プロダクション内での SQL の使用法

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

プロダクション内での SQL の使用法
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

SQL アダプタの概要

SQL クエリまたはプロシージャをプロダクション内から実行することにより、相互運用プロダクションを使用して外部のリレーショナル・データ・ソースのデータを操作することができます。クエリまたはプロシージャを実行する最も簡単な方法は、事前構築済みのビジネス・サービスまたはビジネス・オペレーションをプロダクションに追加する方法です。このようなビジネス・サービスやビジネス・オペレーションを使用することにより、プロダクションは組み込みの SQL アダプタを使用して JDBC または ODBC に準拠したデータベースと通信できます。カスタム・コーディングは必要ありません。ただし、これらのアダプタを使用して外部データ・ソースにアクセスする、カスタムのビジネス・サービスまたはビジネス・オペレーションを構築することもできます。

注釈　ビジネス・サービスおよびビジネス・オペレーションで使用される SQL アダプタはクライアントなので、ユーザ名とパスワードを指定した呼び出しを渡すことにより認証を実行します。オペレーティング・システムによる認証は使用しません。

2

SQL ビジネス・サービスの使用法

インターシステムズでは、SQL を使用してデータをプロダクションに読み込む事前構築済みのビジネス・サービスを 2 つ提供しています。1 つはクエリ用、もう 1 つはストアド・プロシージャ用です。ビジネス・サービスを使用してクエリを実行するには、`EnsLib.SQL.Service.GenericService` をプロダクションに追加します。クエリではなく SQL プロシージャを実行したい場合は、`EnsLib.SQL.Service.ProcService` を使用します。

これらの組み込みのビジネス・サービスは、受信 SQL アダプタ (**EnsLib.SQL.InboundAdapter**) を使用して外部データ・ソースにアクセスします。場合によっては、このアダプタとその結果の処理方法を詳細に制御できるカスタム・ビジネス・サービスを構築しなければならないことがあります。

2.1 データ・ソース名の指定

プロダクションで操作するデータが含まれるデータ・ソースを指定するには、管理ポータルを使用して、以下のビジネス・サービス設定を定義します。

DSN

このデータ・ソース名は、接続先である外部データ・ソースを指定します。InterSystems IRIS® では、定義済みの InterSystems SQL ゲートウェイ接続、JDBC URL、およびオペレーティング・システムで構成した ODBC DSN の 3 種類の形式を自動的に区別します。

管理ポータルの **[システム管理]→[構成する]→[接続性]→[SQLゲートウェイ接続]** ページから構成された JDBC 接続または ODBC 接続の名前とこの名前が一致すると、その仕様のパラメータが使用されます。エントリが構成済みの接続名ではなく、そこにコロン (`:`) が使用されていると、その名前は JDBC URL と見なされます。それ以外の場合は ODBC DSN と見なされます。

以下の例は、JDBC URL を参照する DSN の名前を示しています。

```
jdbc:IRIS://localhost:9982/Samples
```

以下の例は、Microsoft Access データベースを参照する ODBC DSN の名前を示しています。

```
accessplayground
```

このデータ・ソースがパスワードによって保護されている場合は、ユーザ名とパスワードを含むプロダクション認証情報を作成します。その後、これらの認証情報の ID と一致するように **Credentials** を設定します。詳細は、"その他の実行時設定の指定" を参照してください。

JDBC データ・ソースを使用している場合は、以下の設定も適用されます。

Java Gateway Service

このビジネス・サービスで使用する Java ゲートウェイ・サーバを制御する Java ゲートウェイ・サービスの構成名です。基盤となるアダプタ設定は `JGService` です。

重要　この設定は、JDBC と接続中の SQL ゲートウェイ接続を使用している場合でも、すべての JDBC データ・ソースで必要です。JDBC 接続が機能するには、タイプが EnsLib.JavaGateway.Service のビジネス・サービスが存在する必要があります。ビジネス・サービスの SQL アダプタに対して、この構成項目の名前を指定する必要があります。その構成済み設定を使用して、このアダプタが管理する JVM に接続します。

JDBC ドライバ

JDBC ドライバ・クラス名です。

名前付き SQL ゲートウェイ接続を DSN として使用する場合、この値の指定は任意です。ただし、値を指定すると、名前付き JDBC SQL ゲートウェイ接続のプロパティよりも優先して適用されます。

JDBC クラスパス

JDBC ドライバ・クラス名のクラスパスです (Java ゲートウェイ・サービスで構成したクラスパス以外にも必要な場合)。

接続の属性

必要に応じて設定できる一連の SQL 接続属性オプション。ODBC の場合は、これらの形式は次のとおりです。

attr:val,attr:val

例：`AutoCommit:1`

JDBC の場合は、これらの形式は次のとおりです。

attr=val;attr=val

例：`TransactionIsolationLevel=TRANSACTION_READ_COMMITTED`

名前付き JDBC SQL ゲートウェイ接続を DSN として使用する場合、この値の指定は任意です。ただし、値を指定すると、名前付き JDBC SQL ゲートウェイ接続のプロパティよりも優先して適用されます。

2.2 受信クエリの指定

既定では、SQL ビジネス・サービスは、受信アダプタを使用してクエリを定期的に実行します。このアダプタからビジネス・サービスに結果が 1 行ずつ送信されます。

2.2.1 クエリの指定

ビジネス・サービスで使用される基本のクエリを指定するには、Query 設定を使用します。この設定では、基本のクエリ文字列を指定します。置き換え可能なパラメータの表現には、SQL 標準の ? を使用できます。これは、個別の設定で指定します。以下の例を検討します。

```
SELECT * FROM Customer

SELECT p1,p2,p3 FROM Customer WHERE updatetimestamp > ?

SELECT * FROM Sample.Person WHERE ID > ?

SELECT * FROM Sample.Person WHERE Age > ? AND PostalCode = ?
```

2.2.2 パラメータの指定

Parameters 設定では、クエリ文字列内の置き換え可能なパラメータを指定します。この設定は、以下のような、パラメータ値指定子のカンマ区切りリストになります。

```
value,value,value,...
```

指定する値ごとに、`10` や `Gotham City` などの定数リテラル値を使用したり、以下のいずれかを参照したりできます。

- アダプタのプロパティを参照できます。Parameters 設定内で、構文 %property_name を使用します。
- ビジネス・サービスのプロパティを参照できます。構文 $property_name を使用します。

  例えば、タイムスタンプを入れるために、`LastTS` という名前のプロパティをビジネス・サービス・クラスに追加できます。Parameters 設定では、プロパティの値を `$LastTS` として参照します。
- `&%LastKey` などの特別な永続値を参照できます。この値には、アダプタによって最後に処理された行の IDKey 値が含まれています。

  Parameters 設定でパラメータ名がアンパサンド (&) で始まる場合は、その名前は特別な永続値であると見なされます。

  注釈　カスタム・ビジネス・サービスを使用したこれらの値の初期化については、"アダプタの初期化" を参照してください。

2.2.3 文字列データの最大長の指定

ビジネス・サービスの [VARCHAR LOB Boundary] 設定では、**VARCHAR** データ型を使用して InterSystems IRIS に格納できる文字列の最大長を指定します。ソース・データベース内に、指定した値より長い文字列を含む列があると、InterSystems IRIS では、**LOB** (ラージ・オブジェクト) データ型を使用してその文字列が格納されます。ビジネス・サービスの受信アダプタの対応するプロパティは、MaxVarCharLengthAsString です。

デフォルトの最大文字列長は `32767` です。値 `-1` を指定すると、InterSystems IRIS の最大文字列長を使用できます。現在、これは `3641144` ですが、今後のバージョンで変更される可能性があります。InterSystems IRIS の最大文字列長より大きな値を指定した場合、InterSystems IRIS の現在の最大文字列長が使用されます。

2.3 SQL プロシージャ設定

ビジネス・サービスでストアド・プロシージャを呼び出す方法を制御する設定の詳細は、"プロシージャの詳細" を参照してください。

2.4 メッセージについて

外部データ・ソースから取得したデータを受信するために、カスタム・メッセージ・クラスを作成する必要はありません。既定では、SQL データはダイナミック・オブジェクト内にクエリの各列のプロパティと共に配置されます。このオブジェクトを記述する JSON 文字列は、`Ens.StreamContainer` のストリーム値として挿入されるので、その値をプロダクション経由で他のビジネス・ホストに送信できます。

カスタム・メッセージ・クラスを開発し、それを使用してデータをプロダクション経由で転送する場合は、プロパティ名が SQL クエリの列に完全に一致する必要があります。回避策として、SQL `As` キーワードを使用して、列の名前をプロパティ名に変更できます。メッセージ・クラスを定義したら、管理ポータルを開き、ビジネス・サービスの [**メッセージ・クラス**] 設定を使用して、そのメッセージ・クラスを選択します。

データ型とメッセージの詳細は、"メッセージの詳細" を参照してください。

3

SQL ビジネス・オペレーションの使用法

インターシステムズでは、別のビジネス・ホストから受信メッセージを受け取ると SQL を使用してデータをプロダクションに読み込む事前構築済みのビジネス・オペレーションを 2 つ提供しています。ビジネス・オペレーションを使用し、メッセージに応答してクエリを実行するには、`EnsLib.SQL.Operation.GenericOperation` をプロダクションに追加します。クエリではなく SQL プロシージャを実行する場合は、`EnsLib.SQL.Operation.ProcOperation` を使用します。

これらの組み込みのビジネス・オペレーションは、送信 SQL アダプタ (EnsLib.SQL.OutboundAdapter) を使用して外部データ・ソースにアクセスします。場合によっては、このアダプタとその結果の処理方法を詳細に制御するカスタム・ビジネス・オペレーションを構築しなければならないことがあります。

既定では、初めてビジネス・オペレーションのクエリで結果の複数行が返されるときに警告が生成されますが、この警告は次回は表示されません。この動作を変更するには、ビジネス・オペレーションの [Only Warn Once] 設定を無効にします。[Only Warn Once] 設定は、プロパティを設定できない場合の動作を制御するものではなく (初回に警告を生成するだけです)、受信ストリームが文字列プロパティに収まるように切り捨てられた場合の動作を制御するものであることに注意してください。

3.1 データ・ソース名の指定

ビジネス・オペレーションには、接続先のデータ・ソースを指定する設定があります。ビジネス・オペレーションを構成する場合は、この設定に適切な値を設定する必要があります。

DSN

> このデータ・ソース名は、接続先である外部データ・ソースを指定します。InterSystems IRIS® では、定義済みの InterSystems SQL ゲートウェイ接続、JDBC URL、およびオペレーティング・システムで構成した ODBC DSN の 3 種類の形式を自動的に区別します。
>
> 管理ポータルの **[システム管理]→[構成する]→[接続性]→[SQLゲートウェイ接続]** ページから構成された JDBC 接続または ODBC 接続の名前とこの名前が一致すると、その仕様のパラメータが使用されます。エントリが構成済みの接続名ではなく、そこにコロン (:) が使用されていると、その名前は JDBC URL と見なされます。それ以外の場合は ODBC DSN と見なされます。

このデータ・ソースがパスワードによって保護されている場合は、ユーザ名とパスワードを含むプロダクション認証情報を作成します。その後、これらの認証情報の ID と一致するように Credentials を設定します。詳細は、"その他の実行時設定の指定" を参照してください。

JDBC データ・ソースを使用している場合は、以下の設定も適用されます。

JGService

このオペレーションで使用する Java ゲートウェイ・サーバを制御する Java ゲートウェイ・サービスの構成名です。

重要　この設定は、JDBC と接続中の SQL ゲートウェイ接続を使用している場合でも、すべての JDBC データ・ソースで必要です。JDBC 接続が機能するには、タイプが EnsLib.JavaGateway.Service のビジネス・サービスが存在する必要があります。SQL アダプタに対して、この構成項目の名前を指定する必要があります。その構成済み設定を使用して、このアダプタが管理する JVM に接続します。

JDBCDriver

JDBC ドライバ・クラス名です。

名前付き SQL ゲートウェイ接続を DSN として使用する場合、この値の指定は任意です。ただし、値を指定すると、名前付き JDBC SQL ゲートウェイ接続のプロパティよりも優先して適用されます。

JDBCClasspath

JDBC ドライバ・クラス名のクラスパスです (Java ゲートウェイ・サービスで構成したクラスパス以外にも必要な場合)。

ConnectionAttributes

必要に応じて設定できる一連の SQL 接続属性オプション。ODBC の場合は、これらの形式は次のとおりです。

attr:val,attr:val

例：`AutoCommit:1`

JDBC の場合は、これらの形式は次のとおりです。

attr=val;attr=val

例：`TransactionIsolationLevel=TRANSACTION_READ_COMMITTED`

名前付き JDBC SQL ゲートウェイ接続を DSN として使用する場合、この値の指定は任意です。ただし、値を指定すると、名前付き JDBC SQL ゲートウェイ接続のプロパティよりも優先して適用されます。

3.2 クエリの指定

ビジネス・オペレーションの **[クエリ]** 設定の値は、ビジネス・オペレーションがプロダクション内の別のビジネス・ホストから受信メッセージを受け取ったときに実行される SQL 文字列 (`SELECT` や `INSERT INTO` など) です。この文字列には、置き換え可能なパラメータを表す SQL 標準の `?` を使用できます。

[入力パラメータ] 設定は、**[クエリ]** 設定で指定されたパラメータとして渡す値を指定するコンマ区切り文字列です。受信メッセージのプロパティとして受け取った値を渡すこともできます。パラメータをメッセージ・プロパティの値に置き換えるには、プロパティの名前を、先頭に `*` の文字を付けて指定します。この構文は、指定したプロパティが、記述されているダイナミック・オブジェクトのプロパティである限り、受信メッセージが `Ens.StringContainer` または `Ens.StreamContainer` 内の JSON 文字列であっても使用できます。

3.3 その他の実行時設定の指定

SQL ビジネス・オペレーションには、次のような追加の実行時設定があります。

Credentials

指定された DSN への接続を承認できるプロダクション認証情報の ID を識別します。プロダクション認証情報の作成方法は、"プロダクションの構成" を参照してください。

StayConnected

SQL 文の送信や ODBC ドライバ設定の変更などのコマンドを実行している間、接続を開いたままにするかどうかを指定します。

- この設定が 0 の場合は、コマンドを 1 つ実行終了するたびに SQL 送信アダプタの接続が直ちに切断されます。
- この値が正の場合、その値はコマンド完了後のアイドル時間 (秒) になります。アダプタは、このアイドル・タイムの経過後に接続を切断します。
- この設定が -1 の場合、アダプタは起動時に自動接続し、接続したままになります。

このドキュメントの "トランザクションの管理" の説明にあるようにデータベース・トランザクションを管理している場合は、StayConnected を 0 に設定しないでください。

ここに記載されていない設定については、"プロダクションの構成" を参照してください。

3.4 SQL プロシージャ設定

ビジネス・オペレーションでストアド・プロシージャを呼び出す方法を制御する設定の詳細は、"プロシージャの詳細" を参照してください。

3.5 応答メッセージについて

ビジネス・サービスで使用されるメッセージと同様に、ビジネス・オペレーションで取得されたデータを受け取るためにカスタム・メッセージ・クラスを開発する必要はありません。既定では、ダイナミック・オブジェクトを使用して、ビジネス・オペレーションを呼び出したビジネス・ホストにデータが返されます。このダイナミック・オブジェクトには、クエリの各列に対応するプロパティがあり、オブジェクトを記述する JSON 文字列は `Ens.StreamContainer` のストリーム値として挿入されます。ビジネス・オペレーションのクエリが結果の複数行を返す場合は、すべての行がこの JSON 文字列に格納されます。

ダイナミック・オブジェクトではなくカスタム応答メッセージを使用したい場合は、ビジネス・オペレーションの **[レスポンス・クラス]** 設定を使用してカスタム・メッセージ・クラスを指定します。このカスタム・クラスのプロパティは SQL クエリの列と一致する必要がありますが、SQL `As` キーワードを使用してこの要件を回避できます。列の値は、メッセージの対応するプロパティに配置されます。SELECT クエリが複数の行を返す場合、メッセージには最初の行の値だけが含まれます。

カスタム・メッセージ・クラスを使用するかどうかにかかわらず、ビジネス・オペレーションで UPDATE、INSERT、または DELETE のクエリを実行する場合、応答メッセージには 1 つのプロパティ `NumRowsAffected` だけが含まれます。

データ型と応答メッセージの詳細は、"メッセージの詳細" を参照してください。

4

メッセージの詳細

SQL ビジネス・ホストを使用するプロダクションのメッセージに関する基本的な情報は、"SQL ビジネス・サービスの使用法" または "SQL ビジネス・オペレーションの使用法" を参照してください。

4.1 受信ストリーム

プロダクションでメッセージ・クラスまたは応答クラスを定義しておらず、結果セット列または出力パラメータからの受信データがストリーム (CLOB または BLOB) である場合、ストリームはダイナミック・オブジェクトの対応するプロパティの値として追加されます。

同様に、カスタムのメッセージ・クラスまたは応答クラスがストリームである場合は、単に受信ストリームがそのプロパティに設定されます。ただし、プロパティのデータ型が文字列である場合の動作は、プロパティの MAXLEN を基準にした受信データの長さに加え、[Allow Truncating] 設定の値によっても異なります。

受信ストリーム・データの長さが文字列プロパティの MAXLEN を超えない場合、値は文字列としてプロパティに設定されます。受信データの長さがプロパティの MAXLEN を超える場合は、エラーをスローして終了するのが既定の動作です。ただし、[Allow Truncating] 設定が true に設定されている場合は、受信値の先頭の MAXLEN 分の文字がプロパティに文字列として設定され、警告が生成されます。既定では、警告が生成されるのは、特定のプロパティの値を切り捨てる必要がある場合の初回時だけです。ただし、[Only Warn Once] 設定が無効になっている場合は、データが切り捨てられるたびに警告が生成されます。[Only Warn Once] 設定は、プロパティを設定できない場合の動作を制御するものではなく (初回に警告を生成するだけです)、ビジネス・オペレーションのクエリが結果の複数行を返す場合の動作を制御するものであることに注意してください。

4.2 受信文字列

結果セット列または出力パラメータからの受信データが文字列であるにもかかわらず、メッセージまたは応答の対応するプロパティがストリームである場合、値は単純にストリームに書き込まれます。

5
プロシージャの詳細

ビジネス・サービスとビジネス・オペレーションは、どちらもストアド・プロシージャを使用して外部データ・ソースからデータを取得できます。このアプローチは、サービスまたはオペレーションのどちらを使用しているかにかかわらず同様です。`EnsLib.SQL.Service.ProcService` または `EnsLib.SQL.Operation.ProcOperation` をプロダクションに追加した後、[**プロシージャ**] 設定を使用し、プロシージャの名前を、各パラメータに対応する ? を含めて指定します。例えば、「`Sample.Stored_Procedure_Test(?,?)`」と入力できます。

クエリと異なり、ストアド・プロシージャでの呼び出しでは戻り値を持つことができます。入力パラメータを参照渡しすることができ、出力パラメータを存在させることが可能で、プロシージャ全体で 1 つの値を返すことができます。このような戻り値を処理し、入力値と出力値を区別するために、ビジネス・サービスまたはビジネス・オペレーションでは 3 つの設定を使用します。[**パラメータ**] (ビジネス・サービス) または [**入力パラメータ**] (ビジネス・オペレーション) には、プロシージャに渡される値が格納されます。[Output Parameter Names] には、パラメータで返される値に設定されたプロパティ名が格納されます。[**入力/出力**] では、プロシージャ呼び出しのパラメータが入力パラメータであるか、出力パラメータであるか、またはその両方であるかを識別します。

ビジネス・ホストがパラメータと戻り値をどのように処理するかを理解するには、[**入力/出力**] 設定が重要です。ビジネス・ホストは、パラメータのタイプを識別する 3 つの文字を受け入れます。"`i`" (入力)、"`o`" (出力)、および "`b`" (両方/ByRef) です。[**入力/出力**] 設定内のこれらの文字の順序は、プロシージャ呼び出しのパラメータの順序に対応します。例えば、プロシージャ呼び出しが `Sample.Stored_Procedure_Test(?,?,?)`、[**入力/出力**] の値が `ioo` である場合、プロシージャ呼び出しの最初のパラメータは [**パラメータ**] 設定の値を受け入れ、2 番目と 3 番目のパラメータはプロシージャによって返される値になります。これらのパラメータの戻り値は、[Output Parameter Names] 設定で指定した名前に割り当てられます。

プロシージャ全体で 1 つの値を返す場合は、[**入力/出力**] の最初の文字を `o` にする必要があります。すると、[Output Parameter Names] 設定の最初の名前に戻り値が割り当てられます。

この仕組みの詳しい例として、ビジネス・オペレーションが以下のような設定になっている場合を考えてみましょう。

設定	値
プロシージャ	`Sample.Stored_Procedure_Test(?,?,?,?)`
入力パラメータ	`*Value,x`
Output Parameter Names	`ReturnValue,Response,Stream,TruncatedStream`
入力/出力	`oiboo`

[**入力/出力**] の値がプロシージャ内の疑問符の数より多くなっている点に注意してください。これは戻り値も存在するからです。その戻り値は、[Output Parameter Names] の最初の文字列に対応しています。[**入力/出力**] の 2 番目の値は `i` です。これは、プロシージャの最初の値 (最初の ?) が入力パラメータであるからです。この場合、この値は受信メッセージの `Value` プロパティから取得します。2 番目のパラメータは入力と出力の両方です。つまり、ByRef で渡されま

す。値 x が送信されますが、これは [Output Parameter Names] の 2 番目の文字列にも対応しているため、応答メッセージの Response プロパティは、このパラメータで返される値に設定されます。他の 2 つのパラメータは完全に出力パラメータであるため、これらには何も渡されず、戻り値は応答メッセージの Stream プロパティと TruncatedStream プロパティに割り当てられます。

6

カスタム・ビジネス・サービス

この章では、カスタム・ビジネス・サービスを構築する方法について説明します。また、SQL 受信アダプタ (EnsLib.SQL.InboundAdapter) とプロダクション内でのその使用方法についても詳しく説明します。事前構築済みのビジネス・サービスを使用したい場合は、"SQL ビジネス・サービスの使用法" を参照してください。

6.1 全般的な動作

まず、アダプタに対して指定する詳細について理解しておくと役立ちます。EnsLib.SQL.InboundAdapter クラスには、以下のような項目の指定に使用する実行時設定が用意されています。

- アダプタが新規入力をチェックする頻度を制御する、ポーリング間隔
- アダプタの接続先である外部データ・ソース
- 必要に応じてデータ・ソースにユーザ名とパスワードを提供するプロダクション認証情報の ID
- 実行する SQL クエリ
- クエリで使用するオプションのパラメータ

通常、受信 SQL アダプタ (EnsLib.SQL.InboundAdapter) は、クエリを定期的に実行し、結果セットの行を反復し、関連するビジネス・サービスに 一度に 1 行ずつ渡します。ユーザが作成および構成するビジネス・サービスでは、この行を使用してプロダクションの他の部分と通信します。さらに具体的に説明します。

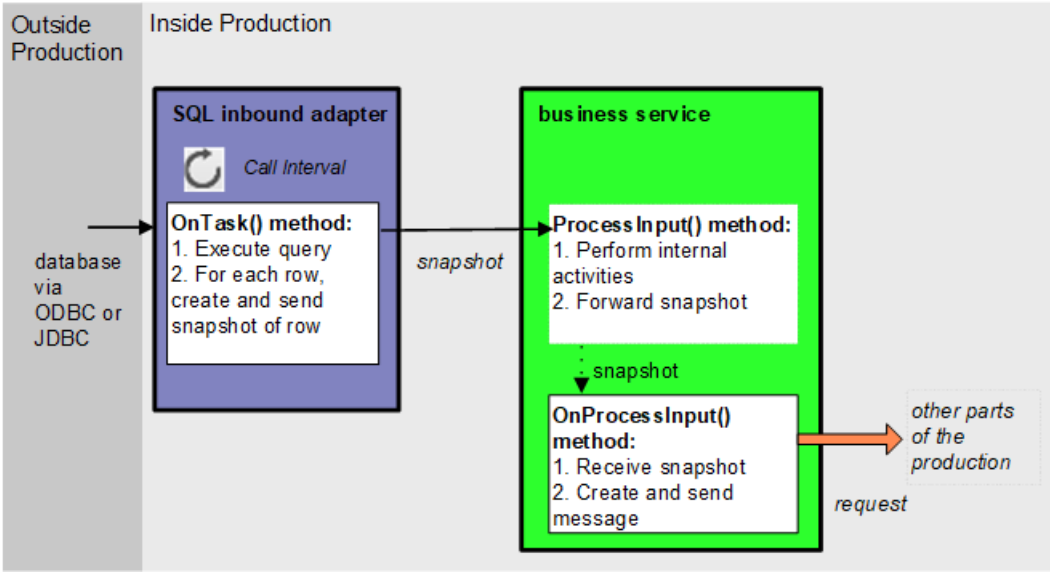
1. アダプタは、定期的にその OnTask() メソッドを実行します。このメソッドは、指定されたクエリを実行します。ポーリング間隔は CallInterval 設定によって決定されます。
2. クエリが行を返す場合は、アダプタが結果セットの行を反復し、行ごとに以下の手順を実行します。
   - この行が既に処理されているものの、変更されていない場合は、無視します。

     特定の行が既に処理されているかどうかを判別するために、アダプタは KeyFieldName 設定の情報を使用します。この章の “新規行のみの処理” を参照してください。
   - この行が既に処理されており (KeyFieldName 設定で処理済みと識別)、エラーが発生した場合は、次回再起動するまでこの行を無視します。
   - それ以外の場合は、アダプタが EnsLib.SQL.Snapshot クラスのインスタンスを構築し、行のデータをインスタンスに入れます。このインスタンスがスナップショット・オブジェクトです。このオブジェクトの詳細は、この章の “デフォルトのスナップショット・オブジェクトの使用法” を参照してください。

     その後アダプタは、関連するビジネス・サービス・クラスの内部 ProcessInput() メソッドを呼び出し、スナップショット・オブジェクトを入力として渡します。

3. ビジネス・サービス・クラスの内部 ProcessInput() メソッドが実行されます。このメソッドは、監視およびエラー・ログの作成などの基本のプロダクション・タスクを実行します。これらのタスクは、すべてのビジネス・サービスに必要です。ビジネス・サービス・クラスが継承するこのメソッドは、カスタマイズや上書きを行いません。
4. 次に、ProcessInput() メソッドがカスタムの OnProcessInput() メソッドを呼び出し、スナップショット・オブジェクトを入力として渡します。このメソッドの要件については、この後の“OnProcessInput() メソッドの実装”で説明します。

以下の図は、全体的なフローを示しています。



6.2 アダプタを使用するビジネス・サービスの作成

このアダプタをプロダクションで使用するには、ここに記載されているように新しいビジネス・サービスを作成します。後で、それをプロダクションに追加して、構成します。存在しなければ、適切なメッセージ・クラスを作成する必要もあります。"プロダクションの開発" の“メッセージの定義”を参照してください。

ビジネス・サービス・クラスの基本要件を以下に列挙します。

- ビジネス・サービス・クラスは Ens.BusinessService を拡張するものでなければなりません。
- クラスの ADAPTER パラメータは EnsLib.SQL.InboundAdapter である必要があります。
- クラスには OnProcessInput() メソッドを実装する必要があります。これについては、“OnProcessInput() メソッドの実装”で説明します。
- 必要に応じて、クラスで OnInit() を実装できます。“アダプタの初期化”を参照してください。
- その他のオプションと一般情報は、"プロダクションの開発" の“ビジネス・サービス・クラスの定義”を参照してください。

以下の例は、必要となる一般的な構造を示しています。

Class Definition

```
Class ESQL.NewService1 Extends Ens.BusinessService
{

Parameter ADAPTER = "EnsLib.SQL.InboundAdapter";

Method OnProcessInput(pInput As EnsLib.SQL.Snapshot, pOutput As %RegisteredObject) As %Status
{

   Quit $$$ERROR($$$NotImplemented)
}

}
```

注釈　スタジオには、上記のようなスタブの作成に使用できるウィザードが用意されています。このウィザードにアクセスするには、[**ファイル**] メニューで [**新規作成**] をクリックし、[**プロダクション**] タブをクリックします。ビジネス・サービスの作成を選択し、関連付ける受信アダプタとして EnsLib.SQL.InboundAdapter を選択します。これにより、ウィザードではこのアダプタに必要な固有の入力引数として EnsLib.SQL.Snapshot が使用されます。

6.3 OnProcessInput() メソッドの実装

カスタム・ビジネス・サービス・クラスにおいて、OnProcessInput() メソッドは以下のシグニチャを持つ必要があります。

```
Method OnProcessInput(pInput As EnsLib.SQL.Snapshot,
                      pOutput As %RegisteredObject) As %Status
```

ここで、pInput は、アダプタからこのビジネス・サービスに送信するスナップショット・オブジェクトです。これは、EnsLib.SQL.Snapshot のインスタンスです。また、pOutput は、メソッド・シグニチャに必要な汎用出力引数です。

OnProcessInput() メソッドは、以下の一部またはすべてを実行する必要があります。

1. ビジネス・サービスから送信されることになる要求メッセージのインスタンスを作成します。

   メッセージ・クラスの作成方法は、"プロダクションの開発" の "メッセージの定義" を参照してください。

2. 要求メッセージに対し、スナップショット・オブジェクトの値を使用して適切にプロパティを設定します。このオブジェクトはクエリによって返される個別行に対応します。詳細は、"デフォルトのスナップショット・オブジェクトの使用法" を参照してください。

3. ビジネス・サービスの適切なメソッドを呼び出して、要求をプロダクション内の宛先に送信します。具体的には、SendRequestSync()、SendRequestAsync()、または (あまり一般的ではない) SendDeferredResponse() を呼び出します。詳細は、"プロダクションの開発" の "要求メッセージの送信" を参照してください。

   これらの各メソッドは、ステータス (具体的には、%Status のインスタンス) を返します。

4. 必要に応じて、以前のアクションのステータスを確認し、そのステータスに基づいて対応します。

5. 必要に応じて、ビジネス・サービスが受信した応答メッセージを調査し、それに基づいて対応します。

6. 適切なステータスを返します。

簡単な例を以下に示します。

Class Member

```
Method OnProcessInput(pInput As EnsLib.SQL.Snapshot,
   pOutput As %RegisteredObject) As %Status
{

 set req=##class(ESQL.request).%New()
 set req.CustomerID=pInput.Get("CustomerID")
 set req.SSN=pInput.Get("SSN")
 set req.Name=pInput.Get("Name")
 set req.City=pInput.Get("City")
 set sc=..SendRequestSync("ESQL.operation",req,.pOutput)

 quit sc
}
```

この例では、EnsLib.SQL.Snapshot の Get() メソッドを使用して、個々の列のデータを取得します。詳細は、“デフォルトのスナップショット・オブジェクトの使用法”を参照してください。

6.4 デフォルトのスナップショット・オブジェクトの使用法

EnsLib.SQL.InboundAdapter と連携して動作するビジネス・サービス・クラスを作成すると、アダプタからカスタムの OnProcessInput() メソッドにスナップショット・オブジェクト (EnsLib.SQL.Snapshot のインスタンス) が渡されます。このインスタンスには、クエリから返されたデータのうちの 1 行分のデータが含まれています。通常、OnProcessInput() メソッドでは、このオブジェクトに含まれているデータを使用します。

注釈　EnsLib.SQL.Snapshot クラスには、複数行を管理するためのプロパティおよびメソッドが用意されています。ただし、複数行のスナップショットは、このドキュメントで後述するように、送信アダプタを使用する操作にのみ関連しています。

以下に、カスタムの OnProcessInput() メソッド内で一般に使用されるメソッドについて説明します。

Get()

```
method Get(pName As %String, pRow=..%CurrentRow) returns %String
```

現在行 (この場合の唯一の行) で名前 pName を持つ列の値を返します。

GetColumnId()

```
method GetColumnId(pName As %String) returns %Integer
```

名前 pName を持つ列の順序位置を返します。不慣れなテーブルを使用する際に、このメソッドは役に立ちます。

GetData()

```
method GetData(pColumn As %Integer, pRow=..%CurrentRow) returns %String
```

現在行 (この場合の唯一の列) で、pColumn で指定されている位置にある列の値を返します。左から順に、最初の列が 1、2 番目の列が 2 などとなります。

GetColumnName()

```
method GetColumnName(pColumn As %Integer = 0)
```

pColumn で指定されている位置にある列の名前を返します。

GetColumnSize()

```
method GetColumnSize(pColumn As %Integer = 0)
```

pColumn で指定されている位置にあるデータベース・フィールドのサイズ (文字数で表した幅) を返します。

GetColumnType()

```
method GetColumnType(pColumn As %Integer = 0)
```

pColumn で指定されている位置にある列の SQL タイプを返します。例えば、VARCHAR、DATE、または INTEGER などです。

注釈　SQL タイプ名は、データベースのベンダごとに異なります。

以下に、Get() メソッドを使用してスナップショットからデータを抽出し、抽出したデータを使用して要求メッセージを設定する方法を示します。

```
set req=##class(ESQL.request).%New()
set req.CustomerID=pInput.Get("CustomerID")
set req.SSN=pInput.Get("SSN")
set req.Name=pInput.Get("Name")
set req.City=pInput.Get("City")
```

6.5 アダプタの初期化

受信アダプタを初期化するには、カスタム・ビジネス・サービス・クラスの OnInit() メソッドをカスタマイズします。このメソッドはビジネス・ホストの起動時に実行されます。デフォルトでは、このメソッドは何も実行しません。

```
Method OnInit() As %Status
```

アダプタを初期化する最も一般的な理由は、クエリのパラメータとして使用する値を初期化することです。これについては、“受信クエリの指定”で説明します。以下に、関連するメソッドと例を示します。

6.5.1 永続値の初期化

EnsLib.SQL.InboundAdapter には、ビジネス・サービスの再起動と再起動間に保存される永続値を初期化する以下のメソッドが用意されています。

```
ClassMethod InitializePersistentValue
            (pConfigName As %String,
             pPersistentValueName As %String = "%LastKey",
             pNewValue As %String)
             As %String
```

このメソッドを使用して、アダプタに関連付けられている名前と値のペアを初期化し、その名前をクエリのパラメータに使用します。このメソッドでは、指定された永続的な名前と値のペアの現在の値を調べます。値が現時点で NULL の場合は、それを pNewValue と等しくなるように設定します。

デフォルトでは、名前を省略すると、メソッドにより、永続的な名前と値のペア **&%LastKey** が初期化されます。%LastKey には、アダプタによって最後に処理された行の IDKey 値が含まれています。

場合によっては、代わりに InitializeLastKeyValue() が必要な場合があります。このメソッドは、アダプタの一時プロパティ **%LastKey** を初期化します。このプロパティは、ビジネス・サービスが起動するたびにリセットされます。関連する SetPersistentValue() メソッドおよび GetPersistentValue() メソッドについては、**EnsLib.SQL.InboundAdapter** のクラスに関するドキュメントも参照してください。

6.5.2 例

&%LastKey 永続値を初期化するには、ビジネス・サービスの OnInit() メソッドをカスタマイズして以下を含めます。

Class Member

```
Method OnInit() As %Status
{
   #; initialize persistent last key value
   Do ..Adapter.InitializePersistentValue(..%ConfigName,,0)
   Quit $$$OK
}
```

&TopSales 永続値を初期化するには、ビジネス・サービスの OnInit() メソッドをカスタマイズして以下を含めます。

Class Member

```
Method OnInit() As %Status
{
   #; must initialize so the query can do a numeric comparison
   Do ..Adapter.InitializePersistentValue(..%ConfigName,"TopSales",0)
   Quit $$$OK
}
```

6.6 ビジネス・サービスの追加と構成

ビジネス・サービスをプロダクションに追加するには、管理ポータルを使用して以下の操作を行います。

1. カスタム・ビジネス・サービス・クラスのインスタンスをプロダクションに追加します。
2. ビジネス・サービスを有効化します。
3. PoolSize 設定を 1 にします。

   PoolSize が 1 より大きい場合、アダプタは多数のレコードを 2 回処理します。
4. 特定の外部データ・ソースと通信を行うためのアダプタを構成します。構成設定の詳細は、"SQL ビジネス・サービスの使用法" を参照してください。
5. プロダクションを実行します。

6.7 新規行のみの処理

同じデータに対してクエリを実行し続けるのは望ましくない場合がよくあるため、EnsLib.SQL.InboundAdapter には、処理済みの行の追跡に使用できるツールがいくつか用意されています。ここでは、これらのツールおよびその実際の使用方法について説明します。

注意　　PoolSize 設定は必ず 1 に設定します。1 より大きい場合、アダプタは多数のレコードを 2 回処理します。

6.7.1 利用可能なツール

EnsLib.SQL.InboundAdapter には、処理済みの行を追跡するための以下のツールが用意されています。

- KeyFieldName 設定を指定すると、アダプタがどの行を処理したかを示すデータが InterSystems IRIS グローバルに追加されます。この設定は、時間が経過しても再利用されない値を含むフィールドを参照します。このフィールドは、

クエリから返される結果セットに含まれている必要があります。アダプタは、そのフィールドのデータを使用して、行が以前に処理されたことがあるかどうかを評価します。

注釈　行を削除すると、InterSystems IRIS によって、その行の KeyFieldName 値が、処理される行を追跡するグローバルから削除されます。後でその行を同じ KeyFieldName 値で戻すと、InterSystems IRIS ではその行が再び処理されます。

- アダプタには、永続値、&%LastKey が用意されています。この永続値には、最後に処理された行の Key Field Name の値が含まれています。この特別な永続値は、ビジネス・サービスの再起動時に保存されます。
- アダプタの一時プロパティ %LastKey には、アダプタが最後に処理した行の KeyFieldName の値が格納されます。このアダプタ・プロパティは、関連するビジネス・サービスを再起動するたびに作成されます。

最後とその前の 2 つのオプションが実際に役立つのは、新規行ごとに単調に値が増加するフィールドを KeyFieldName が参照している場合に限られます。“アダプタの初期化”も参照してください。

6.7.2 行を再処理しないための実用的な方法

同じデータを再処理しないようにするための実用的な方法は、以下の 3 つです。

- アダプタの KeyFieldName 設定を使用します。この設定を指定すると、時間が経過しても再利用されない値を含むフィールドが参照されます。このフィールドは、クエリから返される結果セットに含まれている必要があります。KeyFieldName を指定すると、アダプタはこのフィールド内のデータに基づいて、行が既に処理されているかどうかを評価します。SELECT に擬似フィールド %ID を指定し、このフィールドを KeyFieldName として指定する場合は、KeyFieldName を ID として指定する必要があり、%（パーセント記号）は省略する必要があることに注意してください。

  デフォルトは ID です。

  例えば、Query を以下のように指定できます。

  ```
  SELECT ID,Name,Done from Sample.Person
  ```

  そして、KeyFieldName を以下のように指定します。

  ```
  ID
  ```

  InterSystems IRIS では、ポーリング・サイクルごとに多数の行が選択されますが、選択された行のうち新規のものは少ないので、この方法は非効率的です。

- 特別な永続値 &%LastKey（またはアダプタの一時プロパティ %LastKey）を参照するクエリ・パラメータを使用するクエリを使用します。例えば、Query を以下のように指定できます。

  ```
  SELECT ID,Name,Done from Sample.Person WHERE ID>?
  ```

  次に、Parameters を以下のように指定できます。

  ```
  &%LastKey
  ```

  “アダプタの初期化”も参照してください。

- クエリの実行後、ソース・データを削除または更新し、クエリから同じ行が返されないようにします。そのためには、DeleteQuery 設定を使用します。デフォルトでは、EnsLib.SQL.InboundAdapter は、メイン・クエリ（Query 設定）の実行後、メイン・クエリで返された行ごとに 1 回 DeleteQuery を実行します。

  このクエリには、置換可能なパラメータ（疑問符）が厳密に 1 つのみ含まれている必要があります。アダプタはこのパラメータを KeyFieldName が指定する値に置換します。

このクエリでは、ソース・データを削除または更新し、アダプタのメイン・クエリによって同じ行が選択されないようにすることができます。

例えば、Query を以下のように指定できます。

```
SELECT ID,Name,Done from Sample.Person WHERE Done=0
```

そして、DeleteQuery を以下のように指定します。

```
UPDATE Sample.Person SET Done=1 WHERE ID=?
```

6.8 行の再処理

多くの場合、SQL 受信アダプタで以前に処理された行の変更点を通知する必要があります。そのための最も簡単な方法は以下のとおりです。

- 関連するテーブルに、指定された行が変更されたかどうかを記録する列を追加します。
- 更新トリガをデータ・ソースにインストールし、適宜その列を更新します。
- アダプタが使用するクエリ内で、この列の値を使用し、指定された行を選択するかどうかを判断します。

注意　　PoolSize 設定は必ず 1 に設定します。1 より大きい場合、アダプタは多数のレコードを 2 回処理します。

6.9 クエリ設定の使用例

ここでは、上記の設定を使用する例を示します。

6.9.1 例 1 : KeyFieldName の使用

最も単純な例では、Customer テーブルから行を選択します。このテーブルには主キーである CustomerID フィールドがあります。これは、新しいレコードごとに自動的に増加する整数です。重要なのは新規行のみであるため、このフィールドを KeyFieldName として使用します。プロダクションでは、アダプタに以下の設定を使用します。

設定	値
Query	SELECT * FROM Customer
DeleteQuery	なし
KeyFieldName	CustomerID
Parameters	なし

プロダクションが起動すると、アダプタが Customer テーブルのすべての行を自動的に選択して処理します。トレースが有効であれば 1 行が処理されるたびにトレース・メッセージが作成され、ログ記録が有効であればイベント・ログにエントリが追加されます。このメッセージは以下のようなテキストです。

```
Processing row '216'
```

ここで、'216' は、処理される行の CustomerID です。

プロダクションの起動後、各ポーリング・サイクル時に、アダプタはすべての行を選択しますが、CustomerID フィールドに新しい値が含まれている行のみを処理します。

6.9.2 例 2 : &%LastKey または %LastKey の使用

この例は、上記のバリエーションです。この場合は、メイン・クエリが行のサブセットを選択します。すべての行を選択するよりも効率的です。

設定	値
Query	SELECT * FROM Customer WHERE ID>?
DeleteQuery	なし
KeyFieldName	CustomerID
Parameters	&%LastKey

各ポーリング・サイクル時に、アダプタは &%LastKey の値を決定し、その値を SQL に渡します。

“アダプタの初期化”も参照してください。

注釈　一連の行をアダプタが選択するときには、KeyFieldName で指定された順序で各行が処理されることもあれば、されないこともあります。例えば、あるポーリング・サイクルで、CustomerID が 101、102、103、104、および 105 の行を選択するとします。ただし、顧客 103 を (顧客 105 の代わりに) 最後に処理するとします。このポーリング・サイクル終了後の &%LastKey の値は 103 です。したがって、次のサイクルでは、アダプタは再処理はしないものの、顧客 104 および 105 を再度選択します (この方が、上記例のようにすべての行を再度選択するよりも効率的です)。強制的にアダプタが特定の順序で行を処理するようにするには、以下のように、クエリ内に ORDER BY 節を追加します。

```
SELECT * FROM Customer WHERE ID>? ORDER BY CustomerID
```

この場合、&%LastKey の値は常に最大の CustomerID に設定されるので、複数回選択される行はなくなります。

6.9.3 例 3 : DeleteQuery の使用

この例では、Customer テーブルに Done というフィールドがあります。このフィールドには、アダプタが所定の行を以前に選択したことがあるかどうかを記録できます。

設定	値
Query	SELECT * FROM Customer WHERE Done=0
DeleteQuery	UPDATE Customer SET Done=1 WHERE CustomerID=?
KeyFieldName	CustomerID
Parameters	なし

上記の例と同様、この例では、各行が選択されるのは 1 度だけです。

Tipヒント　削除クエリは各行が処理されるたびに 1 回実行されるので、このクエリを使用すると処理が遅くなる可能性があります。定期的な間隔でバッチ削除 (バッチ更新) を実行するとより効率的です。

6.9.4 例 4 : 複合キーの使用

テーブルの複数のフィールドを主キーとしてまとめて処理する必要がある場合が多々あります。例えば、Month フィールドや Year フィールドを含む統計テーブルがあるとします。クエリでは、アダプタの一意のキーとして月と年を一緒に処理する必要があるとします。このような場合は、関連するフィールドを連結し、AS 節を使用して複合フィールドの別名を提供するクエリを使用します。

例えば、SQL*Server の場合、以下のように始まるクエリを使用できます。

```
SELECT Stats, Year||Month as ID ...
```

アダプタ内の結果セットに ID というフィールドが含まれるようになります。このフィールドを KeyFieldName に使用できます。

注釈　　連結の構文は、作業しているデータベースによって異なります。

6.9.5 例 5 : KeyFieldName の指定なし

場合によっては、KeyFieldName を使用したくないこともあります。KeyFieldName が NULL の場合、アダプタは行を区別せず、エラーがある行や既に正常に処理された行をスキップしません。

以下に例を示します。

設定	値
Query	`Select * from Cinema.Film Where TicketsSold>?`
DeleteQuery	なし
KeyFieldName	なし
Parameters	&TopSales（これは、ビジネス・サービスの OnProcessInput() メソッド内で定義される、TopSales という特殊な永続値を参照します）

ビジネス・サービスは以下のとおりです。

```
Class Test.SQL.TopSalesService Extends Ens.BusinessService
{

Parameter ADAPTER = "EnsLib.SQL.InboundAdapter";

Parameter REQUESTCLASSES As %String = "EnsLib.SQL.Snapshot";

Method OnInit() As %Status
{
   #; must initialize so the query can do a numeric comparison
   Do ..Adapter.InitializePersistentValue(..%ConfigName,"TopSales",0)
   Quit $$$OK
}

Method OnProcessInput(pInput As EnsLib.SQL.Snapshot,
   Output pOutput As Ens.Response) As %Status
{
   Kill pOutput Set pOutput=$$$NULLOREF

   for j=1:1:pInput.ColCount {
   }
   for i=1:1:pInput.RowCount {
      for j=1:1:pInput.ColCount {
      }
   }
   Set tSales=pInput.Get("TicketsSold")
   Set:tSales>$G($$$EnsStaticAppData(
..%ConfigName,"adapter.sqlparam","TopSales")) ^("TopSales")=tSales
   Quit $$$OK
```

```
}
}
```

6.10 その他の実行時設定の指定

EnsLib.SQL.InboundAdapter には、以下のその他の実行時設定が用意されています。

呼び出し間隔

アダプタのポーリング間隔を秒単位で指定します。これは、このアダプタが入力をチェックする頻度を指定します。

ポーリング時にアダプタが入力を検出した場合は、適切な InterSystems IRIS オブジェクトを作成し、関連するビジネス・サービスにそのオブジェクトを渡します。一度に複数の入力が検出された場合は、検出されなくなるまで、アダプタが入力を順に処理します。アダプタは、検出した入力の項目ごとに 1 つの要求をビジネス・サービスに送信します。その後、アダプタはポーリング間隔が経過するのを待ってから、再び入力をチェックします。プロダクションが実行中であり、ビジネス・サービスが有効化され、アクティブになるようにスケジュールされている場合、このサイクルは常に継続されます。

各入力の処理の間の CallInterval 期間をアダプタで延期できるようにビジネス・サービスのプロパティを設定することが可能です。詳細は、"プロダクションの開発" を参照してください。

CallInterval のデフォルト値は 5 秒です。最小値は 0.1 秒です。

Credentials

指定された DSN への接続を承認できるプロダクション認証情報の ID を識別します。プロダクション認証情報の作成方法は、"プロダクションの構成" を参照してください。

StayConnected

SQL 文の送信や ODBC ドライバ設定の変更などのコマンドを実行している間、接続を開いたままにするかどうかを指定します。

- この設定が 0 の場合は、SQL コマンドを 1 つ実行終了するたびにアダプタの接続が直ちに切断されます。
- この設定が -1 の場合、アダプタは起動時に自動接続し、接続したままになります。この値は、例えば、データベース・トランザクションを管理している場合などに使用します。これについては、このドキュメントの “トランザクションの管理” で説明します。

この設定には正の値も使用できますが、その場合の値は SQL コマンド実行後のアイドル時間 (秒) になるので、ポーリングによって機能する SQL 受信アダプタでは役に立ちません (アイドル時間がポーリング間隔 [CallInterval] よりも長い場合、アダプタは常に接続された状態のままです。アイドル時間がポーリング間隔よりも短い場合は、ポーリング間隔ごとにアダプタの切断と再接続が発生します。つまり、アイドル時間には実質的に意味がなくなります)。

ここに記載されていない設定については、"プロダクションの構成" を参照してください。

6.11 受信アダプタで以前に処理された行のリセット

開発およびテスト中に、以前実施したテストを繰り返すために、ビジネス・サービスのアダプタをリセットすることが役に立つ場合があります。そのためには、ここで説明する以下のメソッドのいずれかを使用します。これらは、EnsLib.SQL.InboundAdapter で継承されるクラス・メソッドです。

注意　　動作中のプロダクションでは、通常、これらのメソッドは使用しません。

ClearRuntimeAppData()

```
ClassMethod ClearRuntimeAppData(pConfigName As %String)
```

指定された構成名を持つビジネス・サービスの実行時データをすべて消去します。アダプタの %ConfigName プロパティを使用すると、現在構成されているビジネス・サービスの名前にアクセスできます。このデータは、ビジネス・サービスが起動するたびに自動的に消去されます。

ClearStaticAppData()

```
ClassMethod ClearStaticAppData(pConfigName As %String)
```

構成名で指定されているビジネス・サービスの静的データを消去します。このデータには、永続的な最後のキー値など、アダプタに関連付けられている永続値すべてが含まれています。

ClearAllAppData()

```
ClassMethod ClearAllAppData(pConfigName As %String)
```

このメソッドは ClearRuntimeAppData() および ClearStaticAppData() クラスのメソッドを実行します。

7

カスタム・ビジネス・オペレーション

この章では、カスタム・ビジネス・オペレーションを構築する方法について説明します。また、SQL 送信アダプタ (EnsLib.SQL.OutboundAdapter) とプロダクション内でのその使用方法についても詳しく説明します。事前構築済みのビジネス・オペレーションを使用したい場合は、"SQL ビジネス・オペレーションの使用法" を参照してください。

7.1 デフォルトの動作

プロダクション内で、送信アダプタは、ユーザが作成および構成するビジネス・オペレーションに関連付けられます。このビジネス・オペレーションはプロダクション内からメッセージを受信し、メッセージ・タイプを調べ、適切なメソッドを実行します。このメソッドは、通常、関連するアダプタのメソッドを実行します。

SQL 送信アダプタ (EnsLib.SQL.OutboundAdapter) には、接続先となるデータ・ソースとそのデータ・ソースに必要なログイン詳細の指定に使用する設定が用意されています。また、以下のような一般的な SQL アクティビティを実行するメソッドも用意されています。

- クエリの実行
- ストアド・プロシージャの実行
- 挿入、更新、および削除の実行

7.2 アダプタを使用するビジネス・オペレーションの作成

EnsLib.SQL.OutBoundAdapter を使用するビジネス・オペレーションを作成するために、新しいビジネス・オペレーション・クラスを作成します。後で、それをプロダクションに追加して、構成します。

存在しなければ、適切なメッセージ・クラスを作成する必要もあります。"プロダクションの開発" の "メッセージの定義" を参照してください。

ビジネス・オペレーション・クラスの基本要件を以下に列挙します。

- ビジネス・オペレーション・クラスは、Ens.BusinessOperation を拡張するものでなければなりません。
- ADAPTER パラメータは EnsLib.SQL.OutboundAdapter である必要があります。
- INVOCATION パラメータは、使用する呼び出しスタイルを指定する必要があります。以下のいずれかを使用します。

  - Queue は、メッセージが 1 つのバックグラウンド・ジョブ内で作成され、元のジョブが解放された段階でキューに配置されます。その後、メッセージが処理された段階で、別のバックグラウンド・ジョブがそのタスクに割り当てられます。これは最も一般的な設定です。
  - InProc は、メッセージが、作成されたジョブと同じジョブで生成、送信、および配信されることを意味します。このジョブは、メッセージが対象に配信されるまで送信者のプールに解放されません。これは特殊なケースのみに該当します。

- クラスでは、少なくとも 1 つのエントリを含むメッセージ・マップを定義します。メッセージ・マップは、以下の構造を持つ XData ブロック・エントリです。

```
XData MessageMap
{
<MapItems>
  <MapItem MessageType="messageclass">
    <Method>methodname</Method>
  </MapItem>
  ...
</MapItems>
}
```

- クラスでは、メッセージ・マップ内で名前が付けられたすべてのメソッドを定義します。これらのメソッドは、メッセージ・ハンドラと呼ばれます。通常、これらのメソッドは、ビジネス・オペレーションの Adapter プロパティのプロパティおよびメソッドを参照します。
- その他のオプションと一般情報は、"プロダクションの開発" の “ビジネス・オペレーション・クラスの定義” を参照してください。

以下の例は、必要となる一般的な構造を示しています。

Class Definition

```
Class ESQL.NewOperation1 Extends Ens.BusinessOperation
{
Parameter ADAPTER = "EnsLib.SQL.OutboundAdapter";

Parameter INVOCATION = "Queue";

Method SampleCall(pRequest As Ens.Request, Output pResponse As Ens.Response) As %Status
{
  Quit $$$ERROR($$$NotImplemented)
}

XData MessageMap
{
<MapItems>
  <MapItem MessageType="Ens.Request">
    <Method>SampleCall</Method>
  </MapItem>
</MapItems>
}
}
```

注釈　スタジオには、上記のようなスタブの作成に使用できるウィザードが用意されています。このウィザードにアクセスするには、[ファイル] メニューで [新規作成] をクリックし、[プロダクション] タブをクリックします。ビジネス・オペレーションの作成を選択し、関連付ける送信アダプタとして EnsLib.SQL.OutboundAdapter を選択します。

7.3 SQL オペレーションを実行するメソッドの作成

EnsLib.SQL.OutboundAdapter で使用するビジネス・オペレーション・クラスを作成する場合の最大のタスクは、通常、メッセージ・ハンドラ、つまり、各種 SQL オペレーションを実行するメソッドの記述です。通常、これらのメソッドは、ビジネス・オペレーションの Adapter プロパティのプロパティおよびメソッドを参照します。以下に例を示します。

```
 set tSC = ..Adapter.ExecuteUpdate(.numrows,sql)
```

メソッドは以下のようになります。

```
/// Insert into NewCustomer table
Method Insert(pReq As ESQL.request, Output pResp As ESQL.response1) As %Status
{
 kill pResp
 set pResp=$$$NULLOREF

 set sql="insert into NewCustomer (Name,SSN,City,SourceID) values (?,?,?,?)"

 //perform the Insert
 set tSC = ..Adapter.ExecuteUpdate
   (.nrows,sql,pReq.Name,pReq.SSN,pReq.City,pReq.CustomerID)

 //create the response message
 set pResp=##class(ESQL.response1).%New()
 set pResp.AffectedRows=nrows

 if 'tSC write " failed ",tSC quit tSC
 quit 1
}
```

これらのメソッドを作成するには、EnsLib.SQL.OutboundAdapter クラスのメソッドとプロパティに精通している必要があります。これらのツールについては、“SQL のアダプタ・メソッドの作成”の章で詳しく説明します。

7.4 1 件のメッセージによる複数の SQL 文の処理

アダプタ構成は、ビジネス・オペレーションで受信したメッセージ 1 件について 1 つの SQL 文を実行する単純なケースを扱うように設計されています。 指定した 1 つのメッセージで複数の SQL 文をビジネス・オペレーションで実行するには、OnMessage() メソッドで次のような形式を使用します。

```
OnMessage(..)
{
    Set tStayConn=..Adapter.StayConnected
    Set ..Adapter.StayConnected=-1

    //... your ..Adapter SQL Operations here...

   Set ..Adapter.StayConnected=tStayConn
   If 'tStayConn&&..Adapter.Connected Do ..Adapter.Disconnect()
   Quit tSC
}
```

7.5 ビジネス・オペレーションの追加と構成

ビジネス・オペレーションをプロダクションに追加するには、管理ポータルを使用して以下の操作を行います。

1. カスタム・ビジネス・オペレーション・クラスのインスタンスをプロダクションに追加します。
2. ビジネス・オペレーションを有効化します。

3. 特定の外部データ・ソースと通信を行うためのアダプタを構成します。これらの構成設定の詳細は、"SQL ビジネス・オペレーションの使用法" を参照してください。
4. プロダクションを実行します。

8

SQL のアダプタ・メソッドの作成

この章では、EnsLib.SQL パッケージに用意されているツールを使用して、SQL タスクを実行するアダプタ・メソッドを記述する方法について説明します。通常、送信アダプタを使用する場合にこのようなメソッドを記述します。

8.1 概要および状況

さまざまな場合に、SQL タスクを実行するメソッドを記述する必要があります。最も一般的な場合は、以下のとおりです。

- SQL 送信アダプタを使用する場合は、メッセージ・ハンドラを記述し、このメソッドをアダプタのメッセージ・マップに追加します。そうすると、例えば、ビジネス・オペレーションが特定のタイプのメッセージを受信した場合に、メッセージ・ハンドラが特定のテーブルにレコードを追加できます。
- ビジネス・ホストのスタートアップまたはティアダウンをカスタマイズする場合は、カスタムの OnInit() または OnTearDown() メソッドによって特定のテーブルを初期化またはクリーンアウトできます。

このようなタスクを実行するには、カスタム・メソッドで SQL 受信アダプタと送信アダプタのメソッドを使用します。どちらも、EnsLib.Common クラスからメソッドのコア・セットを継承しています。これらのメソッドでは、クエリの実行、ストアド・プロシージャの実行、レコードの挿入などができます。

8.2 パラメータの使用

クエリ、更新、またはプロシージャを実行する際にパラメータを使用する場合は、使用している ODBC ドライバに関する情報を入手する必要があります。入手する情報は以下のとおりです。

- このドライバが ODBC SQLDescribeParam 関数をサポートしているかどうか。ほとんどのドライバはこの関数をサポートしています。
  - サポートしている場合は、SQL アダプタ・メソッド ExecuteQuery() および ExecuteUpdate() を使用できます。これらの各メソッドでは、任意の数のパラメータ名を受け入れ、SQLDescribeParam を呼び出し、その結果得た情報を使用して自動的にまた適切にこれらのパラメータをバインドします。
  - サポートしていない場合は、代替メソッド ExecuteQueryParmArray() および ExecuteUpdateParmArray() を使用する必要があります。このような場合は、パラメータとそのすべての属性を含む多次元配列を作成して渡す必要があります。
- このドライバが ODBC SQLDescribeProcedureColumns 関数をサポートしているかどうか。ほとんどのドライバはこの関数をサポートしています。

- サポートしている場合は、SQL アダプタ・メソッド ExecuteProcedure() を使用できます。このメソッドでは、任意の数のパラメータ名を受け入れ、SQLDescribeProcedureColumns を呼び出し、その結果得た情報を使用して自動的にまた適切にこれらのパラメータをバインドします。
- サポートしていない場合は、代替メソッド ExecuteProcedureParmArray() を使用する必要があります。このような場合は、パラメータとそのすべての属性を含む多次元配列を作成して渡す必要があります。

  ドライバが SQLDescribeProcedureColumns をサポートしていない場合は、使用する各パラメータが入力、出力、または入出力のどのタイプのパラメータかを指定する必要もあります。

8.2.1 パラメータ属性

ODBC ドライバが SQLDescribeParam または SQLDescribeProcedureColumns 関数をサポートしていない場合、SQL 文でパラメータを使用するには、パラメータとそのすべての関連属性を含む InterSystems IRIS® 多次元配列を作成する必要があります。InterSystems IRIS はこれらの値を使用して、各パラメータを適切にバインドする方法をドライバに問い合わせます。

- SQL データ型 ― InterSystems IRIS では、通常、整数 (SqlType 値) によって表現されます。InterSystems IRIS インクルード・ファイル **EnsSQLTypes.inc** には、一般に使用される値の定義が含まれています。以下にいくつか例を示します。
  - 1 は SQL_CHAR を表します。
  - 4 は SQL_INTEGER を表します。
  - 6 は SQL_FLOAT を表します。
  - 8 は SQL_DOUBLE を表します。
  - 12 は SQL_VARCHAR を表します。

  このインクルード・ファイルには、SqlDB2BLOB や SqlDB2CLOB などの拡張データ型もリストされています。InterSystems IRIS では、これらのデータ型もサポートしています。

  しかし、データベース・ドライバのドキュメントを参照して、ドライバが使用する値に、InterSystems IRIS が認識しない非標準の値が含まれていないか確認してください。
- 精度 ― 数値パラメータの場合は、通常、パラメータのデータ型で使用される最大桁数を意味します。例えば、CHAR(10) 型のパラメータの場合、精度は 10 です。数値以外のパラメータの場合は、通常、パラメータの最大長を意味します。
- スケール ― 数値パラメータの場合は、小数点の右側に格納できる最大桁数を意味します。数値以外のパラメータには使用できません。

8.2.2 InterSystems IRIS の多次元配列でのパラメータの指定

ExecuteQueryParmArray()、ExecuteUpdateParmArray()、および ExecuteProcedureParmArray() メソッドを使用するには、最初に、パラメータとその値を保持する InterSystems IRIS 多次元配列を作成します。次に、メソッド・シグニチャに示されるような引数リストで、この配列を使用します。この配列には任意の数のパラメータを含めることができ、配列の構造は以下のようにする必要があります。

ノード	コンテンツ
arrayname	パラメータ数を示します。
arrayname(integer)	integer が指定する位置にあるパラメータの値。
arrayname(integer,"SqlType")	このパラメータの SqlType (必要な場合)。これは SQL データ型に対応する数字です。オプションについては、前の節を参照してください。詳細は、“SqlType と CType の値”を参照してください。
arrayname(integer,"CType")	このパラメータのローカル InterSystems IRIS タイプ (必要な場合)。これは SQL データ型に対応する数字です。オプションについては、前の節を参照してください。詳細は、“SqlType と CType の値”を参照してください。
arrayname(integer,"Prec")	このパラメータの精度 (必要な場合)。前の節を参照してください。デフォルトは 255 です。
arrayname(integer,"Scale")	このパラメータのスケール (必要な場合)。前の節を参照してください。デフォルトは 0 です。
arrayname(integer,"IOType")	プロシージャのフラグを上書きする必要がある場合の、このパラメータの IOType。これは、ExecuteProcedureParmArray() メソッドのみで使用されます。 · 1 は入力パラメータを表します。 · 2 は入出力パラメータを表します。 · 4 は出力パラメータを表します。
arrayname(integer,"SqlTypeName")	ドライバからパラメータ値を取得するための呼び出しと、SqlType 添え字のみが与えられている場合の CType の計算で使用されます。
arrayname(integer,"LOB")	このパラメータが“ラージ・オブジェクト”であるかどうかを指定するブーリアン値。
arrayname(integer,"Bin")	このパラメータが文字データではなくバイナリ・データを含むかどうかを指定するブーリアン値。

重要　　パラメータ配列を使用するクエリを複数実行する場合は、各クエリの実行前にパラメータ配列を破棄して、再作成します。

ExecuteQueryParmArray()、ExecuteUpdateParmArray()、および ExecuteProcedureParmArray() メソッドは、最初に、指定されたパラメータ配列に記述子の添え字が含まれているかどうかをチェックします (InterSystems IRIS では特に、最初のパラメータに "CType" または "SqlType" の添え字がないかチェックします)。次に、以下の手順を実行します。

· 配列に記述子の添え字が含まれていない場合、メソッドは必要に応じて ODBC 関数の SQLDescribeParam または SQLDescribeProcedureColumns 関数のどちらかを呼び出し、その関数によって返される値を使用します。

· 配列に記述子の添え字が含まれている場合、メソッドはその添え字を使用します。

また、ExecuteProcedure() による DescribeProcedureColumns の呼び出しを阻止することもできます (デフォルトでは呼び出されます)。そのためには、pIO 引数の末尾にアスタリスク (*) を付加します。この章の“ストアド・プロシージャの実行”を参照してください。

8.2.2.1 SqlType と CType の値

任意のパラメータで "SqlType" と "CType" の両方の添え字を指定できます。そのほうが "SqlType" の添え字のみを使用するよりも簡単です。

任意の指定パラメータに対して、使用値が以下のように決定されます。

シナリオ	"SqlType" の添え字	"CType" の添え字	使用される実際の "SqlType" 値	使用される実際の "CType" 値
1	指定	指定なし	"SqlType" 添え字の値	"SqlType" 値から自動計算されます。
2	指定なし	指定	"SqlType" 添え字の値 ("CType" 添え字からのコピーによって自動的に定義されます)	
3	指定なし	指定なし	可能な場合は、データ・ソースに対するクエリによって自動的に値が決定されます。それ以外の場合は 12 (SQL_VARCHAR) が使用されます。	
4	指定	指定	"SqlType" 添え字の値	"CType" 添え字の値

8.3 クエリの実行

受信アダプタまたはビジネス・サービス内でクエリを実行できます。クエリを実行するには、アダプタの ExecuteQuery() または ExecuteQueryParmArray() メソッドを使用します。これらのメソッドは、EnsLib.SQL.GatewayResultSet および EnsLib.SQL.Snapshot ヘルパ・クラスを使用します。2 つのクラスの相違点は以下のとおりです。

- 結果セット (EnsLib.SQL.GatewayResultSet のインスタンス) は初期化が必要です。初期化されると、データ・ソースに対してライブ・データ接続するようになります。
- 一方、スナップショット (EnsLib.SQL.Snapshot のインスタンス) は、さまざまな方法で作成および設定できる静的オブジェクトです。例えば、結果セットのデータ、つまり、すべての行または行のサブセット (任意の行位置から開始) のいずれかを使用して作成できます。後の章で、スナップショットを作成するの他の方法について説明します。

注釈　ここでは、結果セットとスナップショットの取得方法について説明します。使用法については説明しません。これらのオブジェクトの使用法の詳細は、“結果セットの使用法”の章と“スナップショットの使用法”の章を参照してください。

8.3.1 モードの使用

ExecuteQuery() または ExecuteQueryParmArray() メソッドを使用すると、メソッドの呼び出し方法に応じて結果セットまたはスナップショットのどちらかを (参照によって) 取得できます。これらのメソッドを使用するには、以下の手順を実行します。

1. アダプタが DSN に接続していることを確認します。
2. スナップショット・オブジェクトを受信するには、以下の手順を実行します。
   a. EnsLib.SQL.Snapshot の新しいインスタンスを作成します。
   b. オプションで、そのインスタンスの FirstRow プロパティと MaxRowsToGet プロパティの値を指定します。

3. ExecuteQuery() または ExecuteQueryParmArray() メソッドを呼び出して、以下の引数をメソッドに渡します。
   a. スナップショット・インスタンス (ある場合)
   b. クエリを含む文字列
   c. クエリおよびメソッドのどちらか適切な方のパラメータ (次の節を参照)

スナップショット・インスタンスを用意しない場合は、メソッドから結果セットが返されます。スナップショット・インスタンスをメソッドに渡すと、メソッドは新しい結果セットを作成し、この結果セットを使用してスナップショット・インスタンスを設定し (行の選択に FirstRow プロパティおよび MaxRowsToGet プロパティを使用し)、次にスナップショットを返します。

8.3.2 メソッドの構文

クエリを実行するには、以下のメソッドのいずれかを使用します。

ExecuteQuery()

```
Method ExecuteQuery(ByRef pRS As EnsLib.SQL.GatewayResultSet,
                    pQueryStatement As %String,
                    pParms...) As %Status
```

クエリを実行します。クエリ文字列と任意の数のパラメータを指定します。結果は、最初の引数に参照渡しで返されます。前述のとおり、結果は EnsLib.SQL.GatewayResultSet または EnsLib.SQL.Snapshot のインスタンスになります。

2 番目の引数は、実行するクエリ文です。このクエリ文には、置き換え可能なパラメータを表す標準の SQL ? を使用できます。この文には UPDATEは使用しないでください。

ExecuteQueryParmArray()

```
Method ExecuteQueryParmArray(ByRef pRS As EnsLib.SQL.GatewayResultSet,
                    pQueryStatement As %String,
                    ByRef pParms) As %Status
```

クエリを実行します。このメソッドは上記のメソッドと似ていますが、パラメータの指定方法が異なります。このメソッドの場合は、この章の前述の“InterSystems IRIS の多次元配列でのパラメータの指定”で説明したとおり、InterSystems IRIS の多次元配列 (pParms) にパラメータを指定します。

ExecuteQueryParmArray() メソッドは、パラメータを指定する必要があるときに、使用している ODBC ドライバが ODBC SQLDescribeParam 関数をサポートしていない場合に使用します。

8.3.3 例

以下に、クエリを実行するメソッドの例を示します。

```
Method GetPhone(pRequest As ESQL.GetPhoneNumberRequest,
  Output pResponse As ESQL.GetPhoneNumberResponse) As %Status
{
   Set pResponse = ##class(ESQL.GetPhoneNumberResponse).%New()
   //need to pass tResult by reference explicitly in ObjectScript
   //Use an adapter to run a query in the Employee database.
   Set tSC = ..Adapter.ExecuteQuery(.tResult,
"Select "_pRequest.Type_" from Employee where EmployeeID="_pRequest.ID)

 //Get the result
   If tResult.Next() {
     Set pResponse.PhoneNumber = tResult.GetData(1)
   } Else {
   //Handle no phone number for example
     Set pResponse.PhoneNumber = ""
   }
   Quit $$$OK
}
```

8.4 更新、挿入、および削除の実行

データベースの更新、挿入、または削除を実行するには、以下のメソッドのいずれかを使用します。

ExecuteUpdate()

```
Method ExecuteUpdate(Output pNumRowsAffected As %Integer,
                pUpdateStatement As %String,
                pParms...) As %Status
```

INSERT、UPDATE、または DELETE 文を実行します。この文には、使用するパラメータを任意の数だけ渡すことができます。注：

- 対象の行数が最初の引数に出力として返されます。
- 2 番目の引数は、実行する INSERT、UPDATE、または DELETE 文です。このクエリ文には、置き換え可能なパラメータを表す標準の SQL ? を使用できます。

ExecuteUpdateParmArray()

```
Method ExecuteUpdateParmArray(Output pNumRowsAffected As %Integer,
                pUpdateStatement As %String,
                ByRef pParms) As %Status
```

INSERT、UPDATE、または DELETE 文を実行します。このメソッドは上記のメソッドと似ていますが、パラメータの指定方法が異なります。このメソッドの場合は、この章の前述の“InterSystems IRIS の多次元配列でのパラメータの指定”で説明したとおり、InterSystems IRIS の多次元配列 (pParms) にパラメータを指定します。

ExecuteUpdateParmArray() メソッドは、パラメータを指定する必要があるときに、使用している ODBC ドライバが ODBC SQLDescribeParam 関数をサポートしていない場合に使用します。

8.4.1 例

以下の例では、ExecuteUpdate() メソッドを使用しています。

```
/// Insert into NewCustomer table
Method Insert(pReq As ESQL.request,
   Output pResp As ESQL.response1) As %Status
{
 kill pResp
 set pResp=$$$NULLOREF

 set sql="insert into NewCustomer (Name,SSN,City) values (?,?,?)"

 //perform the Insert
 set tSC = ..Adapter.ExecuteUpdate(.nrows,sql,pReq.Name,pReq.SSN,pReq.City)

 //create the response message
 set pResp=##class(ESQL.response1).%New()
 set pResp.AffectedRows=nrows

 if 'tSC write " failed ",tSC quit tSC
 quit 1
}
```

8.4.2 ExecuteUpdateParmArray の使用例

以下の例は、この章内の前の部分で示した ExecuteUpdate() の例と同等です。ここでは ExecuteUpdateParmArray() メソッドを使用しています。

```
/// Insert into NewCustomer table
Method InsertWithParmArray(pReq As ESQL.request,
   Output pResp As ESQL.response1) As %Status
{
 kill pResp
 set pResp=$$$NULLOREF

 set sql="insert into NewCustomer (Name,SSN,City) values (?,?,?)"

 //set up multidimensional array of parameters
 //for use in preceding query
 set par(1)=pReq.Name
 set par(2)=pReq.SSN
 set par(3)=pReq.City

 //make sure to set top level of array,
 //which should indicate parameter count
 set par=3

 //perform the Insert
 set tSC = ..Adapter.ExecuteUpdateParmArray(.nrows,sql,.par)

 //create the response message
 set pResp=##class(ESQL.response1).%New()
 set pResp.AffectedRows=nrows

 if 'tSC write " failed ",tSC quit tSC
 quit 1
}
```

8.5 ストアド・プロシージャの実行

ストアド・プロシージャを実行するには、以下のメソッドのいずれかを使用します。

ExecuteProcedure()

```
Method ExecuteProcedure(ByRef pResultSnapshots As %ListOfObjects,
                Output pOutputParms As %ListOfDataTypes,
                pQueryStatement As %String,
                pIO As %String = "",
                pInputParms...) As %Status
```

ストアド・プロシージャを実行する SQL CALL 文を実行します。任意の数のパラメータを渡すことができます。注：

- 結果は、EnsLib.SQL.Snapshot オブジェクトのリストとして、最初の引数に参照渡しで返されます。
- EnsLib.SQL.Snapshot の新しいインスタンスのリストを作成し、そのリストを最初の引数としてメソッドに渡すことができます。その場合は、メソッドがこれらのインスタンスを設定し、FirstRow プロパティと MaxRowsToGet プロパティの値を使用してそれぞれに相当する行のセットを選択します。次に、メソッドがインスタンスのリストを返します。
- 2 番目の引数は、すべてのスカラ出力パラメータと入出力パラメータの出力値のリストです。プロシージャがスカラ戻り値を返し、文がそれを取得した場合は、この値が最初の出力値になります。
- 3 番目の引数は、ストアド・プロシージャを実行する SQL CALL 文です。このクエリ文には、置き換え可能なパラメータを表す標準の SQL ? を使用できます。

重要　ストアド・プロシージャ名では大文字と小文字が区別されます。また、SQL クエリが必要とする入力パラメータまたは入出力パラメータごとに、pQueryStatement 文で引数を指定する必要があります。

- 4 番目 (オプション) の引数は、各パラメータのタイプ (入力、出力、または入出力) を示します。この引数を指定する場合は、文字 `i`、`o`、および `b` で構成される文字列を使用します。所定の位置にある文字が、対応するパラメータのタイプを示します。例えば、`iob` は、最初のパラメータが入力、2 番目のパラメータが出力、および 3 番目のパラメータが入出力であることを意味しています。

  Tipヒント　デフォルトでは、アダプタは ODBC 関数 DescribeProcedureColumns を呼び出してパラメータに関する情報を取得し、この関数で返されたパラメータ・タイプとここに指定されているパラメータ・タイプが異なる場合は、警告をログに記録します。アダプタがこのチェックを実行しないようにするには、この文字列の最後にアスタリスク (*) を追加します。

  すべてのデータベースで、これらすべてのパラメータ・タイプがサポートされているわけではありません。接続しているデータベースでサポートされているタイプのみを使用してください。

ExecuteProcedureParmArray()

```
Method ExecuteProcedureParmArray(ByRef pResultSnapshots As %ListOfObjects,
                Output pOutputParms As %ListOfDataTypes,
                pQueryStatement As %String,
                pIO As %String = "",
                ByRef pIOParms) As %Status
```

ストアド・プロシージャを実行する SQL CALL 文を実行します。このメソッドは上記のメソッドと似ていますが、パラメータの指定方法が異なります。このメソッドの場合は、この章の前述の “InterSystems IRIS の多次元配列でのパラメータの指定” で説明したとおり、InterSystems IRIS の多次元配列 (pParms) にパラメータを指定します。

ExecuteProcedureParmArray() メソッドは、パラメータを指定する必要があるときに、使用している ODBC ドライバが ODBC SQLDescribeParam 関数をサポートしていない場合に使用します。

以下の点にも注意してください。

- パラメータに関しては、pIOParms 配列内、および pIO 引数内で入出力タイプを指定した場合は、pIOParms 配列に指定されたタイプが優先されます。
- pIOParms 配列内に入出力タイプを指定する場合は、すべての出力パラメータに対して、対応する配列ノードを未定義のままにします。

SQL アダプタが Java ゲートウェイを介して JDBC を使用するように構成している場合は、ラージ・オブジェクト値が格納されている出力パラメータはストリームとして返されます。旧バージョンの InterSystems IRIS との互換性を確保するために、グローバルがこれらのラージ・オブジェクト出力パラメータを文字列として返すように設定できます。ただし、このようにグローバルを設定した場合でも、そのオブジェクトが文字列の最大許容サイズを超えている場合は、ストリームとして返されます。この互換性動作を SQLservice 構成項目に対して設定するには、^Ens.Config("JDBC","LOBasString","SQLservice") というグローバルを 1 に設定します。

8.5.1 例

以下のコードは、出力パラメータ、入力パラメータ、およびもう 1 つ別の出力パラメータという 3 つのパラメータを持つストアド・プロシージャを実行します。入力パラメータは、要求メッセージ `pReq.Parameters.GetAt(1)` の Parameters プロパティから抽出されます。出力パラメータは無視されます。

```
Set tQuery="{ ?=call Sample.Employee_StoredProcTest(?,?) }"
Set tSC = ..Adapter.ExecuteProcedure(.tRTs,.tOutParms,tQuery,"oio",pReq.Parameters.GetAt(1))
Set tRes.ParametersOut = tOutParms
```

この例では、tRTs が以前作成された結果セットを表しています。

8.6 ステートメント属性の指定

SQL アダプタを使用する場合は、任意のドライバ依存のステートメント属性を指定できます。そのためには、以下のように操作します。

- 接続が確立していない場合は、以下のような属性と値のペアのカンマ区切りリストの形式で、アダプタの StatementAttrs プロパティを設定します。

  ```
  attribute:value,attribute:value,attribute:value,...
  ```

  以後作成されるすべての文は、これらの属性を継承します。

  以下に例を示します。

  ```
  Set ..Adapter.StatementAttrs = "QueryTimeout:10"
  ```

- 接続が既に確立している場合は、アダプタの SetConnectAttr() メソッドを呼び出します。このメソッドは、2 つの引数 (属性名および必要な値) を取り、ステータスを返します。以下に例を示します。

  ```
  Set tout= ..Adapter.SetConnectAttr("querytimeout",10)
  ```

  ネットワーク・エラーが検出された場合、デフォルトでは、アダプタは再接続と再試行を試みます。AutoCommit などの接続属性を設定している場合は、次の処理を行って、この再接続/再試行ロジックが実行されるようにしてください。つまり、SetConnectAttr() から返されたステータスをテストし、エラーが生じた場合は、そのステータス値をビジネス・オペレーションから返します。

注釈　使用する属性が、接続しているデータベースの ODBC ドライバでサポートされていることを確認してください。InterSystems のドキュメントには、これに関するリストは含まれていません。

ステートメント属性を設定するのに最も有効な場所は、以下のとおりです。

- SQL 送信アダプタを使用する場合は、ビジネス・オペレーションのメッセージ・ハンドラ・メソッド内
- ビジネス・ホストの OnInit() メソッド内

8.7 トランザクションの管理

SQL アダプタには、正式なデータベース・トランザクションの管理に使用できる以下のメソッドが用意されています。

SetAutoCommit()

```
Method SetAutoCommit(pAutoCommit) As %Status [ CodeMode = expression ]
```

このアダプタ接続の自動コミットをオンまたはオフに設定します。これは、DSN 接続の確立後にのみ機能します。

自動コミットを接続時点で設定する場合は、ビジネス・サービスまたはビジネス・オペレーションの OnInit() メソッドをカスタマイズします。カスタム・メソッドで、ConnectAttrs プロパティを設定します。

自動コミットを有効にする場合は、StayConnected を 0 に設定しないでください。この設定は、コマンドを処理している間、リモート・システムに接続したままにしておくかどうかを指定します。

- この設定が SQL 受信アダプタでどのように使用されるかについての詳細は、“SQL 受信アダプタの使用法” の章の “その他の実行時設定の指定” を参照してください。
- この設定が SQL 送信アダプタでどのように使用されるかについての詳細は、“SQL 送信アダプタの使用法” の章の “その他の実行時設定の指定” を参照してください。

ネットワーク・エラーが検出された場合、デフォルトでは、アダプタは再接続と再試行を試みます。AutoCommit などの接続属性を設定している場合は、次の処理を行って、この再接続/再試行ロジックが実行されるようにしてください。つまり、SetAutoCommit() から返されたステータスをテストし、エラーが生じた場合は、そのステータス値をビジネス・オペレーションから返します。

Commit()

```
Method Commit() As %Status
```

前回のコミット以降の (このアダプタ・プロセス内の) すべてのデータベース・アクティビティをコミットします。

Rollback()

```
Method Rollback() As %Status
```

前回のコミット以降の (このアダプタ・プロセス内の) すべてのデータベース・アクティビティ をロールバックします。

以下の例では、上記メソッドを使用する単純なトランザクションを示しています。当然のことながら、実稼働品質のコードには堅牢なエラー処理が含まれます。例えば、以下のメソッドを Try-Catch 文の `Try` ブロックにラップし、`Rollback` メソッドを `Catch` ブロックに配置すると、エラー発生時にトランザクションをロールバックできます。

Class Member

```
Method TransactionExample(pRequest As common.examples.msgRequest2,
  Output pResponse As common.examples.msgResponse) As %Status
{
  #include %occStatus
  //initialize variables and objects
  set tSC = $$$OK
  set pResponse = ##class(common.examples.msgResponse).%New()

  #; start the transaction. Set autocommit to 0
  set tSC = ..Adapter.SetAutoCommit(0)

  //Example UPDATE, INSERT, DELETE
  set tQueryIns="insert into common_examples.mytable(name,age,datetime)"
                _" values ('SAMPLE"_$random(9999)_"',40,'"_$zdt($h,3)_"')"

  set tSC = ..Adapter.ExecuteUpdate(.tAffectedRows,tQueryIns)

  // finalize transaction
  set tSC=..Adapter.Commit()

  return $$$OK
 }
```

注釈　トランザクションを作成するデータベース・アクティビティを考慮することが重要です。これらのアクティビティが単一のビジネス・ホストに含まれている場合は、上記メソッドを使用してトランザクション管理を設定するだけですみます。ただし、データベース・アクティビティが複数のビジネス・ホストに含まれている場合は、コードを (通常はビジネス・プロセス内に) 記述して正しいロールバックをシミュレートする必要があります。

8.8 データベース接続の管理

アダプタのデータベース接続を管理するには、アダプタの以下のプロパティとメソッドを使用できます。

8.8.1 プロパティ

以下のプロパティでは、データベース接続に関する情報を制御または提供します。

Connected

%Boolean

この読み取り専用プロパティは、アダプタが現在接続されているかどうかを示します。

ConnectAttrs

%String

必要に応じて設定できる一連の SQL 接続属性オプション。ODBC の場合は、これらの形式は次のとおりです。

attr:val,attr:val

例：`AutoCommit:1`

JDBC の場合は、これらの形式は次のとおりです。

attr=val;attr=val

例：`TransactionIsolationLevel=TRANSACTION_READ_COMMITTED`

このプロパティをビジネス・オペレーションまたはビジネス・サービスの OnInit() メソッドで設定し、接続時にこのオプションを使用するように指定します。

ConnectTimeout

%Numeric

このプロパティは、接続の試行ごとに待機する秒数を指定します。デフォルト値は 5 です。

StayConnected

%Numeric

このプロパティは、リモート・システムに接続したままにしておくかどうかを指定します。

- この設定が SQL 受信アダプタでどのように使用されるかについては、“SQL 受信アダプタの使用法” の章の “その他の実行時設定の指定” を参照してください。
- この設定が SQL 送信アダプタでどのように使用されるかについては、“SQL 送信アダプタの使用法” の章の “その他の実行時設定の指定” を参照してください。

DSN

%String

このデータ・ソース名は、接続先である外部データ・ソースを指定します。以下の例は、Microsoft Access データベースを参照する DSN の名前を示しています。

```
accessplayground
```

8.8.2 メソッド

データベース接続を管理するには、以下のメソッドを使用します。

Connect()

```
Method Connect(pTimeout As %Numeric = 30) As %Status
```

DSN プロパティの現在の値で指定されているデータ・ソースに接続します。

Disconnect()

```
Method Disconnect() As %Status
```

データ・ソースから切断します。

TestConnection()

```
Method TestConnection()
```

データ・ソースへの接続をテストします。

また、アダプタ・クラスにも、上記の節に記載されているプロパティの設定に使用できるセッター・メソッドがいくつか用意されています。

9

結果セットの使用法

EnsLib.SQL.GatewayResultSet クラスは、InterSystems IRIS® で特定の用途に使用される結果セットを表しています。このクラスの初期化済みのインスタンスは、データ・ソースに対してライブ・データ接続しています。このクラスには、結果セットの内容を調べるメソッドと静的なスナップショットを返すメソッドが用意されています。

この章では、EnsLib.SQL.GatewayResultSet クラスを使用する方法について説明します。

結果セットから行を含むスナップショットを取得することもできます。“スナップショットの使用法”を参照してください。

9.1 結果セットの作成と初期化

SQL 結果セットを作成および初期化するには、以下の手順を実行します。

1. SQL アダプタ (EnsLib.SQL.InboundAdapter または EnsLib.SQL.OutboundAdapter) で、DSN に接続します。
2. アダプタの ExecuteQuery() メソッドまたは ExecuteQueryParmArray() メソッドを使用します。EnsLib.SQL.GatewayResultSet のインスタンスを参照渡しで受け取ります。

注釈　%New() クラス・メソッドを使用する場合は、結果セットは作成できますが、初期化されないのでデータを入れることができません。結果セットを初期化するには、ここで説明している手順を使用します。

9.2 結果セットの基本情報の取得

EnsLib.SQL.GatewayResultSet の以下のプロパティは、結果セットに関する基本情報を提供します。

- ColCount プロパティは、結果セットの列数を示します。
- QueryStatement プロパティは、この結果セットで使用されるクエリ文を示します。

9.3 結果セットのナビゲート

結果セットは、データ行で構成されます。行をナビゲートするには、以下のメソッドを使用できます。

Next()

```
method Next(ByRef pSC As %Status) returns %Integer
```

カーソルを次の行に進め、行のデータをキャッシュします。カーソルが結果セットの末尾にある場合は、0 を返します。

SkipNext()

```
method SkipNext(ByRef pSC As %Status) returns %Integer
```

カーソルを次の行に進めます。カーソルが結果セットの末尾にある場合は、0 を返します。

9.4 結果セットの現在行の調査

結果セットの現在行を調べるには、以下のメソッドを使用します。

Get()

```
method Get(pName As %String) returns %String
```

現在行で pName という名前を持つ列の値を返します。

GetData()

```
method GetData(pColumn As %Integer) returns %String
```

現在行で pColumn で指定されている位置にある列の値を返します。

GetColumnName()

```
method GetColumnName(pColumn As %Integer = 0)
```

pColumn で指定されている位置にある列の名前を返します。

注釈　ソース・データに無名の列が含まれている場合は、結果セットが、自動的に、これらの列に xCol_n の形式で名前を設定します。

10
スナップショットの使用法

EnsLib.SQL.Snapshot クラスは、さまざま方法で作成および設定する静的オブジェクトを表します。このクラスには、データ調査用のメソッドが用意されています。結果セットよりもこの静的オブジェクトの方が、多くのメソッドを利用できます。この章では、EnsLib.SQL.Snapshot クラスを使用する方法について説明します。

10.1 スナップショットの作成

SQL 受信アダプタを使用する場合は、デフォルトで、スナップショット・オブジェクトをビジネス・サービス内で自動的に受け取ります。クエリの各行について、アダプタは、ビジネス・サービスの ProcessInput() メソッドを呼び出す際にスナップショット・オブジェクトを作成して引数として送信します。以前説明したように、デフォルトでは、このスナップショットには 1 つの行しか含まれていません。

別の方法でも、スナップショットを作成し、設定できます。これらの方法の一部については、前の章で説明されていますが、この章で新たに説明するものもあります。

10.1.1 ライブ接続からのスナップショットの作成

ほとんどの場合、データ・ソースへライブ接続することになります。具体的には、SQL アダプタ (EnsLib.SQL.InboundAdapter または EnsLib.SQL.OutboundAdapter) から始めます。アダプタ内で、DSN に接続します。次に、以下のいずれかを実行します。

- アダプタの ExecuteProcedure() メソッドまたは ExecuteProcedureParmArray() メソッドを使用します。これらの各メソッドは、スナップショットとして結果を返します。

  これらのメソッドについては、前の章で説明しています。

- 結果セットを作成し (前の章を参照)、結果セットの GetSnapshot() メソッドを使用します。このメソッドには、以下のシグニチャがあります。

  ```
  method GetSnapshot(ByRef pSnap As EnsLib.SQL.Snapshot,
     pFetchAll As %Boolean = 0) returns %Status
  ```

  最初の引数に参照渡しでスナップショット・オブジェクトを返します。既存のスナップショット・オブジェクトをメソッドに渡すと、メソッドはそのオブジェクトの FirstRow および MaxRowsToGet プロパティを使用してスナップショットに配置する行を決定します。スナップショット・オブジェクトを渡さない場合は、メソッドはデフォルト値を使用します。

10.1.2 静的データからのスナップショットの作成

DSN に接続せずに、静的データからスナップショットを作成することもできます。そのためには、以下のいずれかのテクニックを使用します。

- CreateFromFile()、CreateFromStream() または CreateFromResultSet クラス・メソッドを使用します。
- スナップショットの新しいインスタンスを (%New() クラス・メソッドを使用して) 作成し、次に ImportFile()、ImportFromStream() メソッドまたは ImportFromResultSet() メソッドを使用します。

以下のリストに、これらのメソッドの詳細を示します。これらのメソッドはすべて、EnsLib.SQL.Shapshot クラスに属しています。

CreateFromFile()

```
classmethod CreateFromFile(pFilename As %String,
   pRowSeparator As %String,
   pColumnSeparator As %String,
   pColumnWidths As %String,
   pLineComment As %String,
   pStripPadChars As %String,
   pColNamesRow As %Integer,
   pFirstRow As %Integer,
   pMaxRowsToGet As %Integer,
   Output pStatus As %Status) as Snapshot
```

新しいスナップショット・オブジェクトを作成し、テーブル形式のテキスト・ファイルからのデータと共にロードします。引数は以下のとおりです。

- pFilename は、インポートするファイルの名前を指定します。これは、唯一の必須引数です。
- pRowSeparator は、以下のいずれかです。
  - ある行と次の行とを区切る文字。デフォルトは、改行文字です。
  - マイナス記号が頭に付いた数字。文字数単位での行の長さを示します。
- pColumnSeparator は、以下のいずれかです。
  - ある列と次の列とを区切る文字。デフォルトの文字はありません。
  - 数字 0。これは、列が pColumnWidths 引数で決定されることを意味します。次の引数を参照してください。
  - マイナス記号が頭に付いた数字。各行でスキップする先頭の文字数を示します。この場合は、列は、pColumnWidths 引数で決定されます。次の引数を参照してください。
- pColumnWidths は、以下のいずれかです。
  - ファイルのフィールドが位置で指定される場合は、列幅 (文字数) のカンマ区切りリスト。
  - ファイルで列区切りを使用している場合は、列数。
- pLineComment は、その文字列以降の行の残りの部分が無視されることになる文字列を指定します。指定された行でこの文字列が検出されると、スナップショットは、行の残りの部分を解析して列に入れることはしません。
- pStripPadChars は、フィールドの最初と最後から切り取る文字を意味します。デフォルトは、空白文字です。
- pColNamesRow は、列名を含む行がある場合に、その行のインデックスを指定します。
- pFirstRow は、スナップショットに入れる (ファイルからの) 最初の行のインデックスを指定します。
- pMaxRowsToGet は、スナップショットに入れる最大行数を指定します。

- pStatus は、スナップショットを作成する際にメソッドが返すステータスです。

CreateFromStream()

```
classmethod CreateFromStream(pIOStream As %IO.I.CharacterStream,
   pRowSeparator As %String,
   pColumnSeparator As %String,
   pColumnWidths As %String,
   pLineComment As %String,
   pStripPadChars As %String,
   pColNamesRow As %Integer,
   pFirstRow As %Integer,
   pMaxRowsToGet As %Integer,
   Output pStatus As %Status) as Snapshot
```

新しいスナップショット・オブジェクトを作成し、テーブル形式のストリームからのデータと共にロードします。CreateFromFile() に関するコメントを参照してください。

CreateFromResultSet

```
classmethod CreateFromResultSet(pRS,
  pLegacyMode As %Integer = 1,
  pODBCColumnType As %Boolean = 0,
  pFirstRow As %Integer,
  pMaxRowsToGet As %Integer,
  Output pStatus As %Status) as Snapshot
```

新しいスナップショット・オブジェクトを作成し、結果セットからのデータと共にロードします。CreateFromFile() および ImportFromResultSet に関するコメントを参照してください。

ImportFile()

```
 method ImportFile(pFilename As %String,
   pRowSeparator As %String = $C(10),
   pColumnSeparator As %String = $C(9),
   pColumnWidths As %String = "",
   pLineComment As %String = "",
   pStripPadChars As %String = " "_$C(9),
   pColNamesRow As %Integer = 0) as %Status
```

テーブル形式のテキスト・ファイルからデータをインポートします。CreateFromFile() に関するコメントを参照してください。

ImportFromStream()

```
method ImportFromStream(pIOStream As %IO.I.CharacterStream,
   pRowSeparator As %String = $C(10),
   pColumnSeparator As %String = $C(9),
   pColumnWidths As %String = "",
   pLineComment As %String = "",
   pStripPadChars As %String = " "_$C(9),
   pColNamesRow As %Integer = 0) as %Status
```

ここで、pIOStream はインポートするストリームです。CreateFromFile() に関するコメントを参照してください。

ImportFromResultSet()

```
method ImportFromResultSet(pRS,
   pLegacyMode As %Integer = 1,
   pODBCColumnType As %Boolean = 0) as %Status
```

結果セットをスナップショット・インスタンスにインポートします。引数は以下のとおりです。

- pRS は EnsLib.SQL.GatewayResultSet のインスタンス、または %SQL.StatementResult や %SQL.ISelectResult (%SQL.IResult) など、%SQL パッケージ内の結果セットです。

- pLegacyMode は、メタ・データの検索方法を指定します。この引数が 0 の場合、InterSystems IRIS® では最初に %GetMetadata の使用が試行されます。これにより、従来の結果セット・クラスに対して別のソースのメタデータが検出されます。デフォルトは 1 です。これは、以前の動作を維持しながら、**%SQL.*** および従来のクラスをサポートします。
- pODBCColumnType は、ColumnType の設定方法を制御します。pODBCColumnType が 1 の場合、ColumnType テキストは ODBC タイプの列タイプ・テキストに設定され、clientType には設定されません。

10.1.2.1 例

以下のコンテンツを持つファイルを検討します。

```
col1,col2,col3
value A1,value A2,value A3
value B1,         value B2           ,value B3
```

以下のコードは、このファイルを読み取り、これを使用してスナップショットを作成し、簡単なコメントをターミナルに書き込みます。使用される引数がファイル名および列区切りのみであることに注意してください。

ObjectScript

```
 set filename="c:/demo.txt"
 set snap=##class(EnsLib.SQL.Snapshot).%New()
 do snap.ImportFile(filename,,",")
 d show
 quit


show
 w "number of rows in snapshot=",snap.RowCount,!
 while snap.Next()
  {
      w "current row=",snap.%CurrentRow,!
      w "data in first column=",snap.GetData(1),!
      w "data in second column=",snap.GetData(2),!
      w "data in third column=",snap.GetData(3),!
      }
 quit
```

このルーチンからの出力は、以下のとおりです。

```
number of rows in snapshot=3
current row=1
data in first column=col1
data in second column=col2
data in third column=col3
current row=2
data in first column=value A1
data in second column=value A2
data in third column=value A3
current row=3
data in first column=value B1
data in second column=value B2
data in third column=value B3
```

デフォルトでは、改行が行区切りとして使用されます。また、デフォルトで、先頭および末尾の空白が各フィールドから削除されることにも注意してください。

10.1.3 スナップショットの手動作成

以下のように、スナップショットを手動で作成することもできます。

1. スナップショットの新しいインスタンスを作成します (%New() クラス・メソッドを使用)。
2. SetColNames() メソッド、SetColSizes() メソッド、および SetColTypes() メソッドを使用して列の名前、サイズ、およびタイプを指定します。

3. AddRow() メソッドを使用してデータ行を追加します。

以下のリストに、これらのメソッドの詳細を示します。これらのメソッドはすべて、EnsLib.SQL.Shapshot クラスに属しています。

AddRow()

```
method AddRow(pCol...) returns %Status
```

指定されたデータを含む行を追加します。引数リストは、フィールド別の行データです。例えば、以下の例では、スナップショットに行を追加します。この場合の列名はそれぞれ、ID、Name、および DOB です。

```
set sc=snapshot.SetColNames("1023","Smith,Bob","06-06-1986")
```

SetColNames()

```
method SetColNames(pColName...) returns %Status
```

引数で指定されている順に、列名を設定します。例えば、以下の例では、列名をそれぞれ、ID、Name、および DOB と設定します。

```
set sc=snapshot.SetColNames("ID","Name","DOB")
```

SetColSizes()

```
method SetColSizes(pColSize...) returns %Status
```

引数で指定されている順に、列のサイズ (文字数単位での幅) を設定します。

SetColTypes()

```
method SetColTypes(pColType...) returns %Status
```

引数で指定されている順に、列のタイプを設定します。

注釈　SQL タイプ名は、データベースのベンダごとに異なることに注意してください。使用中のデータベースに適したタイプ名を使用します。SetColTypes() メソッドは、タイプ名のチェックは実行しません。

10.2 スナップショットの基本情報の取得

スナップショットの以下のプロパティは、基本情報を提供します。

- %CurrentRow プロパティは、現在行を示す整数です。
- AtEnd プロパティは、現在行が最終行である場合は真に、最終行でない場合は偽になります。
- ColCount プロパティは、スナップショットの列数を示します。
- RowCount プロパティは、スナップショットの行数を示します。このプロパティは、コメント文字列で始まらない行がある場合に、その行のみをカウントします。コメントを含むスナップショットを作成するには、CreateFromFile() メソッドと関連メソッドを使用し、pLineComment 引数に値を指定します。コメント文字列で始まらないものの、後の位置にコメント文字列が含まれる場合に、行がカウントされます。

10.3 スナップショットのナビゲート

スナップショットは、データ行で構成されます。行をナビゲートするには、以下のメソッドを使用できます。

Next()

```
method Next(ByRef pSC As %Status) returns %Integer
```

カーソルを次の行に進めます。カーソルがスナップショットの末尾にある場合は、0 を返します。

Rewind()

```
method Rewind() returns %Status
```

カーソルをスナップショットの先頭行に戻します。

10.4 スナップショットの現在行の調査

スナップショットの現在行を調べるには、以下のメソッドを使用します。

Get()

```
method Get(pName As %String, pRow=..%CurrentRow) returns %String
```

指定の行 (デフォルトでは現在行) で名前 pName を持つ列の値を返します。

GetData()

```
method GetData(pColumn As %Integer, pRow=..%CurrentRow) returns %String
```

指定の行 (デフォルトでは現在行) でpColumn で指定されている位置にある列の値を返します。

GetColumnName()

```
method GetColumnName(pColumn As %Integer = 0)
```

pColumn で指定されている位置にある列の名前を返します。

GetColumnId()

```
method GetColumnId(pName As %String) returns %Integer
```

名前 pName を持つ列の順序位置を返します。不慣れなテーブルを使用する際に、このメソッドは役に立ちます。

GetColumnSize()

```
method GetColumnSize(pColumn As %Integer = 0)
```

pColumn で指定されている位置にあるデータベース・フィールドのサイズ (文字数で表した幅) を返します。

GetColumnType()

```
method GetColumnType(pColumn As %Integer = 0)
```

pColumn で指定されている位置にある列のタイプを返します。

注釈　　SQL タイプ名は、データベースのベンダごとに異なります。

10.5 スナップショットのリセット

既存のスナップショット・オブジェクトがある場合は、そのオブジェクトからデータや定義を消去できます。そのためには、Clean() メソッドを使用します。このメソッドはステータスを返します。これは、スナップショットを破棄し、%New() を使用して新たなスナップショットを作成するよりも、若干効率的です。