を自動的に推奨してくれる機能も備えています。

これを効果的に利用することで、ROLAPの活用のほかにMOLAPとしても同様にその役割を果たすことから、多くのマイニング・アプリケーションが開発されています。

「特徴3. 豊富な分析用関数によって複雑な処理を高速処理」で提供される関数は、DWHでは一般的に使われる関数であり、他社データベースでは非常に多くの時間がかかるものを、高速に処理することが可能です。

ROLAP、MOLAP、データ・マイニングの用語説明

OLAPとはオンライン解析処理のことを言い、大量のデータを対象に、時間や場所、人といったさまざまな切り口、次元(ディメンション)からデータを仮説に基づき分析する手法を指します。この手法を用いて多角的な分析を簡易にしたデータベースを主に多次元データベースといえます。多角的な分析は、多次元データベースのみならず、リレーショナル・データベースにおいても実現可能ですが、多次元データベースの方が専門性が高いことが多くあります。

また、多次元データベースは、その明細情報をもたないことが多いという特徴から、ドリルダウン(集計データから当該の明細データを表示する機能をサポートすることは少ないため、汎用性はリレーショナル・データベースの方が高いといえます。

多次元データベースにあらかじめデータを蓄積し、検索処理要求に対応するOLAPのシステム形態をMOLAP、事前に多次元データを用意せず(多次元データベースに格納せず)、検索実行時にリレーショナル・データベースの検索機能を用いて解析機能を行うシステム形態をROLAPと言います。

データ・マイニングとは情報を探し出すプロセスのことを指します。付加価値の無い、純粋なデータのかたまりから、情報に値するものを選定し、その中のばらつきを発見し、傾向を導きだし、その根拠を元にビジネスを優位な方向に導く、それがデータ・マイニングです。

最近では、標準的なビジネス分析(財務、会計など)用に、マイニング・ツール、アプリケーション製品が各社から提供されています。

3 IBM Red Brick Warehouseのコスト・パフォーマンス

純粋にソフトウェア・ライセンス料金を他社と比較した場合、Red Brickは標準的な価格帯を採用しています。

しかし、ハードウェアを含めたトータル金額で他社製データベースと比較しますと、大幅なコストダウンにつながります。なぜなら、Red Brickは圧倒的な性能の優位性をもっていますので、サーバーへの投資コストを削減することが可能で、費用対効果の比較においては如実に差別化が図られているからです。

また、目に見えない要素としては、管理作業の軽減があります。極端に表現しますと、2-3ヵ月放置していても機能を損なうことはなく、特別な管理作業を必要としない性質をもっています。

このように、DWH運営全体でのTCO(トータル・コスト・オブ・オーナーシップ)を削減することが可能です。

4 IBM Red Brick Warehouse 技術の概要

Red Brickは、データマート、DWH、およびオンライン解析処理(以降OLAPと表記)アプリケーション用に設計されたリレーショナル・データベース管理システム(以降RDBMSと表記)です。Red Brickは、長年にわたってほとんどのテクノロジー・コンポーネントを独自に採用しています。他社製のデータベースも機能的に追いついてきていますが、今もなお、Red Brickにしか存在しない機能が提供されています。

このようなテクノロジー・コンポーネント開発の担当の多くは、Teradata、Oracleなどのほかのデータベース分野で先駆的な立場にいました。彼らは、今日IBM Red Brick Warehouseとなった最先端技術を作り上げたのです。ここでは、Red Brickの主要な技術について概要を説明し、それぞれがどのように連携して動作するか解説します。

4-1 STARjoinおよびインデックス

次元モデルにはファクト・テーブル、およびディメンション(次元)テーブルが含まれます。ファクト・テーブルとはビジネスに関する定量的なデータ、つまり必要なファクト(事実)を保持するテーブルで、ディメンション・テーブルとは、次元属性を表現するテーブルを意味します。図4-1が示すように、ディメンション・テーブルとファクト・テーブルは、外部キーによってお互いを参照し関連付けられます。

図4-1は、ファクト・テーブルとディメンション・テーブルの関係、およびそれらに関連付ける外部キーを示しています。これらのテーブルは、サンプル・データベースで使用されているものです。Billing_CDRテーブルがファクト・テーブルで、その他はディメンション・テーブルです。

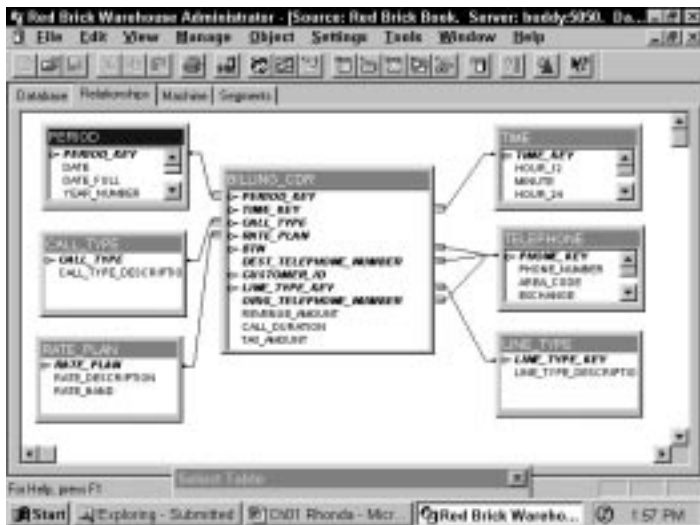


図4-1ファクト・テーブルとディメンション・テーブルの関係、およびそれらに関連付ける外部キー

次元モデル

STARスキーマの本質、利点、そして内容に関して、最近、非常に多くの混乱と誤解が生じています。STARスキーマとは、ラルフ・キンボール氏が発明した用語で、大規模なファクト・テーブルの周りに存在するディメンション・テーブルが、星(STAR)型のパターンを形成する仕組みを指します(図4-1を参照)。ここでは、次のような概念を伝えるために次元モデル、およびSTARスキーマという用語を同じ意味で使用しています。STARスキーマとは次元モデルで、それ自体を構成するテーブルの個数や種類に制限はありません。DWHを構築するときに基本となりビジネスの手順を正確にモデル化するのに使用できます。

ご存じの方もいらっしゃると思いますが、すべてのデータベースは、ヘアワイズ・ジョインと呼ばれる方法でテーブルをジョイン(結合)します。Red Brickは、リレーショナル・データベースの性能を向上するには、インデックスの性能を向上することが必要であると考えました。そのため、STARindex、およびSTARjoinアルゴリズムが密接にかかわるようになったのです。STARjoinテクノロジーにより、複数のリレーショナル・テーブルは高速でジョインされ、単一受け渡し処理によりクエリー時間が大幅に節約されることになりました。

ベアワイズ・ジョイン

ベアワイズ・ジョインは、汎用データベースがテーブルをジョインする以外に他の仕事もします。Red Brickの経験が少しでもあるユーザーは、"ベアワイズ・ジョイン問題の困難" という表現を聞いたことがあるかもしれません。それは、複数受け渡しによる解決方法で、テーブルをジョインするのに複数回の受け渡しが実行されます。この問題は、DWHがまだ成熟していない時期に、汎用データベースが持つ技術的な制限によるものでした(いまだにその問題を持つものもありますが)。このような方法でテーブルをジョインするさまざまな汎用データベースでは、実際の次元モデルでもRed Brickほどの性能が得られません。

ベアワイズ・ジョインの問題となっているのは、使用される技術がDWHを意識していないことと、オンライン・トランザクション処理と同じように、一度に2つのテーブルにアクセスしてウェアハウスへのクエリーへ応答することです。

500,000,000行を持つSales(セールス)ファクト・テーブル、1,200行を持つPeriod(期間)ディメンション・テーブル、および8,000行を持つCustomer(顧客)ディメンション・テーブルを例に説明しましょう。ペンシルベニア州のすべての顧客に対する、最新四半期の売上高を知る必要があるとします。汎用データベースは、クエリーを完了するのに、2つのテーブルを場合によってはランダムにアクセスし、必要なテーブルがジョインされるまで、一時的なりザルト・セットを次々に生成しようとする。

この状況を例に即して説明すると、次のようになります。

SalesテーブルとPeriodから処理が開始されます。1,200行のテーブルと500,000,000行のテーブルがジョインされた結果、100,000,000行が生成されることになります。これは、ディスクをオーバーフローさせ、大容量のメモリーを消費するのに十分な量です。

次に、一時的なりザルト・セットが生成され、Customerテーブルにジョインされます。つまり、100,000,000行のテーブルが生成されジョインされ、その結果、50,000行のりザルト・セットが生成される可能性があります。おわかりのように、このような処理は無駄な入出力処理を要し、CPUに多大な負荷をかけるため、一般的には勧められません。

汎用データベースも長年の間には改良され、サイズの小さなテーブルをジョインし大規模なファクト・テーブルへジョインする前にデカルト積を生成するような、さまざまな最適化機能を装備しています。しかし、このような機能もベアワイズ・ジョインに変わりなく、多数のテーブルを読み取らなければならないという複雑さが増えるため、性能が低下する傾向にあります。

4-2 TARGETIndex、TARGETJoin

Red Brickのビットマップ・インデックス・テクノロジーは、TARGETIndex、およびTARGETJoinにより構成されています。ビットマップ・インデックスとは、ビット・ベクトルとも呼ばれ、列ごとの独自の値を圧縮ビット表現方式で追跡し、その値によりポインター、つまりベクトルでテーブルの各行を参照し、情報の保持を実行する方法です。汎用データベース・ベンダーのほとんどは、このようなインデックスを使用しますが、Red Brickは、追加型のテクノロジーではなくサーバーの一部として組み込むように、開発されています。

TARGETIndexは、選択肢が少ない単一列に存在します。制約の選択肢が少ないと、テーブルからは多数の行が返されることとなります。例えば、顧客テーブルのGENDER(性別)列は、2つの値、男性/女性を持つことが考えられます。このように選択肢が少ない条件は、高位のデータ分野でデータ分布が大きく偏っている場合にも当てはまります。このような状況のよい例として、年齢が挙げられます。

TARGETJoinは、候補となる次元列のリストを特定するのにTARGETIndexを使用するジョイン・アルゴリズムです。各リスト内の列は、ファクト・テーブルから取得されます。



TARGETIndexは、選択肢の少ない制約で列にインデックスを生成するのに使用されます。Home Owner(自宅所有)列に対する制約は、おそらく、“ある、ない、不明”の3つの値を持つでしょうが、このように選択肢が少ないと、10,000,000行のテーブルから多数の行が返されることとなります。

4-3 インデックスの採用

STARIndex、TARGETIndex、その他のインデックスを必要に応じて作成しておくことで、システムはクエリー要求があった時点で、自動的に最適なインデックスを採用し適用します。

4-4 IBM RISQL

DWH市場の興味深いものの一つは、標準的なビジネス解析をSQL(標準クエリー言語)で処理しようとする発想です。ご存じのとおり、SQLは、SQLデータベースへのアクセスに標準的に使用され、リレーション・セット論理に基づいています。しかし、セット論理での基本的機能だけでは、ランク付けや、ある時点での平均、合計などのビジネス上の現実的な問題点を解決するには不十分です。ほかのリレーショナル・データベース・エンジンにおいて、標準SQL言語の拡張機能を見たり、実際に使ったことがある方もいると思いますが、Red Brickもその例外ではありません。Red Brick拡張機能は、従来のデータベース・テクノロジーでは解決するのが不可能、または困難な、現実世界でしばしば直面するビジネス解析の問題点を解決することを目的に設計されました。IBM Red Brick Intelligent SQL(以降RISQLと表記)拡張機能により、一般的に使用されることの多いビジネス上の目的は単純な式で表され、連続計算により値と複数の列が返されるようになりました。

CUME : 1セットの値に対する列で、ある時点での合計を算出します。リセットするには、ORDER BY節のRESET BYを使用します。

MOVINGAVG : 特定のセットの値に対し、n個の行ごとに平均値を算出します。リセットするには、ORDER BY節のRESET BYを使用します。

MOVINGSUM : 特定のセットの値に対し、n個の行ごとに合計値を算出します。リセットするには、ORDER BY節のRESET BYを使用します。

NTILE : 定義した範囲で結果をランク付けします。TERTILEは、結果を3層(高、中、低)に分類するのに対し、NTILEは、必要なだけ層を生成することができます。NTILE、およびRANKを参照してください。

RANK : 値に対応して、1~10、低~高などの、ランク付けを階層的に実行します。

RATIOTOREPORT : ある値のセットの合計に対応して、ある値が占める割合を算出します。

TERTILE : ある値のセットに対応して、3層のランク付け(高、中、低)を実行します。

RESET BY : RISQL表示関数で算出された値を、ORDER BY節で参照される列に基づいてゼロに初期化します。RESET BYは、ORDER BY節の副節です。

< 4-4-1 RISQL Reporter >

RISQL Reporterはコマンド・ライン・インターフェースで、合計、副合計、および顧客列見出しを加算してSQL出力形式を指定することができます。列の幅、間隔も指定することができます。

< 4-4-2 Web解析標準 >

バージョン6.0を発表したのはそれほど昔のことではありませんが、このリリースで、Web解析機能がRed Brickに追加され、Web、およびe-コマース・アプリケーションの高度な要求に対応するようになりました。この標準機能は、One to Oneマーケティング、e-コマース、およびCRMアプリケーション構築のためのプラットフォームを提供します。

4-5 クエリー式

Red Brickは、SELECT、FROM、WHERE、GROUP BY、HAVING、およびWHENなどの、すべての標準的なクエリー式を実装しています。この標準的なクエリー式を、あらかじめ理解しておく必要がありますが、WHEN節の機能の一部について、ここで説明します。WHEN節は、すべてのSET機能が実行され、HAVING節の条件が適用され、すべてのRISQL表示機能が実行された後で、リザルト・セットに条件を設定します。

< 4-5-1 スカラー関数、マクロ >

RISQL拡張機能の一部である豊富なスカラー関数により、複合式を作成することができます。スカラー関数は、一度に1つの行で処理されます。入れ子にしたり、引数としてRISQL表示関数を取ったりすることができます。スカラー関数は、次のように分類されます。

条件関数

CASE
COALESCE
DECODE
IFNULL
NULLIF

数値スカラー関数

ABS
CEIL
DEC
FLOAT
FLOOR
INT
SIGN
EXP
LN
SQRT
LENGTH

文字列スカラー関数

CONCAT
LOWER
LTRIM
RTRIM
STRING
SUBSTR
TRIM
UPPER

日時スカラー関数

CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
DATE
DATEADD
DATEDIFF
DATENAME
EXTRACT
TIME
TIMESTAMP
CURRENT_USER

統計関数のマクロ

Power
Log
Log10

Variance(分散)
Standard deviation(標準偏差)
Population variance(人口分散)
Population standard deviation(人口標準偏差)

4-6 参照整合性

参照整合性は、テクノロジー・コンポーネントではありませんが、Red Brickの基本的な要素なので、説明しておく必要があります。参照整合性(RI)とは、外部キーを適切なマスター・テーブルで確認し、存在する外部キーの有効性を調べる処理のことを意味します。この有効性の確認は、非常に重要になってきます。

RIチェックは、ロード時に実行されるのが一般的です。よく知られていることですが、汎用データベースにとって、このRIチェックは多くの問題の原因となり、全体のロード時間が大幅に増加してしまいます。そのため、データを取得する場合、RIを無効にすることがよく行われます。つまり、ロードのパフォーマンスが正常なデータのどちらかを選択しなければならなくなるのです。RIチェックを無効にすることは、実際にはDWHの品質を犠牲にすることになります。

Red Brickでは、どちらかを選択する必要はありません。RIチェックは、ロード処理に組み込まれているからです。RIチェックは高速で処理され、関連付けられるべき行が正しく関連付けられていることを保証します。RIがどうして大切なのか、次に実際の例を挙げて説明します。

ある小売業者は、各店舗の在庫レベルを毎週更新し、商品の注文量を決定する必要がありました。一般的なデータベースを使用して、データをロードするのに約2日間かかりましたが、その間に、110,000,000の行のうち10パーセント以上が失われてしまいました。これは、RIを無効にしていたためです。Red Brickを使うと、データのロードは約6時間で終了し、すべての行が正しく処理されました。問題のない行は正常にロードされ、問題のある行は、RIやデータ有効確認機能によりロードされませんでした。

注意

RIチェック処理が実行されるのは、データがトランザクション・データベースに書き込まれるときなので、データがウェアハウスにロードされる時に再び実行される必要はない、というような話を聞いたことがあるかもしれません。このことは間違っています。RI処理が既に実行されていたとしても、データは、実際にはデータ・ルールではなく、ビジネス・ルールによって再編成されていることを考えると、この話に信憑性がないことは明らかです。“疑わしきものは信じず”です。

4-7 Table Management Utility(テーブル管理ユーティリティ)

DWHに格納されるすべてのデータは、外部システムに存在します。このようなデータには、患者の通院記録、交換機のデータ、小売データ、または購入データなどが含まれ、その種類は異なります。このようなウェアハウスへロードするすべてのデータを収集し、準備しておくことは、重要な手順のひとつで、そのためには多くのことを実行します。

- ・ ディスク・ドライブ、ネットワーク、磁気テープ、メインフレームとの接続など、さまざまなデータ・ソースからデータが読み取れるようにしておく必要があります。
- ・ さまざまな外部形式のデータを、内部データベース形式に変換する方法を理解しておく必要があります。外部形式には、EBCDICデータ、バック/ゾーン10進数データ、固定/非固定長の行やその他の形式が含まれます。
- ・ 無効なデータ、重複行、その他の不正なデータを選別し取り除いておく必要があります。
- ・ セグメント化、および物理ディスク配置の構成条件に基づいて物理記憶域にデータを書き込む方法を理解しておく必要があります。

Table Management Utility(以降TMUと表記)は、データのロード、およびテーブル、インデックス、そしてデータベースの整合性を保持するのに使用されます。前述の必要条件を満たすため、データをロードするという主要な機能以外にも、以下の機能を提供します。

- ・ データのアンロード
- ・ テーブル/インデックスの再構築
- ・ DDLの生成
- ・ データベースの更新

TMUについて最も重要なのは、DWH構築に特有な、大切なデータ処理を扱うよう設計されている点です。後になって考え出されたり、マーケティングの要求で継ぎ足されたようなものではありません。TMUは非常に役に立つ機能なのです。

Red Brickデータベースをロードするには、次の4つの手順に従います。

1. データの変換
2. 参照整合性チェックの実行
3. インデックスの作成/更新
4. データの書き込み/更新

Red Brickは、データを変換するとき入力ファイルのデータ型を、対応する列のデータ型に変換します。この処理で、範囲、日付、時刻などが確認されます。その次に、どのウェアハウスにとっても成功へのキーとなる、参照整合性チェックが処理されます。

この後でインデックスが次々に生成され、データが更新されます。ローダーには、重複した行、および新規の行の処理に関するいくつかのオプションがあります。これらのオプションはロード・モードとよばれ、次の種類があります。

APPEND : 新規の行を既存のテーブルに挿入します。入力ストリームの行がテーブルに既に存在する場合、その行は削除されます。主キーがない行も同様に処理されます。

INSERT : データを空のテーブルにロードします。サーバーは、テーブルが空であることを認識する場合、インデックス作成の方法を判断します。適切な外部キーによる参照がない行は削除されます。

REPLACE : テーブルの内容を削除してから、空になったテーブルにデータをロードします。このモードを使用するには注意が必要です。大抵のユーザーは、一度はこのモードで失敗することがありますので、十分に気を付けてください。

MODIFY : 新規の行を挿入するか、既存の行を更新します。

UPDATE : 既存の行を更新します。新規の行は削除されます。

数多くの構成パラメーターにより、処理のパフォーマンスが向上します。削除された行を保存するオプションもあるので、問題が解決してから、このような行を再ロードすることも可能です。Red Brickの顧客が、ロード・パフォーマンスだけを理由にRed Brickを選んでいるというのは、興味深い事実でもあります。

< 4-7-1 自動行作成 >

いかなるデータにも、間違いはあるものです。ウェアハウスの初期の構築段階には、多くの次元列が失われ、完全なデータ・セットをロードすることができないことがよくあります。ほかのデータベース製品だと、RIチェックを実行したとしても、問題となる行は削除されてしまいます。最初にロードを行ったら、問題のある列を調べ、適切な次元列を追加し以前に無視した行を再ロードする必要があります。これは時間もかかる面倒な作業です。

Red Brickの自動列作成機能(autorowgen)により、データのロード中に参照される次元テーブルに自動的に列を作成することができます。autorowgenを使用すると、行が次元テーブルに自動的に追加され、失われた値で主キーを作成し、定義済みのデフォルト値で残りの列を生成することが可能となります。この機能は、以下の3つの特徴的な動作を実行します。

1. 失われた次元を生成し参照整合性を保持します。
2. 参照されるテーブルで、参照整合性を損なうような値をデフォルト値に置き換えます。
3. このような動作を組み合わせて、テーブル単位の処理を実行します。

< 4-7-2 Auto Aggregate(自動集約機能) >

DWHでは、さまざまな方法で集約テーブルや要約テーブルを使用します。Red Brickは、必要なときにだけ集約テーブルを作成するよう設計されていますが、Red BrickのVista機能により、DBAは、必要なときを判断するのがより簡単になりました。Auto Aggregate機能は、基本的に詳細データから集約テーブルを構築します。ローダーは、すべての詳細行がデータベースにロードされるときに処理を実行するので、ロード処理の後ではなく、ロード中に集約テーブルを構築します。このように集約テーブルを構築すると、処理が必要となるのは新規のデータによる変更のみで、集約テーブル全体を再構築する必要はなくなります。

< 4-7-3バージョンング >

バージョンングは、ユーザーのクエリーに応答する時間に影響することなく、DWHのリアルタイム更新を可能にするRed Brickの機能です。バージョンングは、クエリー実行中に、同じデータの異なるバージョンに対する、データ変更のトランザクションを許可します。データの更新が可能な場合、基本データがバージョンごとに更新されます。

4-8 Warehouse Administrator

Red Brickのグラフィック・ユーザー・インターフェース管理ツールにより、ODBC経由でRed Brickデータベースに接続し、ほとんどのDBAの作業を実行することができます。以下は、その一部です。

- DDLの再作成
- ユーザー、ロール、およびマクロの管理
- 列の追加、削除などの、テーブル、およびインデックスの管理
- ビュー、階層構造、およびシノニムの保持
- Vista制御の設定、およびRed Brickの集約テクノロジーの更新
- Vista Advisor 解析処理
- システム・テーブルの表示
- SQLウィンドウによる、対話的なSQLコマンド、およびクエリーの実行

ただし、Administratorでは、パフォーマンスを監視することはできませんが、この機能は2002年の次期バージョンで追加される予定です。

4-9 IBM Red Brick Vista

Red Brick Vistaは、事前計算された集約データを管理するメカニズムです。ベース(詳細) テーブルにクエリーを実行するには、必ずVistaを使用します。その理由は次のとおりです。

ユーザーに、集約を使用する時期、方法を教える必要がありません。ユーザーが意識する必要がないからです。

ビジネスが変化しても、集約以外に変更する必要がなくなります。

ユーザーがクエリーを実行すると、コスト分析により、クエリーを捕捉しリライト(書き換え)して既存の集約テーブルを使用してもよいかが判断され、パフォーマンスが向上します。サーバーは、クエリー実行の統計をログに記録するので、現在の集約戦略が適切かどうか、検討することができます。Red Brick Vistaは、次の2つのコンポーネントで構成されています。

- クエリー・リライト・システム Aggregate Navigatorと呼ばれ、集約テーブルを使用するのに可能な場合、クエリーを捕捉しリライトします。
- Advisor 収集した情報のログを記録し解析する機能を提供します。これにより、既存、および構築する予定の集約のコスト、利点を判断するのにクエリーを実行することができます。

4-10 DST/ECC

動的統計テーブル(Dynamic Statistic Tables : DST)は、サーバーが実行中の間だけ存在し、データベースの活動を監視するのに使用できます。このテーブルは、メモリーにのみ存在し、定期的に更新されますが、Red Brickのほかのテーブルと同様に表示し処理します。つまり、ほかのコンポーネントと同様にクエリーやジョインを実行することができるのです。

DSTの構成は次のとおりです。

- DST_COMMANDS : サーバーに送信された各コマンドに関する累積情報を含みます。
- DST_DATABASES : データベース全体の活動や、データベースの場所に関する情報を保持します。
- DST_LOCKS : 各セッションが保留するか待機するロックに関する情報を含みます。
- DST_SESSIONS : データベースに接続される各セッションに関する情報、および、累積的な統計とセッション・ピーク統計を含みます。
- DST_USERS : サーバーが起動されてからデータベースにアクセスした各ユーザーの情報を含みます。

Red BrickのEnterprise Control、およびCoordinationコンポーネントが提供するメカニズムにより、データベース、ユーザー、およびパスワードを、Administratorの、見やすいグラフィカル・インターフェース、またはRISQLのプロンプトのコマンド・ラインから管理することができます。

4-11 接続性、セキュリティ

Red Brickは、サーバーに接続するのに2つの方法があります。それは、ODBC、およびJDBCです。Red Brickが標準接続としてODBCを採用したとき、処理が遅くなるのではないかとこの心配がありました。しかし実際には、Red Brickを使用するユーザーのほとんどが理解するとおり、処理が遅くなることはありません。それどころか、満足のいくパフォーマンスが得られています。ごく最近になって、JDBCがサポートされるようになりました。

< 4-11-1 ODBC >

ODBCは標準データベース接続性プロトコルで、これにより、ほとんどのクライアント・ツールがRed Brickに接続することができます。さらに、すべてのRed Brickクライアント・ツール(RISQL、およびWarehouse Administrator)は、UNIX[®]、またはWindows[®]のどちらのプラットフォームでも、サーバーとの通信にODBCを使用します。

< 4-11-2 JDBC >

Java[™]データベース接続性(JDBC)は、標準アプリケーション・プログラミング・インターフェース仕様で、これによりJavaプログラムはRed Brickデータベースにアクセスすることができます。このインターフェース標準を使用して、データベースへの接続、SQLクエリーの送信、サーバーが返した結果の処理を実行するアプリケーションを開発することが可能となります。IBM Red Brick JDBC APIは、2層、および3層構成をサポートし、Java開発に柔軟性を提供します。

5 IBM Red Brick Warehouseの新機能

Red Brickが登場して以来、15年以上DWHに専用の機能を追加してきました。このような伝統は、現在IBMに受け継がれています。以下は、新機能の中でも優れたものを説明しています。

バージョン6.0の新機能

- ・ JDBC Type 4ドライバーのサポート
- ・ ロード中の、データのクエリー凍結ビュー
- ・ 領域の有効活用によるWebデータの処理
- ・ クエリーの結果の高速エクスポート
- ・ バージョニング付き 並列ロード中の高速RIチェック
- ・ バージョン ログ表示ユーティリティ
- ・ バージョン 6.0.3の新機能
- ・ デフォルト・セグメントに対する複数PSU
- ・ “読み込み”レベル・ロッキング
- ・ Windows 2000のサポート

バージョン 6.10の新機能

- ・ 集約の自動更新
- ・ ODBC 3.0
- ・ JDBC 2.0
- ・ 平行MODIFY機能
- ・ RIチェックのメモリー管理の改良
- ・ SET演算子に対するクエリーのリライト
- ・ ランダム・サンプリング
- ・ MODIFYおよびUPDATEに対するインデックス作成の最適化

バージョン 6.20の新機能(2002年を予定)

- ・ TMUベースのバックアップ & リストア
- ・ XMLベースのデータロード・エクスポート
- ・ パフォーマンス監視ツール

6 参考資料

ここまで、データウェアハウス、Red Brickの製品紹介と、全体の概要を説明してきましたが、ここからは、Red Brickが独自に持つキーとなる技術の一部を、具体的に説明します。特に

- ・ インデックス、ジョイン技術
- ・ 集計テーブルの管理技術

の2点についてご紹介します。

6-1 スター・インデックス (STARindex)

<6-1-1 スター・ジョイン (STARjoin) 型検索を最適化する >

DWHに最適なモデルである、ディメンショナル・モデル (= スター・スキーマ) を検索することをスター・ジョイン型検索といいます。ここで、うれしい問題が起きました。それは、ディメンショナル・モデルが従来のERモデルとは異なり、そのジョインの順番に結果が左右されない、ということです。従来のERモデルでは、ちょっとした検索でも、多数のテーブルをジョインしなければならないこと、そして、そのジョインの順番が暗黙のうちに決定されています。この特性のために、ERモデル上でのジョインは、多数のテーブル群から一組ずつ取り出してはジョインを施していくペアワイズ・ジョインしか事実上使用できず、このための

インデックス技術の研究が行われてきました。しかし、Red Brickのジョインでは、ジョインの順番などないのです。つまり、複数テーブル間のジョインを一度に行うことができれば、最も効率が良くなるはずですが、このアルゴリズムを最も早くから研究し、インデックスとして完成させたもの、それがスター・インデックス (STARindex) です。

<6-1-2 スター・インデックスのジョインと従来型ジョイン >

従来型ジョインでは、2つの方法しか採用することができませんでした。それは、ペアワイズ・ジョインとクロス・ジョインです。これについて、性能における大まかな試算を図式6-1で行ってみました (これでもかなり、従来型ジョインに有利な条件にしてあります)。

この式6-1に示すように、ペアワイズ・ジョインとクロス・ジョインは、どちらも大変なコストがかかるものです。Red Brickは、世界初そして現在でも唯一、スター・インデックスによるスター・ジョイン戦略を持つRDBMSエンジンです。それは、単に、従来の方式をCPUの力や数に任せて強引に並列化したものではありません。従来のジョインが、1, 2, 3, 4, 5, ... とやってゆくものとすれば、Red Brickのスター・ジョインはまさに1でやってしまうアルゴリズムを開発し、この仕組みをネイティブに搭載しているのです。このため、Red Brickでは、そのドキュメント・マニュアル内などでも、スター・インデックスを使用したジョインのみをスター・ジョインと呼び、他のジョインと区別しています。

対象テーブル	売上履歴 (100万件)、製品 (500アイテム)、店 (200店舗)、日付 (300日) の4テーブル。
検索条件	製品、店、日付 それぞれに検索条件をかけ、それらに一致した売上履歴を読み出す。(無論、検索対象となるカラムには、すべてインデックスが張ってあるものとする)
仮定	検索条件により、製品、店、日付のそれぞれのテーブルから、1割のレコード (行) が選択 (ヒット率10%) され、最終的に、売上履歴テーブルからは、1000件 (ヒット率0.1% = 10% × 10% × 10%) が選ばれるものとする。

式：ペアワイズ・ジョイン

((売上履歴 ジョイン 製品) ジョイン 日付) ジョイン 店

10%のヒット率であるから、

製品 抽出 50行 よって 売上履歴 抽出 100,000行
 日付 抽出 30行 よって 売上履歴 抽出 10,000行
 店 抽出 20行 よって 売上履歴 抽出 1,000行

結果を得るまでに、売上履歴テーブルからだけでも、約100,000 + 10,000 + 1,000 = 111,000行もの抽出が必要。

式：クロス・ジョイン

売上利益 ジョイン (製品 クロス・ジョイン 日付 クロス・ジョイン 店)

上記同様10%のヒット率であるから、

製品 抽出 50行
 日付 抽出 30行
 店 抽出 20行

よって (製品 クロス・ジョイン 日付 クロス・ジョイン 店) 抽出 30,000行

(クロス・ジョインはすべての組み合わせを発生するため50 × 30 × 20 = 30,000行)

上記30,000行 ジョイン 売上履歴 抽出 1,000行

結果を得るまでに、新たに生成される行数だけでも、30,000行もの生成が必要。これに売上履歴からの最終生成を加えると、約30,000 + 1,000 = 31,000行もの抽出が必要。

式：スター・インデックスを使用したスター・ジョイン

製品 抽出 50行
 日付 抽出 30行
 店 抽出 20行

上記100行 スター・インデックスによるスター・ジョイン 売上履歴 抽出 1,000行

結果を得るまでに、1,100行のみの抽出で済む。

式6-1: スター・インデックスのジョインと従来型ジョイン 試算式

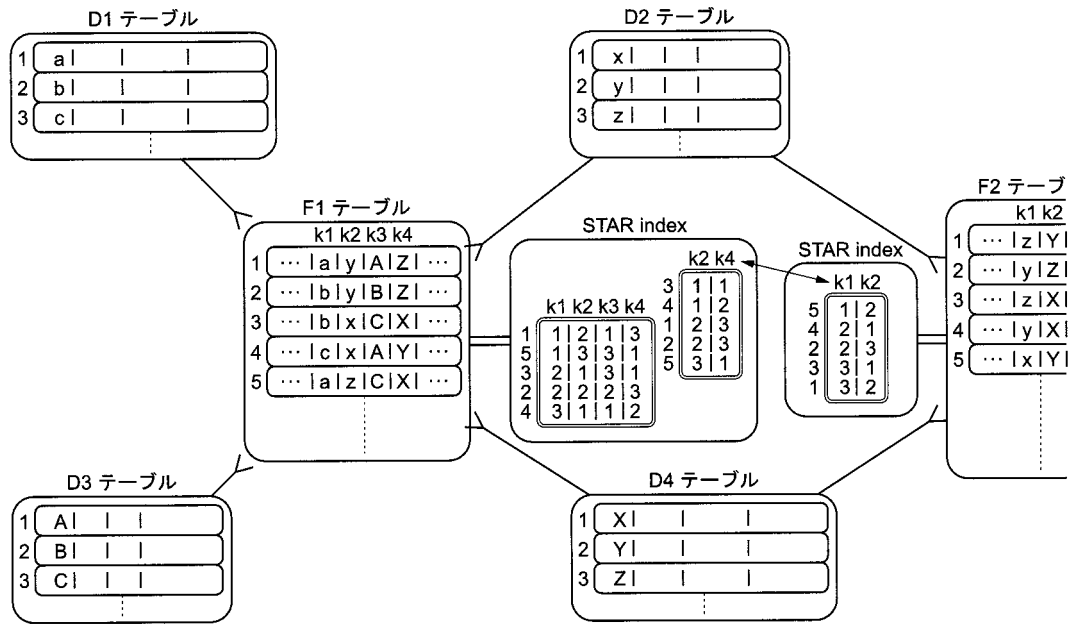


図6-1-3:スター・インデックス構造例

<6-1-3 スター・インデックスと単なる複合キー・インデックスとの違い>

スター・インデックスの構造は、図6-1-3にて、その概観を表してあります。ファクト・テーブル上のキーは、ディメンション・テーブルのキーと関係付けられます。この関係を、一般にRDBMSでは、ファクト・テーブル上に複数の外部キーを持つことによって表現し、これを複合キーと呼びます。他社DBMSは、この複合キーをそのまま単体に単体キーとみなしてインデックス化(一般にBツリー化)するだけです。つまり、ディメンション・テーブルD1,D2,D3,D4のそれぞれのテーブルのキーa1,b1,c1が $a1+b1+c1=X1$ となり、このようなXnのかたまりとして実装されます。このため、検索条件が、この外部キーの組み合わせに完全に一致しない限り、インデックスとしての性能は発揮されません。一般の複合任意検索では、上の例でいうテーブルBとCだけを使用するなど、さまざまな組み合わせが考えられるために実質的な性能向上は、これだけでは望めないこととなります。

<6-1-4 スター・インデックスの構造>

スター・インデックスは、この複合キーを単体に単体キーとしてインデックス化することをやめ、それを関係するディメンション・テーブルの論理行番号のマトリックスとして表します。複合キーの境目を認識できるマトリックスであるために、任意の外部キーの組み合わせに対しても柔軟に対応できることは当然ですが、特筆すべきは、論理行で成り立っている点です。これは、いわばスター・ジョイン処理の最終ステージである結合結果を、最初から持っていることに相当するのです。速いのは当たり前というわけです。

また、従来型ジョインでは、最初から巨大なファクト・テーブルとの突き合わせを避けるために、制約をかけたディメンション・テーブル群を先に掛け合わせる(直積をとる)などの方法をとりました。(式6-1のクロス・ジョインを参照)しかし、このクロス・ジョイン戦略は、選択され制約のかかったディメンション・テーブルの、どれか一つでも大きかったりすると、その組み合わせ(直積)の数が莫大になり、実用に耐えませんでした。しかし、スター・インデックスのマトリッ

クスの行は、ファクト・テーブル上の実績データの行に対応するわけですから、実際に意味のある組み合わせのみを有していることとなります。つまり、本当に必要な組み合わせだけが存在しているために、ディメンション・テーブルの大きさには左右されません。また、論理行で構成されている性質上、それらはバイナリーで表現されており、通常のインデックスに比べて大変小さいサイズになると同時に、その生成も他と比較して高速です。

<6-1-5 スター・インデックスと複雑なスキーマ>

スター・インデックスは、別のファクト・テーブル上のスター・インデックスとも突き合わせる事ができます。例えば、複雑なスキーマを代表するものとして、単純スター・スキーマが複数結合したようなマルチ・ファクト・テーブルを考えた場合、この利点が最大に活かされるでしょう。スター・インデックスは、このようなファクト to ファクトのジョインに対しても、最高のパフォーマンスを得ることができるのです。

<6-1-6 1つのファクト・テーブルに複数のスター・インデックス>

実は、スター・インデックスは、ファクト・テーブル上に定義されているすべての外部キーをスター・インデックス化する必要はありません。もし、ディメンション・テーブルの数が非常に多いにもかかわらず、よく検索する次元(ディメンション・テーブル)は少数に限定されているならば、その限定されている外部キーだけでスター・インデックスを作り出しておいてもいいのです。このように限定がユーザーごとに異なって存在するならば、それぞれ違った形のスター・インデックスを任意に作成することができます。つまり、1つのファクト・テーブル上に任意に複数の異なるスター・インデックスを作っておくことが可能です。ちなみに、ファクト to ファクトのような複雑ジョインの際には、この中で突き合わせに最適な組み合わせのスター・インデックスを自動的に選択するために、より高速な処理が行われます。

6-2 ターゲット・インデックス(TARGETindex)

<6-2-1 単純検索も高速に>

スター・インデックスは、ファクト・テーブルとディメンション・テーブルとのさまざまな関係に注目して作られました。しかし同時に、それぞれ単体としての検索に対しても注目をせざるを得ませんでした。それは、Red Brickにおいて実践のコンサルティングを続ける中で、巨大なディメンション・テーブルが存在することを知っていたからです。単体としての巨大テーブルに対する検索性能のために、Red Brickではビット・ベクトル・インデックスの技術を世界最高のレベルまで高めました。それが、Red Brickのみが持つターゲット・インデックス(TARGETindex)です(後に、このターゲット・インデックスを使用した、新しいジョイン・アルゴリズムであるターゲット・ジョイン(TARGETjoin)を完成させ、スノーフレークと呼ばれるような、くずれたスキーマに対しても世界最高の性能を出しています)

<6-2-2 ビットマップ・インデックスとBツリー・インデックスの適用領域>

従来のBツリー・インデックスが、その構造上もっとも苦手とするデータの集まりに対して、TARGETindexは最高の性能を發揮します。近年、他社DBMSも、このBツリーの穴を補うべく、ビット・ベクトル技術を使用したビットマップ・インデックスを搭載し、大幅な性能向上をうたっています。しかし、通常のビットマップ・インデックスでは、Bツリーの苦手とする領域をすべてカバーすることは不可能です。

<6-2-3 ターゲット・インデックスの適用領域>

Bツリー・インデックスが効果的なのは、例えば、あるテーブル中で、インデックス化の対象となるデータ値が、すべて異なるような場合です。つまり、そのテーブルの全レコード数が10,000件だった場合に、10,000件すべてが異なった値を持つようなとき、最も効果を發揮します。このような状況を一般に、カーディナリティーが大きいといいます(カーディナリティーとは、テーブルの1つの列内の一意な値の個数を、そのテーブルの全行数で割った値です)それでは、通常のビットマップ・インデックスが得意とする領域とはいうと、このカーディナリティーが極端に小さい場合に限られます。一般に一意な値の個数が2~100ぐらいまでが限度であり、これ以上はその構造上使用できません。そうすると、このビットマップ・インデックスとBツリー・インデックスの中間に位置する領域、つまり一意である値の総数が、約100~10,000ぐらいの間にあるようなテーブルに関しては、苦手とするBツリーを使用するか、またはインデックスそのものを付けないでおくしかありません。

ターゲット・インデックスなら、この領域を完全にカバーします。これを図示したものが、図6-2-3です。

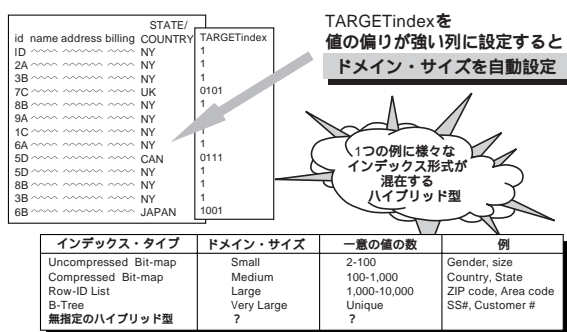


図6-2-3: ターゲット・インデックスのドメイン・サイズ

RDBMSでは、対象となるデータ領域をドメインと呼びます。Red Brickは、このドメインに対するカーディナリティーのサイズを、小・中・大と3種類に分け、それぞれに最適なインデックス構造を施しました。また、驚くことに、これら3種類の構造を一つの列の中で、その値の偏りに応じて自動的に割り付けるハイブリッド型も実現しました。このハイブリッド型によって、値の極端な偏りに対しても最小の資源で、最高の性能を發揮することが可能になったのです(例えば、一意の値の総数が10,000あり、そのうちの9,000種以上は、たかだか1行しか使われていないような場合です) また一般に、ユーザーはドメイン・サイズがあらかじめ予想できるような場合には、ターゲット・インデックスを作成する際に、このSMALL(小) MEDIUM(中) LARGE(大)を指定しますが、予想できない場合は、ドメイン・サイズを指定しなければいけません。何も指定しなければ、自動的にハイブリッド型が選ばれ、Red Brickが効果的にインデクシングを行います。

このターゲット・インデックスとBツリー・インデックスを組み合わせることで、それぞれの領域(ドメイン・サイズ)にとって最小の資源で、最高の性能を出すインデックスを生成でき、これによって巨大テーブルに対して、強い検索条件でも弱い検索条件でも、きわめて短いレスポンス・タイムをかえすことに成功しました。

6-3 ターゲット・ジョイン(TARGETjoin)

<6-3-1 スター・ジョイン(STARjoin)を補完するスター型新ジョイン・アルゴリズム>

Red Brickでは、一般にスター・ジョインとは、スター・インデックス(STARindex)を使用して行うスター型ジョインを示します。このため、ターゲット・インデックス(TARGETindex)を主体としたスター型ジョインを、ターゲット・インデックスと呼び区別しています。一般に他社DBMSのいうスター・ジョインは、Red Brickでいうスター・ジョイン(STARjoin)よりも、このターゲット・ジョインの方が、ほんの少しだけ近い位置にあるかもしれません。

ターゲット・ジョインは、その基本となるインデックスであるターゲット・インデックスの利点を、スター・スキーマ上でも最大限に活かすために生まれました。例えば、そのビット・ベクトル技術を活用し、非常に多くのディメンション・テーブルに大変緩い制約条件(緩い制約とは、条件に一致する行が多数あることをいいます)を課すようなときに効果的です。また、ハイブリッド型や、Bツリーまでも同時に使用できるために、データに偏りがあるような巨大ディメンション・テーブルに対して、大変強い制約条件(強い制約とは、条件に一致する行が少ないことをいいます)を課す場合などにも威力を發揮するでしょう。

<6-3-2 ターゲット・ジョインの設定>

ターゲット・インデックスのための準備は至極簡単です。スター・インデックス同様に、ファクト・テーブル上の外部キーに対してターゲット・インデックスを作成するだけです。ただし、1つのスター・インデックスが、その複数外部キーの任意のセット(組み合わせ)を指定するのに対し、1つのスター・インデックスは、1つの外部キーを指定します。つまり、現実的にはファクト・テーブル上の複数のターゲット・インデックスが集まって、ターゲット・ジョインは実行されます。

<6-3-3 Red Brickの動的ジョイン戦略>

ディメンション・テーブルが非常に多く、よく使用する次元が特定できないような場合、とりあえず、このターゲット・インデックスを張っておくことによって、ターゲット・ジョインが自動的に起動されるでしょう。また、同時にスター・インデックスが存在した場合などでも、やみくもにスター・インデックスを使用するのではなく、動的に検索のヒット率等を考慮しながら、最適と考えられるジョインを選択します(一般に他社DBMSでは、その最新のエンジンであっても、最初に最適化によって、SQLが書きかえられ、これ以後は戦略を全く変えずに実行されます)スノーフレイクとも呼べないほどの、くずれたスキーマに対しては最高の性能が出るのはこのためです。

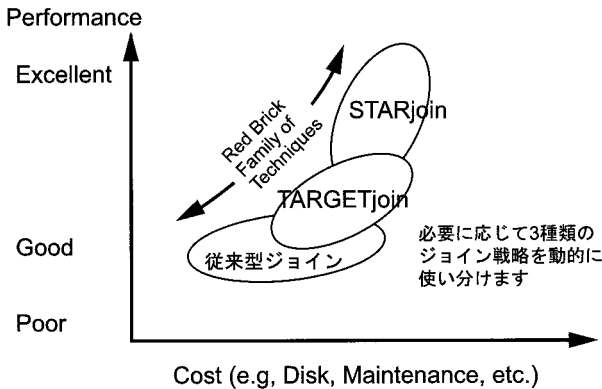


図6-3-3 : Red Brick Warehouseの3種類のジョイン関係表

<6-3-4 ディメンショナル・モデル再考>

しかし、ユーザーの観点からみて、より重要なことは、くずれたスキーマを作らないことです。このためには、ディメンション・テーブルにくっついているアウトボード・テーブルといわれる子テーブル群を、親のディメンション・テーブル内の属性におさめてしまうような非正規化が必要です。このような前向きな非正規化を現実のものとするには、その非正規化されたディメンション・テーブル中の複数属性の組み合わせに対しても、高速に検索が行われなくてはならないわけです。ラルフ・キンボール氏は「我々は、正規化されていない単体のディメンション・テーブル中の属性セットに対して、非常に高速にインタラクティブなブラウジングができなければならない。この高速性能を、特に、顧客リストが数百万行にも及ぶほどの大規模ディメンション・テーブルにまで可能とさせることが、大変重要なことである」と、その著書の中で述べました。現在のRed Brickのインデックス技術は、このラルフ・キンボール氏の理想を具現化している唯一のRDBMSエンジンです。

6-4 並列処理とセグメント化

<6-4-1 並列処理>

Red Brickは、複数プロセッサにのみ有効な並列処理だけでなく、作業を複数のプロセスに自動的に分割することによって、単一プロセッサ・システムにも並列処理を取り入れています。例えば、データの格納状況に応じて、アクセスを複数のプロセスに分割し、1つのプロセスがディスクの入出力を待機している間に、他のプロセスを処理し、検索を含むすべてのアクセスの効率と速度を上げるといったパイプライン・パラレル処理を行います。この中で、必要に応じて一般的なパーティション・パラレル処理も動的に発生させ、並列処理が必要な部分にだけ並列処理を施すといった動作を自動的に行います。

<6-4-2 ディメンショナル・セグメンテーション>

Red Brickでは、ディメンショナル・セグメンテーションと呼ぶ機能があります。これは、テーブルのデータやインデックスを、それぞれ任意の独立した物理格納ユニットに分散させ、データのロード中でも部分的にアクセスを可能とし、検索性能やインクリメンタル・バックアップおよびインクリメンタル・リストアの性能を上げるものです。とくに、テーブル単位ではなく、より小さな単位のセグメント・レベルでのロックができるために、オフ・ライン・ロードと検索処理を同時に行うことができます。

<6-4-3 行データとインデックス>

行データは、任意の列の値の範囲だけでなく、ハッシュによって複数のセグメントに分散することも可能です。インデックスは列の値によってセグメント化することを基本としますが、スター・インデックスは、各セグメントのエントリー数がほぼ同数になるように、均等分散させることもできます。

<6-4-4 スーパースキャン(Red Brick SuperScan™)>

Red Brickのディメンショナル・セグメンテーションを使用して、複数のドライブにデータを格納し、並列処理によって複数のディスクに同時にアクセスできるため、性能が大幅に向上します。また、1つのディスクにデータが格納されている場合でも、あるプロセスが読み込んだデータを他のプロセスが活用するRed Brick SuperScanにより、各プロセスのディスク・アクセスの遅延を短縮します。

6-5 有用な集約データ(IBM Red Brick Vista)

< 6-5-1 Red Brick Vistaの発表 >

この機能は、日本では1998年6月17日に記者発表され、7月1日に正式出荷開始したRed Brick Warehouse 5.1(日本語版)に、搭載された機能で、この機能に対する自信を、その概説書のなかで、こう述べています「...以上のような機能を持つRed Brick Vistaは、DWHにとって画期的な技術であると自負しています。なぜなら、これによって、Red Brickは、データキューブによるOLAPが苦手とするデータ・ロードおよびサマリー生成のためのロール・アップ処理を、数十倍から数百倍以上の速さでこなすことに加えて、従来以上の大容量性を維持しつつ、OLAPに匹敵する検索性能と、凌駕するメンテナンス性を実現できたからです。」

事実これが、RDBMSの上で行われたことは、まさに、生データや明細データといわれる基本データと、サマリーと呼ばれる集約データを、同時に扱うことのできる史上初のエンジン誕生なのかもしれません。

< 6-5-2 集約の重要性 >

意思決定支援のクエリーで頻繁に実行される処理には、月単位、および四半期単位の総売上、商品、または顧客単位の収益、あるいは、ほかの種類のグループ解析など、集約合計の計算があります。集約がないと、このようなクエリーでは、何十万、何百万の行を読み取り、結果を計算してグループ化する必要があります。クエリーが実行されている間、結果を待つことしかできません。

例えば、年ごとの、顧客別の売上合計が必要な場合、その答えを得るには、何億もの詳細行を読み取る必要があります。そのような大量の詳細行を、もっとも高速に読み取る方法は、一度にすべてを実行しないことです。この場合、最適な方法は、同じクエリーを、SUM、AVG、MIN、およびMAXなどの必要な集約情報を含む、サイズが大幅に小さいテーブルに対し実行し、読み取る行の個数を減らすことです。そのためには、集約テーブルに対してクエリーを実行することになります。集約を構築したら、ユーザーおよび開発者に集約テーブルが有効であることを通知し、処理を高速化することができます。

< 6-5-3 IBM Red Brick Vistaのソリューション >

「クエリーを今日実行した。処理が遅かった。DBAに連絡した。クエリーを次の日に実行した。今度は速かった。変更されたものは何もない。テーブルも追加されていないし、クエリーの実行方法も同じまま。例外処理のリストもない。でも、クエリー処理は高速になった。」

このような状況は、クエリーを実行する前にコスト・ベースの解析を行い、パフォーマンスを向上させるためにクエリーを捕捉し書き換える必要があるかを判断することで、達成可能となります。こうしたクエリーの捕捉および書き換えを、ユーザーに知られることなく実行する機能が、Red Brick、およびVistaの使用を強くお勧めする理由の一つです。

さらに、Vistaは、クエリー実行の統計を記録し、集約戦略の有効性、および改善方法を検討する材料を提供します。Vistaには、次の4つの機能が含まれておりGUI環境および、SQLにて以下の機能を実行できます。

1. 集約を定義し作成する方法
2. 集約テーブルの使用のため、クエリーを捕捉し書き換えるクエリー書き換えシステム。この処理は、ユーザー、またはアプリケーションによって発行されたクエリーに認識されることはありません。
3. 集約の使用を追跡し、新規の集約を推奨するAdvisorサブシステム、または、クエリーのログを記録したり解析する機能。この機能は、既存の集約テーブル、および作成すると役に立つと思われる新規の集約テーブルの、推奨サイズと利点を提供します。
4. 基本テーブルのロード時に、集約テーブルを更新しメンテナンスする、メンテナンス機能

本解説書の作成にあたり、以下の文献を参考にしました。
「Red Brickデータウェアハウス構築」株式会社シイエム・シイ R. ホカット 著
「ビジネス・インテリジェンス最前線」日経BP社 高千穂 彰 著

IBM, IBM Red Brick Warehouse, Red Brick, RISQL, STARindex, STARjoin, SuperScan, TARGETindex, TARGETjoin, Vistalは、IBM Corporationの商標。
Windowsは、Microsoft Corporationの米国およびその他の国における商標。
JavaおよびすべてのJava関連の商標およびロゴは、Sun Microsystems, Inc.の米国およびその他の国における商標または登録商標。
UNIXは、The Open Groupがライセンスしている米国およびその他の国における登録商標。
他の会社名、製品名、サービス名等は、それぞれ各社の商標または登録商標。



日本アイ・ビー・エム株式会社
〒106-8711 東京都港区六本木3-2-12
02-02 Printed in Japan

仕様は事前の予告なしに変更することがあります。 製品、サービスなどの詳細については、弊社もしくはIBMビジネス・パートナーの営業担当員にご相談ください。

