# Actian Zen v15 の
# パフォーマンス改善について


株式会社エージーテック

2022 年 9 月 30 日

## 免責事項

株式会社エージーテックは本書の使用を、利用者またはその会社に対して「現状のまま」でのみ許諾するものです。株式会社エージーテックは、いかなる場合にも本書に記載された内容に関するその他の一切の保証を、明示的にも黙示的にも行いません。本書の内容は予告なく変更される場合があります。

## 商標


すべての Pervasive ブランド名および製品名は、Pervasive Software Inc. の米国およびその他の国における登録商標または商標です。また、すべての Actian のブランド名は、Actian Corporation の米国およびその他の国における登録商標または商標です。
文中の社名、商品名等は各社の商標または登録商標である場合があります。

Actian Zen v15 のパフォーマンス改善について
最終更新：2022 年 9 月 30 日

目次


はじめに 4

Actian Zen パフォーマンス改善の概要 5

ファイル I/O の効率化 5

ファイルのリビルド 5

キャッシュのサイズ拡大 6

ページサイズの最大化による I/O 効率向上 8

データ圧縮による I/O 効率向上 8

ファイル I/O の分散によるパフォーマンス向上 8

ファイルをあらかじめ開いておく 9

インメモリでの使用 １０

クライアント／サーバー構成での改善 １１

ネットワークドライバーの設定 １１

Btrieve API 使用時のパフォーマンス改善 １２

ODBC 接続での効率化 １４

その他留意点 １５

# 1.はじめに

AGBP 技術資料として PSQL v12 までの情報を提供しておりますが、多くのお客様がダウンロードされており、パフォーマンスの改善に関する情報を求められているお客様が多数お見えになると感じております。

本書では、以前提供していた内容も含め、Actian Zen の新機能を含めた最新の情報でパフォーマンス改善のポイントをご案内いたします。

# 2. Actian Zen パフォーマンス改善の概要

パフォーマンスを改善したい場合は、どこがボトルネックになっているかを調べる必要があります。

・ネットワークの帯域幅および使用率
・アプリケーションでのコーディング技法
・使用可能なメモリ（空きメモリやキャッシュのサイズ、キャッシュ利用効率）
・ハードウェア性能
・アプリケーションでの使用パターン（大量の書き込み、大量の読み込み、トランザクションの使用/不使用、レコード サイズの大小、複雑または単純なクエリ、ODBC、Btrieve API のみ、など）
・CPU サイクルで競合する他のソフトウェア
・データベース エンジンの設定
・データファイルの状態
・同時にサーバーを使用するクライアント数
.... etc.
上記は一例になりますが、複数の現象が重なって遅延に繋がる事もあり、現象に合わせた対処が必要になります。また、上記の通り、Actian Zen を取り巻く環境が原因になるケースも多いです。
本書では、Actian Zen 側でパフォーマンスを改善できる方法を中心にいくつかご紹介します。

# 3. ファイル I/O の効率化

## ◆ファイルのリビルド

長年にわたり追加・削除を繰り返すと、ファイル内に未使用領域が発生したり、インデックスの偏りが発生する場合があります。Defragmenter または Rebuild を行うと、未使用領域が詰められる他、インデックスの再配置が行われるため速度の向上が期待できます。
Defragmenter では、運用を行いながらファイルの再構成が可能です。

## ◆Defragmenter

「Defragmenter」は、データの断片化を検出して修正することが可能なツールです。データファイル内のレコードやインデックスを配置し直したり、未使用領域を除去したりして、再びデータへ効率良くアクセスできるようにします。

「Rebuild」ツールを使ってデータベースの最適化を図ることも可能ですが、業務を停止する必要があります。しかし、「Defragmenter」では、業務を停止する必要がなく、データベースエンジンの実行中にこの機能を使用することができます。

従いまして、大きなファイルサイズのデータベースを最適化する際も、要する時間を考慮する必要がなく、業務に支障をきたすことがありません。

＜制限事項＞

・最適化中にデータベースおよびスキーマ定義を変更することはできません。

・最適化を実行中のファイルに対して、データバックアップを目的とする Backup Agent 操作や Continuous オペレーションを実行することはできません。

・クライアント キャッシュ エンジンを使用しており、サーバー上で既にファイルを開いている場合は、サーバーの最適化が実行できなくなり、最適化の要求でエラーが返されます。最適化できるようにするには、サーバーを再起動してクライアント接続をクリアする必要があります。

・最適化は現在、バックアップ操作に Microsoft のボリューム シャドウ コピー サービス（VSS）を使用する環境にあるサーバー エンジンについてはサポートしません。

・システムデータ v2 を使用するファイルの最適化は行えません。

## ◆自動最適化

「Defragmenter」の設定を行わなくとも、自動的に最適化を行うことが可能です。

エンジンのプロパティで［パフォーマンスチューニング］→［自動最適化］をオンに設定します。

この設定を行うことで、エンジンが定期的に断片化の監視を行い開いているファイルで、次の条件に該当しないファイルを順次最適化します。

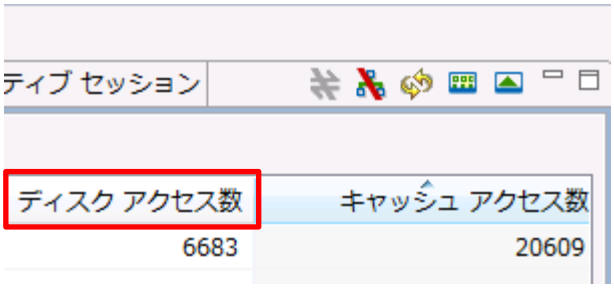
·24 時間以内に最適化されている

·サイズが 10 MB 未満

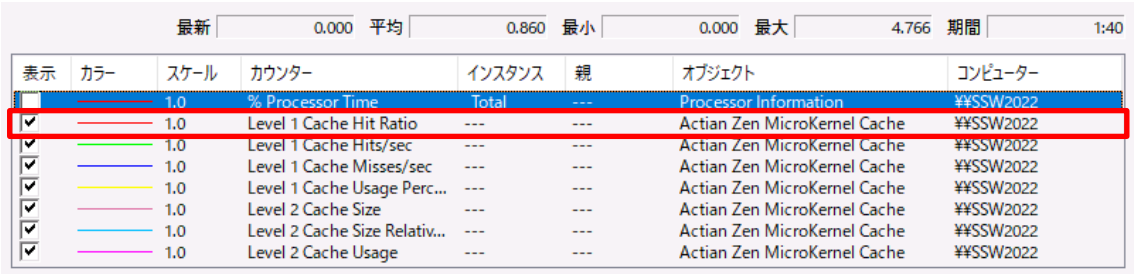
## ◆キャッシュのサイズ拡大

次のような場合には、キャッシュサイズの拡大が必要です。

・Monitor の「アクティブユーザー」の「ディスクアクセス数」が連続的に増える。




・パフォーマンスモニターのカウンター「Level 1 Hit Ratio」の平均値が減少している。




データベースエンジンは 2 つのレベルのメモリ キャッシュシステムを使用してデータ操作のパフォーマンスを向上させます。2 つのキャッシュは「L1（レベル 1）キャッシュ」と「L2（レベル 2）キャッシュ」と呼ばれます。

L1 キャッシュは、もっとも基本的なキャッシュシステムです。
Actian Zen は、ページ単位でファイル I/O を行なっていますが、このページ単位のデータが L1 キャッシュに格納され、読み込みおよび書き込みに使用されます。

L2 キャッシュは、L1 キャッシュに空きが無くなった際、ファイルへの書き込みが行われた際に
L1 キャッシュから破棄されるデータが移されます。直接ファイルから L2 キャッシュには読み込みません。
また、L2 キャッシュは読み込みにのみ使用され、書き込みには使用されません。
ユーザーの要求に応じエンジンがディスクからページを読み取る頻度が高くなるほど、パフォーマンスは低下します。すべてのデータが L1 キャッシュに格納された場合は最高のパフォーマンスが得られます。

L1 キャッシュサイズは、「キャッシュ割当サイズ」から変更できます。
L1 キャッシュサイズはサイズが固定で、物理メモリ容量の 20 %が使用されます。この値は Zen インストール時に設定されその後自動では変更されないため、後日メモリを増設した場合は手動での変更を行う必要があります。

L2 キャッシュサイズは、物理メモリの 60% を最大として動的に変更されます。
（「Microkernel の最大メモリ使用量」で設定します。「Microkernel の最大メモリ使用量」には L1 キャッシュも含まれます。）
オペレーティングシステム、その他アプリケーションで使用するメモリ使用量を考慮して、割り当てられるメモリバッファがある場合は、キャッシュサイズの拡大を検討してください。

キャッシュサイズは、下記の「キャッシュ割当サイズ」から変更できます。


---

＜＜＜ メモ ＞＞＞

キャッシュサイズを大きくすることは、パフォーマンスの向上に不可欠ですが、物理メモリの空きを考慮してください。他のアプリケーションや OS の必要メモリとあわせて物理メモリ以上になると、ページングが発生してパフォーマンスが低下します。

また、同時に使用するデータファイルのファイルサイズの合計値を超えるキャッシュサイズは、使用されない無駄なキャッシュ領域が発生しますので、ファイルサイズの合計値を超えない値をキャッシュサイズに指定してください。

L2 キャッシュは、他のアプリケーションでメモリを必要とした際に、サイズの縮小を行うことがあります。

パフォーマンスログで次のカウンターが減少するタイミングでパフォーマンスが低下する場合、L2 キャッシュを使用せずに L1 キャッシュのみにすることで改善することがあります。

Level 2 Cache Raw Usage
Level 2 Cache Size Relative to Memory
Level 2 Cache Raw Size

---

## ◆ページサイズの最大化による I/O 効率向上

大量レコードの読み込み・更新を行う場合、ファイル I/O が連続的に発生することでパフォーマンスに影響することがあります。
ディスク I/O は小さな単位で行うよりも、できるだけ大きな単位で行ったほうが効率的です。
小さな単位では、連続したブロックがディスク上では分散し I/O 毎にシークが発生することで読み込み時間が大幅に長くなります。
大きな単位では、同じデータを読み込む場合にも、ディスク上に連続して存在する可能性が高くなることから、シーク等のオーバーヘッドが減少でき、パフォーマンスの向上につながります。
これには、ページサイズの最大化が有効です。
ファイルを作成する際、デフォルトではページサイズが 4096 バイトとなります。
ファイル形式 9.5 では、ページサイズの最大サイズが 16384 バイトに設定ができ、この場合には、ページサイズ 4096 バイトの場合と比べ、シークの回数が 1/4 にできる可能性があります。

## ◆データ圧縮による I/O 効率向上

ページサイズを大きくすることに加え**圧縮**機能を併用すると、ページ当たりのレコード数が増加することで、更なるパフォーマンスアップが期待できます。

---

＜＜＜ メモ ＞＞＞
圧縮機能を使用する場合、圧縮・展開のために CPU 負荷が高くなります。
近年の CPU コアが多数の環境であれば通常問題ありませんが、CPU コア数の少ない環境では、CPU がボトルネックとなる可能性があります。
ストレージに SSD を使用していた場合、非圧縮の方が早くなることがあります。

---

これらの設定を変更するには、リビルドユーティリティを使用します。

## ◆ファイル I/O の分散によるパフォーマンス向上

他に、ファイル I/O の効率を高める方法として、データファイルと Actian Zen が内部で使用するトランザクションログファイルを異なるディスクに保存します。
トランザクションログファイルは、トランザクションの整合性を保証するために Btrieve ファイルの更新前に変更内容が記録されます。

複数のクライアントが同時に更新（Insert、Update、Delete）を行うと、Btrieve ファイルに加え、トランザクションログファイルへのアクセスが発生することから、ディスクへの負荷が高くなります。
Btrieve ファイルと Actian Zen が内部的に使用するファイルを異なるディスクに置くことで負荷が分散でき、パフォーマンスが向上します。

---

＜＜＜ メモ ＞＞＞
ディスクは物理的に異なることが必要です。
1 つのディスクを複数のパーティションに分けた場合、複数のパーティションのアクセスが同時に発生すると、パーティションにまたがりシークが発生するため、パフォーマンスが低下します。

---

OS のページファイルも異なるドライブに置けば、更にディスクの負荷が低くなり、ディスク I/O の効率アップに繋がります。
サーバーが仮想マシンの場合には、ディスクアクセスがボトルネックになることが多くなります。データファイルを保存するディスクは、パススルーディスクあるいは、仮想マシン上で直接 iSCSI ディスクに接続して使用することを推奨します。

## ◆ファイルをあらかじめ開いておく

アプリケーションの処理内容にもよりますが、次のような特定のケースでは、著しくパフォーマンスが低下することがあります。
１．SQL を連続で実行するケース
　例：データの登録に Insert を数千回実行
２．一連の処理の中で、Btrieve API の ファイル OPEN/CLOSE を繰り返し行うケース
　例：1 件のデータ更新の際、複数のファイルにアクセスしその都度、Btrieve API の ファイル OPEN/CLOSE を行う。
このような処理を数百、数千件繰り返し行う。
あるお客様からのお問い合わせでは、SQL での Insert を数千回繰り返した場合に、7 倍程度速度に違いがありました。
（SQL エンジンは、SQL 命令を実行する毎に、SQL 命令で処理するテーブルに対し、OPEN/CLOSE が発生します。）
多くの場合、このパフォーマンス低下の原因は PSQL エンジンから OS に対してのファイルの OPEN/CLOSE が、通常の読み込み、追加、更新、削除と比べると、数十倍時間が掛かることです。

（読み込み、追加、更新、削除は、キャッシュ上で処理が完結する場合、非常に高速です。）
PSQL エンジン内では、Btrieve API を呼び出すプログラムまたは SQL エンジン（以降、「Btrieve API セッション」と表記します）から Btrieve ファイルの OPEN/CLOSE 命令を受けると、別の Btrieve API セッションがすでにその Btrieve ファイル（テーブル）を開いている（操作中）ようであれば、再度 OS に対してのファイル OPEN/CLOSE は行わないため、Btrieve ファイルの OPEN/CLOSE はとても速くて軽い処理となります。
しかし、別の Btrieve API セッションが同じ Btrieve ファイルを開いていないようであれば、PSQL エンジンは時間が掛かる OS に対してのファイル OPEN/CLOSE が必要となり、パフォーマンス低下の原因となります。
この対策には、プログラム内で Btrieve API の OPEN/CLOSE 処理を減らす、またはシステム上のどこかのプログラムで予めファイルを開いておくことが有効です。
予めファイルを開いておくことで、処理内容によっては数倍速くなります。
特に、Workgroup では、OS に対してのファイルの OPEN/CLOSE の際にゲートウェイロケーター ファイルを確認するため、更にパフォーマンスに影響します。
プログラムで Btrieve API の OPEN/CLOSE 処理を減らすことは、多数の工数が掛かり、困難とのご意見もあり、弊社では簡単にファイルを開いておくツールを用意いたしました。
指定のファイルを開き、終了指示があるまで Wait するプログラムと終了指示を行うプログラムで構成されます。

**bfopen.exe**：カレント上の bfopen.ini に記述したパスに存在するファイルを OPEN し Wait します。
**bfopenend.exe**：bfopen.exe を終了します。

bfopen.exe の使い方は、ダウンロードファイル bfopen.zip に同梱しております bfopen_readme.txt に記載しておりますので、こちらを参照してください。

---

＜＜＜ メモ ＞＞＞
Zen v15 では、設定に「ファイルを閉じるまでの待ち時間」が追加され、ファイルをクローズしても一定時間（デフォルトでは 50 ミリ秒）待ってから Close を行います。この機能により多くのケースでは、ファイルを予め開いておかなくても、パフォーマンスが向上します。
実行間隔によっては、ファイルを予め開いておくことで、パフォーマンスが向上します。

---

### ◆インメモリでの使用

ファイル I/O を最小化する究極の方法は、すべてのファイルをキャッシュに読み込むことです。このためには、Btrieve ファイルの合計サイズを超えるキャッシュを用意し、全ファイルの読み込みを行う必要があります。
Actian Zen では、butil コマンドにファイルをキャッシュに読み込む機能が追加になりました。
butil コマンドを使用することで簡単に Btrieve ファイルをキャッシュに読み込めます。
butil -cache ファイル名

---

＜＜＜ メモ ＞＞＞
butil -cache は、キャッシュに空きが無いと読み込まれません。また、他のファイル同様キャッシュが不足した際に、長期間アクセスの無いページから削除されます。

---

全ての Btrieve ファイルがキャッシュ上に読み込まれている場合、ファイル I/O は書き込みのみとなります。
（実際には、トランザクションログファイルへの I/O が発生します）
書き込みについては、システムトランザクションの設定（「オペレーション バンドル制限」「起動時間制限」）により頻度が変わります。
同じレコードを頻繁に変更するようなケースでは、システムトランザクションの間隔を長くすることで I/O が減らせる可能性があります。
※システムトランザクションの間隔が長くなると、停電等の障害発生時にデータが失われる可能性が高くなります。

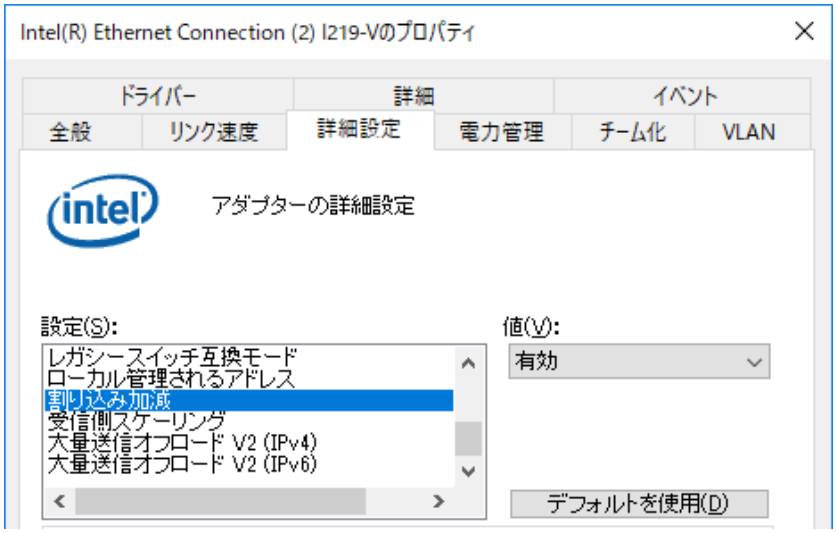
## 4. クライアント／サーバー構成での改善

クライアント/サーバー構成で使用する場合、ネットワークの影響や通信のオーバーヘッドにより、スタンドアローン環境と比べ数倍から数十倍遅くなることがあります。

### ◆ネットワークドライバーの設定

新しいマシンに変更して、物理メモリサイズやキャッシュサイズが増えているにも関わらず、パフォーマンスが劇的に遅くなるケースがあります。その場合は、ネットワークドライバーの設定を疑ってみてください。
具体的には「割込み加減」が有効の場合、ネットワークのパフォーマンスが著しく低

下する場合があります。
LAN ドライバーの「詳細設定」タブのプロパティの「割込み加減」を無効に変更して、速度が改善されるかどうかお試しください。
※「割込み加減」はハードベンダーによっては、「割り込みモデレーション」「Interrupt Moderation」「割り込み調停」などの名称になっている場合があります。




また、 Windows 7 以降の OS では、SNP 機能が有効になっていますが、これによってネットワークのパフォーマンスが低下する場合があります。下記の設定を行って SNP を無効にしてパフォーマンスが改善できるかどうかお試しください。
管理者コマンドプロンプトから下記の 3 つのコマンド実行

```
netsh int tcp set global chimney=disabled
netsh int tcp set global rss=disabled
netsh int tcp set global netdma=disabled
```

事例として、設定変更後に 3 倍以上のパフォーマンスの改善が見られるケースもありました。
※ネットワークドライバーの設定を複数行うことで 10 倍以上改善した例もあります。

## ◆Btrieve API 使用時のパフォーマンス改善

クライアント・サーバー環境では、通信が介在するためスタンドアローン環境とくらべ、どうしても遅くなります。
次に挙げる状況では、クライアントキャッシュを使用することで通信回数を減らすことができ、パフォーマンスが改善します。

- 連続的にデータの読み込みを行なう
- 更新処理が少ない

● レコードロックを行なわない

※ クライアントキャッシュは Btrieve レベルの I/O をページ単位でキャッシュし、サーバーからキャッシュのページを受け取ります。従って、サーバー上で動作する SQL エンジンが読み書きするキャッシュページは、クライアントキャッシュには送られません。SQL では、後述の Client Reporting Engine を使用することでクライアント側のキャッシュが利用されるようになります。

注意事項といたしまして、複数の端末からデータを書き換えている場合、お互いに書き換えた結果が得られるまでに最大 5 秒のタイムラグが発生します。そのため、同じレコードを複数の端末から参照したり、更新するような処理を行うシステム（例：在庫数量や金額の集計などを行うシステム）では、必要なプログラム内のところだけレコードロックまたはトランザクションによる並行性制御機能を使用するか、

キャッシュエンジンを使用しない運用をお願いします。

詳細は

https://www.agtech.co.jp/products/actian/docs_portal/Zen/15.1/index.html#page/advops%2Fconfig_client_security.htm%23

を参照して下さい。

---

＜＜＜ メモ ＞＞＞

ロックを使わない複数台からの同時更新は Update でステータス 80 が返されるので、更新データに整合性がとれなくなる事はありません。

---

キャッシュエンジン以外でも、拡張オペレーション（Extended Operation）を使用するとサーバーとの通信を大幅に減らすことができます。

拡張オペレーションでは、フィルターを指定して条件にマッチしたデータのみを取り出すことや、レコードの一部分のみを取り出すことができ、効率よくデータの読み込みが可能です。

拡張オペレーションでレコードの必要な部分だけを読み込めば、１回のオペレーションで数十から数百レコード読み込むことも可能です。

C# あるいは、VB.NET をご使用の場合、Btrieve Classes for .NET の Extended クラスを使用することで、より簡単に拡張オペレーションを利用できます。

Actian Zen では、拡張オペレーションで SQL の LIKE と同等な機能が新たにサポートされました。

この機能を使用すると、ある文字列（例えば「新製品」）が含まれるレコードを検索する場合に、従来ではすべてのレコードを読み込んでアプリケーションでサーチが必

要でしたが、エンジンで該当データのみを抽出して転送を行うことから、大幅なパフォーマンス向上が期待できます。

例として、demodata の Room テーブルで次の SQL を実行した場合と同様の抽出を Btrieve API で行ってみましょう。
select * from “Room” where building_name LIKE ‘%Harper%’

フィルターの構造体は、次のように定義します。

```
typedef struct
{
    BTI_CHAR    fieldType;
    BTI_SINT    fieldLen;
    BTI_SINT    fieldOffset;
    BTI_CHAR    comparisonCode;
    BTI_CHAR    connector;
    BTI_CHAR    excomp;
    BTI_SINT    vallen;
    BTI_CHAR    value[16];
} TERM_HEADER;
```

value は LIKE で指定する文字列になります。

例：%Harper%

この例では、SQL で「LIKE '%Harper%'」と指定した場合と同じになります。
value のサイズは指定する文字列（NULL を含む）よりも大きければ OK です。
vallen には、value のバイト数を設定します。
また、value に指定する文字列は、NULL で終端していることが必要です。

＜拡張オペレーションのサンプル＞
demodata の room テーブルから Building_Name に「Harper」が含まれるレコードを表示するサンプルです。見つかったレコードの Buikding_Name（2 バイト目から 25 バイト）を表示します。

```
#include "stdafx.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <btrapi.h>
#include <btrconst.h>

#define FILE1_NAME "btrv:///demodata?table=room"

#define EXIT_WITH_ERROR     1
#define TRUE                1
#define FALSE               0
#define VERSION_OFFSET      0
#define REVISION_OFFSET     2
#define PLATFORM_ID_OFFSET  4
#define MY_THREAD_ID        50

/* Don't pad our structures */
#if defined(__BORLANDC__)
#pragma option -a-
#else
#if defined(_MSC_VER) || defined(__WATCOMC__) || defined(BTI_LINUX) || defined(BTI_LINUX_64)
|| defined(BTI_MACOSX) || defined(BTI_MACOSX_64)
#pragma pack(1)
#endif
#endif

/*************************************************************************
Type definitions for version Structures
*************************************************************************/
typedef struct
{
         BTI_SINT  Version;
         BTI_SINT  Revision;
         BTI_CHAR  MKDEId;
} VERSION_STRUCT;

/*************************************************************************
Definition of record from 'Room'
*************************************************************************/
typedef struct
{
         BTI_CHAR  Building_Name_null;
         BTI_CHAR  Building_Name[25];
         BTI_CHAR  Number_null;
         BTI_ULONG  Number;
         BTI_CHAR  Capacity_null;
         BTI_WORD  Capacity;
         BTI_CHAR  Type[20];
} room_STRUCT;

/*************************************************************************
Structure type definitions for Get Next Extended operation
*************************************************************************/
typedef struct
{
         BTI_SINT    descriptionLen;
         BTI_CHAR    currencyConst[2];
         BTI_SINT    rejectCount;
         BTI_SINT    numberTerms;
} GNE_HEADER;
```

```
typedef struct
{
         BTI_CHAR    fieldType;
         BTI_SINT    fieldLen;
         BTI_SINT    fieldOffset;
         BTI_CHAR    comparisonCode;
         BTI_CHAR    connector;
         BTI_CHAR    excomp;
         BTI_SINT    vallen;
         BTI_CHAR    value[16];
} TERM_HEADER;

typedef struct
{
         BTI_SINT    maxRecsToRetrieve;
         BTI_SINT    noFieldsToRetrieve;
} RETRIEVAL_HEADER;

typedef struct
{
         BTI_SINT    fieldLen;
         BTI_SINT    fieldOffset;
} FIELD_RETRIEVAL_HEADER;

typedef struct
{
         GNE_HEADER              gneHeader;
         TERM_HEADER             term1;
         RETRIEVAL_HEADER        retrieval;
         FIELD_RETRIEVAL_HEADER  recordRet;
} PRE_GNE_BUFFER;

typedef struct
{
         BTI_SINT      recLen;
         BTI_LONG      recPos;
         room_STRUCT roomRecord;
} RETURNED_REC;

typedef struct
{
         BTI_SINT      numReturned;
         RETURNED_REC  recs[20];
} POST_GNE_BUFFER;

typedef union
{
         PRE_GNE_BUFFER  preBuf;
         POST_GNE_BUFFER postBuf;
} GNE_BUFFER,BTI_FAR* GNE_BUFFER_PTR;

/* restore structure packing */
#if defined(__BORLANDC__)
#pragma option -a.
#else
#if defined(_MSC_VER) || defined(__WATCOMC__) || defined(BTI_LINUX) || defined(BTI_LINUX_64)
|| defined(BTI_MACOSX) || defined(BTI_MACOSX_64)
#pragma pack()
#endif
#endif

int main()
{
```

```
        /* Btrieve function parameters */
        BTI_BYTE posBlock1[128];
        BTI_BYTE posBlock2[128];
        BTI_BYTE dataBuf[255];
        BTI_WORD dataLen;
        BTI_BYTE keyBuf1[255];
        BTI_BYTE keyBuf2[255];
        BTI_WORD keyNum = 0;

        BTI_BYTE btrieveLoaded = FALSE;
        BTI_LONG personID;
        BTI_BYTE file1Open = FALSE;
        BTI_BYTE file2Open = FALSE;
        BTI_SINT status;
        BTI_SINT getStatus = -1;
        BTI_SINT i;
        BTI_SINT posCtr;

        VERSION_STRUCT  versionBuffer[3];
        GNE_BUFFER_PTR  gneBuffer;
        room_STRUCT   roomRecord;
        printf("**************** Btrieve C/C++ Interface Demo ****************¥n¥n");
        memset(versionBuffer, 0, sizeof(versionBuffer));
        dataLen = sizeof(versionBuffer);

        status = BTRV(
                 B_VERSION,
                 posBlock1,
                 &versionBuffer,
                 &dataLen,
                 keyBuf1,
                 keyNum);

        if (status == B_NO_ERROR)
        {
                 printf("Btrieve Versions returned are: ¥n");
                 for (i = 0; i < 3; i++) {
                          if (versionBuffer[i].Version != 0)
                          {
                                   printf("   %d.%d %c¥n", versionBuffer[i].Version,
                                            versionBuffer[i].Revision,
                                            versionBuffer[i].MKDEId);
                          }
                 }
                 printf("¥n");
                 btrieveLoaded = TRUE;
        }
        else
        {
                 printf("Btrieve B_VERSION status = %d¥n", status);
                 if (status != B_RECORD_MANAGER_INACTIVE)
                 {
                          btrieveLoaded = TRUE;
                 }
        }

        /* clear buffers */
        if (status == B_NO_ERROR)
        {
                 memset(dataBuf, 0, sizeof(dataBuf));
                 memset(keyBuf1, 0, sizeof(keyBuf1));
        }
```

```
/* open room table */
if (status == B_NO_ERROR)
{

          strcpy((BTI_CHAR *)keyBuf1, FILE1_NAME);
          keyNum = 0;
          dataLen = 0;

          status = BTRV(
                    B_OPEN,
                    posBlock1,
                    dataBuf,
                    &dataLen,
                    keyBuf1,
                    keyNum);

          printf("Btrieve B_OPEN status (%s) = %d¥n", FILE1_NAME, status);
          if (status == B_NO_ERROR)
          {
                    file1Open = TRUE;
          }
}

/* now extract data from the original file, insert into new one */
if (status == B_NO_ERROR)
{
          /* getFirst to establish currency */
          keyNum = 0;
          memset(&roomRecord, 0, sizeof(roomRecord));
          dataLen = sizeof(roomRecord);

          getStatus = BTRV(
                    B_GET_FIRST,
                    posBlock1,
                    &roomRecord,
                    &dataLen,
                    keyBuf1,
                    keyNum);

          printf("Btrieve B_GET_FIRST status  = %d¥n¥n", getStatus);
}

gneBuffer = (GNE_BUFFER_PTR)malloc(sizeof(GNE_BUFFER));
if (gneBuffer == NULL)
{
          printf("Insufficient memory to allocate buffer");
          return(EXIT_WITH_ERROR);
}
memset(gneBuffer, 0, sizeof(GNE_BUFFER));
memcpy(&gneBuffer->preBuf.gneHeader.currencyConst[0], "UC", 2);
while (getStatus == B_NO_ERROR)
{
          gneBuffer->preBuf.gneHeader.rejectCount = 60000;
          gneBuffer->preBuf.gneHeader.numberTerms = 1;
          posCtr = sizeof(GNE_HEADER);

          /* fill in the first condition */
          gneBuffer->preBuf.term1.fieldType = 11;
          gneBuffer->preBuf.term1.fieldLen = 30;
          gneBuffer->preBuf.term1.fieldOffset = 1;
```

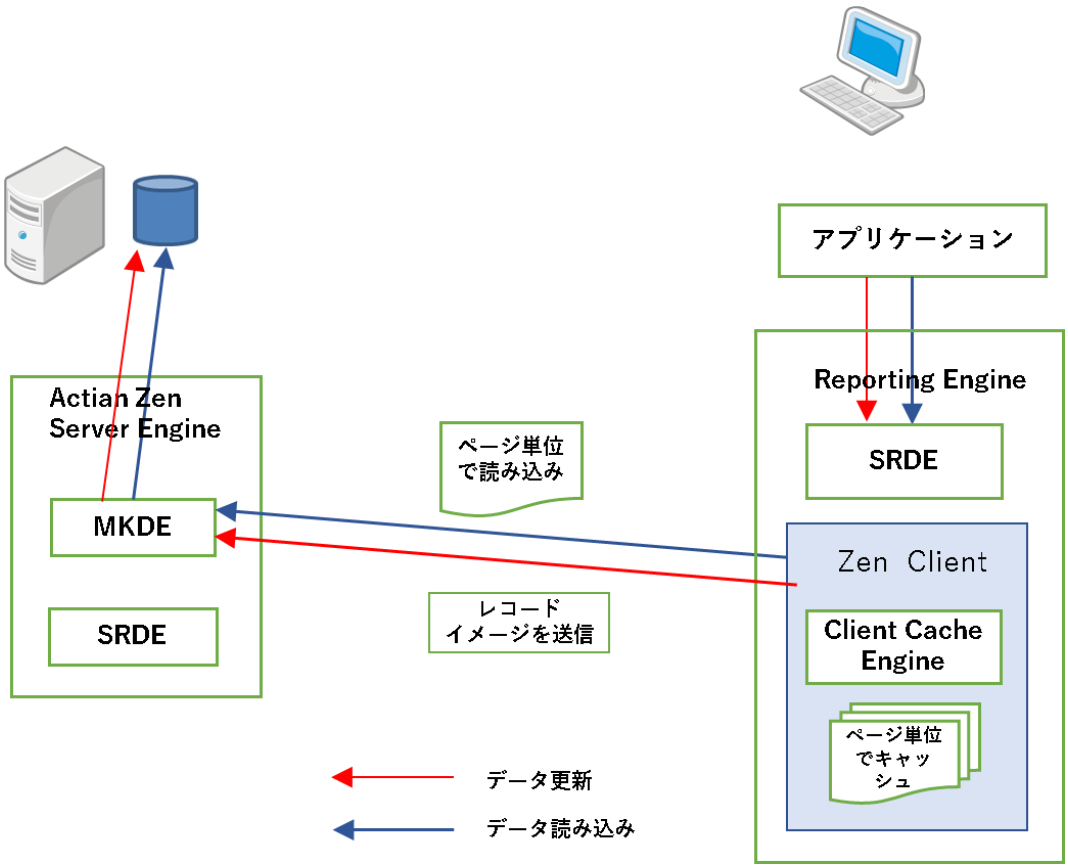
```
                    gneBuffer->preBuf.term1.comparisonCode = 7;
                    gneBuffer->preBuf.term1.connector = 0;
                    gneBuffer->preBuf.term1.excomp = 1;
                    gneBuffer->preBuf.term1.vallen = 16;
                    memcpy(&gneBuffer->preBuf.term1.value[0], "%Harper%", 8);
                    posCtr += sizeof(TERM_HEADER);

                    /* fill in the projection header to retrieve whole record */
                    gneBuffer->preBuf.retrieval.maxRecsToRetrieve = 20;
                    gneBuffer->preBuf.retrieval.noFieldsToRetrieve = 1;
                    posCtr += sizeof(RETRIEVAL_HEADER);
                    gneBuffer->preBuf.recordRet.fieldLen = sizeof(room_STRUCT);
                    gneBuffer->preBuf.recordRet.fieldOffset = 0;
                    posCtr += sizeof(FIELD_RETRIEVAL_HEADER);
                    gneBuffer->preBuf.gneHeader.descriptionLen = posCtr;

                    dataLen = sizeof(GNE_BUFFER);
                    getStatus = BTRV(
                              B_GET_NEXT_EXTENDED,
                              posBlock1,
                              gneBuffer,
                              &dataLen,
                              keyBuf1,
                              keyNum);

                    /* Get Next Extended can reach end of file and still return some records
*/
                    if ((getStatus == B_NO_ERROR) || (getStatus == B_END_OF_FILE)|| (getStatus
== 60))
                    {
                              printf("GetNextExtended  (Building_Name  Like  ¥'%%Harper%%¥')
returned %d records.¥n", gneBuffer->postBuf.numReturned);
                              for (i = 0; i < gneBuffer->postBuf.numReturned; i++)
                              {
                                        printf("名前：%s¥n", &gneBuffer->postBuf.recs[i].room
Record.Building_Name);

                              }
                    }

                    memset(gneBuffer, 0, sizeof(GNE_BUFFER));
                    memcpy(&gneBuffer->preBuf.gneHeader.currencyConst[0], "EG", 2);
          }

          free(gneBuffer);
          gneBuffer = NULL;

          /* close open files */
          if (file1Open)
          {
                    dataLen = 0;

                    status = BTRV(
                              B_CLOSE,
                              posBlock1,
                              dataBuf,
                              &dataLen,
                              keyBuf1,
                              keyNum);

                    printf("Btrieve B_CLOSE status () = %d¥n", status);
          }
```

```
        if (btrieveLoaded)
        {
                dataLen = 0;
                status = BTRV(
                        B_STOP,
                        posBlock1,
                        dataBuf,
                        &dataLen,
                        keyBuf1,
                        keyNum);

                printf("Btrieve STOP status = %d¥n", status);
                if (status != B_NO_ERROR)
                {
                        status = EXIT_WITH_ERROR;
                }
        }

        return(status);
}
```

## ◆ODBC 接続での効率化

ODBC 接続で大量レコードの読み込みを行うと、他のデータベースとくらべ大幅に遅くなることがあります。
これは、一度にデータを転送するバッファーのデフォルト値が小さいことが原因です。
データを転送するバッファーのサイズは、ODBC アドミニストレーターでデータソースのプロパティを開き、詳細ボタンで表示される「詳細の接続属性」にある「配列フェッチ バッファー サイズ」で変更できます。
デフォルトでは、8KB と小さいため、大量のデータを転送するには、効率がよくありません。これは、都度発生するオーバーヘッドの割合が高くなるためです。
「配列フェッチ バッファー サイズ」は、最大で 64KB まで設定が可能で、最大サイズとすることで、オーバーヘッドの割合を減少しパフォーマンスを向上することが可能です。

Actian Zen では、新たに Client Reporting Engine が追加され、クライアント側の SQL エンジンが SQL データの抽出を行うことで、負荷分散が可能となります。
また、キャッシュエンジンの機能を内包していることで、ページ単位にサーバーからデータを読み込み、クライアント側のキャッシュにあるデータを、そのまま使用することで、読み込みのパフォーマンスが向上します。
Client Reporting Engine は、ODBC だけではなく、ADO.NET、JDBC の SQL インターフェイス全てで使用できます。
大量にデータの更新（追加、変更、削除）を行う場合には、レコード毎にサーバーの MKDE と通信を行うため、パフォーマンスが低下します。
処理によって、クライアント側のデータベースを使用するか、サーバー側のデータベースを使用するか選択してください。




# 5.その他留意点

ファイル I/O に関連してパフォーマンス上留意すべき点として、ページングが発生しないような設定での運用が必要です。
ページングが発生するとパフォーマンスの低下が顕著となるため、ページングが発生しない範囲でキャッシュサイズを設定します。
また、トランザクションで大量データの追加・更新を行う等で大量のロックが必要となる場合は、メモリの使用量が想定以上に増加することがありますから、ご注意ください。
※ロックで必要となるメモリは、「Microkernel の最大メモリ使用量」で設定したメモリとは別に使用されることが判明しています。
大量のロックが必要となるロジックは避けてください。
※SQL エンジン（SRDE）が作業用に使用するメモリは、「Microkernel の最大メモリ使用量」で設定したメモリとは別に使用されます。