COBOL アプリケーションにおける Actian Zen リレーショナル機能の利用

AG-Tech ホワイトペーパー

2021 年 6 月

目次

はじめに 3

なぜ Actian Zen なのか 4

SQL アクセス 4

複数の開発環境の使用 4

SQL アクセスを可能にする 5

COBOL Schema Executor を使用する 5

手順 1：データの評価 5

手順 2：不適合の解決 7

手順 3：データの記述 8

手順 4：デプロイメント 9

終わりに 10

お問い合わせ先と商標に関する情報 11

はじめに

このホワイト ペーパーは、Actian Zen の ISAM 側(トランザクショナル インターフェイス Btrieve)を使用している COBOL アプリケーション開発者が、既存の COBOL アプリケーションを拡張して Actian Zen のリレーショナル側を完全に利用するための支援となるよう作成されています。

COBOL アプリケーション開発者は、アプリケーション内で Actian Zen アクセスを完全に可能にする利点について聞いたり、顧客から Actian Zen アクセスの要望があったとしても、何年にも渡る作業と顧客からのフィードバックを反映した一連の既存コードがあることが多く、追加機能を利用するためにそのコードを書き直したいとは思いません。このホワイト ペーパーでは、Actian Zen の全機能を利用するための方法について、さらに、既存アプリケーションにおける投資を保ちながらも、コードの書き直しのコストとリスクを回避する方法についても説明します。

これは アプリケーションの COBOL ランタイムが、基となるデータ API として Btrieve を使用することを前提としています。このランタイムが現在 Btrieve API(または Actian Zen)を使用していない状況で、Btrieve API のパフォーマンスや低所有コストを利用したい場合は、別途ご提供しているホワイトペーパー「データベース移行: COBOL から Actian Zen」で、COBOL アプリケーションから Actian Zen への移行に関する概要説明をご参照ください。

ISAM と Btrieve

ISAM はレコードの格納と検索の点で Btrieve API と類似するので容易に Btrieve API に適合します。このため、多くの場合、COBOL コンパイラはデータ アクセス メカニズムとして Btrieve を使用します。ISAM のように、Btrieve API にはファイルとテーブル間に 1 対 1 のリレーションシップがあり、Btrieve ではアプリケーションがレコード レイアウトを定義(および再定義)することが可能です。必要なのは、キー情報を知っていることだけです。

Btrieve を使用する COBOL ランタイム上に構築されたアプリケーションは既に Actian Zen アプリケーションです。これらのアプリケーションでは、Actian Zen と Btrieve が認められる、圧倒的なパフォーマンス、広範なプラットフォーム サポートおよび安価な TCO(Total Cost Of Ownership:総所有コスト)を利用します。ただし、Actian Zen で提供される強力な機能のうち、Btrieve のみのアプリケーションですぐには使用できないものがあります。また、Actian Zen では、SQL クエリ機能や別の言語のデータへのアクセス、各種ツール、および ODBC、OLE DB、ADO.NET や JDBC のような標準インターフェイスを使用したアプリケーション サーバーも提供します。

ISAM と SQL

SQL と ISAM ではデータ アクセス要件が異なります。SQL の Database Management System(DBMS)では各フィールドが明確に定義され、あらゆるデータが厳密に 1 個のフィールドに属している必要があります。ISAM アプリケーションで SQL アクセスを有効にする場合は、ISAM アクセスの要件(現在、データの要件が満たされている)と SQL アクセスの要件との相違を解決する必要があります。データによっては、その移行で必要な作業が DDF ファイルとして知られる Actian Zen カタログ テーブルのデータベース スキーマを作成するだけという場合もあります。それ以外の場合、データが SQL によって完全にアクセスできるようになる前に大幅なアプリケーション変更が必要となる可能性があります。このような場合でも、部分的なアクセスは可能で、時間をかけて追加的に完全なアクセスへ移行することができます。

このホワイト ペーパーでは SQL アクセスを伴う COBOL アプリケーションを可能にするための一般的および具体的なガイダンスを提供します。特に、Actian Zen には、COBOL OCCURS コンストラクト、部分的 REDEFINES および可変長レコード レイアウトを使用したデータへのリレーショナル アクセスのサポートを提供するために大いに役立つユーティリティや例があります。これは、COBOL 開発者向けに SQL アクセスに必要なことについて大きく取り上げていると考えています。それ以外に、このホワイト ペーパーでは一般的な対処法と、より完全な SQL アクセスを可能にするための例について説明します。

なぜ Actian Zen なのか

COBOL 開発者が Actian Zen の機能を利用することを望む理由は大きく 2 つあります。それは SQL アクセスおよび幅広い開発者環境との統合です。

SQL アクセス

ほとんどの場合、COBOL 開発者が COBOL アプリケーションに追加したい機能は SQL アクセスです。SQL の柔軟性や標準性は、レポート作成における新しい要件や変化する要件を理想的に満たすことできます。ユーザーが必要とする新しいレポート作成機能に応じてその都度、設計、コード記述、テストするよりも、臨時にレポート作成要件のソリューションとしてデータベースを照会する SQL を使用する方がはるかに簡単です。

従来のCOBOL アプリケーションはサイクルに対応しており、ユーザーが新しいレポートを要求する場合には、IT 部員がその要求を待ち行列へ追加し、ユーザーによる(あるいはユーザーを介さないで)数回のフィードバック サイクルの後、最終的に新しいレポートを作成するという事象が発生します。SQL を使用すれば、ユーザーは直接または操作された SQL クエリのどちらからでも臨時のレポートを作成できます。この柔軟性によって、SQL アクセスがなく、ユーザーが独自のレポートを作成できないアプリケーションに対して競争上の優位性を提供します。

しかし、アプリケーションのパフォーマンスにおいては(トランザクショナル アクセス方法に比べ)同等のレベルを提供することが難しいため、コンパイラ/ランタイムの組み合わせによってサポートされるとしても、多くの開発者はアプリケーション全体を SQL に変換しようとは思いません。

場合によっては、機能を拡張することによってアプリケーションの価値が高まるため、SQL サポートを追加する意味があると思われることもあるでしょう。それ以外の状況で主に懸念されることは、アプリケーションの機能を高めることよりも、ユーザーの当面のニーズに合っているかということです。この場合には、SQL を使用するサードパーティ製ツールが非常に貴重な資源となり得ます。ODBC(標準的な SQL アクセス方法)を介して Actian Zen を存分に使用できるレポート作成ツールが多数あります。データベースで SQL アクセスが可能になれば、これらのツールはアプリケーションから独立して使用することができます。多くの場合、ODBC と SQL ではユーザーの要件を迅速かつ確実に満たす方法を提供することが可能で、アプリケーションの大々的なリエンジニアリングが必要になることはありません。また、これらのツールのなかにはエンドユーザー向けのものもあるため、顧客はそのツールを自ら用いて必要な機能を追加することができます。

複数の開発環境の使用

複数の開発環境からアプリケーション スイートが構築されることがますます一般的になっています。たとえば、アプリケーションの中核部分は COBOL で作成しているが、最新の機能は Delphi または Visual Studio 開発環境を使用して追加していることがあります。Btrieve API は最も一般的な開発環境内で使用することができます。また、Delphi、C++ Builder、および Visual Studio(C#、VB.NET、C++ 環境向け)などのビジュアル環境に対しても、Actian Zen はより親しみやすい環境として選択できます。

特に、ADO.NET データ プロバイダーでは、VB.NET やその他の .NET ベースのアプリケーションを用いたアプリケーション開発に最も適しているナビゲーショナルおよび SQL インターフェイスを提供します。Zen Direct Access Components(旧称 PDAC)を使用すれば、Delphi または C++ Builder 開発者が直感的に理解できるナビゲーショナルおよび SQL アクセスが行えます。

上記のコンポーネントはそれぞれ Actian Zen ソフトウェア開発キット(SDK)の一部です。各コンポーネントには詳しいマニュアルが付属しており AG-Tech の SDK Library から無償でダウンロードすることができます。アプリケーションでこれらのコンポーネントを使用する場合は、SQL アクセスに必要な情報と同様のデータ スキーマ定義が必要となるので、データベースへの SQL アクセスを可能にすれば、さらにこれらのナビゲーショナル アクセスも可能となります。

SQL アクセスを可能にする

データベースへの SQL アクセスを完全に行えるようにする場合、データの編成によって、アプリケーションの変更が不要なこともあれば、大幅な変更が必要なこともあります。完全な SQL アクセスでなくても大きな効果が得られることが多いので、徐々にアプリケーションを完全な SQL アクセス対応に移行してもよいでしょう。SQL アクセスを完全に使用できるようにするか、あるいは徐々に移行するか、どちらを選択するにしろ SQL エンジンがデータにアクセスできるようにするには、先にデータ構造が明確に定義されていなければいけません。

最初に、データベース(アプリケーションで使用するファイル)を評価し、SQL と互換性がある部分と、互換性がないのでアクセスするためには SQL 用に変更が必要である部分を特定します。アクセスしたいデータが SQL と互換性を持つようになってから、データを SQL エンジンに記述させるために、メタデータ(列名、テーブル名などのデータに関する情報)を用意する必要があります。これらのタスクが完了すると、SQL アクセスが可能になりユーザーはランタイム クエリの能力と柔軟性を味わうことができるようになります。

Actian Zen では、COBOL OCCURS コンストラクト、部分的 REDEFINES および可変レコード レイアウトを使用して特定のデータ型に対して SQL アクセスを可能にする各種ツールを提供します。まず、これらのツールの使用方法について説明し、次に、その他の COBOL データ型へのSQL アクセスを可能にする一般的な説明に進みます。

COBOL Schema Executor を使用する

Actian Zen には COBOL Schema Executor というコマンド ライン ユーティリティがあります。これは ISAM データを正規化された SQL テーブルと解釈するために、Actian Zen リレーショナル インターフェイスが使用するシステム テーブルを作成するユーティリティです。COBOL Schema Executor を使用するには、まず XML ファイルを使用し SQL エンジンへデータを記述する必要があります。Actian Zen には単純なテーブルのデータ、OCCURS コンストラクトを含むデータ、部分的 REDEFINES を含むデータ、および可変レコード レイアウトのデータを定義するためのサンプル XML ファイルがあります。この COBOL Schema Executor の使用に関する手順は以下のとおりです。

手順 1:データの評価

ISAM の実装で、完全な SQL アクセスに対して一般的に発生する障害が主に 2つあります。これは、データベースのデータ型にマップしない COBOL コンパイラ データ型と、複数の定義を持つレコードについてです。SQL DBMS ではクエリを評価するためにデータを完全に理解することが必要なため、そのデータのデータ型について説明し SQL DBMS で理解できるようにしなければなりません。ISAM モデルでは、キーの一部ではないデータはデータベース自身ではなく、アプリケーションでのみ理解されます。これは、キーではないデータがかならずしもデータベースでサポートされるデータ型とは限らないということです。最初の作業は、アプリケーションで使用されるコンパイラの各データ型に相当するデータベースのデータ型の確定です。

データ型のマッピング

COBOL アプリケーションに SQL アクセスを有効にする場合、以下の 3 種類のデータ型が関与すると思われます。

1) COBOL データ型(“9999 COMP” など)

2) Btrieve データ型。これは自動的に SQL 型へマップすることができます。

3) SQL データ型

Btrieve データ型から SQL データ型へのマッピングを確認する場合は、『SQL Engine Reference』マニュアルの「Zen で使用できるデータ型」セクションを参照してください。

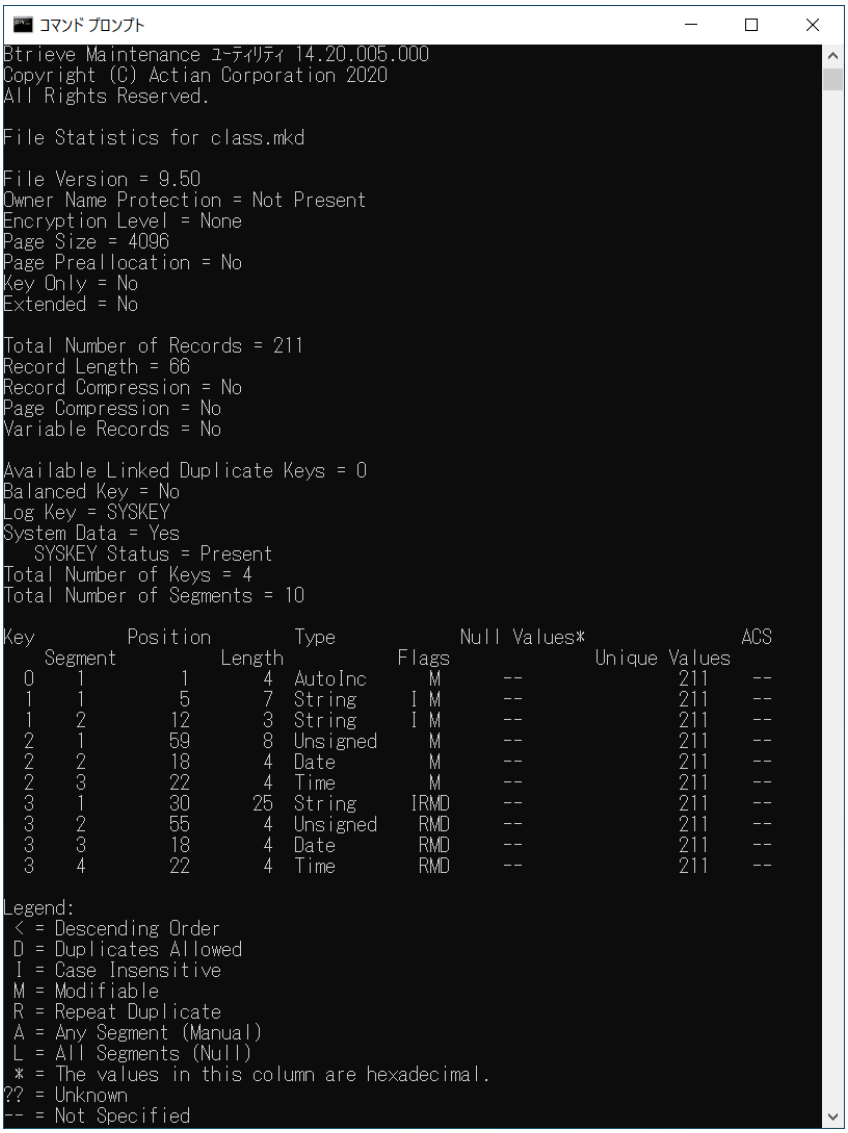
データ型のマッピングを判断する最良の方法は COBOL コンパイラ ベンダーに尋ねることです。コンパイラは COBOL 型のデータを存続させる責任があるため、必ずこの情報を表すために使用するネイティブのデータ型を知っているはずです。たとえキーではないデータがアプリケーションの仕様であっても、キー データは正しくソートされるようにするために DBMS によって理解される必要があります。COBOL ベンダーがキーに対してさえネイティブではないデータ型を使用することが可能であっても、Btrieve API を使用したときにデータが正しくソートされていれ

ば、ほぼ間違いなく SQL API がデータ型を理解しています。コンパイラ ベンダーはいくつかのデータ型をサポートし、それ以外のデータ型はサポートしないことが考えられます。

必要なデータ型のマッピングをコンパイラ ベンダーが提供できない場合でも、Actian Zen ファイルのヘッダーに格納されているキー データ型の情報を使用してそのマッピングを判断することができます。この情報は Actian Zen ユーティリティによって取得できるため、アプリケーションで使用する各データ型に対して 1 個のキーを作成し、結果ファイルのキー情報を見ることによってデータ型のマッピングを判断できます。

まず、アプリケーションで使用されるすべてのデータ型を含むファイルを新規作成します（通常、データ長は影響ありません）。それらのデータ型のフィールドをキーとしてファイル記述（File Description）に宣言し、書き込み用のファイル（作成する必要があります）を開きます。このアプリケーションの実行後には、このファイルがファイル システムにあります。次に、このファイルに対して Actian Zen ユーティリティである BUTIL を実行し、COBOL の各データ型のネイティブのデータ型を調べます。「butil -stat <ファイル名>」と入力します。この <ファイル名> 部分には新たに作成するファイルの完全なパスを指定します。データ ファイルの各キーに対して 1 個のキー エントリがあり、その隣にネイティブのデータ型宣言が示されます。コンパイラが特定のデータ型へ直接マップしないときは、コンパイラの最も近いデータ型、またはコンパイラのデフォルトの型（おそらく文字列またはバイナリ ― バイナリ順でソートされる汎用的な非文字列）になります。

図 1：Btrieve Maintenance ユーティリティ



```
コマンド プロンプト
Btrieve Maintenance ﾕｰﾃｨﾘﾃｨ 14.20.005.000
Copyright (C) Actian Corporation 2020
All Rights Reserved.

File Statistics for class.mkd

File Version = 9.50
Owner Name Protection = Not Present
Encryption Level = None
Page Size = 4096
Page Preallocation = No
Key Only = No
Extended = No

Total Number of Records = 211
Record Length = 66
Record Compression = No
Page Compression = No
Variable Records = No

Available Linked Duplicate Keys = 0
Balanced Key = No
Log Key = SYSKEY
System Data = Yes
   SYSKEY Status = Present
Total Number of Keys = 4
Total Number of Segments = 10

Key          Position        Type            Null Values*                ACS
    Segment          Length           Flags              Unique Values
  0    1         1       4  AutoInc      M      --               211    --
  1    1         5       7  String     I M      --               211    --
  1    2        12       3  String     I M      --               211    --
  2    1        59       8  Unsigned     M      --               211    --
  2    2        18       4  Date         M      --               211    --
  2    3        22       4  Time         M      --               211    --
  3    1        30      25  String     IRMD     --               211    --
  3    2        55       4  Unsigned    RMD     --               211    --
  3    3        18       4  Date        RMD     --               211    --
  3    4        22       4  Time        RMD     --               211    --

Legend:
 < = Descending Order
 D = Duplicates Allowed
 I = Case Insensitive
 M = Modifiable
 R = Repeat Duplicate
 A = Any Segment (Manual)
 L = All Segments (Null)
 * = The values in this column are hexadecimal.
?? = Unknown
-- = Not Specified
```

複数レコード定義

完全な SQL アクセスに対する 2 番目の障害はレコード定義です。ISAM データ モデルではキー データに対して強制的に単一定義を行いますが、キーではないデータに対しては再定義を行うことができます。リレーショナル モデル（SQL がベースとなる）では、データの各フィールドが正確に 1 種類の情報（請求日など）を示すことが必要です。

手順 2：不適合の解決

データを評価した後、アプリケーションにはデータ型のマッピングと再定義されたレコードの事例リストがある場合があります。SQL によって完全にデータへアクセスできるようにするにはこれを解決する必要があります。この事例リストがない場合は問題ありません。手順 3 の「データの記述」に進むことができます。

不適合のデータ型にはそれぞれ可能な解決策がいくつかあります。この解決策は必要な作業量やアプリケーションへの影響によって異なります。通常、最も完全な解決策には最も労力を要します。完全な SQL アクセスが可能となるように不適合を解決する場合は、より多くの作り替えが伴い、次善策を講じるよりもアプリケーションへの影響が大きくなります（純粋な SQL ベースのアプリケーションでは、実際には完全にアクセスするか、あるいはまったくアクセスしない状況の場合があります。しかし、ナビゲーショナル/ISAM のやり方を基にしている場合は、部分的な SQL アクセスが可能なので、不完全性の問題を持ち込むことなく SQL の利点を得ることができます。）。

サポートされないデータ型の解決

コンパイラが、正しくソートされない特定のコンパイラ データ型用にキーを作成している場合は、サポートされないデータ型があります。これは正しくソートされないことに加え、このデータ型に対する SQL 計算（SUM など）も不正になることを意味します。サポートされないデータ型が 1 つ以上あり、そのデータ型が頻出するものでもなく、また特に重要でもない場合、短期的にはそれらのデータ型にコンパイラのデフォルトをそのまま使い続けることが妥当です。このアプローチの難点は、これらのデータ型のフィールドに対する SQLアクセスでは不要なデータが示されるという点です。

たとえ、キーの作成時にコンパイラが Btrieve へデータ型を正しく記述しない（つまり、コンパイラはデータ型のマッピングで不正確な選択を行っている）としても、コンパイラのデータ型のバイト レイアウトがデータベースのデータ型のバイト レイアウトと一致することもあります。この場合、正しいデータ型を使用してアプリケーション外でファイルを作成することができます。アプリケーションでファイルを作成すると、結果的にデータ型は不正になります。セットアップや維持が難しいため、この解決法は理想的ではありませんが、場合によっては便利な方法でもあります。このアプローチを使用するには、コンパイラのデータ型とデータベースのデータ型が幅広い値にわたってバイナリ レベルで同一であることを確認してください。

データ型が不適合で、そのデータ型の使用頻度が高く、非常に重要である場合（または目標が次善策を講じるのではなく完全な SQL アクセスの場合）、アプリケーションの変更が必要になるかもしれません。ファイル記述（File Description）で使用されるデータ型を適合するデータ型に変更することはそれほど困難ではありません。厳密な表現が重要な場合は、たいてい特定のデータが、保有しているレコード（レポート レイアウトなど）へ移動します。この場合、新しいデータ型に対する要件は、コンパイラがそのデータ型を前のデータ型と質的に類似するよう扱うことだけです。たとえば、ファイル記述（File Description）に PICTURE 句で 9(5) COMP という宣言があったとすると、コンパイラは正しく 9(5) をマップしますが 9(5) COMP はマップしません。

この型を 9(5) へ変更してもほかのコードにはほとんど影響ありません（しかし、物理レコード レイアウトは変更されるのでデプロイメントには影響があります）。

再定義されたレコードの解決

COBOL Schema Executor が、再定義されたレコードを処理するためのニーズに合わない場合は、「部分的レコード定義」、「複数定義」、および「オフライン正規化」という 3 つの対処法を試してください。

部分的レコード定義 ― 再定義されたレコードへの対処法の 1 つは、SQL を使用してレコードの定義の中から 1 つのレコード定義だけ公開することです。たとえば、アプリケーションには制御ファイルがあり、システム 1、2、3、および 4（例：総勘定元帳、支払台帳、売掛金および買掛金）に対してそれぞれ 1 つのレコード定義を持つことができます。システム 2 への SQL アクセスを可能にすることだけが目的であれば、SQL への制御ファイルの記述で、システム 2 向けに定義するのが最良の手段です。目的がこれよりも明確でない場合でも、アクセスしたいデータが合併した定義へ隔離されるときがあります。レコード定義 1 として 5-16 のバイトへの SQL アクセスが総勘定元帳レポートに必要で、レコード定義 2 として 21-26 のバイトへの SQL アクセスが買掛金レポートに必要な場合、実際にはファイルに存在しないレコード定義をSQL に対して記述することもあり得ますが、関心のある最大量のデータへのアクセスが可能です。これの欠点は、17 から 20 のバイトが無意味（SQL クエリでは不要データとして示される）であることですが、必要なフィールドを最も多く得ることができるでしょう。

複数定義 ― 再定義されたレコードへの一般的な別の対処法は、特定のファイルに対して複数の定義を持つことです。SQL エンジンに対してデータを記述する場合、同じファイルを参照するいくつかのテーブル定義を、それぞれ固有のテーブル名を付けて提供することができます。これらの記述はすべて同じファイルを指すよう設定してください。この対処法の利点は、アプリケーションを変更することなくすべてのデータへ SQL アクセスするという点です。これの欠点は、レコード タイプの列に対して必ず適切なフィルターを適用する必要がある点です。これを行わないと、これらのテーブルへのクエリでは別のタイプのレコードのたびに不要データが返されます。もう 1 つの欠点は、SQL アクセスが COBOL アプリケーションとは異なるテーブル識別子を使用する必要があるという点です。

たとえば、ファイル A にレコード タイプ 1 と 2 があるとします。Actian Zen には 2 つのテーブル定義を記述し、A1 と A2 という名前を付けます。すべてのテーブル アクセスは A1 または A2 という名前を使用して実行されますが、A1 のすべてのレコードに対するクエリではすべてのレコードが返され、A2 を対象としたすべてのレコードに対するクエリでは A1 固有のフィールドにある不要データがあるでしょう。SQL の WHERE 句やフィルターは適切なレコードのみを返すクエリを構築するために使用できますが、SQL アクセスがアプリケーションによって操作されていない限り、すべてのクエリにはそのビジネス ロジックを含める必要があります。この問題を軽減するためにビューを使用できます（Actian Zen マニュアルで SQL ビューについての説明を参照してください）。

オフライン正規化 ― もう 1 つの対処法は、恐らく比較的静的な制御レコードにのみ適している、補助的なオフライン正規化です。まず、再定義されたレコードをそれぞれ個別ファイルに書き込む補助的なアプリケーション作成します。次に、その作成したファイルを SQL エンジンに記述します。この対処法の利点は、既存のコードに変更を加える必要がなく、非常に小さなアプリケーションが 1 つ必要なだけという点です。この欠点は、これが自動的に動作するものではなく、処理が必要だという点です（制御データを変更したときには必ずアプリケーションを実行することを覚えておく必要があります）。これは制御レコードの変更プログラムから連鎖するもので、わずかなコード変更を必要としますが、不正データに対してレポートが作成される可能性を減らします。

この問題を完全に解決する方法は、複数のファイルにまたがる再定義されたレコードをすべて正規化することです。これは大幅なアプリケーション変更（ロジックへの変更も含む）が必要です。アプリケーションのビジネス ロジックは再定義によって異なることもあります。たとえば、単独のカーソルを使い、関連する 2 種類のレコードをキー順で並べることによってレポート作成作業を容易にするためにレコードの再定義を使用できます。このアプローチは SQL の世界では動作しません。これは各ファイルの記述に独立したカーソルがあるためです。このデータに対するレポート作成には 2 つのファイルのリンクが必要です。

手順 3：データの記述

開発者はこの時点で、要求される SQL アクセスを可能にするには Actian Zen エンジンへデータを記述する必要があるということを理解しているでしょう。Actian Zen ではテーブルを記述にするために役立つ DDF Builder というツールを提供します。この Eclipse ベースのユーティリティは、対応するデータベース ファイルの基礎を成すメタデータを記述するデータ辞書の作成、変更、表示に使用できます。

DDF Builder ではメタデータとデータ ファイルの両方が変更できるため、作業を開始する前にデータをバックアップしておくことが重要です。

DDF Builder で使用される用語についても理解しておくことが大切です。リレーショナル モデルは、データベースを構成するファイルの定義についてよく知っていることを前提としている点で ISAM とは異なることを覚えておいてください。また、リレーショナル モデルではファイルはテーブルと呼ばれ、フィールドは列と呼ばれます。

データベースを作成し、既存のデータ ファイルへの記述を追加することから始めます。DDF Builder を使用すると、SQL のデータ型とディスクリプター（オフセット、長さなど）を使用してデータを記述することができます。また、データをマップしたらその構造をプレビューして適合性と正当性を検証することもできます。

フィールドはデフォルトでヌル値が可能となるよう設定され、データがないことを正確に記録することができます。このオプションによって、ヌルの後続にあるすべてのフィールドのオフセットが変更されるので、デフォルト設定を受け入れるとレコード レイアウトが大幅に変わります。ご自分のアプリケーションの COBOL コンパイラが真のヌルをサポートしないようであれば、すべての列に対してヌル値を許可しない（[ヌル]チェックボックスをオフにする）ようにしてください。最後に、データの記述で相対パスを使ってデプロイメントをより簡潔にしてください。DDF Builder でファイルを記述したら、Zen Control Center（ZenCC）を使用して SQL レイアウトおよび構造を表示することができます。

SQL の背景のためには、キーのリレーションシップを記述しないようにすることができます。リレーショナル モデルでは、主キー/外部キーのリレーションシップは参照整合性（RI）を意味します。参照整合性については Actian Zen のマニュアルで詳しく説明していますが、簡単に言うと、COBOL アプリケーションでは DBMS が恐らく参照整合性をサポートしないことを前提としています。これを変更すると、アプリケーションで予期しないエラーが発生するかもしれません。主キー/外部キーのリレーションシップを確立することは、暗黙のうちに新しいデータベース ルールを追加することになり、レコードの削除および変更のタイミングや方法に影響を与えます。

手順 4：デプロイメント

デプロイメントを開始する前に、製品データを必ずバックアップするようにしてください。

レコード レイアウトに変更がない、またはコンパイラ/ランタイムを "ごまかす" ための対処法がない場合は、製品データへの変更はありません。

次の手順ではデータ定義をデプロイします。これには、データベースをエンティティとして定義する、そしてデータベースを構成するデータ ファイルを定義するという 2 つの工程があります。多くの場合、特にデプロイメントが少ないときには、手順 3 で示したように ZenCC を使用して空のデータベースを作成した方が簡単です。現在、データ ファイルの定義をデプロイする方法が最低 3 つあります。

1 つ目の方法は、DDF Builder を使用してデータを記述し、デプロイメント マシン（通常はサーバー）ごとにヌルの問題を解決することです。これは、多数のデプロイメントがある場合は集中的に時間が取られる可能性があるので、一般的にこの問題を解決する最良の方法ではありません。

2 つ目の方法は、最も速い解決策で、開発マシン（手順 3 が実行されたマシン）から DDF をデプロイメント マシンへコピーすることです。ただし、これは DDF に相対パスが含まれているか、両者のマシン上のパスが同一である場合にのみ動作します。

3 つ目の方法は、SQL に精通している場合で、SQL スクリプトを使用（および保存）しデプロイメント マシンで記述を作成することです。

データ ファイルの定義をデプロイすると、ZenCC、Crystal Reports またはあらゆる SQL ツールをサーバー上でロードしデータにアクセスできるようになります。これは非常に画期的なことですが、さらにクライアント DSN という手順があります。DSN はODBC アクセスに必要です。DSN なしでも ADO、ADO.NET やその他いくつかの方法を使用することも可能ですが、サード パーティ製ツールの多くは ODBC を使用しています。これは、ODBC が SQL アクセスには最も幅広く使用されている標準だからです。サーバーに設定された DDF を使用してネットワーク経由でクライアント アクセスを行うことができます（これはさらにナビゲーショナル API が使用できるようになるためには理想的）が、クライアント DSN がクライアントにセットアップされている場合は、一般に ODBC アクセスの方がよく動作します。クライアント DSN については Actian Zen のマニュアルで説明していますが、内容はリリースごとに少しずつ変わっています。

レコード レイアウトまたはキーのデータ型がこの処理の結果として変更された場合、既存のデータは変換する必要があります。データがコンパイラに反した形でマップする必要があった場合（つまり、コンパイラ/ランタイムでは常に PIC 9(5) を文字列として保存するが、ネイティブの型である DECIMAL として保存するべき）、COBOL 外でファイルを作成し、COBOL アプリケーションでそのファイルを入れます。このような理由から、この手段は非常に堅牢なものとは言えず、最後の手段として使う解決策の 1 つとして検討してください。データが同一の場合、コンパイラによるデータ型の認識が挿入、更新および読み取りに影響を及ぼすことはありませんが、ファイルの作成要求には影響があります。

終わりに

ここまでくれば、作業は完了です。アプリケーションには新たな機能がもたらされましたが、既存のコード ベースにおける投資は保たれます。既存のアプリケーションは（機能とパフォーマンスの観点で）変わることなく動作するはずです。しかも、SQL の能力や新たに増えたプラットフォームのサポートを使用して、以前よりも多くの機能をユーザーにご提供します。

お問い合わせ先と商標に関する情報

株式会社エージーテック
本　社：〒101-0054 東京都千代田区神田錦町1-21-1 ヒューリック神田橋ビル3F
TEL：03-3293-5300（代表）FAX：03-3293-5270

本資料の内容の著作権はすべて株式会社エージーテックに帰属します。株式会社エージーテック以外の方がこの資料および内容をコピー・配布すること、また本資料および内容を用いて研修・著述やそれに準ずる行為を行うことは禁止します。

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。