Zap-Over: インターネット上に分散したビッグデータのSVL によるブラウジング手法

古庄 晋二† 飯沢 篤志‡

†株式会社ターボデータラボラトリー 〒231-0004 横浜市中区元浜町 3-21-2
‡リコーIT ソリューションズ株式会社 〒104-6042 東京都中央区晴海 1-8-10
E-mail: †s-furusho@turbo-data.co.jp, ‡atsushi.iizawa@jp.ricoh.com

あらまし インターネット上に分散する、オーナが異なったりスキーマや単位などが異なったりする、様々なビッグデータ（およびオープンデータ）を、手元で自由に組み合わせてブラウズや検索を行うための仕組み；Zap-Over を提案する。Zap-Over によって、ビッグデータの用途が大きく広がり、公開が容易になる。Zap-Over ではデータ提供側とデータ利用側で以下を行う。1）データ提供側は、元テーブルを Zap-Over テーブル表現(Zap-Over Table Representation: Zap-OverTR) に変換し、ファイルとしてインターネットに公開する。2）データ利用側は、インターネット上にある上記ファイルを任意に選択して仮想的にユニオンして 1 つのビッグデータとし、これをレコード順および任意の項目の値順のビューとしてブラウズし、検索する。

この Zap-Over は、テーブルのカラム内の値を取り出して重複を省きソートした Sorted Value List (SVL)というデータ構造を中心に構築されている。

キーワード インターネット，分散，ビッグデータ、オープンデータ

1. はじめに

最近「ビッグデータ」と呼ばれるデータが多くなり、様々な分野で活用されるようになってきた。ビッグデータには、気象観測データ、トラフィックデータ、その他 IoT データ、POS データ、流通データ、生産管理データ、通信・通話のログ、などがある。ビッグデータはそのサイズゆえ転送が困難であるため、転送せずにそれぞれの発生源で保持しておく場合が多く、その結果世界各地にビッグデータが分散することになる。これらのビッグデータはオーナが異なったり、スキーマや単位が異なったりすることが想定される。

一方、利用者側は世界各地に分散した様々なビッグデータをその都度、自由に組み合わせて使用したいと思っている。

本論文では、第 2 章で Zap-Over の先行技術である Zap-In を概説し、第 3 章で提案手法である Zap-Over を紹介する。第 4 章で関連技術・関連研究を述べ、第 5 章でまとめる。

2. 既存技術(Zap-In)

2-1. Zap-In の開発目的と課題

Zap-In の開発目的は、通常のスタンドアロン型のデータベースの処理を高速にすることである。

通常の RDB システムは、処理を高速化するのにインデックス技術を用いている。データベースのカラムのうちのいくつかに、ハッシュテーブルやＢツリー等のインデックスを構築し、そのカラムの検索処理などを高速化できる。このインデックスによる高速化なしには効率的なデータベースは構築できないのが現状である。特にデータ量が大きくなると、インデックスなしでは効率低下は顕著になる。また、インデックスをつけるカラムをどれにするかが、システム全体の高速性のカギになる。

このインデックス技術には、以下の問題点がある。

1. インデックスが付いていないカラムでは、処理が高速にならない。
2. インデックスを付けるカラムは多くできない。（通常は５～６個以下）
3. どのカラムにインデックスを付けるかはデータベースシステムの設計時に決めるので、設計時に想定していなかった使い方が要求された場合には、別なカラムの高速化が必要になり設計し直しになってしまう。
4. インデックスは処理前に予め付けるものなので、あるデータベース処理の結果として得られるテーブルにはインデックスが付いていない。このため、新テーブルを利用する次のデータベース処理は高速にならない。
5. インデックス用の記憶領域がかなり必要である。

Zap-In の開発では、従来の RDB の機能を変えずに、このインデックス技術に代わるデータベース処理高速化技術を確立することを課題とした。

2-2. Zap-In の概要

上記の目的のため、Zap-In では以下のようになっている。

1. リレーショナル演算ができる
2. インメモリ型データベースである。
3. カラム指向データベース(Columnar Database)である。
4. 全てのデータベース処理に先立って、Zap-In テーブル表現 (Zap-In Table Representation: Zap-InTR) に変換しておく。Zap-InTR は Sorted Value List (SVL) を基本とするものである。
5. 通常の意味のインデックスあるいはインデックス記憶領域はない。
6. 非手続き型言語である SQL ではなく、手続き型言語である Python を採用する。

Zap-In テーブル表現（Zap-InTR）とそれに合わせたアルゴリズムによって以下のように、今までのインデックス技術よりもはるかに優れた高速性が得られた。

1. 従来の RDB に比べてはるかに高速。
2. 検索（値検索・範囲検索）・ソート・ユニオン・ジョイン・集計が高速化される。
3. 通常のインデックス技術では効果がないユニオンでも有効。

さらに、以下の特長が得られた。

1. SVL で実データの重複をなくし、しかも整数化しているので、全体のメモリ使用量が削減される。これは、特にインメモリ型では大きなメリットになり、ビッグデータをメインメモリで操作できることになる。
2. データベース処理結果で得られるテーブルも Zap-InTR であるため、このテーブルを利用する次のデータベース処理も高速。（2-8 参照）
3. このため、多段階のデータベース処理が必要なシステムでは、既存 RDB よりもはるかに高速。

この多段階処理の高速性を活かした、特に有用な用途として以下がある。

1. 多段階の処理を行うバッチ処理　（BOM 展開(Development of Bill Of Materials) や Material Requirements Planning(MRP)など）
   これらは、最終結果までに多数の中間テーブルを作り出す。中間テーブルも Zap-InCDR なのでそれに対する処理も高速である。
2. 対話型処理（分析、クレンジングなど）
   対話型分析では、処理結果のテーブルを見てから次の処理を決め、その結果テーブルを処理して次のステップに進むことを繰り返す。そのため多段階の処理になる。クレンジングも同様。
3. データ移行
   データ移行では中間テーブルを作成する場合が多く、多段階の処理になる。

また、スプレッドシート GUI の操作を記録し Python コードに変換できるので、システム開発が高効率になっている。

2-3. Zap-In の性能評価

富士通で行われた Supply Chain Management(SCM) 処理におけるベンチマークで、多段階の処理である BOM 展開(Development of Bill Of Materials)、ジョインのパフォーマンスが重要である MRP(Material Requirement Planning)においては、PL/SQL を用いた PC サーバ上のシステムに対し、Zap-In を用いたノート PC 上のシステムによる処理が 500～700 倍のパフォーマンスを記録した。

2-4. Zap-In テーブル表現 (Zap-InTR)

Zap-In では、元テーブルを最初に Zap-InTR へ変換してから、さまざまなデータベース処理を行う。

図 1 に、あるテーブルの従来の一般的なデータ構造（図中の上側）と、それを Zap-InTR にしたもの（図中の下側）を示す。元テーブルの２つのカラム”Name”と”Age”は、２つの別の Zap-In カラムデータ表現(Zap-In Column Data Representation: ZAP-InCDR) に分かれて表現されている。各 Zap-InCDR は VCD と SVL から成る。これらの Zap-InCDR とは別に１つの RNL が存在する。

A) SVL (Sorted Value List):

各カラムに現れる「値」をユニーク・昇順に格納した配列である。以下の機能を担う。

1. カラムに現れる値を、値の大きさの順番にあたる整数に変換する。(これによりカウンティングソートが使えるのでソート時間が O(n)になる)
2. 集計の際の次元を定義する。（次元の値、サイズが即座にわかる）
3. ジョイン・マッチングのキーを定義する。

B) VCD (Value Number Oriented Column Data):

値番号化カラムデータ VCD は、SVL を使って元カラムを変換したものである。レコードで指定される元の値が SVL の何番目の値であるかが入っている。

C) RNL (Record Number List):

データベース演算の処理結果として得られるカラムの情報を入れる配列である。

RNL は、カラムの情報で、以下をレコード番号で保持する。

1. 処理対象レコードはどれか、
2. 処理対象レコードの順序

処理結果の RNL はまた、次のデータベース処理の入力として使われる（後述）。

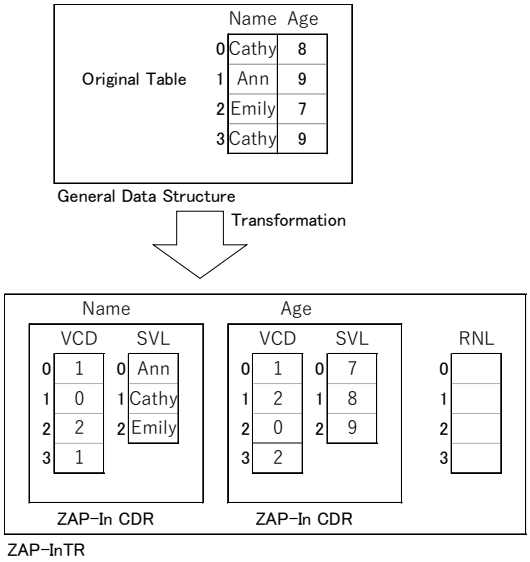


図 1. Zap-In テーブル表現 (Zap-InTR)

Zap-In CDR から元のテーブルのあるレコードを取得するには、各カラムについて

<1> まず、取得したいレコード位置に該当する VCD の要素を読み出す。

<2> 次にその VCD の要素が指し示す SVL の要素を取得する。

という様にすればよい。これを各カラムについて行う。

2-5. Zap-In の値サーチのアルゴリズム

図 2 に Zap-In での値サーチアルゴリズムを示す。例として、検索条件 “Age” = 9 の値サーチで説明する。

<1> まず、検索条件(“Age” = 9) の値 9 で SVL を逆引きし、2 を得る。
SVL はソートされているので、逆引きは O(log(n))で高速に実行できる。

<2> 次に、得た 2 で VCD を逆引き（上から順に）し、得たレコード番号（複数）を検索結果書き込み用の RNL へ上から順に書き込んでゆく。
これで RNL に 1 と 3 が書き込まれ、これが検索結果になる。つまりレコード 1 とレコード 3 の”Age” が 9 である。

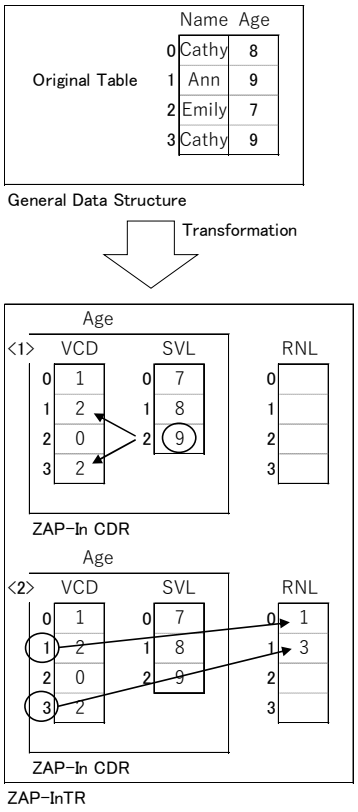


図 2. 値サーチのアルゴリズム

2-6. Zap-In の範囲サーチのアルゴリズム

図 3 に Zap-In での範囲サーチアルゴリズムを示す。

複数の値をサーチすれば良いので基本は値サーチと同じだが、1 回の工程でその複数の値のサーチができるように一時的作業領域としてフラグ配列を用意して利用する。

例として、検索条件“Age”>7 の値サーチで説明する。

<1> まず、SVL と同じサイズのフラグ配列（SVL-Flag）を用意し、0 で初期化しておく。

<2> 次に、検索条件に合致する SVL 要素をリストアップするために、それに対応する SVL-Flag の要素を 1 にする。検索条件(“Age”> 7) に該当する SVL 上の値は連続区間になっている（SVL はソート済みだから）から、SVL の全ての要素をチェックする必要はなく、区間の上端と下端を特定すれば良い。このためこのフラグのセットは高速に行える。また、検索条件が値の範囲ではなく逐次値だとしても SVL はソート済みなので、高速に行える。

<3> VCD の一番上の要素から順番に、VCD→SVL-Flag とたどって、SVL-Flag が 1 であればそのレコード番号を検索結果書き込み用の RNL へ上から順に書き込んでゆく。

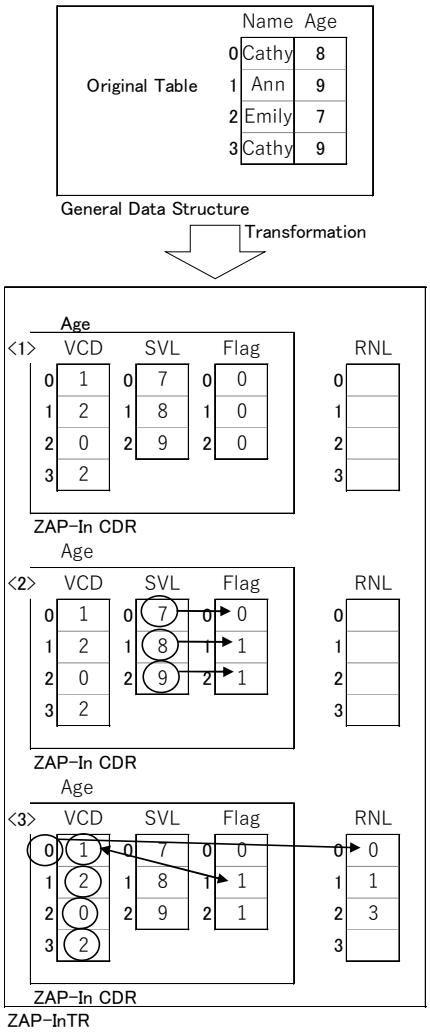


図 3. 範囲サーチのアルゴリズム

以上の<1><2><3>の処理は単純であり、特にインメモリのシステムでは高速に進めることができる。また、RNL の一番上の要素から順番に行うことで、レコードの順序が検索結果でも保たれるという利点がある。Zap-In では、サーチ対象の件数のみに依存し、ヒット件数には依存しない。だから、ヒット件数が多い場合にも有効である。

2-7. Zap-In のその他の処理のアルゴリズムの概要

サーチ以外のデータベース処理（ソート・ユニオン・ジョイン・集計・集合演算・など）についても、Zap-InTR を有効に活用したアルゴリズムが開発されている。[1]

2-8. RNL を用いた処理のカスケード（多段処理）

RNL について説明を加える。

各データベース処理の結果は RNL に置かれたレコード番号として得られる。例えばサーチの結果は、合致するレコード（複数）のレコード番号が RNL に入っており、ソート処理の結果は、全レコード番号が RNL にソート結果の順に入っている。

この RNL を通して各 Zap-InCDR を見ると処理結果の各レコードの値が得られる（図 3）。この時、元の各 Zap-InCDR の方はまったく変化していない。言い換えれば、元の各 Zap-InCDR は変化せずに存在し続け、データベース処理をするごとに得られる結果テーブルの情報が生成された RNL に入っている。

この結果新テーブルに対して次のデータベース処理をすることを考えてみよう。まず始めに RNL をたどる工程を一段行ってから Zap-InCDR へアクセスすれば、あとのデータ構造は Zap-InCDR そのままである。その後は今まで述べてきた Zap-In データベースアルゴリズムをそのまま使う事ができる。つまり、処理結果もまた Zap-InCDR に基づく Zap-InTR であると言え、それを使った次の処理も同じく超高速で行うことができる。このような多段処理の高速化は、インデックスでは実現できない。

なお、RNL として各要素にそのレコード番号をそのまま入れる（RNL[i]=i ）のような初期化をしたものを用意すれば、その RNL が保持するテーブル情報は元テーブルを示すことになる。つまり、まず RNL をたどる工程を一段行ってから Zap-InCDR へアクセスというやり方を、最初の処理でも同じ形で行うことができ、処理の統一化ができることになる。

3. 提案手法(Zap-Over)

3-1. Zap-Over の開発目的と課題

Zap-Over の開発目的は、以下の２つである。

1. 遠隔地にあるデータベースを、ネットワーク経由で高速にブラウズと検索ができること
2. 別々の遠隔地にある複数のデータベースを、組み合わせてブラウズと検索ができること

従来技術では、以下のような問題点がある。

1. 別々の遠隔地にある複数のデータベースを組み合わせてデータベース処理をするためには、まず１箇所にデータを集めて統合してから処理することになる。データが巨大な場合には、データを集める（転送あるいは空輸）にも統合するにも長い時間とコストがかかる。ビッグデータの全部を引き渡すので対価も大きくなる。
2. データベース側にある DBMS に、クライアント端末からリクエストを送って処理をしてもらうシステムになるので、同時多数アクセスがあると DBMS に負荷が集中してしまう。

そこで、以下のようなシステムが求められる。

1. 遠隔地にあるデータベースに対して、サーチ・ソートなどのブラウジングができる。
2. 別々の遠隔地にある複数のデータベースをユニオンやジョインでき、その結果に対してブラウジングができる。
3. データベース側の処理が軽く、処理は主に端末側で行われる。そうすれば、同時多数アクセスがあっても負荷が集中しない。

3-2 Zap-Over の概要

上記の目的を実現するため、Zap-Over では以下のように実現している。

1. ブラウジングのための実処理は、その時の画面に表示するのに必要充分なだけのデータ処理を行い、その処理に必要なだけのデータのみをフェッチしてくる。画面がスクロールされると、新たな表示に必要充分なだけの処理とデータフェッチを行う。(3-6.参照)
2. 以上が効率良く高速に実行できるように、元テーブルをあらかじめ Zap-Over テーブル表現 (Zap-Over Table Representation: Zap-OverTR) に変換してサーバ側に置いておく。
3. Zap-OverTR は Sorted Value List (SVL) を基本とするもので、Zap-InTR の拡張である。(3-3.参照)
4. テーブルの読み取りのみでき、更新はできない。

Zap-OverTR とそれに合わせたアルゴリズムによって、上記の機能に加えて以下のような特長も得られた。

1. テラバイト級のビッグデータが取り扱える。
2. SVL 採用により処理は超高速。
3. 仮想的にユニオンした結果に対して、レコード順および任意のカラムの値順にブラウズできる。
4. データとしてテキストを入れれば全文検索システムができ、しかも超高速である。

特に有用な用途として以下がある。

1. **オープンデータ**: 世界の各機関がデータを公開するのに Zap-Over を採用すれば、以下の利点がある。
   A) 複数のオープンデータを組み合わせての利用ができる。
   B) 利用に必要な分だけのデータを転送するので、有料の場合でも利用者側への料金請求を従量課金にできる。
   C) サーバ側はファイルサーバでよく、設備投資が小さく、維持管理も簡単である。
2. **国際企業の支社間のデータ利用**: 各地の支社に分散して管理されているデータを、「仮想的に統合して」しかも「世界各地から」データブラウジングでき、データの利用効率が上がり、管理コストを削減できる。
3. **IoT データ**: 現状のシステムでは、各地に分散配置された IoT 端末（あるいは POS レジ、自動販売機、など）のデータを、全て１ヶ所に集めて管理し処理する。Zap-Over を採用すれば、データを IoT 端末に置いたままで必要な時に Zap-Over 端末からアクセスして、データをブラウジングできる。
4. **データセンター内での利用**: データが地理的には分散していない同一データセンター内であっても、多数の独立したデータを必要に応じて自由に組み合わせてデータブラウジングしたい用途に利用できる。(3-7.参照)

3-3. Zap-Over テーブル表現 (Zap-OverTR)

図 4 に Zap-OverTR を示す。

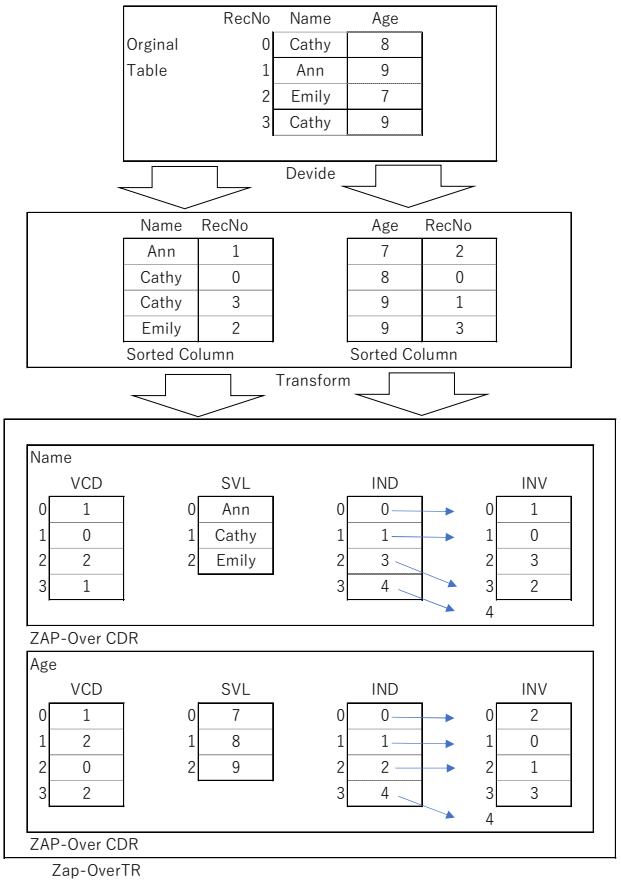


図 4. Zap-Over テーブル表現(Zap-Over TR)

・上段は元テーブルの従来の一般的なデータ構造である。
・中段はそれをカラム毎に分割しソートした形式である。RecNo は元のレコード番号である。
・下段は Zap-OverTR にしたものである。

元テーブルの２つのカラム "Name" と "Age" は、２つの別の Zap-Over カラムデータ表現(Zap-Over Column Data Representation: Zap-OverCDR) に分かれて表現されている。各 Zap-OverCDR は以下から成る。

1. **VCD (Value Number Oriented Column Data)**: Zap-In の VCD と同じ。
2. **SVL (Sorted Value List)**: Zap-In の SVL と同じ。
3. **INV** 転置レコード索引(Inverted Record Index) : 中段の RecNo と同じ。
4. **IND** 間接レコード索引(Indirect Record Index) : SVL の各要素が INV のどの範囲に存在しているかの情報を持つ。SVL の i 番目の要素は、INV[IND[i]]～INV[IND[i+1]-1]に存在する。IND は SVL よりも１つだけ大きいサイズの配列である。

3-5. 複数テーブルをユニオンした結果をレコード順に表示するアルゴリズム

図 5 に２つのテーブルを仮想ユニオンした結果を、そのままレコード順に表示する場合を示す。

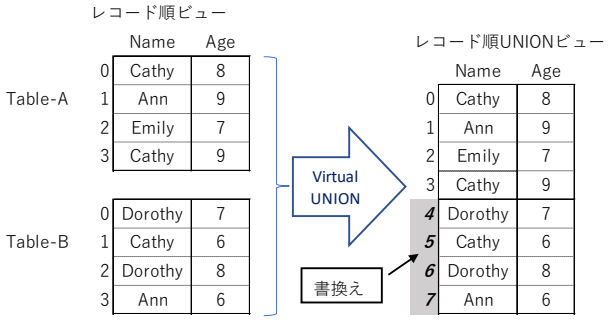


図 5. 複数テーブルのレコード順表示

ユニオンの２つめ以降のテーブルのレコードが表示される時には、その元々のレコード番号にそれより前方にあるテーブルのレコード数の合計を加算して書き替えればよい。これを使って、画面に表示する分がどのテーブルのどのレコードかが容易に算出できるので、そのレコードだけを端末からサーバへ要求すればよい。

3-6. 複数テーブルをユニオンした結果を値順に表示するアルゴリズム

図 6 に、２つのテーブルを仮想ユニオンした結果を、”Name”の値順にソートしたものを表示する場合の画面表示イメージを示す。

	"Name"	"Age"	INV
0	Ann	9	1
1	**Ann**	**6**	**7**
2	Cathy	8	0
3	Cathy	9	3
4	**Cathy**	**6**	**5**
5	**Dorothy**	**7**	**4**
6	**Dorothy**	**8**	**6**
7	Emily	7	2

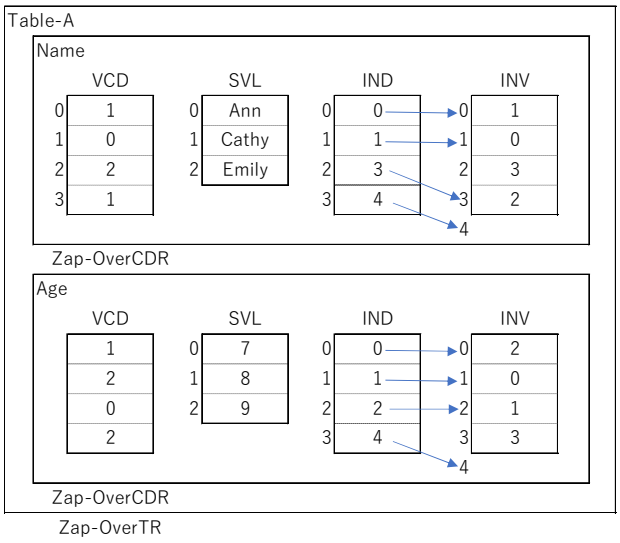
図 6. 複数テーブルの値順表示イメージ
（レコード番号が書き替えられた部分を太字で表示）

この値順ビューでの表示順番を利用者に好ましいものにするために、”Name”の値を第 1 のソートキー、ユニオンした際のテーブルの番号を第 2 のソートキー、テーブル内でのレコード番号を第 3 のソートキーとしてソートされたものにしたい。

図 7 はその際の Zap-OverTR におけるレコード番号の書き替えを表している。あるテーブルのレコード番号は、それより前方のテーブルのレコード数の合計を加算して書き替えられている。

たとえば、複数のテーブルを仮想ユニオンしソートした結果の 10000 行目から 10050 行目を画面に表示したい場合を考える。これは以下のようにすれば良い。

<1> 最初に仮想ユニオン処理で各 Zap-OverCDR の INV の書き替えが決まる。
<2> ソートキーの Zap-OverCDR の SVL から適当な ”値” を決める。（ソートキーの Zap-OverCDR は各テーブルごとに存在するので適宜マージしながら）
<3> それが何行目～何行目に当たるかを各テーブルの Zap-OverCDR で SVL→IND→INV とたどって調べる。
<4> この行範囲が目的の表示行を含むようになるまで、<2><3>を試行錯誤（２分法など）する。これで目的の表示行が、どのテーブルのどの値かがわかる。
<5> あとは INV を詳しく調べれば、目的の表示行に出すべきレコードの番号がわかる。



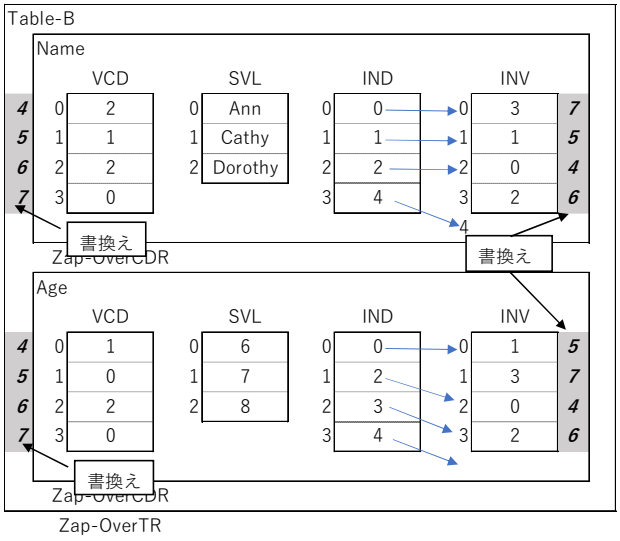


図 7. 複数テーブルの値順表示のための内部構造

以上のようにすれば、表示のためにどのテーブルのどのレコードをフェッチすれば良いのかがわかる。この処理は軽いので、画面スクロールして表示範囲を変えても十分な速さで追従できる。

3-7. 東京国税局でのパフォーマンス改善

東京国税局では、世界百数十カ国と海外送受金に関するデータを毎月交換し、マネーロンダリングを追跡している。

以前は全てのデータを RDB に格納していた。各国からの 100 以上のデータベースを統合するのに長時間を要し、その結果のデータは肥大化していた。また、項目数が数百にもなるため、全ての項目にインデックスを付けることができず、インデックスの付けられていない項目を使うことが頻発し、検索のパフォーマンスは低いものであった。1 回の検索は概ね 15 分から 20 分を要し、同時に検索を行える人数も 1 名から 2 名であった。

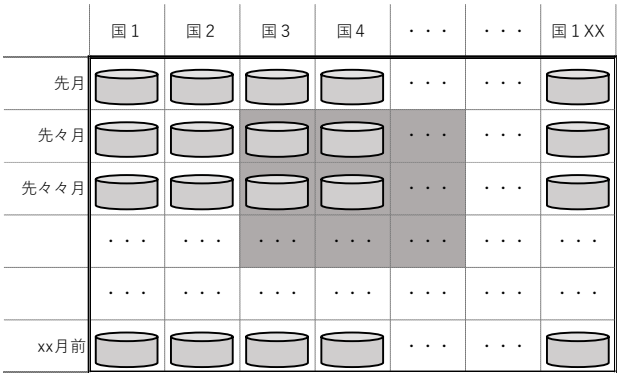


図 8. 典型的なビッグデータファイルの持ち方

Zap-Over の導入で図 8 に示すシステムにした。ビッグデータファイルが月ごと、国ごとに分割されたままで Zap-OverTR で保存されており、必要な月・国を選んで、該当のビッグデータファイルをユニオンして仮想的なビッグデータを生成し、検索を行う。この Zap-Over 利用システムでは、1 回の検索は概ね 10 秒未満（約 100 倍高速化）、同時に検索をかけられるユーザ数も 20 名程度になった。（約 10 倍の人数）

4. 関連技術・関連研究

4.1 Zap-In の関連技術・関連研究

カラム指向データベースでは SVL を導入している例が多い。代表的なカラム指向データベースを取り上げ、Zap-In との差異をまとめる。

Sybase-IQ[2]はインメモリではなくディスクベースのシステムである。ランダムアクセスできないので性能低下を補うために様々なインデックス(9 種類)が搭載されている。またカラム指向ではカラムの中間に新データを挿入するのに時間がかかるため、新データの中間挿入は禁止し、末尾に追加のみに制限している。

C-Store[4]、Vertica[5]もディスクベースである。大規模データ対応のためシェアドナッシング方式の超並列方式を導入しているが、このため機能的な制約が大きく、またシングルタスクとなりジョブがキューイングされるため多数のユーザの同時使用はしにくい。また、上と同じく新データは末尾に追加のみに制限している。

SAP-HANA[6][7]はインメモリであり、Zap-In の特許をライセンスしている。原則的としてインデックスは使わない。インメモリを活かして新データの中間挿入ができるが、1 つのカラムを多数の小ブロックに分割して格納することで、中間挿入の速度を上げて OLTP

に対応している。しかしこのために、ソート他のデータベース処理では速度が低下している。

Zap-In はインメモリでありインデックスは使わない。HANA の様な中間挿入高速化は行わず OLTP 性能は上がっていないが、データ構造がシンプルなためその他の処理では非常に高速である。

なお、呼称が類似している「分散ソート済みカラム向データベース」と呼ばれるものがあり、その代表例として BigQuery[3] がある。BigQuery でソート済みなのはノードに割り付けられたリージョンとリージョン内のレコードであり、各項目全体に渡ってソートされているわけではない。また SVL は存在しない。

4.2 Zap-Over の関連技術・関連研究

Remote Table Access(RTA)[8]は、世界各地に分散する RDB 形式のオープンデータに SQL でアクセスし、利用者の手元の RDB と組み合わせてデータベース処理ができるようにするものである。データを公開するためには、そのスキーマ等を RTA Library に登録公開しておく。クライアントシステムは利用者からのクエリを分解して、必要なリモート RDB に対して必要な問い合わせを行い、ローカル RDB と合わせて処理結果を得る。ネット上のデータ転送量を少なくするためには、クエリの分解を工夫してそのように最適化することで対応する。

これに対して Zap-Over では、各サーバのデータを Zap-OverTR 形式に変換することで、リモートデータへのアクセスは SQL を用いずにファイルアクセスで実現でき、データ転送量が最小限になっている。

5. まとめ

本論文では、世界各地に分散する、オーナが異なったり、スキーマや単位などが異なったりする様々なビッグデータを、インターネットを経由して端末側から複数を組み合わせて自由に検索・ブラウズできるようにする仕組みとして、Zap-Over を提案した。Zap-Over は、データの値を並び替えた SVL というデータ構造を中心に構築されていて、分散したビッグデータを低コストで利便性よく活用できる。

今後の課題は以下である。

1. Zap-Over で収集したデータを Zap-In で編集・加工・分析できるために、Zap-In から Zap-Over データを利用可能にすること。
2. メタ情報を管理し、単位などが異なるビッグデータファイルの項目を矛盾無くユニオンできるようにする機能を開発すること。
3. 検索結果集合間の論理演算機能の利便性を向上させること。
4. データのエクスポートに関するパフォーマンスと機能を向上させること。

参 考 文 献

[1] 古庄晋二,“汎用超高速データベース処理技術：多様なデータ構造と超並列処理への普遍的アプローチ- RealTimeVIDP”, 東大総研 2005.

[2] ホワイトペーパー, "クラウドにも対応！DWH に最適化された最高の ROI を実現するデータベース Sybase IQ" https://wp.techtarget.itmedia.co.jp/contents/12605

[3] "Google BigQuery のドキュメント" https://cloud.google.com/bigquery/docs/?hl=ja

[4] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, Stan Zdonik, "C-Store: A Column-oriented DBMS", Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005. http://db.csail.mit.edu/projects/cstore/vldb.pdf

[5] “Vertica とは”, Vertica 技術サイト, 2015. http://vertica-tech.ashisuto.co.jp/about-vertica/

[6] "[図説]インメモリーコンピューティング SAP HANA®のテクノロジー解説: 真のリアルタイム経営を支えるリアルタイムデータプラットフォームの実力とは", 2015. https://www.intel.co.jp/content/dam/www/public/ijkk/jp/ja/documents/white-papers/xeon-e7-in-memory-computing-sap-hana-technology-paper.pdf

[7] 三原健一, "HANA はどうやって行を識別しているのか", Oracle 技術者から見た、SAP HANA, 2017. https://enterprisezine.jp/dbonline/detail/10198

[8] 村上柊, 小坂祐介, 五嶋研人, 遠山元道, "RTA:公開型テーブルへの直接問い合わせ機構の提案", DEIM 2017. http://rta-keio.net/papers/DEIM2017_RTA.pdf