データベース移行:

COBOL から Actian Zen

AG-Tech ホワイトペーパー

2021 年 6 月

目次

はじめに 3
COBOL および Actian Zen データベースの比較 4
COBOL から Actian Zen への移行 5
バイナリ互換性の維持 5
ヌルの処理 5
スキーマの移行 7
Actian Zen インデックス(ファイル キー) 7
データの移行 8
コードの移行 9
その他のリレーショナル アクセス 9
終わりに 9
付録:データ型 11
お問い合わせ先と商標に関する情報 12

はじめに

COBOL ベンダーは開発者に対し、一般的にコンパイラ、統合開発環境、および高速で信頼性の高いファイル ベースのデータ管理を勧めます。しかし、COBOL ISAM データベースにはユーザーの要求に一致するための多くの主要な機能、たとえば、簡易レポートおよびビジネス情報交換のためのリレーショナル アクセス、リモート ユーザーをサポートするレプリケーション、安全なバックアップとトラブル復旧が欠けています。

多くの COBOL 開発者はベンダーが提供したデータベースを使用し続けています。それは、ISAM ファイル アクセス用に記述されたアプリケーションをリレーショナル データ管理技術へ移行するためには、多くの場合、複雑な移行作業と移行されないコードの書き直しが必要なためです。別の開発者は、コードの書き直しおよび既存のアプリケーションにリレーショナル レイヤーを追加することによる移行を敬遠します。これは(ISAM および SQL 間の)トランスレーション レイヤーに関わることで、この追加処理は一般的にアプリケーションのパフォーマンスを低下させます。これらの手法は、どちらも理想的とはいえません。

必要なものは、トランザクショナル コードと簡単にインターフェイスがとれる、高パフォーマンスのリレーショナル データ管理システムです。Actian Zen は、トランザクショナルおよびリレーショナル データ アクセスの両方を提供する類のないシステムです。トランザクショナル ISAM から Zen のトランザクショナル API である Btrieve への移行は、インターフェイスの類似性により非常にすばやく行うことができます。ひとたび変換されれば、アプリケーションは元のビジネス ロジックを使用し、一方 Actian Zen 環境は、レポート作成のためのネイティブなリレーショナル アクセスだけでなく SQL アプリケーションと互換性のあるさまざまなインターフェイスを提供します。

この文書の目的は、データ管理技術を Actian Zen にアップグレードしようと考える COBOL 開発者を手助けすることです。Actian Zen を一般的な COBOL ISAM データベースと比較します。これには COBOL アプリケーションを Actian Zen へ移行するのに必要な手順、および、より複雑なデータ パターンに関連する障害を克服する方法が含まれます。この文書では、移行の完了後にアプリケーションが最適なパフォーマンスを維持するために必要な手順についても説明します。

COBOL および Actian Zen データベースの比較

移行を決断する前に、COBOL ISAM データベースと Actian Zen との差異を理解しておくことをお勧めします。2 つの比較を次の表に示します。

表 1：COBOL ISAM / Actian Zen 比較

機能	COBOL	Actian Zen
データ ストレージ システム	ファイル ベースのレコード システム。ISAM アクセス。	完全なリレーショナル機能を提供するファイル ベースのレコード システム。Actian Zen エンジンには完全に機能するデータベース管理システムが含まれ、Actian Zen 製品でサポートされるすべてのプラットフォームで利用できます。
ロック	行レベル	行レベル
ストアド プロシージャ	No	Yes
トリガー	No	Yes
レプリケーション	No	Yes
ライブ バックアップ機能	No	Yes

データ アクセス	COBOL	Actian Zen
ODBC（リレーショナル）	アドオン*	Yes、レベル 2
JDBC	アドオン*	Yes、JDBC 4 Type 4
OLE DB/ADO	アドオン*	Yes、リレーショナルおよびトランザクショナル
ADO.NET	アドオン*	Yes、Visual Studio と統合される ADO.NET データ プロバイダーおよび Entity Framework プロバイダー
トランザクショナル	Yes	Yes、Btrieve API、また、C#、C++、Node.JS、Perl、PHP、Python 用のオブジェクト指向の Btrieve 2 API も用意されています。
Java（直接）	No	Yes、Actian Zen の Java Class Library（JCL）インターフェイス
Embarcadero VCL	No	Yes
Distributed Tuning（DTI および DTO）	No	Yes

*アドオンが利用可能かどうかは、COBOL ベンダーおよび開発環境によって異なります。

COBOL、Btrieve および ODBC データ型のマッピングについては「付録：データ型」をお読みください。

COBOL から Actian Zen への移行

このセクションでは、COBOL 環境から Actian Zen へのスキーマ、データおよびコードの移行に関する方針およびトピックについて説明します。

バイナリ互換性の維持

COBOL アプリケーションは、一般的にデータを低レベルで処理します。つまり、アプリケーションはデータを特定の場所で、レコードを特定のサイズ(バイト数)で探します。したがって、COBOL から Actian Zen に移行する場合、移行されたデータがアプリケーションとバイナリ互換性を保っていることを確実にする必要があります。

ヌルの処理

以下の例は、COBOL からリレーショナル データ ストアへ移行する際にバイナリ互換性を綿密に再検査することの重要性を示しています。Actian Zen は真のヌルをサポートし、テーブルの作成時には、多くのリレーショナル データベースのようにレコード間にヌル バイトを自動的に作成します。COBOL レコードは一般的にヌルを含まないので、移行中にこれに対処することが重要です。

Table_A と呼ぶサンプル レコードを見てください。以下の 3 つのフィールドから成る固定長レコードが含まれています。

1) 長さ 20 の英数字フィールド
2) 長さ 4 の数値フィールド
3) 長さ 37 の英数字フィールド

COBOL のレコード表現では、この構造は次のようになります。

```
01 Record
    05 Field1 PIC X(20)
    05 Field2 PIC S9(9) COMP5
    05 Field3 PIC X(37)
```

サンプルのバイナリ レコードは次のようになります。

文字 20 バイト	整数 4 バイト	文字 37 バイト

このフォーマットでは、レコードの合計は 61 バイトになります。このテーブルを標準 SQL ステートメントを使用して作成するには、以下の構文を使用します。

```
CREATE TABLE Table_A (Field1 Char(20), Field2 Integer, Field3 Char(37))
```

ただし、このステートメントが Actian Zen で実行されると、Actian Zen はフィールドのデータが含まれるバイトの直前にヌル バイトを配置するので、64 バイト長のレコードを持つ SQL テーブルを表すファイルが作成されます。CREATE TABLE ステートメントの実行結果は、次のようなレコード レイアウトになります。

ヌル バイト	文字 20 バイト	ヌル バイト	整数 4 バイト	ヌル バイト	文字 37 バイト

COBOL アプリケーションの元のレコード長が 61 バイトの場合、データ アクセスに上の COBOL 構造を使用すると、元の 61 バイト構造は新しい 64 バイト構造に調整されず、このデータへの SQL クエリは無意味なものになります。

ヌル値のインジケーター バイトは Actian Zen の使用時に除去して、以前の構造に調整する必要があります。これは、2 つの方法のいずれかを使用して行うことができます。最初の方法は、次のような SQL ステートメントを実行します。

```
SET TRUENULLCREATE = OFF
```

これにより、現在 Actian Zen エンジン セッションに、擬似ヌル サポート(レガシー ヌル サポートとも呼びます)に戻すよう指示します。レガシー ヌル サポートでは、フィールド内のヌルを表すのに既知の値を使用します。たとえば、ゼロという値がヌルを表すのに使用されることがあります。

2 番目の方法として、フィールド作成時にヌル値を許可しないよう、SQL エンジンに明示的に指示することができます。この場合、テーブルは真のヌルをサポートして作成されますが、ヌル値を含みません。SQL ステートメントは次のようになります。

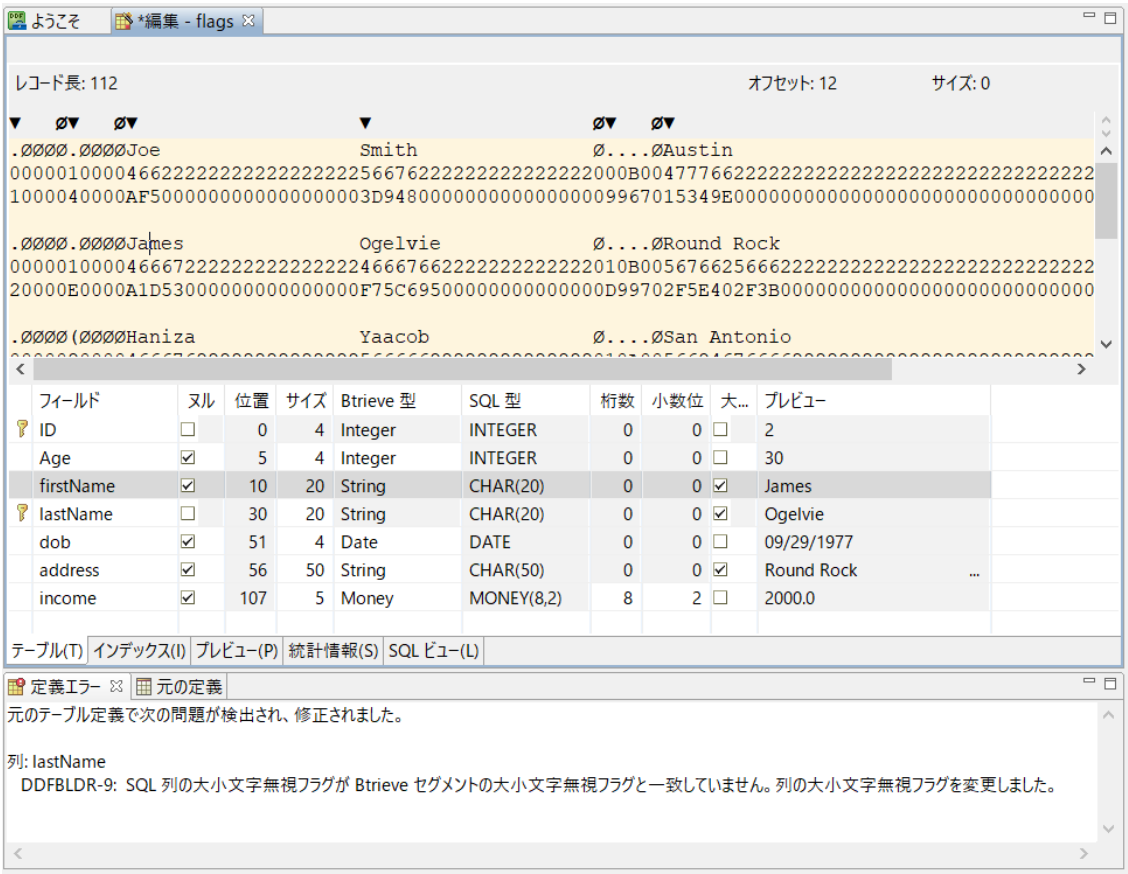
```
CREATE TABLE Table_A (Field1 Char(20) NOT NULL, Field2 Integer NOT NULL,
Field3 Char(37) NOT NULL)
```

これらの SQL ステートメントは、それぞれレコード長が 61 バイトで COBOL 構造に調整されたファイルを生成します。

DDF Builder とのバイナリ互換性の分析

Actian Zen の DDF Builder を使用して、移行中に作成されたテーブルのバイナリ互換性を分析することができます。DDF Builder は Java ベースのユーティリティで、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 を使用すると、SQL のデータ型とディスクリプタ(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互換性と正確性を検証することもできます。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。

図 1:DDF Builder



COBOL アプリケーションがバイナリ レベルでデータにアクセスする場合、互換性が保持されていることは必須です。そうしないとアプリケーションはエラーになります。

スキーマの移行

スキーマを移行するには、アプリケーションのデータ型を分析する必要があります。分析をまとめる最も単純な方法は、アプリケーションで使用されているすべての構造の一覧表を作成することです。例を示します。

```
01 CUST-REC.
     03 CUST-ID _________ PIC X(10)
     03 CUST-NAME _______ PIC X(30)
     03 CUST-ADDR-1 _____ PIC X(40)
     03 CUST-ADDR-2 _____ PIC X(40)
     03 CUST-CITY _______ PIC X(15)
     03 CUST-STATE ______ PIC XX
     03 CUST-ZIP ________ PIC X(10)
     03 CUST-PHONE ______ PIC X(15)
     03 CUST CONTACT ____ PIC X(20)
```

この構造では、レコードごとに 182 バイトが必要です。このテーブルを作成するには、次の SQL ステートメントを使用します。

```
CREATE TABLE CUSTOMERS( ID CHAR(10) NOT NULL,
                        NAME CHAR(30) NOT NULL,
                        ADDR1 CHAR(40) NOT NULL,
                        ADDR2 CHAR(40) NOT NULL,
                        CITY CHAR(15) NOT NULL,
                        STATE CHAR(2) NOT NULL,
                        ZIP CHAR(10) NOT NULL,
                        PHONE CHAR(15) NOT NULL,
                        CONTACT CHAR(20) NOT NULL)
```

このステートメントは、長さ 182 バイトのレコードを持つ CUSTOMERS.MKD というファイルを作成します。このファイルは Customers テーブルに関連付けられます。このアプリケーションでは、NAME というフィールドにキーが定義されています。このキー、またはインデックスは、別の SQL ステートメントを使用して Actian Zen 内で明示的に定義することができます。

```
CREATE UNIQUE INDEX idx_Name on CUSTOMERS(NAME);
```

このステートメントは、Customers ファイルの Name フィールドのすべてのエントリが一意になることを必要とするインデックスを作成します。

スキーマを移行する際は、COBOL コンパイラが Actian Zen データベース機能のすべてを同じようにサポートしないこと、および Zen がすべての COBOL データ型をサポートしないことを心に留めておいてください。移行処理が順調に行くことを確実にするため、ベンダーのマニュアルを見直すことをお勧めします。たとえば、Actian Zen は COMP-6 データ型をサポートしません。移行を考えているアプリケーションが COMP-6 データの拡張使用を行っている場合、COMP-6 型の要素を除去する再加工が必要です。移行の目的の 1 つが、データベリレーショナル アクセスを可能にすることである場合は、事前に、『Zen Programmer's Guide』で「リレーショナル データベース設計」の「設計の段階」のセクションにある正規化ルールをお読みください。

Actian Zen インデックス（ファイル キー）

インデックスまたはファイル キーは、Actian Zen と連携する場合には重要な意味を持ちます。また、複数フィールドキーの関係を理解するのに特に重要です。たとえば、前のサンプル テーブルに、Name、City、および State から成

るキーが追加されていた場合、単一キーまたは単一インデックスを作成することができます。Name、City および State から成る複数フィールド インデックスも、Name フィールドから成る単一フィールド キーと同じように働きます。これは、Name フィールドが複数フィールドキー内で、順序として単一フィールド キーと同じ位置にあるためです。

複数フィールド キーは、COBOL ソース コードの FILE-CONTROL セクションで別個のフィールドとして定義されている必要があります。たとえば、以下の構造が使用されるとします。

```
01 Record
   05 Field1_PIC X(10)
   05 Field2_PIC X(25)
   05 GROUP1
      10 G1F1_PIC X(3)
      10 G1F2_PIC 9(9)
```

GROUP1 は FILE-CONTROL セクションでキー定義として使用されていません。定義は以下のような形式です。

```
RECORD KEY IS MyKEY = G1F1, G1F2
```

複数フィールド定義が使用されない場合、Actian Zen は、ファイルのインデックスをキー定義に一致させられないため、パフォーマンスが悪影響を受けます。インデックスは、変換されたアプリケーションが、速度という点で失敗する最も一般的な理由の 1 つです。アプリケーションが小さなファイルで申し分なく実行できても、ファイル サイズが大きくなるにつれてパフォーマンスが低下する場合、ほとんどの原因はインデックスの定義が欠けているか不正なことです。

データの移行

アプリケーション用のファイルを作成したら、次の手順は ISAM ファイルから既存のデータを Actian Zen ファイルに転送することです。通常、既存の ISAM ファイルからデータを読み取り、それを新しい Actian Zen データ ファイルに書き込む COBOL プログラムを作成することでデータの移行が可能です。

以下のサンプル コードは、富士通 NetCOBOL ISAM ファイルからレコードを読み取って、一致する Actian Zen ファイルに書き出す方法を示しています。

メモ: Btrieve と富士通 NetCOBOL では、数値データの保存方法が異なるため、数値データを使用している場合、Btrieve ファイルへデータを書き込みする直前に NetCOBOL の内部形式から Btrieve ファイルで扱う形式（88 コンソーシアム形式）へ変換します。その変換を簡単に行うために、富士通 NetCOBOL には下記の "#DECFJTO88" 関数が用意されています。

```
*- ファイルを開く
     OPEN INPUT ISAMFILE
     OPEN OUTPUT BTRFILE
*- レコードの読み取りと書き出し
     PERFORM UNTIL END-OF-PROCESSING
        READ ISAMFILE NEXT INTO workarea
          AT END SET END-OF-PROCESSING TO TRUE
          NOT AT END
              CALL "#DECFJTO88" USING workarea
              WRITE btrRECORD FROM workarea
        END-READ
     END-PERFORM
*- ファイルを閉じる
     CLOSE ISAMFILE
     CLOSE BTRFILE
```

データを COBOL から Actian Zen へ変換するのにどのような手段をとったかにかかわらず、変換が完了したらアプリケーションの徹底的なテストを行うことを強くお勧めします。

コードの移行

富士通 NetCOBOL では、COBOL プログラム内の ASSIGN 句でファイルシ ステム タイプを BTRV に設定するだけで、デフォルトのファイル システムの代わりに Btrieve ファイルを使用することで、レコード シーケンシャル アクセスとインデックス アクセスの両方を行うことができます。このファイル システム タイプ設定の簡単な変更を行うと、富士通 NetCOBOL ランタイムは、COBOL の動詞 READ および WRITE を、基になる Btrieve API オペレーション呼び出しに変換します。これは、ほんのわずかなコード変更だけでコード移行ができるということです。

この機能の使用に関する詳細情報については、富士通 NetCOBOL ユーザーガイド第 7 章の「7.9.1 Btrieveファイル」に記載されているベンダーのドキュメントをご覧ください。すでに Btrieve API の使用に慣れている方は、ライブラリやオブジェクト ファイルを使用して中間 DLL にリンクすることで、COBOL プログラム内から Btrieve API を直接呼び出すこともできます。このようなリンクが行われると、COBOLの "CALL" 動詞を使用し、次のような形式の COBOL コマンドを記述することで BtrieveAPI を直接呼び出すことができます。

```
CALL "_BTRV" USING B-OPEN,B-STAT, B-POS-BLOCK, CUST-REC,B-BUF-LEN, B-KEY-VALUE,
B-OPEN-MODE.
```

ベンダー固有のサンプル コード、また Microfocus および Fujitsu コンパイラに必要な中間 DLL、OBJ、LIB ファイルは、AG-Tech サポート ページから入手できます。

その他のリレーショナル アクセス

この文書では、Actian Zen リレーショナル エンジンを介してリレーショナル アクセスも可能な Actian Zen ファイルの作成方法について説明しています。つまり、トランザクショナル メソッドを使用して元の COBOL アプリケーションでアクセスされている同じファイルに、SQL ステートメントを使用してアクセスすることもできるということです。Actian Zen には、COBOL の OCCURS 句、部分的な REDEFINES、および可変レコード レイアウトで定義されたデータに対しても、リレーショナル アクセスを可能にする機能もあります。この詳細については、Actian Zen の『SQL Engine Reference』マニュアルの「COBOL アプリケーションの SQL アクセス」のセクションを参照してください。COBOL ベンダーは、一般的に ODBC 準拠のデータベース アクセスを提供するインターフェイスを勧めます。これらのインターフェイスを使用すると、COBOL プログラマは COBOL ソース コード内で SQL ステートメントを使用することができます。COBOL アプリケーション内での Actian Zen リレーショナル インターフェイスの使用に関する詳細については、ベンダーのマニュアルを参考にしてください。

終わりに

COBOL ISAM ファイルから Actian Zen への移行は、汎用的な処理ですが、いくつかの手順だけは完全を期す必要があります。ソフトウェア設計のように非常に複雑な仕事では、個別の状況の詳細によって、どの手順が取り入れられ、どの手順が省略されるかが決定づけられます。

移行は、アプリケーションが使用するデータ型の徹底的な分析から始める必要があります。Actian Zen ファイルのスキーマを構築してテストする必要があります。データ移動の確実な計画も構築する必要があります。元のアプリケーションのコード変更は、すべて検証およびテストする必要があります。アプリケーションの移行が終わったら、さまざまなインターフェイスを介して Actian Zen へのリレーショナル アクセスが可能になります。元のアプリケーションがトランザクショナル アクセスによって操作を継続していても、ODBC（Open Database Connectivity）、JDBC（Java Database Connectivity）、ADO.NET および OLEDB アプリケーションがデータ ストアに接続可能です。

さらに、このようにデータを利用できるのは、COBOL アプリケーションだけに限りません。Actian Zen は、これ以外のプログラミング言語やレポート ツールからのトランザクショナルおよびリレーショナルのデータ アクセスも豊富にサポートしています。このように、COBOL データを Actian Zen に移行することで、システム設計者はほかの言語やツールを使用してシステム機能を拡張することができます。また、その一方で既存の COBOL アプリケーションの機能や使用法を最小限のコード変更で維持することもできます。

付録：データ型

COBOL	Btrieve データ型	リレーショナル データ型	ODBC
PIC S9(4)	AUTOINCREMENT(2)	SMALLIDENTITY	SQL_SMALLINT(5)
PIC S9(8)	AUTOINCREMENT(4)	IDENTITY	SQL_INTEGER(4)
COMP1	BFLOAT(4)	BFLOAT4	SQL_REAL(7)
COMP2	BFLOAT(8)	BFLOAT8	SQL_DOUBLE(8)
PIC X(n)	BLOB	LONGVARBINARY	SQL_LONGVARBINARY (-4)
PIC X(n)	CLOB	LONGVARCHAR	SQL_LONGVARBINARY (-1)
PIC S9(n) V(2)	CURRENCY	CURRENCY	SQL_DECIMAL(3)
	DATE	DATE	SQL_DATE(9)
PIC S9(n)V(m)	DECIMAL	DECIMAL	SQL_DECIMAL(3)
COMP-1	FLOAT(4)	REAL	SQL_REAL(7)
COMP-2	FLOAT(8)	DOUBLE	SQL_DOUBLE(8)
PIC S9(2) COMP-5	INTEGER(1)	TINYINT	SQL_TINYINT(-6)
PIC S9(4) COMP-5	INTEGER(2)	SMALLINT	SQL_SMALLINT(5)
PIC S9(9) COMP-5	INTEGER(4)	INTEGER	SQL_INTEGER(4)
PIC S9(16) COMP-5	INTEGER(8)	BIGINT	SQL_DECIMAL(3)
	LOGICAL(1)	BIT	SQL_BIT(-7)
	LOGICAL(2)	SMALLINT	SQL_SMALLINT(5)
PIC S9(n)V(m)	NUMERIC	NUMERIC	SQL_NUMERIC(2)
	NUMERICSA	NUMERICSA	SQL_NUMERIC(2)
	NUMERICSTS	NUMERICSTS	SQL_NUMERIC(2)
PIC X(n)	STRING	BINARY	SQL_BINARY(-2)
PIC X(n)	STRING	CHAR	SQL_CHAR(1)
	TIME	TIME	SQL_TIME(10)
	TIMESTAMP	TIMESTAMP	SQL_TIMESTAMP(11)
	UNSIGNED(1) BINARY	UTINYINT	SQL_TINYINT(-6)
	UNSIGNED(2) BINARY	USMALLINT	SQL_SMALLINT(5)
	UNSIGNED(4) BINARY	UINTEGER	SQL_INTEGER(4)
	UNSIGNED(8) BINARY	UBIGINT	SQL_DECIMAL(3)
	WSTRING	NCHAR	SQL_WCHAR
	WZSTRING	NVARCHAR	SQL_WVARCHAR
	ZSTRING	VARCHAR	SQL_VARCHAR(12)
	なし	BIT	SQL_BIT(-7)

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。