InterSystems SQL の使用法

Version 2023.1
2024-01-02

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

InterSystems SQL の使用法
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

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

目次

テーブル一覧

1

InterSystems SQL の使用法

InterSystems SQL は、標準的なリレーショナル機能を完備しています。その機能として、テーブル・スキーマを定義する機能、クエリを実行する機能、ストアド・プロシージャを定義して実行する機能などがあります。InterSystems SQL は、管理ポータルから対話形式で実行できるほか、SQL シェル・インタフェースを使用してプログラムで実行することもできます。埋め込み SQL を使用すると、ObjectScript コードに SQL 文を埋め込むことができます。また、ダイナミック SQL を使用すると、ObjectScript からその実行時にダイナミック SQL 文を実行できます。

1.1 はじめに

- "InterSystems SQL の機能" では、ソフトウェアの規格と相互運用性に関する InterSystems SQL の概要について説明します。
- "InterSystems SQL の基礎" では、テーブルやクエリなどの InterSystems SQL の基本機能、および InterSystems SQL の実行方法について説明します。

1.2 トピック

1.2.1 InterSystems SQL の構文

- "言語要素" では、InterSystems SQL でのリテラル、NULL、演算子、およびコメントの扱いについて説明します。
- "暗黙結合 (矢印構文)" では、SELECT 節で指定した列に対して左外部結合を実行する、InterSystems SQL の簡潔な拡張子について説明します。
- "識別子" では、InterSystems SQL のエンティティに命名するための規則について説明します。

1.2.2 SQL 実行インタフェース

- "埋め込み SQL の使用法" では、ObjectScript のコードに InterSystems SQL のコードを埋め込む方法について説明します。
- "ダイナミック SQL の使用法" では、ObjectScript のコードの実行時に、そこから SQL コードを作成して実行する方法について説明します。

- "SQL シェル・インタフェースの使用法" では、シェル・インタフェースを使用して InterSystems ターミナルで SQL 文を作成し、そこから実行する方法について説明します。
- "管理ポータルの SQL インタフェースの使用法" では、インターシステムズの管理ポータル・インタフェースから SQL 文を実行し、SQL 機能を表示および管理する方法について説明します。

1.2.3 スキーマ定義

- "テーブルの定義" では、テーブルの基本的な要素 (スキーマ名、テーブル名、RowID、主キー) を取り上げ、永続クラスの定義または DDL コマンドの使用によって InterSystems SQL でテーブルを定義する方法について説明します。
- "ビューの定義と使用" では、InterSystems SQL でビューとビュー ID (%VID) を定義する方法について説明します。
- "テーブル間のリレーションシップ" では、InterSystems SQL で外部キーおよび親と子のリレーションシップを定義して維持する方法について説明します。
- "トリガの使用法" では、InterSystems SQL でレコードを追加、変更、削除すると自動的に実行されるトリガについて説明します。
- "照合" では、InterSystems SQL でデータ値を並べる方法と比較する方法を指定する照合タイプについて説明します。

1.2.4 データ管理とクエリ

- "データベースの変更" では、データを挿入、更新、削除する方法、およびトランザクションを使用して複数のデータ変更をグループ化する方法について説明します。
- "データベースの問い合わせ" では、InterSystems SQL でクエリを作成して実行する方法について説明します。
- "ストアド・プロシージャの定義と使用" では、InterSystems SQL でストアド・プロシージャを定義して使用する方法について説明します。
- "ストリーム・データ (BLOB と CLOB) の格納と使用" では、InterSystems SQL で使用するバイナリ・ストリーム・データと文字ストリーム・データについて説明します。

1.2.5 SQL セキュリティ

- "SQL のユーザ、ロール、および特権" では、ユーザの定義、ロールへのユーザの関連付け、ユーザまたはロールへの特権の割り当てなど、InterSystems SQL のセキュリティ機能について説明します。

1.2.6 SQL のインポート/エクスポート

- "SQL コードのインポート" では、InterSystems SQL または他のベンダの SQL コードを記述したテキスト・ファイルから SQL コードをインポートして実行する方法を説明します。
- "SQL データのインポートとエクスポート" では、テキスト・ファイルからテーブルにデータをインポートし、テーブルからテキスト・ファイルにデータをエクスポートする InterSystems SQL ツールについて説明します。

2

InterSystems SQL の機能

InterSystems SQL は、InterSystems IRIS® データ・プラットフォーム・データベースに格納されているデータに対し、徹底した標準リレーショナル・アクセスを提供します。

InterSystems SQL には、以下のような利点があります。

- 優れた性能とスケーラビリティー InterSystems SQL は、従来のリレーショナル・データベース製品に比べ、高い性能とスケーラビリティを有しています。また、ラップトップ・コンピュータから高性能のマルチ CPU システムまで、さまざまなハードウェアやオペレーティング・システム上で動作します。
- InterSystems IRIS オブジェクト・テクノロジとの統合 ー InterSystems SQL は、InterSystems IRIS オブジェクト・テクノロジと密接に統合されています。データへのリレーショナル・アクセス、およびオブジェクト・アクセスの性能を低下させることなく、両方のアプローチを併用できます。
- メンテナンスの低減 ー 従来のリレーショナル・データベースとは異なり、InterSystems IRIS アプリケーションでは、配置したアプリケーションでテーブルを圧縮する必要がありません。
- 標準 SQL クエリをサポート ー InterSystems SQL は、SQL-92 標準構文とコマンドをサポートします。したがって、既存のリレーショナル・アプリケーションを簡単に InterSystems IRIS に移行することができ（特殊な場合を除く）、InterSystems IRIS の優れたパフォーマンスと高性能なオブジェクトをご利用いただけます。

InterSystems SQL は、さまざまな用途に使用できます。

- オブジェクト・ベース・アプリケーションと Web ベース・アプリケーション ー InterSystems IRIS のアプリケーションでは、SQL クエリを使用して検索などの高機能なデータベース操作を実行できます。
- オンライン・トランザクション処理 ー InterSystems SQL は、挿入および更新操作や、トランザクション処理アプリケーションにおける一般的なクエリのタイプで優れた性能を発揮します。
- ビジネス・インテリジェンスとデータウェア・ハウジング ー InterSystems IRIS の多次元データベース・エンジンとビットマップ・インデックス・テクノロジの組み合わせにより、最高のデータ・ウェアハウス形式アプリケーションを提供します。
- アドホック・クエリおよびレポート ー InterSystems SQL に含まれているフル装備の ODBC ドライバと JDBC ドライバを使用して、一般のレポート・ツールやクエリ・ツールに接続できます。
- エンタープライズ・アプリケーションの統合 ー InterSystems SQL ゲートウェイにより、ODBC または JDBC 準拠の外部リレーショナル・データベースに保存されているデータへのシームレスな SQL アクセスが可能になります。これにより、さまざまなソースのデータを InterSystems IRIS アプリケーション内で簡単に統合できます。

2.1 アーキテクチャ

InterSystems SQL の核となる部分は、以下のコンポーネントで構成されます。

- 統一データ・ディクショナリー一連のクラス定義として保存されている、すべてのメタ情報の集積する場所。InterSystems IRIS は、統一ディクショナリに保存されるすべての永続クラスに対するリレーショナル・アクセス(テーブル)を自動的に生成します。
- SQL プロセッサとオプティマイザ ー SQL クエリを解析し分析する一連のプログラム。これは、(高性能コスト・ベース・オプティマイザを使用して) 既存のクエリに最も適した検索方法を決定し、クエリを実行するコードを生成します。
- InterSystems SQL サーバー InterSystems ODBC および JDBC ドライバとのすべての通信を担う、一連の InterSystems IRIS サーバ・プロセス。また、最も頻繁に使用するクエリのキャッシュも管理されています。同じクエリを何度も実行する場合、その実行プランをクエリ・キャッシュから取り出すことができるので、オプティマイザで再度処理する必要がありません。

2.2 機能

InterSystems SQL には、すべての標準的なリレーショナル機能が装備されています。これには、以下のものがあります。

- テーブルやビューを定義します (DDL または Data Definition Language)。
- テーブルやビューに対しクエリを実行します (DML または Data Manipulation Language)。
- INSERT、UPDATE、DELETE 操作などの、トランザクションを実行します。同時操作を実行する際、InterSystems SQL は行レベル・ロックを使用します。
- 効率的にクエリを実行するために、インデックスを定義し使用します。
- ユーザ定義タイプも含め、さまざまなデータ型に対応します。
- ユーザとロールを定義し、それらに特権を与えます。
- 外部キーや、その他の整合性制約を定義します。
- INSERT、UPDATE、DELETE の各トリガを定義します。
- ストアド・プロシージャを定義、実行します。
- クライアント・アクセスには ODBC モード、サーバ・ベース・アプリケーションで使用するには表示モードというように、異なる形式でデータを返します。

注釈　インターシステムズは、今後も引き続き InterSystems SQL のその他の機能のサポートを提供します。このリリースでサポートされていない機能をご希望の方は、今後の新しいリリースにその機能が追加されるかどうかなど、インターシステムズのサポート窓口までお問い合わせください。

2.2.1 SQL-92 準拠

SQL-92 標準は算術演算子の優先順位に関して不明確です。この件に関する前提は SQL の実装間で異なります。InterSystems SQL では、SQL 算術演算子の優先順位についてシステム全体に以下のいずれかの処理方法を適用するようシステムを構成できます。

- 演算子の優先順位はなく、算術式を必ず左から右の順番で解析するよう InterSystems SQL を構成できます。これは、ObjectScript で使用される規則と同じです。例えば、`3+3*5=30` となります。演算順位を指定するには、括弧を使用します。例えば、`3+(3*5)=18` となります。
- ANSI の優先順位を使用して算術式を解析するよう InterSystems IRIS を構成できます。この優先順位では、乗算演算子と除算演算子の優先順位が、加算演算子、減算演算子、および連結演算子よりも高く設定されます。例えば、`3+3*5=18` となります。この優先順位は、必要に応じて括弧を使用することでオーバーライドできます。例えば、`(3+3)*5=30` となります。

SQL 演算子の既定の優先順位は、InterSystems IRIS のバージョンによって異なります。詳細は、"SQL 算術演算子の優先順位" を参照してください。

InterSystems SQL は、下記の例外を除き、すべての初級 SQL-92 標準をサポートします。

- テーブル定義に CHECK 制限を別途追加することはできません。
- SERIALIZABLE 分離レベルはサポートされていません。
- 区切り識別子は、大文字と小文字を区別しません。SQL-92 標準では、大文字と小文字を区別する必要があります。
- HAVING 節に含まれるサブクエリ内で、通常であれば、その HAVING 節で “使用可能な” 集合を参照できるようになっています。これはサポートされていません。

2.2.2 拡張機能

InterSystems SQL は、多数の便利な拡張機能をサポートします。これらの多くは、InterSystems IRIS がデータに対し同時にオブジェクト・アクセスとリレーショナル・アクセスを提供できるという機能に関連します。

以下は、その拡張機能の例の一部です。

- ユーザ定義のデータ型と機能をサポートします。
- 後続のオブジェクト参照のための特別構文が使用できます。
- サブクラスと継承をサポートします。
- 他のデータベースに保存されている外部テーブルに対するクエリをサポートします。
- 最高の性能を実現するために、テーブルの保存構成を管理するさまざまなメカニズムを提供します。

2.3 相互運用性

InterSystems SQL は、他のアプリケーションやソフトウェア・ツールをリレーショナルに相互運用する多くの方法をサポートします。

2.3.1 JDBC

InterSystems IRIS には、標準準拠のレベル 4 (すべて純正 Java コード) JDBC クライアントが含まれています。

InterSystems JDBC ドライバには、以下のような特徴があります。

- 高性能
- 純正 Java の実装
- Unicode サポート

- スレッドの安全性

JDBC をサポートするものであれば、あらゆるツール、アプリケーション、開発環境で InterSystems JDBC をご利用いただけます。互換性について問題が発生した場合やご質問等がございましたら、インターシステムズのサポート窓口までお問い合わせください。管理ポータルから右上の [**お問い合わせ**] ボタンを使用して、インターシステムズのサポート窓口に問い合わせることができます。

2.3.2 ODBC

InterSystems SQL における C 言語の呼び出しレベルのインタフェースは、ODBC です。他のデータベース製品とは異なり、InterSystems ODBC ドライバはネイティブのドライバであり、他のメーカー独自のインタフェース上に構築されたものではありません。

InterSystems ODBC ドライバには、以下のような特徴があります。

- 高性能
- 移植性
- ネイティブの Unicode のサポート
- スレッドの安全性

ODBC をサポートするものであれば、あらゆるツール、アプリケーション、開発環境で InterSystems ODBC をご利用いただけます。互換性について問題が発生した場合やご質問等がございましたら、インターシステムズのサポート窓口までお問い合わせください。管理ポータルから右上の [**お問い合わせ**] ボタンを使用して、インターシステムズのサポート窓口に問い合わせることができます。

2.3.3 埋め込み SQL

ObjectScript では、InterSystems SQL は埋め込み SQL をサポートします。つまり、メソッド (またはその他のコード) の本体内に SQL 文を配置できます。埋め込み SQL を使用すると、単一のレコードを照会できます (複数のレコードを照会するにはカーソルを定義して使用します)。埋め込み SQL は、コンパイルされます。既定では、埋め込み SQL を含むルーチンのコンパイル時ではなく、埋め込み SQL の初回実行時にコンパイルされます (ランタイム)。このため、実行時に SQLCODE エラーを確認することが重要です。埋め込み SQL を、それを含む ObjectScript ルーチンと同時にコンパイルすることもできます。

埋め込み SQL を InterSystems IRIS のオブジェクト・アクセス機能と併せて使用すると、非常に高い性能を発揮できます。例えば、次のメソッドは、特定の **Name** 値を持つレコードの RowID を検索します。

Class Member

```
ClassMethod FindByName(fullname As %String)
 {
    &sql(SELECT %ID INTO :id FROM Sample.MyTable WHERE Name = :fullname)
      IF SQLCODE<0 {SET baderr="SQLCODE ERROR:"_SQLCODE_" "_%msg
                    RETURN baderr }
      ELSEIF SQLCODE=100 {SET nodata="Query returns no data"
                    RETURN nodata }
      RETURN "RowID="_id
 }
```

詳細は、“埋め込み SQL の使用法”の章を参照してください。

2.3.4 ダイナミック SQL

InterSystems IRIS は、標準ライブラリの一部として **%SQL.Statement** クラスを提供しており、(実行時に定義される) ダイナミック SQL 文を実行するときに使用できます。ダイナミック SQL は、ObjectScript メソッド内で使用できます。例えば、以下のメソッドでは、21 世紀に誕生した人物について、指定した人数分のクエリを実行します。このクエリは、1999 年 12

月 31 日より後に誕生したすべての人物を選択し、選択したレコードを誕生日順に並べ替えて、上位 x 件のレコードを選択します。

Class Member

```
ClassMethod Born21stC(x) [ language=objectscript ]
{
  SET myquery=2
  SET myquery(1) = "SELECT TOP ? Name,%EXTERNAL(DOB) FROM Sample.Person "
  SET myquery(2) = "WHERE DOB > 58073 ORDER BY DOB"
  SET tStatement = ##class(%SQL.Statement).%New()
  SET qStatus = tStatement.%Prepare(.myquery)
   IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
  SET rset = tStatement.%Execute(x)
  DO rset.%Display()
  WRITE !,"End of data"
 }
```

クエリを準備すると、そのクエリの最適化されたバージョンがクエリ・キャッシュとして保存されます。それ以降にクエリを呼び出すと、このクエリ・キャッシュが実行され、実行のたびにクエリを再び最適化するオーバーヘッドがなくなります。

詳細は、“ダイナミック SQL の使用法”の章を参照してください。

2.4 制約

以下に、InterSystems SQL の制約を示します。

- NLS を使用して、個々のグローバルおよび現在実行しているプロセスのローカル変数に特定の国言語ロケール動作の $ORDER の動作を指定できます。InterSystems SQL は、あらゆる国言語ロケール内で使用でき、正常に動作します。ただし、現時点では、特定のプロセスの場合に InterSystems SQL が参照する関連グローバルではすべて、現在のプロセスのロケールと同じ国言語ロケールを使用している必要があるという制約があります。このガイドの“照合”の章にある“SQL 照合と NLS 照合”を参照してください。

3

InterSystems SQL の基礎

この章では、InterSystems SQL の機能の概要を示します。特に SQL 標準では取り上げられていないものや InterSystems IRIS® データ・プラットフォームの統一データ・アーキテクチャに関連する情報を示します。この章では、SQL について事前の知識があることが前提となっており、SQL の概念や構文の概要は対象外となっています。

3.1 テーブル

InterSystems SQL では、データはテーブル内に存在します。各テーブルには、多数の列が定義されます。各テーブルには、0 行以上のデータ値が含まれます。以下の用語は、ほぼ同じ意味となります。

データ用語	リレーショナル・データベース用語	InterSystems IRIS 用語
database (データベース)	schema (スキーマ)	package (パッケージ)
	table (テーブル)	persistent class (永続クラス)
field (フィールド)	column (列)	property (プロパティ)
record (レコード)	row (行)	

詳細は、"クラスの定義と使用" の "永続オブジェクトの概要" の章にある "既定の SQL プロジェクションの概要" を参照してください。

テーブルには、基本テーブル (データを含むテーブルで、通常は単にテーブルと呼ばれます) とビュー (1 つまたは複数のテーブルに基づいた論理ビューを表します) という、2 種類の基本となるタイプがあります。

テーブルを定義する方法の詳細は、"テーブルの定義" の章を参照してください。

ビューを定義する方法の詳細は、"ビューの定義" の章を参照してください。

テーブルに対するクエリをより効率的にするには、テーブルにインデックスを定義します。"SQL 最適化ガイド" の "インデックスの定義と構築" の章を参照してください。

参照の整合性を強化するには、テーブルに外部キーとトリガを定義します。"外部キーの定義" および "トリガの定義" の章を参照してください。

3.1.1 スキーマ

SQL スキーマによって、関連する一連のテーブル、ビュー、ストアド・プロシージャ、およびクエリ・キャッシュをグループ分けすることができます。テーブル名、ビュー名、およびストアド・プロシージャ名はスキーマ内で一意である必要がある

ため、スキーマを使用することで、テーブル・レベルでの名前の競合を防ぐことができます。アプリケーションは、複数のスキーマでテーブルを指定できます。

SQL スキーマは、永続クラスのパッケージに対応しています。一般的に、スキーマの名前は、それが対応しているパッケージの名前と同じですが、スキーマ名前付け規約が異なることにより、または異なる名前が意図的に指定されていることにより、これらの名前が異なる場合があります。スキーマからパッケージへのマッピングについては、"SQL からクラス名への変換" で詳しく説明します。

スキーマは特定のネームスペースに対して定義されます。スキーマ名は、そのネームスペース内で一意である必要があります。スキーマ (および対応パッケージ) は、最初の項目が割り当てられると自動的に作成され、最後の項目が削除されると自動的に削除されます。

SQL 名は、修飾、未修飾のどちらも指定できます。修飾名では、スキーマを指定します (`schema.name`)。未修飾名では、スキーマを指定しません (`name`)。スキーマを指定しない場合、InterSystems IRIS は以下のようにスキーマを提供します。

- DDL 操作の場合、InterSystems IRIS では、システム全体の既定のスキーマ名が使用されます。この既定は構成可能です。すべてのネームスペースに適用されます。
- DML 操作の場合、InterSystems IRIS では、ユーザ指定のスキーマ検索パスまたはシステム全体の既定のスキーマ名を使用できます。ダイナミック SQL、埋め込み SQL、および SQL シェルでは、スキーマ検索パスの指定に、異なる手法が使用されます。

ネームスペース内のすべての既存のスキーマを表示するには、以下の操作を実行します。

1. 管理ポータルで、**[システム・エクスプローラ]**、**[SQL]** の順に選択します。ページ上部の **[切り替え]** オプションを使ってネームスペースを選択します。利用可能なネームスペースのリストが表示されます。ネームスペースを選択します。
2. 画面の左側にある **[スキーマ]** ドロップダウン・リストを選択します。これによって、現在のネームスペース内のスキーマのリストが表示されます。このリストからスキーマを選択します。選択した名前が **[スキーマ]** ボックスに表示されます。
3. **[適用先]** ドロップダウン・リストを使用すると、スキーマに属するテーブル、ビュー、プロシージャ、またはクエリ・キャッシュ、あるいはこれらすべてを選択できます。このオプションを設定したら、三角形をクリックして、項目のリストを表示します。項目がない場合は、三角形をクリックしても何も表示されません。

3.2 クエリ

InterSystems SQL 内では、クエリを使用してテーブル内のデータを表示したり、変更したりします。大別すると、クエリにはデータの検索 (SELECT 文) とデータの変更 (INSERT 文、UPDATE 文、DELETE 文) の 2 種類があります。

SQL クエリは、さまざまな方法で使用できます。

- ObjectScript で埋め込み SQL を使用できます。
- ObjectScript でダイナミック SQL を使用できます。
- CREATE PROCEDURE または CREATE QUERY を使用して作成したストアド・プロシージャを呼び出します。
- クラス・クエリを使用できます。詳細は、"クラスの定義と使用" の "クラス・クエリの定義と使用" を参照してください。
- 他のさまざまな環境から ODBC インタフェースまたは JDBC インタフェースを使用できます。

SELECT クエリについては、このドキュメントの "データベースの問い合わせ" の章で説明しています。

クエリは、InterSystems IRIS オブジェクトまたは ObjectScript ルーチンの一部です。

3.3 特権

InterSystems SQL では、特権を使用して、テーブルやビューなどへのアクセスを制限できます。一連のユーザおよびロールを定義して、それぞれに対し (読み取り、書き込みなど) 特権を付与できます。“SQL のユーザ、ロール、および特権”の章を参照してください。

3.4 データ表示オプション

InterSystems SQL では、SelectMode オプションを使用して、データの表示方法や保存方法を指定します。使用可能なオプションは、Logical、Display、および ODBC です。データは、内部的には Logical モードで保存されますが、上記のどのモードでも表示できます。データ型クラスはすべて、LogicalToDisplay()、LogicalToODBC()、DisplayToLogical()、および ODBCToLogical() メソッドを使用することで、内部の Logical 形式と、Display 形式または ODBC 形式間の変換を定義できます。SQL SelectMode が Display である場合、LogicalToDisplay 変換が適用され、返り値は表示用にフォーマットされます。既定の SQL SelectMode は Logical です。そのため、既定では、返り値は、そのストレージ形式で表示されます。

SelectMode は、クエリ結果セットのデータを表示する形式に影響し、WHERE 節などでデータ値を指定する形式にも影響します。InterSystems IRIS では、保存モードおよび指定された SelectMode に基づいて、適切な変換方式が適用されます。指定されたデータ値と SelectMode が一致しないと、エラーが生じるか誤った結果となる可能性があります。例えば、DOB が $HOROLOG 論理形式で格納される日付であり、WHERE 節で `WHERE DOB > 2000-01-01` (ODBC 形式) と指定されている場合は、SelectMode = ODBC ならば、意図した結果が返されます。SelectMode = Display ならば、SQLCODE -146 [ ] が返されます。SelectMode = Logical ならば、`2000-01-01` を論理データ値として解釈しようとするため、0 行が返されます。

ほとんどのデータ型で、3 つの SelectMode モードは、同じ結果を返します。以下のデータ型は、SelectMode オプションによる影響を受けます。

- 日付、時刻、およびタイムスタンプのデータ型。InterSystems SQL は、多くの日付、時刻、およびタイムスタンプのデータ型 (%Library.Date、%Library.Time、%Library.PosixTime、%Library.TimeStamp、および %MV.Date) をサポートしています。これらのデータ型は、%Library.TimeStamp を除いて、Logical モード、Display モード、および ODBC モードで異なる表現を使用します。InterSystems IRIS では、これらの複数のデータ型で日付を $HOROLOG 形式で保存します。この Logical モードの内部表現は、任意の開始日 (1840 年 12 月 31 日) からの日数を示す整数値、コンマ (区切り記号)、および当日の午前 0 時からの経過秒数を示す整数値で構成されます。InterSystems IRIS では、%PosixTime タイムスタンプをエンコードされた 64 ビット符号付き整数として保存します。Display モードでは、日付と時刻は一般に、データ型の FORMAT パラメータで指定された形式、または %SYS.NLS.Format で現在のロケールの既定に設定された日付と時刻の形式で表示されます。アメリカ・ロケールの既定の形式は、`DD/MM/YYYY hh:mm:ss` です。ODBC モードでは、日付と時刻は常に `YYYY-MM-DD mm:mm:ss` という形式で表わされます。また、%Library.TimeStamp データ型も、この ODBC 形式を Logical モードと Display モードに使用します。
- %List データ型。InterSystems IRIS Logical モードでは、2 つの出力不能文字 (リスト内の最初の項目の前で使用されるものと、リスト項目間の区切り記号として使用されるもの) を使用してリストが格納されます。ODBC SelectMode では、リスト項目は、リスト項目間の区切り記号としてコンマを使用して表示されます。Display SelectMode では、リスト項目は、リスト項目間の区切り記号として空白を使用して表示されます。
- VALUELIST および DISPLAYLIST を指定するデータ型。Display モードにしているとき、必須フィールドに DISPLAYLIST と示されているテーブルに値を挿入する場合は、DISPLAYLIST にあるいずれかの項目と完全に同じ値を入力する必要があります。必須以外のフィールドの場合、一致しない値は NULL 値に変換されます。
- 空の文字列、および空の BLOB (ストリーム・フィールド)。Logical モードの空の文字列および BLOB は、非表示文字 $CHAR(0) により表されます。Display モードでは、空の文字列 (“”) により表されます。

以下のように SQL SelectMode を指定します。

- 現在のプロセスでは、SetOption("SelectMode") メソッドを使用します。
- InterSystems SQL シェル・セッションでは、SET SELECTMODE コマンドを使用します。
- 管理ポータルの“クエリ実行”ユーザ・インタフェース ([**システム・エクスプローラ**]、[SQL]) からのクエリ結果セットでは、[表示モード] ドロップダウン・リストを使用します。
- ダイナミック SQL %SQL.Statement インスタンスでは、%SelectMode プロパティを使用します。
- 埋め込み SQL では、ObjectScript の #sqlcompile select プリプロセッサ指示文の設定を使用します。この指示文では、4 つ目の値である Runtime が許可されます。この値は、選択モードを RuntimeMode プロパティ設定の値に (Logical、Display、ODBC のいずれにも) 設定します。**RuntimeMode** の既定値は Logical です。
- SQL コマンド CREATE QUERY、CREATE METHOD、CREATE PROCEDURE、および CREATE FUNCTION では、SELECTMODE キーワードを使用します。
- SQL クエリ内の個々の列では、%EXTERNAL 関数、%INTERNAL 関数、および %ODBCOUT 関数を使用します。

3.5 データ照合

照合は、値を並べる方法と比較する方法を指定するもので、InterSystems SQL と InterSystems IRIS オブジェクトのどちらにも欠かせない要素です。

照合タイプは、フィールド/プロパティ定義の一部として指定できます。特に指定がない場合、文字列フィールド/プロパティは、ネームスペースのデフォルトの照合に設定されます。デフォルトでは、文字列に対するネームスペースのデフォルトの照合は SQLUPPER に設定されています。SQLUPPER 照合では、並べ替えと比較のために文字列は大文字に変換されます。そのため、特に指定がない場合、文字列の並べ替えと比較では大文字と小文字が区別されません。

照合タイプはインデックス定義の一部として指定できます。また、インデックス付きフィールドの照合タイプを使用することもできます。

SQL クエリでは、フィールド名に照合関数を適用することで、フィールド/プロパティに定義されている照合タイプをオーバーライドできます。ORDER BY 節では、クエリの結果セットの順序を指定できます。指定した文字列フィールドが SQLUPPER として定義されている場合、クエリ結果の順序では大文字と小文字が区別されなくなります。

詳細は、"InterSystems SQL の使用法" の“照合”の章を参照してください。

3.6 SQL の実行

InterSystems IRIS は、SQL コードを記述し、実行する多くの方法をサポートしています。これには、以下のものがあります。

- 埋め込み SQL：ObjectScript コードに埋め込んだ SQL コード。
- ダイナミック SQL：**%SQL.Statement** クラスを使用して、ObjectScript から実行する SQL コード。
- Execute() メソッド：**%SYSTEM.SQL** クラスの Execute() メソッドを使用して SQL コードを実行します。
- SQL コードを含む ストアド・プロシージャ：CREATE PROCEDURE または CREATE QUERY を使用して作成します。
- SQL シェル：ターミナル・インタフェースから実行する SQL 文。
- クエリ・インタフェースの実行：管理ポータルから実行する SQL 文。

InterSystems IRIS オブジェクト (クラスおよびメソッド) は、以下の作業に使用できます。

- 永続クラス (SQL テーブル) の定義。
- インデックスの定義。
- クラス・クエリの定義と使用。

4

言語要素

InterSystems SQL は以下の言語要素をサポートします。

- コマンドとキーワード
- 関数 : 内部および外部
- 文字列と数値のリテラル
- NULL および空文字列
- 算術演算子と算術関数
- 関係演算子
- 論理演算子
- コメント

4.1 コマンドとキーワード

InterSystems SQL コマンド (SQL 文とも呼ばれる) は、キーワードで始まり、1 つ以上の引数が続きます。この引数のいくつかは、固有のキーワードで識別できる節または関数である場合があります。

- InterSystems SQL コマンドには、コマンド・ターミネータはありません。ただし例外として、SQL プロシージャ・コードやトリガ・コードなどの特定のケースでは、SQL コマンドは単一のセミコロン (;) で終了します。それ以外の場合、InterSystems SQL コマンドは、セミコロンのコマンド・ターミネータを必要としないかまたは受け入れません。InterSystems SQL でセミコロンのコマンド・ターミネータを指定すると、SQLCODE -25 エラーになります。InterSystems IRIS® データ・プラットフォームにおける TSQL (Transact-SQL) の実装では、セミコロンのコマンド・ターミネータを受け付けますが、必須ではありません。SQL コードを InterSystems SQL にインポートすると、セミコロンのコマンド・ターミネータは削除されます。
- InterSystems SQL コマンドには、空白の制約はありません。コマンド項目をスペースで区切る場合、少なくとも 1 つのスペースが必要です。コマンド項目をコンマで区切る場合、スペースは不要です。算術演算子の前後のスペースは不要です。スペースで区切られた項目間、引数のコンマ区切りリストの項目間、あるいは算術演算子の前後には、改行または複数のスペースを挿入してもかまいません。

InterSystems SQL のキーワードには、コマンド名、関数名、述語条件名、データ型名、フィールド制約、最適化オプション、および特殊変数などがあります。また、AND、OR、および NOT 論理演算子、NULL 列値インジケータ、および ODBC 関数構文 ({d dateval} および {fn CONCAT(str1,str2)} など) があります。

- キーワードでは、大文字と小文字は区別されません。慣例により、キーワードはこのドキュメントでは大文字で表されますが、InterSystems SQL には大文字/小文字の制限はありません。
- 多数のキーワードが SQL 予約語となっています (すべてではありません)。InterSystems SQL では、明確に解析できないキーワードのみが予約されています。SQL 予約語は、区切り識別子として使用できます。

4.2 関数 : 内部および外部

関数は演算を実行し、値を返します。一般に、InterSystems SQL では、関数は SELECT 文で select-item として指定されるか、WHERE 節で指定され、テーブルのフィールド値またはリテラル値に対して演算を実行します。

- 内部 : InterSystems SQL では、多数の内部 (システム提供) 関数をサポートしています。これには、数値関数、文字列関数、日時関数などが含まれます。これらの関数については、"InterSystems SQL リファレンス" を参照してください。この章では、算術関数と三角関数についても説明します。

  集約関数は、列のすべての値を評価し、単一の集約値を返す SQL 内部関数です。集約関数は、"InterSystems SQL リファレンス" で個別に説明されています。

- 外部 : InterSystems SQL では、以下の例に示すように、ユーザ指定の ObjectScript 関数呼び出し (外部関数) もサポートしています。

  ObjectScript

```
MySQL
  &sql(SELECT Name,$$MyFunc() INTO :n,:f FROM Sample.Person)
    IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
    ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
  WRITE "name is: ",n,!
  WRITE "function value is: ",f,!
  QUIT
MyFunc()
  SET x="my text"
  QUIT x
```

  ユーザ指定の (外部) 関数の使用がシステム全体のオプションとして構成されている場合、SQL 文はユーザ指定の (外部) 関数のみ呼び出すことができます。既定値は “いいえ” です。既定では、ユーザ指定関数を呼び出そうとすると、SQLCODE -372 エラーが発生します。SQL による外部関数の使用をシステム全体で構成するには、$SYSTEM.SQL.Util.SetOption() メソッドを `SET status=$SYSTEM.SQL.Util.SetOption("AllowExtrinsicFunctions",1,.oldval)` のように使用します。現在の設定を確認するには $SYSTEM.SQL.CurrentSettings() を呼び出します。これにより、`[SQL ]` オプションが表示されます。

  ユーザ指定関数を使用して、% ルーチン (% 文字で始まる名前を持つルーチン) を呼び出すことはできません。実行しようとすると SQLCODE -373 エラーが発行されます。

4.3 リテラル

InterSystems SQL リテラルの構文は、以下のとおりです。

```
literal ::=       number | string-literal  number ::=
{digit}[.]digit{digit}[E[+|-]digit{digit}]   digit ::=      0..9  string-literal ::=
       std-string-literal | ObjectScript-empty-string  std-string-literal ::=
 ' {std-character-representation} '  std-character-representation ::=
nonquote-character | quote-symbol  quote-symbol ::=      ''  ObjectScript-empty-string
 ::=       ""
```

リテラルは、実際の値を表す文字列です。数値または文字列のいずれかになります。

- 数値は、区切り文字を必要としません。0 ～ 9 の数字、小数点文字、指数記号、正符号、負符号で構成できます。1 つの数値には 1 つの小数点文字しか使用できません。小数点文字は、数値の基数部分にのみ使用でき、指数部分には使用できません。小数点の後に数字を続ける必要はありません。先頭および末尾のゼロは使用可能です。指数 (科学的記数法) の記号は文字 E です。大文字と小文字のどちらの E も使用可能ですが、大文字の E の使用が優先されます。正符号あるいは負符号を、基数または指数の前に置くことができます。また、複数の正符号および負符号を、基数の前に置くことができます。SQL は、これらの符号を演算子として扱います。単一の正符号および負符号のみを指数の前に置くことができます。SQL は、この符号をリテラルの一部として扱います。コンマまたは空白は、数値内で使用できません。
- 文字列リテラルは、あらゆる種類の文字の文字列を囲む 1 組みの区切り文字で構成されます。優先的に使用される区切り文字は、一重引用符 (以下を参照) です。文字列内で区切り文字をリテラルとして指定するには、'Mary''s office' のように、その文字を重複して使用します。

空文字列はリテラル文字列です。2 つの一重引用符 ('') で表されます。NULL はリテラル値ではありません。値が存在しないことを表します。詳細は、この章の "NULL および空文字列" を参照してください。

注釈　埋め込み SQL では、## で始まるいくつかの文字シーケンスは、文字列リテラル内での使用が許可されません。これについては、"埋め込み SQL の使用法" の章の "リテラル値" で説明しています。この制約は、ダイナミック SQL などの、他の SQL の呼び出しには適用されません。

4.3.1 文字列の区切り文字

文字列の区切り文字としては、一重引用符 (') 文字を使用します。二重引用符文字 (") の使用は、SQL 互換性に対してはサポートされますが、区切り識別子の標準と競合するため、使用しないことを強くお勧めします。二重引用符文字のペア "" は無効な区切り識別子として解析され、SQLCODE -1 エラーを生成します。

文字列内のリテラル文字として一重引用符文字を指定するには、これらの文字のペアをリテラル・エスケープ・シーケンスとして指定します。例えば、'a ''normal'' string' と指定します。

4.3.2 連結

二重の垂直バー (||) は、好ましい SQL 連結演算子です。2 つの数値、2 つの文字列、または 1 つの数値と 1 つの文字列を連結するために使用できます。

アンダースコア文字 (_) は、ObjectScript の互換性のために SQL 連結演算子として指定されます。連結演算子は、2 つの文字列を連結するためにのみ使用できます。

2 つの演算子が両方とも文字列であり、両方の文字列の照合タイプが同じ場合、結果として得られる連結文字列には、その照合タイプが含まれます。その他の場合はいずれも、連結の結果の照合タイプは EXACT です。

4.4 NULL および空文字列

値を指定しないことを示すには、NULL キーワードを使用します。NULL は、何らかの理由でデータ値が指定されていないか存在しないことを示す、SQL で常に推奨されている方法です。

長さゼロの SQL 文字列 (空文字列) は、2 つの一重引用符で指定します。空文字列 ('') は NULL と同じではありません。

注釈　長さがゼロの SQL 文字列をフィールドの入力値またはフィールドの既定値とすることはお勧めできません。ObjectScript では、このような文字列が、$CHAR(0) 文字列を使用した長さ 1 の文字列として扱われます。データ値がないことを表すには NULL を使用します。これは、ObjectScript で空文字列 ("") に相当します。詳細は、"ObjectScript と SQL" を参照してください。

長さゼロの SQL 文字列は、SQL コーディングでは使用しないでください。ただし、多くの SQL 処理では末尾の空白スペースが削除されるため、空白文字 (スペースおよびタブ) のみを含むデータ値が、長さゼロの SQL 文字列となる場合があります。

さまざまな SQL 長さ関数がそれぞれ異なる値を返すことに注意してください。LENGTH、CHAR_LENGTH、および DATALENGTH は、SQL の長さを返します。$LENGTH は、ObjectScript 表現での長さを返します。後述の "NULL の長さ" を参照してください。LENGTH は末尾の空白スペースを数えません。他のすべての長さ関数は、末尾の空白スペースを数えます。

4.4.1 NULL の処理

NOT NULL データ制約は、1 つのフィールドが 1 つのデータ値を取得することを必要とします。値ではなく NULL を指定することはできません。この制約は、空文字列値の使用を妨げません。詳細は、"CREATE TABLE" コマンドを参照してください。

SELECT 文の WHERE 節または HAVING 節内の IS NULL 述語は、NULL 値を選択します。空文字列値は選択しません。

IFNULL 関数は、フィールド値を評価して、フィールドが NULL に評価された場合は 2 つ目の引数に指定されている値を返します。この関数が、空文字列の値を NULL 以外の値として扱うことはありません。

COALESCE 関数は、指定されたデータから最初の NULL でない値を選択します。空文字列は、NULL でない値として処理されます。

CONCAT 関数または連結演算子 (||) が 1 つの文字列と 1 つの NULL を連結すると、結果は NULL になります。詳細は、以下の例を参照してください。

SQL

```
SELECT {fn CONCAT('fred',NULL)} AS FuncCat,   -- returns <null>
       'fred'||NULL AS OpCat                  -- returns <null>
```

AVG、COUNT、MAX、MIN、および SUM 集約関数は、処理を実行するときに NULL 値を無視します (すべてのフィールドに NULL 値を持つレコードは存在できないため、COUNT＊はすべての行をカウントします)。SELECT 文の DISTINCT キーワードは、その処理に NULL を含みます。指定したフィールドに NULL 値が存在する場合、DISTINCT は NULL 行を返します。

AVG、COUNT、および MIN 集約関数は、空文字列値の影響を受けます。MIN 関数は、ゼロ値を持つ行がある場合でも、空文字列を最小値と判断します。MAX および SUM 集約関数は、空文字列値の影響を受けません。

4.4.2 式内の NULL

ほとんどの SQL 関数で NULL をオペランドとして指定すると、NULL が返されます。

NULL をオペランドとして持つ任意の SQL 算術演算は、NULL 値を返します。したがって、7+NULL=NULL となります。これには、二項演算子の加算 (+)、減算 (-)、乗算 (*)、除算 (/)、整数除算 (¥)、モジュロ (#)、および単項演算子符号のプラス (+) およびマイナス (-) が含まれます。

算術演算で指定された空文字列は、0 (ゼロ) 値として処理されます。6/'' など、空の文字列による除算 (/)、整数除算 (¥)、またはモジュロ (#) では、<DIVIDE> エラーが返されます。

4.4.3 NULL の長さ

SQL 内では、NULL の長さは未定義です (<null> を返します)。しかし、空文字列の長さは、長さゼロとして定義されています。詳細は、以下の例を参照してください。

SQL

```
SELECT LENGTH(NULL) AS NullLen,   -- returns <null>
       LENGTH('') AS EmpStrLen    -- returns 0
```

この例で示すように、SQL LENGTH 関数は SQL の長さを返します。

以下の例に示すように、ASCII 関数を使用して、長さゼロの SQL 文字列を NULL に変換できます。

SQL

```
SELECT LENGTH(NULL) AS NullLen,                  -- returns <null>
       LENGTH({fn ASCII('')}) AS AsciiEmpStrLen, -- returns <null>
       LENGTH('') AS EmpStrLen                   -- returns 0
```

ただし、標準 SQL に対する特定の InterSystems IRIS の拡張は、NULL と空文字列のそれぞれの長さを処理します。$LENGTH 関数は、これらの値の InterSystems IRIS 内部表現を返します。NULL は、長さゼロの定義済みの値として表され、SQL の空文字列は、長さゼロの文字列として表されます。この機能は、ObjectScript と互換性があります。

SQL

```
SELECT $LENGTH(NULL) AS NullLen,    -- returns 0
$LENGTH('') AS EmpStrLen,           -- returns 0
$LENGTH('a') AS OneCharStrLen,      -- returns 1
$LENGTH(CHAR(0)) AS CharZero        -- returns 0
```

これらの値の内部表現が有効な別の場所は、%STRING、%SQLSTRING、および %SQLUPPER 関数内です。これらの関数は、空白スペースを値に追加します。NULL は実際には値を持たないため、それに空白を追加すると、長さが 1 の文字列が作成されます。一方、空文字列は文字値を持つため、それに空白を追加すると、長さが 2 の文字列が作成されます。以下の例を参照してください。

SQL

```
SELECT CHAR_LENGTH(%STRING(NULL)) AS NullLen,  -- returns 1
CHAR_LENGTH(%STRING('')) AS EmpStrLen          -- returns 2
```

この例では、LENGTH ではなく CHAR_LENGTH が使用されることに注意してください。LENGTH 関数は末尾の空白を削除するため、`LENGTH(%STRING(NULL))` は長さ 0 を返します。`LENGTH(%STRING(''))` は、%STRING が末尾の空白ではなく先頭の空白を付加するため、長さ 2 を返します。

4.4.4 ObjectScript と SQL

SQL の NULL が ObjectScript に出力された場合、その SQL の NULL は、ObjectScript の空文字列 (""）、つまり長さゼロの文字列で表されます。

長さゼロの SQL 文字列データが ObjectScript に出力された場合、そのデータは、長さ 1 の文字列である $CHAR(0) を含む文字列で表されます。

ObjectScript

```
   &sql(SELECT NULL,''
        INTO :a,:b)

  WRITE !,"NULL length: ",$LENGTH(a)         // returns 0
  WRITE !,"empty string length: ",$LENGTH(b) // returns 1
```

ObjectScript では、通常、値が存在しないことを空文字列 (""）で示します。この値が埋め込み SQL に渡された場合は、以下の例に示すように、NULL 値として処理されます。

ObjectScript

```
  set x=""
  set myquery="SELECT NULL As NoVal,:x As EmpStr"
  set tStatement=##class(%SQL.Statement).%New()

  set qStatus = tStatement.%Prepare(myquery)
  if $$$ISERR(qStatus) {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}

  set rset = tStatement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
    write "NoVal:",rset.%Get("NoVal")," length ",$LENGTH(rset.%Get("NoVal")),! // length 0
    write "EmpStr:",rset.%Get("EmpStr")," length ",$LENGTH(rset.%Get("EmpStr")),! // length 0
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}

  write "End of data"
```

定義されていない入力ホスト変数を指定した場合、埋め込み SQL はその値を NULL として処理します。

以下の例では、空白を追加された SQL 空文字列は、文字列長 2 として渡されます。

ObjectScript

```
   &sql(SELECT %SQLUPPER('')
        INTO :y )
  WRITE !,"SQL empty string length: ",$LENGTH(y)
```

4.5 算術演算子と算術関数

InterSystems SQL は以下の算術演算子をサポートします。

演算子	説明
+	加算演算子。例えば、17+7 は 24 に等しくなります。
-	減算演算子。例えば、17-7 は 10 に等しくなります。これらの文字のペアは、InterSystems SQL コメント文字であることに注意してください。このため、複数の減算演算子つまりマイナス記号を指定するには、スペースまたは括弧を使用する必要があります。例えば、17- -7 または 17-(-7) は 24 と等しくなります。
*	乗算演算子。例えば、17*7 は 119 に等しくなります。
/	除算演算子。例えば、17/7 は 2.428571428571428571 に等しくなります。
¥	整数除算演算子。例えば、17\7 は 2 に等しくなります。
#	モジュロ演算子。例えば、17 # 7 は 3 に等しくなります。# 文字は有効な識別子文字でもあるため、これをモジュロ演算子として使用するには、前後にスペースを入れ、オペランドと区切って指定する必要があります。
E	指数 (科学的記数法) 演算子。大文字小文字のいずれでも使用できます。例えば、7E3 は 7000 に等しくなります。指数が大きすぎると、SQLCODE -7 “指数が範囲外です” というエラーが発生します。例えば、1E309 や 7E308 の場合などです。
()	グループ化演算子。算術演算子の入れ子に使用します。演算子は ANSI 演算子の優先順位に従って実行されます。例えば、17+7*2 は 31 ですが、(17+7)*2 は 48 となります。
||	結合演算子。例えば、17||7 は 177 に等しくなります。

算術演算は、キャノニック形式の数字に対して実行されます。

4.5.1 結果のデータ型

データ型が異なる 2 つの数値に対して算術演算を実行した場合、結果のデータ型は以下のように決定されます。

加算 (+)、減算 (-)、整数除算 (¥)、およびモジュロ (#) の場合 :

データ型	NUMERIC	INTEGER	TINYINT	SMALLINT	BIGINT	DOUBLE
NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	DOUBLE
INTEGER	NUMERIC	BIGINT	BIGINT	BIGINT	BIGINT	DOUBLE
TINYINT	NUMERIC	BIGINT	SMALLINT	INTEGER	BIGINT	DOUBLE
SMALLINT	NUMERIC	BIGINT	INTEGER	INTEGER	BIGINT	DOUBLE
BIGINT	NUMERIC	BIGINT	BIGINT	BIGINT	BIGINT	DOUBLE
DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE

乗算 (*) または除算 (/) の場合 :

データ型	NUMERIC	INTEGER	TINYINT	SMALLINT	BIGINT	DOUBLE
NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	DOUBLE
INTEGER	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	DOUBLE
TINYINT	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	DOUBLE
SMALLINT	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	DOUBLE
BIGINT	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	DOUBLE
DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE

任意のデータ型の 2 つの数値を連結すると、VARCHAR 文字列になります。

ダイナミック SQL では、SQL 列のメタデータを使用して、結果セット・フィールドのデータ型を確認できます。数値データ型の詳細は、SQL データ型を参照してください。

4.5.2 演算子の優先順位

SQL-92 標準は演算子の優先順位に関して不正確です。このことについての前提は SQL 実装間で異なります。InterSystems SQL は、以下のいずれかのタイプの優先順位をサポートするよう構成できます。

- InterSystems IRIS 2019.1 以降の場合、InterSystems SQL では、既定で算術演算子の ANSI 優先順位がサポートされます。これは、システム全体の構成設定です。ANSI の優先順位を構成した場合、演算子 "*"、"¥"、"/"、および "#" は、演算子 "+"、"-"、および "||" よりも優先順位が高くなります。優先順位の高い演算子は、優先順位の低い演算子より先に実行されます。したがって `3+3*5` は 18 になります。この優先順位は、必要に応じて括弧を使用することでオーバーライドできます。したがって `(3+3)*5` は 30 になります。

  既定の ANSI 優先順位は、InterSystems IRIS 2019.1 のクリーン・インストールの場合にサポートされます。InterSystems IRIS 2018.1 を InterSystems IRIS 2019.1 にアップグレードした場合は、演算子の優先順位はそのまま InterSystems IRIS 2018.1 の既定である厳密な左から右の順序に構成されます。

- InterSystems IRIS 2018.1 の場合、InterSystems SQL では、既定で算術演算子に優先順位はありません。既定では、InterSystems SQL には演算子の優先順位はなく、算術式は必ず左から右の順番で実行されます。これは、ObjectScript で使用される規則と同じです。したがって `3+3*5` は 30 になります。演算順位を指定するには、括弧を使用します。したがって `3+(3*5)` は 18 に等しくなります。開発者は、注意深く括弧を使用して、意図を明示する必要があります。

$SYSTEM.SQL.Util.SetOption() メソッドを次のように使用して、システム全体でどちらのタイプの SQL 演算子の優先順位も構成できます。`SET status=$SYSTEM.SQL.Util.SetOption("ANSIPrecedence",1,.oldval)` は、ANSI の優先順位を設定し、`SET status=$SYSTEM.SQL.Util.SetOption("ANSIPrecedence",0,.oldval)` は、厳密な左から右への評価を設定します。現在の設定を確認するには $SYSTEM.SQL.CurrentSettings() を呼び出します。これにより、`[ANSI ]` オプションが表示されます。この SQL オプションの変更内容は、すぐにシステム全体で有効になります。このオプションを変更すると、すべてのクエリ・キャッシュがシステム全体で削除されます。

SQL の優先順位を変更しても、ObjectScript には影響しません。ObjectScript は常に、厳密に左から右へ算術演算子を実行します。

4.5.3 精度と小数桁数

NUMERIC 結果の 精度 (数値内の最大桁数) は、以下のようになります。

- 加算または減算は、resultprecision=max(scale1, scale2) + max(precision1−scale1, precision2−scale2)+1 というアルゴリズムによって決定されます。計算値 resultprecision が 36 より大きい場合、有効桁数の値は 36 に設定されます。
- 乗算は、resultprecision=min(36, precision1+precision2+1) というアルゴリズムによって決定されます。
- 除算 (value1 / value2) は、resultprecision=min(36, precision1−scale1+scale2+max(6, scale1+precision2+1)) というアルゴリズムによって決定されます。

NUMERIC 結果の スケール (小数部の最大桁数) は、以下のようになります。

- 加算または減算は、resultscale=max(scale1, scale2) というアルゴリズムによって決定されます。
- 乗算は、resultscale=min(17, scale1+scale2) というアルゴリズムによって決定されます。
- 除算 (value1 / value2) は、resultscale=min(17, max(6, scale1+precision2+1)) というアルゴリズムによって決定されます。

データ型、有効桁数、および小数桁数の詳細は、SQL の "データ型" を参照してください。

4.5.4 算術関数と三角関数

InterSystems SQL は以下の算術関数をサポートします。

関数	説明
ABS	数値式の絶対値を返します。
CEILING	数値式以上の最も近い整数が返されます。
EXP	数値式の Log 指数値 (基数 e) の値を返します。
FLOOR	数値式以下の最大の整数が返されます。
GREATEST	コンマ区切りの数値のリストから最大の数値を返します。
ISNUMERIC	式が有効な数値かどうかを示すブーリアン値コードを返します。
LEAST	コンマ区切りの数値のリストから最小の数値を返します。
LOG	数値式の自然対数 (基数 e) 値を返します。
LOG10	数値式の基底 10 の対数値を返します。
MOD	除算演算の剰余値を返します。# 演算子と同じです。
PI	数値定数 pi を返します。
POWER	数値式の指定した累乗の値を返します。
ROUND	指定した桁数に丸めた (または切り捨てた) 数値式を返します。
SIGN	数値式の値が正、ゼロ、または負のいずれになるかを示す数値コードを返します。
SQRT	数値式の平方根の値を返します。
SQUARE	数値式の 2 乗の値を返します。
TRUNCATE	指定された桁数に切り捨てた数値式を返します。

InterSystems SQL は以下の三角関数をサポートします。

関数	説明
ACOS	数値式のアークコサインを返します。
ASIN	数値式のアークサインを返します。
ATAN	数値式のアークタンジェントを返します。
COS	数値式のコサインを返します。
COT	数値式のコタンジェントを返します。
SIN	数値式のサインを返します。
TAN	数値式のタンジェントを返します。

InterSystems SQL は以下の角度変換関数もサポートします。

関数	説明
DEGREES	ラジアンを角度に変換します。
RADIANS	角度をラジアンに変換します。

4.6 関係演算子

条件式からブーリアン値が求められます。条件式では以下の関係演算子を使用できます。

演算子	説明
=	等値演算子
!= <>	不等値演算子。2 つの構文形式は機能的に同じです。
<	より小さい演算子
>	より大きい演算子
<=	以下演算子
>=	以上演算子

これらの等値演算子は、テーブルのフィールド値を比較するときにはフィールドの既定の照合を使用します。InterSystems IRIS の既定では大文字と小文字が区別されません。2 つのリテラルを比較するときには、大文字と小文字が区別されます。

浮動小数点数の比較の際には、等値演算子 (等しい、等しくない) は避ける必要があります。浮動小数点数 (データ型のクラス %Library.Decimal および %Library.Double) は固定有効桁数ではなく、バイナリ値として格納されます。変換の際に、丸め処理によって 2 つの浮動小数点数となり、正確には等しくない同等の数値となる場合があります。未満であるか、より大きいかのテストを使用して、2 つの浮動小数点数が所定の精度と“同等”であるかどうかを判断してください。

4.6.1 包含関係演算子と後続関係演算子

InterSystems SQL では、Contains 比較演算子と Follows 比較演算子もサポートされています。

演算子	説明
[	包含関係演算子。オペランドを含んでいる値をすべて返します（オペランドと等しい値が含まれます）。この演算子は、EXACT（大文字と小文字を区別する）照合を使用します。否定は NOT[です。

- 包含関係演算子は、指定された文字または文字列が値に含まれているかどうかを判断します。大文字と小文字が区別されます。
- %STARTSWITH 述語条件は、指定された文字または文字列で値が始まるかどうかを判断します。大文字と小文字は区別されません。
- InterSystems SQL Search を使用して、指定された単語や語句が値に含まれているかどうかを判断できます。SQL Search は、コンテキスト認識のマッチングを実行します。大文字と小文字は区別されません。

演算子	説明
]	後続関係演算子。照合シーケンスでオペランドに続く値をすべて返します。オペランドの値自体は除かれます。この演算子は、フィールドの既定の照合を使用します。InterSystems IRIS の既定では大文字と小文字が区別されません。否定は NOT] です。

例えば、`SELECT Age FROM MyTable WHERE Age ] 88` は 89 以上を返しますが、9 も返します。9 は照合シーケンスでは 88 の後であるためです。`SELECT Age FROM MyTable WHERE Age > 88` は 89 以上を返し、9 は返しません。'ABC' などの文字列オペランドは、'ABCA' など、追加の文字を含む文字列の前に照合されます。このため、オペランド文字列を] 演算子または > 演算子から除外するには、文字列全体を指定する必要があります。`Name ] 'Smith,John'` は、'Smith,John' を除外しますが、'Smith,John P' は除外しません。

4.7 論理演算子

SQL 論理演算子は、True または False として評価する条件式で使用されます。これらの条件式は SELECT 文の WHERE 節と HAVING 節、CASE 文の WHERE 節、JOIN 文の ON 節、および CREATE TRIGGER 文の WHEN 節で使用されます。

4.7.1 単項否定演算子

単項否定論理演算子を使用して、論理的に逆の条件を指定します。以下に例を示します。

SQL

```
SELECT Name,Age FROM Sample.Person
WHERE NOT Age>21
ORDER BY Age
```

SQL

```
SELECT Name,Age FROM Sample.Person
WHERE NOT Name %STARTSWITH('A')
ORDER BY Name
```

NOT 演算子を条件の前に配置できます（上記のとおり）。または、NOT を単一文字演算子の直前に配置できます。例えば、`NOT<`、`NOT[` などです。NOT とそれによって否定する単一文字演算子の間には、スペースを入れてはなりません。

4.7.2 AND 演算子および OR 演算子

一連の複数条件で 2 つのオペランド間に AND 論理演算子および OR 論理演算子を使用できます。これらの論理演算子は、キーワードまたは記号で指定できます。

演算子	説明
AND	&
OR	!

記号演算子とオペランドの間にスペースは必要ありません (ただし読みやすくするためにスペースを入れることをお勧めします)。キーワード演算子の前後にはスペースが必要です。

これらの論理演算子は、単項否定論理演算子と組み合わせることができます。`WHERE Age<65 & NOT Age=21` のように使用します。

以下の 2 つの例では、論理演算子を使用して、年齢に基づいた課税をスケジュールします。年齢が 20 ～ 40 歳の人は 3 年ごとに、40 ～ 64 歳の人は 2 年ごとに、そして 65 歳以上の人は毎年課税します。以下の例では結果が同じになります。最初の例はキーワードを使用し、2 番目の例は記号を使用しています。

SQL

```
SELECT Name,Age FROM Sample.Person
WHERE Age>20
      AND Age<40 AND (Age # 3)=0
      OR Age>=40 AND (Age # 2)=0
      OR Age>=65
ORDER BY Age
```

SQL

```
SELECT Name,Age FROM Sample.Person
WHERE Age>20
      & Age<40 & (Age # 3)=0
      ! Age>=40 & (Age # 2)=0
      ! Age>=65
ORDER BY Age
```

括弧を使用すると論理演算子をグループにできます。これにより、グループ・レベルを設定できます。評価は、最下位のグループから最上位のグループの順で行われます。以下の最初の例では、AND 条件は 2 番目の OR 条件にのみ適用されます。MA からすべての年齢の人物を返し、NY から 25 歳未満の人物を返します。

SQL

```
SELECT Name,Age,Home_State FROM Sample.Person
WHERE Home_State='MA' OR Home_State='NY' AND Age < 25
ORDER BY Age
```

グループ条件で括弧を使用すると、異なる結果になります。以下の例は、MA または NY から、年齢が 25 歳未満の人物を返します。

SQL

```
SELECT Name,Age,Home_State FROM Sample.Person
WHERE (Home_State='MA' OR Home_State='NY') AND Age < 25
ORDER BY Age
```

- SQL の実行は、簡易版のロジックを使用します。条件が偽になると、残りの AND 条件はテストされません。条件が真になると、残りの OR 条件はテストされません。

- ただし、SQL は、WHERE 節の実行を最適化するため、同じグループ・レベルでの複数条件の実行順序は予測できず、有効ではありません。

4.8 コメント

InterSystems SQL は、1 行のコメントと複数行のコメントの両方をサポートします。コメント・テキストには、任意の文字や文字列を含めることができます。ただし当然ながら、コメントの末尾を示す文字は使用できません。

注釈　埋め込み SQL のマーカ構文 (`&sql<marker>(...)<reversemarker>`) を使用すると、SQL コメントの内容に制約が課されます。マーカ構文を使用する場合には、SQL コード内のコメントに文字シーケンス “`)<reversemarker>`” は使用できません。詳細は、このドキュメントの “埋め込み SQL の使用法” の章の ″&sql 指示文″ を参照してください。

preparse() メソッドを使用して、コメントを削除した SQL DML 文を返すことができます。さらに、preparse() メソッドは各クエリ引数を ? 文字に置き換えて、これらの引数の %List 構造を返します。以下の例で、preparse() メソッドは、1 行および複数行のコメントと空白が削除された解析バージョンのクエリを返します。

ObjectScript

```
SET myq=4
SET myq(1)="SELECT TOP ? Name /* first name */, Age "
SET myq(2)="   FROM Sample.MyTable -- this is the FROM clause"
SET myq(3)="   WHERE  /* various conditions "
SET myq(4)="apply */ Name='Fred' AND Age > 21 -- end of query"
DO ##class(%SQL.Statement).preparse(.myq,.stripped,.args)
WRITE stripped,!
WRITE $LISTTOSTRING(args)
```

4.8.1 1 行コメント

1 行のコメントを指定するには、行頭に 2 つのハイフンを記述します。コメントは独立した行として記述できるほか、SQL コードと同じ行に記述することもできます。SQL コードの後にコメントを続ける場合は、コードの末尾に空白を 1 つ以上置き、その後にハイフン 2 つのコメント演算子を記述します。コメントには、ハイフン、アスタリスク、スラッシュも含め、あらゆる文字を使用できます。その行の最後までがコメントです。

以下の例では、複数の 1 行コメントが記述されています。

SQL

```
-- This is a simple SQL query
-- containing -- (double hyphen) comments
SELECT TOP 10 Name,Age, -- Two columns selected
Home_State -- A third column
FROM Sample.Person -- Table name
-- Other clauses follow
WHERE Age > 20 AND -- Comment within a clause
Age < 40
ORDER BY Age,  -- Comment within a clause
Home_State
-- End of query
```

4.8.2 複数行コメント

複数行のコメントを指定するには、/* 開始区切り文字と */ 終了区切り文字を使用します。コメントは、1 行または複数の別々の行として記述することも、SQL コードと同じ行で開始または終了することもできます。少なくとも 1 つの空白を使用して、コメント区切り文字を SQL から区切る必要があります。コメントには、ハイフン、アスタリスク、スラッシュも含め、あらゆる文字を記述できます。ただし、*/ 文字の組み合わせは明らかに例外です。

注釈　構文 /*#OPTIONS */ (/* と # の間にスペースなし) は、コメント・オプションを指定します。コメント・オプションは、コメントではありません。これは、クエリ・オプティマイザが SQL クエリのコンパイル時に使用するコード・オプションを指定します。コメント・オプションは、JSON 構文 (通常は、/*#OPTIONS {"optionName":value} */ などの key:value ペア) を使用して指定します。

以下の例では、複数の複数行コメントが記述されています。

SQL

```
/* This is
   a simple
   SQL query. */
SELECT TOP 10 Name,Age /* Two fields selected */
FROM Sample.Person  /* Other clauses
could appear here */ ORDER BY Age
/* End of query */
```

埋め込み SQL コードをコメントアウトする場合、必ずコメントを &sql 指示文の前または括弧内で始めます。以下の例では、2 つの埋め込み SQL コード・ブロックが正しくコメントアウトされています。

ObjectScript

```
   SET a="default name",b="default age"
   WRITE "(not) Invoking Embedded SQL",!
   /*&sql(SELECT Name INTO :a FROM Sample.Person) */
   WRITE "The name is ",a,!
   WRITE "Invoking Embedded SQL (as a no-op)",!
   &sql(/* SELECT Age INTO :b FROM Sample.Person */)
   WRITE "The age is ",b
```

4.8.3 コメントとして保持される SQL コード

埋め込み SQL 文は、ルーチンの .INT コード・バージョンのコメントとして保持することができます。これをシステム全体で行うには、$SYSTEM.SQL.Util.SetOption() メソッドを SET status=$SYSTEM.SQL.Util.SetOption("RetainSQL",1,.oldval) のように設定します。現在の設定を確認するには、$SYSTEM.SQL.CurrentSettings() を呼び出します。これにより、[.INT SQL] の設定が表示されます。既定は 1 (“はい”) です。

このオプションを [はい] に設定すると、ルーチンの .INT コード・バージョンで SQL 文をコメントとして維持できます。また、“はい” に設定することで、コメントのテキストの中でその SQL 文によって使用されている非 % 変数をすべてリストにできます。このリストの変数を ObjectScript プロシージャの PUBLIC 変数リストにもリストし、NEW コマンドで再び初期化する必要があります。詳細は、このドキュメントの “埋め込み SQL” の章の ″ホスト変数″ を参照してください。

5
暗黙結合 (矢印構文)

InterSystems SQL には、関連するテーブルから値を取得するための省略手段として、特殊な -> 演算子があります。これにより、特定のケースで、明示的に JOIN を指定するという複雑な作業を省くことができます。この矢印構文を明示的な結合構文の代わりに使用したり、明示的な結合構文と組み合わせて使用したりできます。矢印構文は左外部結合を実行します。

矢印構文は、クラスのプロパティまたは親テーブルのリレーションシップ・プロパティの参照に使用できます。その他のタイプのリレーションシップおよび外部キーでは矢印構文はサポートされません。ON 節では矢印構文 (->) を使用できません。

シャード・テーブルが関与するクエリでは、矢印構文を使用できます。

詳細は、"InterSystems SQL リファレンス" の "JOIN" のページを参照してください。

5.1 プロパティの参照

“参照されたテーブル” から値を取得するための省略手段として、-> 演算子を使用できます。

例えば、Company と Employee という 2 つのクラスを定義するとします。

Class Definition

```
Class Sample.Company Extends %Persistent [DdlAllowed]
{
/// The Company name
Property Name As %String;
}
```

また、Employee は以下のように定義します。

Class Definition

```
Class Sample.Employee Extends %Persistent [DdlAllowed]
{
/// The Employee name
Property Name As %String;

/// The Company this Employee works for
Property Company As Company;
}
```

Employee クラスは、Company オブジェクトへの参照のプロパティを含みます。オブジェクト・ベースのアプリケーションでは、ドット構文を使用してこの参照を表すことができます。例えば、会社員 (employee) が所属する会社 (company) を見つけるには、以下を使用します。

ObjectScript

```
 Set name = employee.Company.Name
```

SQL 文で同じ作業を実行するには、OUTER JOIN により Employee と Company のテーブルを結合します。

SQL

```
SELECT Sample.Employee.Name, Sample.Company.Name AS CompName
FROM Sample.Employee LEFT OUTER JOIN Sample.Company
ON Sample.Employee.Company = Sample.Company.ID
```

-> 演算子を使用すると、同様の OUTER JOIN 操作をより簡潔に実行できます。

SQL

```
SELECT Name, Company->Name AS CompName
FROM Sample.Employee
```

テーブルに参照列がある場合はいつでも -> 演算子を使用できます。この列の値は、参照されたテーブルの ID です (本来は、外部キーの特殊なケースです)。この場合は、Sample.Employee の Company フィールドには、Sample.Company テーブル内のレコードの ID が含まれています。クエリで列の式を使用できる場所であれば、どこでも -> 演算子を使用できます。例えば、以下の WHERE 節を考えます。

SQL

```
SELECT Name,Company AS CompID,Company->Name AS CompName
FROM Sample.Employee
WHERE Company->Name %STARTSWITH 'G'
```

これは以下と同じです。

SQL

```
SELECT E.Name,E.Company AS CompID,C.Name AS CompName
FROM Sample.Employee AS E, Sample.Company AS C
WHERE E.Company = C.ID AND C.Name %STARTSWITH 'G'
```

この場合、対応するクエリは INNER JOIN を使用しています。

以下の例では、矢印構文を使用して Sample.Person 内の Spouse フィールドにアクセスします。この例で示されるように、Sample.Employee 内の Spouse フィールドには、Sample.Person 内のレコードの ID が含まれています。この例では、従業員の Home_State または Office_State とその従業員の配偶者の Home_State が同じであるレコードを返します。

SQL

```
SELECT Name,Spouse,Home_State,Office_State,Spouse->Home_State AS SpouseState
FROM Sample.Employee
WHERE Home_State=Spouse->Home_State OR Office_State=Spouse->Home_State
```

GROUP BY 節で -> 演算子を使用できます。

SQL

```
SELECT Name,Company->Name AS CompName
FROM Sample.Employee
GROUP BY Company->Name
```

ORDER BY 節で -> 演算子を使用できます。

SQL

```
SELECT Name,Company->Name AS CompName
FROM Sample.Employee
ORDER BY Company->Name
```

または、ORDER BY 節で -> 演算子列の列エイリアスを参照します。

SQL

```
SELECT Name,Company->Name AS CompName
FROM Sample.Employee
ORDER BY CompName
```

以下の例で示すように、複合矢印構文がサポートされています。この例では、Cinema.Review テーブルには Film フィールドが含まれており、それには Cinema.Film テーブルの行 ID が含まれています。Cinema.Film テーブルには Category フィールドが含まれており、それには Cinema.Category テーブルの行 ID が含まれています。したがって、ReviewScore がある各フィルムの CategoryName を返すには、`Film->Category->CategoryName` によってこれら 3 つのテーブルにアクセスします。

SQL

```
SELECT ReviewScore,Film,Film->Title,Film->Category,Film->Category->CategoryName
FROM Cinema.Review
ORDER BY ReviewScore
```

5.2 子テーブルの参照

-> 演算子を使用して、子テーブルを参照できます。例えば、LineItems が Orders テーブルの子テーブルであるとき、以下を指定できます。

SQL

```
SELECT LineItems->amount
FROM Orders
```

Orders には、LineItems というプロパティはありません。LineItems は、`amount` フィールドが含まれる子テーブルの名前です。このクエリによって、各 Order 列の結果セットに複数の行が生成されます。これは、以下と同等です。

SQL

```
SELECT L.amount
FROM Orders O LEFT JOIN LineItems L ON O.id=L.custorder
```

ここで、`custorder` は、LineItems テーブルの親参照フィールドです。

5.3 矢印構文の特権

矢印構文を使用する場合、両方のテーブルの参照対象データに対する SELECT 特権が必要です。テーブルレベルの SELECT 特権、または参照対象列に対する列レベルの SELECT 特権が必要です。列レベルの特権では、参照対象テーブルの ID および参照対象列に対する SELECT 特権が必要です。

以下の例は、必要な列レベルの特権を示しています。

SQL

```
SELECT Name,Company->Name AS CompanyName
FROM Sample.Employee
GROUP BY Company->Name
ORDER BY Company->Name
```

上記の例では、Sample.Employee.Name、Sample.Company.Name、および Sample.Company.ID に対して列レベルの SELECT 特権が必要です。

ObjectScript

```
  SET tStatement = ##class(%SQL.Statement).%New()
  SET privchk1="%CHECKPRIV SELECT (Name,ID) ON Sample.Company"
  SET privchk2="%CHECKPRIV SELECT (Name) ON Sample.Employee"
CompanyPrivTest
  SET qStatus = tStatement.%Prepare(privchk1)
    IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
  SET rset = tStatement.%Execute()
  IF rset.%SQLCODE=0 {WRITE !,"have Company privileges",! }
  ELSE {  WRITE !,"No privilege: SQLCODE=",rset.%SQLCODE,! }
EmployeePrivTest
  SET qStatus = tStatement.%Prepare(privchk2)
    IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
  SET rset = tStatement.%Execute()
  IF rset.%SQLCODE=0 {WRITE !,"have Employee privilege",! }
  ELSE {  WRITE !,"No privilege: SQLCODE=",rset.%SQLCODE }
```

6
識別子

識別子は、テーブル、ビュー、列 (フィールド)、スキーマ、テーブル・エイリアス、列のエイリアス、インデックス、ストアド・プロシージャ、トリガなどの SQL エンティティの名前です。識別子は、そのコンテキスト内で一意であることが必要です。例えば、同じスキーマ内の 2 つのテーブル、または同じテーブル内の 2 つのフィールドは、同じ名前を持つことはできません。ただし、異なるスキーマ内の 2 つのテーブル、または異なるテーブル内の 2 つのフィールドには同じ名前を指定できます。ほとんどの場合、異なるタイプの SQL エンティティに同じ識別子名を使用できます。例えば、スキーマ、そのスキーマ内のテーブル、およびそのテーブル内のフィールドには、すべて同じ名前を使用でき、競合が発生することはありません。ただし、同じスキーマ内のテーブルおよびビューには、同じ名前を指定できません。

InterSystems IRIS® データ・プラットフォームの SQL 識別子は、一連の名前付け規約に従います。識別子の使用方法によっては制約が厳しくなる場合があります。識別子は大文字と小文字を区別しません。

識別子は単純な識別子か区切り識別子のいずれかです。InterSystems SQL は既定で単純な識別子と区切り識別子の両方をサポートします。

6.1 単純な識別子

単純な識別子の構文は以下のとおりです。

```
simple-identifier ::= identifier-start { identifier-part }
      identifier-start ::= letter | % | _
      identifier-part ::=  letter | number | _ | @ | # | $
```

6.1.1 名前付け規約

identifier-start は、SQL 識別子の最初の文字です。これは以下のいずれかである必要があります。

- 大文字または小文字。非数字文字とは、ObjectScript $ZNAME 関数の検証に合格する任意の文字であると定義されます。既定では、A から Z までの大文字 (ASCII 65-90)、a から z までの小文字 (ASCII 97-122)、およびアクセント記号付きの文字 (ASCII 215 と 247 を除く ASCII 192-255) です。InterSystems IRIS では、有効な Unicode (16 ビット) 文字を SQL 識別子内で使用できます。単純な識別子は、大文字と小文字を区別しません (ただし、下記を参照してください)。規約により、最初は大文字で表示されます。

  日本のロケールでは、アクセント記号付きのラテン文字を識別子でサポートしていません。日本の識別子には、日本語の文字に加え、ラテン文字 A-Z と a-z (65-90 と 97-122) とギリシャ語の大文字 (913-929 および 931-937) を使用できます。

- アンダースコア (_)

- パーセント記号 (%)。% 文字で始まる InterSystems IRIS 名 (%Z または %z で始まるものは除く) は、システム要素として予約されているので、識別子として使用しないでください。詳細は、″サーバ側プログラミングの入門ガイド″ の “識別子のルールとガイドライン” の章を参照してください。

identifier-part は、SQL 識別子の後続文字です。これら残りの部分は、以下に示した 0 個以上の文字で構成されます。

- 文字 (Unicode 文字を含む)
- 数字。数字文字は、0 から 9 までの数字で定義されます。
- アンダースコア (_)
- アット記号 (@)
- ポンド記号 (#)
- ドル記号 ($)

一部の記号文字は、演算子としても使用されます。SQL では、# 記号はモジュロ演算子として使用されます。SQL では、アンダースコア文字は 2 つの文字列の連結に使用できます。この使用法は ObjectScript との互換性のために用意されており、推奨される SQL 連結演算子は || です。識別子文字としての記号の解釈は、必ず演算子としてのその解釈より優先されます。記号文字の演算子としての解析のあいまいさは、演算子の前後にスペースを配置することで解決できます。

単純な識別子は、空白や上記で指定されている記号文字以外の非英数文字を含むことができません。インターシステムズの SQL インポート・ツールは、インポートされたテーブル名から空白を削除します。

注釈　SQL カーソル名は、識別子の名前付け規約に従いません。カーソルの名前付け規約の詳細は、″DECLARE 文″ を参照してください。

InterSystems SQL には、単純な識別子として使用できない予約語があります。これらの予約語のリストについては、″InterSystems SQL リファレンス″ の “予約語” のセクションを参照してください。ある単語が予約語かどうかをテストするには、$SYSTEM.SQL.IsReservedWord() メソッドを使用します。ただし、区切り識別子は、SQL 予約語と同じであってもかまいません。

これらの名前付け規約に反する識別子は、SQL 文の中で区切り識別子として表示する必要があります。

6.1.2 文字の大文字小文字

InterSystems SQL 識別子は、既定では大文字と小文字を区別しません。InterSystems SQL は、最初に識別子をすべて大文字に変換してから識別子を比較することで、大文字と小文字の無区別を実装しています。これは、使用されている名前の実際の文字には何の影響もありません (他の SQL の実装では、識別子の大文字と小文字の区別が異なる方法で処理される場合があります。したがって、大文字と小文字を意識した識別子の使用を避けることをお勧めします)。

InterSystems SQL のカーソル名とパスワードでは大文字と小文字が区別されることに注意してください。

6.1.3 有効な識別子のテスト

InterSystems IRIS は、文字列が有効な識別子であるかどうかをテストする、%SYSTEM.SQL クラスの IsValidRegularIdentifier() メソッドを提供しています。このメソッドは、文字の使用と予約語の両方に対してテストします。また、最大 200 文字の長さテストも実行します (これは、間違った入力を防止するために使用する任意の長さで、識別子の検証ではありません)。以下の ObjectScript の例は、このメソッドの使用法を示しています。

ObjectScript

```
WRITE !,$SYSTEM.SQL.IsValidRegularIdentifier("Fred")
WRITE !,$SYSTEM.SQL.IsValidRegularIdentifier("%Fred#123")
WRITE !,$SYSTEM.SQL.IsValidRegularIdentifier("%#$@_Fred")
WRITE !,$SYSTEM.SQL.IsValidRegularIdentifier("_1Fred")
WRITE !,$SYSTEM.SQL.IsValidRegularIdentifier("%#$")

WRITE !,$SYSTEM.SQL.IsValidRegularIdentifier("1Fred")
WRITE !,$SYSTEM.SQL.IsValidRegularIdentifier("Fr ed")
WRITE !,$SYSTEM.SQL.IsValidRegularIdentifier("%sqlupper")
```

最初の 3 つのメソッド呼び出しは 1 を返します。これは有効な識別子であることを示しています。4 番目と 5 番目のメソッド呼び出しも 1 を返します。これらは、有効な識別子です。ただし、テーブル名またはフィールド名として使用する場合は有効ではありません。最後の 3 つのメソッド呼び出しは、0 を返します。これは無効な識別子であることを示しています。最後の 3 つのメソッドのうち 2 つが無効なのは、数字で始まっているのと、スペースが含まれているという点で文字規則に違反しているためです。最終のメソッド呼び出しが 0 を返すのは、指定された文字列が予約語であるためです。これらの規則テストは最小要件で、あらゆる SQL 使用で有効な識別子として検証されているわけではないことに注意してください。

このメソッドは、ODBC または JDBC からストアド・プロシージャ %SYSTEM.SQL_IsValidRegularIdentifier(″nnnn″) として呼び出すこともできます。

6.1.4 ネームスペース名

ネームスペース名 (データベース名とも呼ばれます) は識別子の名前付け規約に従います。また、句読点文字および最大長に関する追加の制限があります。詳細は、″CREATE DATABASE″ コマンドを参照してください。

ネームスペース名には区切り識別子を指定でき、これは SQL 予約語と同じであってもかまいません。ただし、同じネームスペース名の句読点制限が、単純な識別子と区切り識別子の両方に適用されます。

6.1.5 識別子とクラスのエンティティ名

SQL テーブル名、ビュー名、フィールド名、インデックス名、トリガ名、およびプロシージャ名は、英数字以外の文字を削除することによって、対応する永続クラス・エンティティの生成に使用されます。クラス・エンティティおよびグローバルに生成される名前は、これらの変換ルールに従います。

注釈　　ネームスペース名と SQL スキーマ名および対応するパッケージ名は、これらの変換ルールには従いません。

- 句読点文字が含まれている点だけが異なる識別子は有効です。クラス・オブジェクト名に句読点文字を含めることはできないため、InterSystems IRIS ではすべての句読点文字を削除することによって、対応する一意のオブジェクト名が生成されます。識別子の句読点文字を削除することで、一意でないクラス・オブジェクト名が生成される場合は、最後の英数字をインクリメントされる文字接尾語と置き換えることによって、一意の名前が生成されます。

  テーブル、ビュー、フィールド、トリガ、およびプロシージャ・クラス・メソッド名の場合、これは 0 で始まる整数の接尾語です。例えば、myname および my_name は、myname および mynam0 を生成し、my#name を追加すると mynam1 が生成されます。生成される一意の名前の数が 10 を超えるときには (mynam9)、A で始まる大文字の接尾語に置き換えることで追加の名前が作成されます (mynamA)。テーブルとビューは同じネームスペースを共有するため、テーブルまたはビューに対して同じ接尾語カウンタがインクリメントされます。

  インデックス名の場合、この接尾語は A で始まる大文字です。例えば、myindex および my_index は myindex および myindeA を生成します。

  接尾語で終了する名前を定義した場合 (my_name0 や my_indexA など)、InterSystems IRIS は次の未使用の接尾語にインクリメントすることによって一意の名前を生成します。

- 最初の文字が句読点で、2 番目の文字が数字である識別子は、テーブル名、ビュー名、およびプロシージャ名としては無効です。フィールド名およびインデックス名としては有効です。SQL フィールド名またはインデックス名の最初

の文字が句読点 (% または _) で、2 番目の文字が数字である場合、InterSystems IRIS は、対応するプロパティ名の最初の文字として小文字の “n” を追加します。

- 句読点文字のみで構成された識別子、2 つのアンダースコア文字で始まる識別子 (__name)、または連続する 2 つのポンド記号を含む識別子 (nn##nn) は、通常、SQL エンティティ名としては無効です。すべてのコンテキストで使用を避ける必要があります。

変換元/変換先の文字のペアのリストを作成して、SQL 識別子内の特定の文字を対応するオブジェクト識別子内で他の文字に変換するように構成することができます。DDL 実行時に SQL 識別子をオブジェクト識別子に変換する場合、“変換元” 文字列の文字は、“変換先” 文字列の対応する文字に変換されます。これらのシステム全体での文字変換により、識別子で指定可能な文字の規則が異なる環境間での識別子の使用が容易になります。
$SYSTEM.SQL.Util.SetDDLIdentifierTranslations() メソッドを使用して、変換元/変換先の文字のペアを設定します。現在の設定を確認するには、$SYSTEM.SQL.CurrentSettings() を呼び出します。

6.1.5.1 クラス定義での SQL 名の指定

SQL エンティティを投影する永続クラスを定義する場合、各 SQL エンティティの名前は、対応する永続クラス定義要素の名前と同じです。別の SQL テーブル名、フィールド名、またはインデックス名を作成する場合は、SqlTableName、SqlFieldName、または SqlName (インデックスの場合) キーワードを使用して、クラス定義内で SQL 名を指定します。以下はその例です。

Class Member

```
Property LName As %String [SqlFieldName = "Family#Name"];
```

Class Member

```
Index NameIdx As %String [SqlName = "FullNameIndex"];
```

6.1.6 識別子の長さの考慮事項

SQL 識別子の最大長は 128 文字です。InterSystems IRIS は、SQL 識別子を対応するオブジェクトのエンティティにマップするときに、そのエンティティに対応する最大 96 文字のプロパティ、メソッド、クエリ、またはインデックス名を作成します。2 つの SQL 識別子の最初の 96 文字が同一の場合、InterSystems IRIS は、それに対応するオブジェクト名の 96 番目の文字を 0 から始まる整数に置き換えて一意の名前を作成します。

スキーマおよびテーブル名の最大長については、さらなる考慮事項と制限があります。このドキュメントの “テーブルの定義” の章の ″テーブル名とスキーマ名″ を参照してください。

6.2 区切り識別子

区切り識別子の構文は以下のとおりです。

```
delimited-identifier ::= " delimited-identifier-part { delimited-identifier-part }
"
    delimited-identifier-part ::= non-double-quote-character | double-quote-symbol
    double-quote-symbol ::= ""
```

区切り識別子は、区切り文字で囲まれた一意の識別子です。InterSystems SQL は、区切り文字として二重引用符 (″) をサポートします。区切り識別子は通常、単純な識別子の名前付けの制約を回避するために使用します。

リテラル値を区切るには、InterSystems SQL は一重引用符 (’) を使用します。これにより、区切り識別子は二重引用符 (″) で、リテラル値は一重引用符 (’) で指定されなければなりません。例えば、’7’ は数値リテラルの 7 ですが、″7″ は区

切り識別子です。SQL 文を二重引用符で囲む場合 (ダイナミック SQL の場合など)、その文字列内の二重引用符文字を二重にする必要があります。

SQL の空文字列は、必ず一重引用符文字のペア '' として指定する必要があります。区切り識別子のサポートが有効な場合、二重引用符文字のペア "" は無効な区切り識別子として解析され、SQLCODE -1 エラーを生成します。

6.2.1 区切り識別子の有効な名前

区切り識別子は、一意の名前でなければなりません。区切り識別子は大文字と小文字を区別しません。規約により、識別子の最初の文字は大文字で表示されます。

区切り識別子は、SQL 予約語と同じであってもかまいません。区切り識別子は、通常、SQL 予約語との名前の競合の問題を避けるために使用されます。

区切り識別子には、空白も含めた、ほぼすべての印刷可能文字を含むことができます。ほとんどの区切り識別子の名前に、コンマ (,)、ピリオド (.)、キャレット (^)、および 2 文字の矢印シーケンス (->) を含めることはできませんが、区切り識別子のロール名およびユーザ名にこれらの文字を含めることはできます。区切り識別子のクラス名には、ピリオド (.) を含めることができます。いずれの区切り識別子でも、先頭にアスタリスク (*) を使用することはできません。%vid は、区切り識別子に使用できません。これらの名前付け規約に違反すると、SQLCODE -1 エラーが返されます。

テーブル名、スキーマ名、列名、またはインデックス名として使用される区切り識別子は、有効なクラス・エンティティ名に変換できる必要があります。このため、1 つ以上の英数字が含まれる必要があります。数字 (または句読点とそれに続く数字) で始まる区切り識別子は、“n” 接頭語を付けて対応するクラス・エンティティ名を生成します。

以下の例で示しているクエリは、列とテーブルの両方の名前に区切り識別子を使用します。

SQL

```
SELECT "My Field" FROM "My Table" WHERE "My Field" LIKE 'A%'
```

区切り識別子は二重引用符で区切られ、文字列リテラル A% は一重引用符で区切られることに注意してください。

区切り識別子をテーブル名に指定する場合は、テーブル名とスキーマ名を別々に区切る必要があります。したがって、"schema"."tablename" または schema."tablename" は、有効な識別子ですが、"schema.tablename" は有効な識別子ではありません。

6.2.2 区切り識別子のサポートの無効化

既定で、区切り識別子のサポートは有効になっています。

区切り識別子のサポートが無効の場合は、二重引用符の中の文字は、文字列リテラルとして処理されます。

SET OPTION コマンドに SUPPORT_DELIMITED_IDENTIFIERS キーワードを指定して、システム全体に区切り識別子のサポートを設定できます。

システム全体に区切り識別子のサポートを設定できますが、そのためには、$SYSTEM.SQL.Util.SetOption() メソッドの DelimitedIdentifiers オプションを使用します。区切り識別子は既定でサポートされています。

現在の設定を確認するには、$SYSTEM.SQL.CurrentSettings() を呼び出します。

6.3 SQL 予約語

SQL には、識別子として使用できない予約語が多数あります。これらの予約語のリストについては、″InterSystems SQL リファレンス″ の “予約語” のセクションを参照してください。

7

埋め込み SQL の使用法

InterSystems IRIS® データ・プラットフォームで使用される ObjectScript コード内に SQL 文を埋め込むことができます。これらの埋め込み SQL 文は、実行時に最適化された実行可能なコードに変換されます。

埋め込み SQL には以下の 2 種類があります。

- 単純な埋め込み SQL クエリは、1 つの行の値のみを返すことができます。また、単純な埋め込み SQL は、単一行を対象とする挿入、更新、および削除や、その他の SQL 操作にも使用できます。
- カーソル・ベースの埋め込み SQL クエリは、クエリ結果セットの繰り返し処理を行って、複数の行の値を返すことができます。また、カーソル・ベースの埋め込み SQL は、複数行を対象とする更新および削除の SQL 操作にも使用できます。

注釈　埋め込み SQL は、ターミナルのコマンド行に入力することも、XECUTE 文で指定することもできません。コマンド行から SQL を実行するには、$SYSTEM.SQL.Execute() メソッドまたは SQL シェル・インタフェースを使用します。

7.1 埋め込み SQL のコンパイル

埋め込み SQL は、これを含むルーチンのコンパイル時にはコンパイルされません。埋め込み SQL のコンパイルは、SQL コードの最初の実行時（ランタイム）に行われます。最初の実行によって、実行可能なクエリ・キャッシュが定義されます。これはダイナミック SQL のコンパイルと似ています。この際、SQL 準備処理が実行されるまで SQL コードはコンパイルされません。

埋め込み SQL コードは、ルーチンの初回実行まで、SQL テーブルおよび他のエンティティに対して検証されません。このため、ルーチンのコンパイル時に存在しないテーブルまたはその他の SQL エンティティを参照する埋め込み SQL を含む、ルーチンまたは永続クラスのメソッドをコンパイルすることができます。この理由により、ほとんどの SQL エラーはコンパイル時ではなく実行時に返されます。

ルーチンのコンパイル時に、埋め込み SQL に対して SQL 構文チェックが実行されます。埋め込み SQL に無効な SQL 構文があった場合、ObjectScript コンパイラが失敗し、コンパイル・エラーが生成されます。

管理ポータルの SQL インタフェースを使用して、SQL コードを実行することなく、埋め込み SQL で指定された SQL エンティティの存在をテストすることができます。これについては、"埋め込み SQL コードの検証" で説明します。埋め込み SQL コードの検証では、SQL 構文の検証と SQL エンティティの存在チェックの両方が行われます。"埋め込み SQL コードの検証" に説明されているように、`/compileembedded=1` 修飾子を使用して埋め込み SQL コードを含むルーチンをコンパイルすることで、ランタイム実行の前に 埋め込み SQL コードを検証することを選択できます。

埋め込み SQL 文が正常に実行されると、クエリ・キャッシュが生成されます。その埋め込み SQL の以降の実行では、埋め込み SQL ソースをリコンパイルする代わりに、クエリ・キャッシュが使用されます。これにより、クエリ・キャッシュのパフォーマンス上のメリットが埋め込み SQL にもたらされます。これらのクエリ・キャッシュは、各テーブルの管理ポータルで **[カタログの詳細]** の [クエリキャッシュ] リストに表示されます。

カーソル・ベースの埋め込み SQL 文のランタイム実行は、OPEN コマンドを使用して初めてカーソルを開くときに行われます。管理ポータルの [SQL 文] リストに示されているように、実行のこの時点で、最適化されたクエリ・キャッシュ・プランが生成されます。**[SQL 文]** リストに表示される **[場所]** は、埋め込み SQL コードを含むルーチンの名前です。埋め込み SQL を実行しても、**[クエリキャッシュ]** リストにエントリは生成されません。`%sqlcq.USER.cls1` などのクラス名を持つこれらのリストは、ダイナミック SQL クエリによって作成されます。

注釈　InterSystems IRIS の旧バージョンで使用されていた #sqlcompile mode プリプロセッサ文は非推奨になりました。解析はされますが、ほとんどの埋め込み SQL コマンドに対して処理は行いません。ほとんどの埋め込み SQL コマンドは、#sqlcompile mode の設定に関係なく、実行時にコンパイルされます。ただし、少数の埋め込み SQL コマンドでは、#sqlcompile mode=deferred に設定することに引き続き意味があります。すべてのタイプの埋め込み SQL コマンドが実行時に強制的にコンパイルされるようになるからです。

7.1.1 埋め込み SQL とマクロ・プリプロセッサ

埋め込み SQL は、メソッド内およびトリガ内 (ObjectScript を使用するように定義されている場合)、または ObjectScript の MAC ルーチン内で使用できます。MAC ルーチンは InterSystems IRIS マクロ・プリプロセッサによって処理され、INT (中間) コードに変換された後、実行可能な OBJ コードにコンパイルされます。これらの処理は、埋め込み SQL コード自体のコンパイル時ではなく、埋め込み SQL を含むルーチンのコンパイル時に実行されます。埋め込み SQL コードは実行時までコンパイルされません。詳細は、"ObjectScript の使用法" の "ObjectScript マクロとマクロ・プリプロセッサ" の章を参照してください。

埋め込み SQL 文自体に InterSystems IRIS マクロ・プリプロセッサ文 (# コマンド、## 関数、または $$$ マクロ参照) が含まれる場合、これらの文はルーチンがコンパイルされ、実行時に SQL コードで使用できるようになったときにコンパイルされます。これにより、ObjectScript コード本文を含む CREATE PROCEDURE、CREATE FUNCTION、CREATE METHOD、CREATE QUERY、CREATE TRIGGER のいずれかの文が影響を受ける可能性があります。

7.1.1.1 埋め込み SQL のインクルード・ファイル

埋め込み SQL 文では、参照するマクロ・インクルード・ファイルを、実行時にシステムにロードする必要があります。

埋め込み SQL のコンパイルは最初の参照まで延期されるため、埋め込み SQL クラスがコンパイルされるコンテキストは、埋め込み SQL を含むクラスやルーチンのコンパイル時環境ではなく、実行時環境になります。実行時現在のネームスペースが、埋め込み SQL を含むルーチンのコンパイル時のネームスペースと異なる場合、コンパイル時のネームスペースのインクルード・ファイルは実行時のネームスペースに表示されないことがあります。この場合、以下が発生します。

1. インクルード・ファイルが実行時のネームスペースに表示されない場合、埋め込み SQL のコンパイルですべてのインクルード・ファイルが削除されます。SQL コンパイルでインクルード・ファイルが必要になることはめったにないため、実行時の埋め込み SQL のコンパイルは多くの場合、インクルード・ファイルなしでも成功します。
2. インクルード・ファイルの削除後にコンパイルに失敗した場合、InterSystems IRIS エラーでルーチンのコンパイル時のネームスペース、埋め込み SQL の実行時のネームスペース、および実行時のネームスペースで表示されないインクルード・ファイルのリストが報告されます。

7.1.1.2 #SQLCompile マクロ指示文

マクロ・プリプロセッサには、埋め込み SQL で使用するための 3 つのプリプロセッサ指示文が用意されています。

- #sqlcompile select は、SELECT 文から返されたときのデータ表示の形式を指定します。また、INSERT 文または UPDATE 文、あるいは SELECT 入力ホスト変数に指定した場合はデータ入力の必須形式を指定します。`Logical`

(既定)、Display、ODBC、Runtime、Text (Display と同義)、および FDBMS (下記参照) の 6 つのオプションがサポートされています。#sqlcompile select=Runtime の場合、$SYSTEM.SQL.Util.SetOption("SelectMode",n) メソッドを使用してデータの表示方法を変更できます。n 値には、0 = 論理、1 = ODBC、または 2 = 表示を指定できます。

#sqlcompile select オプションの指定に関係なく、INSERT または UPDATE は、保存のために、指定されたデータの値を対応する論理形式に自動的に変換します。

#sqlcompile select オプションの指定に関係なく、SELECT は、述語の照合のために、入力ホスト変数の値を対応する論理形式に自動的に変換します。

以下の各例では、クエリの表示に #sqlcompile select を使用する方法を示します。これらの例は、DOB (誕生日) 値を表示してから、SelectMode を ODBC 形式に変更し、再び DOB を表示します。最初の例では、SelectMode を変更しても表示には影響しません。2 番目の例では、#sqlcompile select=Runtime であるため、SelectMode を変更すると表示が変更されます。

ObjectScript

```
#sqlcompile select=Display
&sql(SELECT DOB INTO :a FROM Sample.Person)
   IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
   ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
WRITE "1st date of birth is ",a,!
DO $SYSTEM.SQL.Util.SetOption("SelectMode",1)
WRITE "changed select mode to: ",$SYSTEM.SQL.Util.GetOption("SelectMode"),!
&sql(SELECT DOB INTO :b FROM Sample.Person)
WRITE "2nd date of birth is ",b
```

ObjectScript

```
#sqlcompile select=Runtime
&sql(SELECT DOB INTO :a FROM Sample.Person)
   IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
   ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
WRITE "1st date of birth is ",a,!
DO $SYSTEM.SQL.Util.SetOption("SelectMode",1)
WRITE "changed select mode to: ",$SYSTEM.SQL.Util.GetOption("SelectMode"),!
&sql(SELECT DOB INTO :b FROM Sample.Person)
WRITE "2nd date of birth is ",b
```

SelectMode オプションの詳細は、このドキュメントの“InterSystems IRIS SQL の基礎”の章にある“データ表示オプション”を参照してください。

- #sqlcompile select=FDBMS は、埋め込み SQL で FDBMS と同様にデータをフォーマットできるようにするために用意されています。クエリの WHERE 節に定数値がある場合、FDBMS モードでは、それは Display 値であると見なされ、DisplayToLogical 変換を使用して変換されます。クエリの WHERE 節に変数がある場合は、FDBMS モードでは、FDBMSToLogical 変換を使用してそれが変換されます。FDBMSToLogical 変換メソッドは、Internal、Internal_$c(1)_External、および $c(1)_External という 3 つの FDBMS 変数形式を処理するように設計されている必要があります。クエリで選択して変数に格納する場合は、それによって LogicalToFDBMS 変換メソッドが呼び出されます。このメソッドでは、Internal_$c(1)_External が返されます。

- #sqlcompile path (または #import) は、データ管理コマンド (SELECT、CALL、INSERT、UPDATE、DELETE、TRUNCATE TABLE など) 内の未修飾テーブル名、ビュー名、およびストアド・プロシージャ名を解決するために使用されるスキーマ検索パスを指定します。スキーマ検索パスを指定しない場合、または指定されたスキーマ内にテーブルが見つからない場合、InterSystems IRIS は、既定のスキーマを使用します。#sqlcompile path および #import は、データ定義文 (ALTER TABLE、DROP VIEW、CREATE INDEX、CREATE TRIGGER など) では無視されます。データ定義文では、未修飾名の解決には既定のスキーマが使用されます。

- #sqlcompile audit は、埋め込み SQL 文の実行をシステム・イベント監査ログに書き込む必要があるかどうかを指定するブーリアン・スイッチです。詳細は、"埋め込み SQL の監査" を参照してください。

これらのプリプロセッサ指示文の詳細は、"ObjectScript の使用法" の "システム・プリプロセッサ・コマンド・リファレンス" のセクションを参照してください。

7.2 埋め込み SQL の構文

埋め込み SQL 指示文の構文を以下に示します。

7.2.1 &sql 指示文

埋め込み SQL 文は、以下の例に示すように、&sql() 指示文により、他のコードから区別されています。

ObjectScript

```
NEW SQLCODE,a
WRITE "Invoking Embedded SQL",!
&sql(SELECT Name INTO :a FROM Sample.Person)
   IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
   ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
WRITE "The name is ",a
```

結果は、1 つ以上のホスト変数を指定する INTO 節を使用して返されます。この場合、ホスト変数には :a という名前が付けられています。詳細は、この章の “ホスト変数” のセクションを参照してください。ここには、SQLCODE とホスト変数との間のやりとりについての説明があります。

&sql 指示文は大文字と小文字を区別しないため、&sql、&SQL、&Sql などを使用できます。&sql 指示文の直後には、開始の括弧が続く必要があります (間にスペース、改行、またはコメントは使用できません)。&sql 指示文は、以下の例のように、ラベルと同じ行で使用することができます。

ObjectScript

```
Mylabel  &sql(
       SELECT Name INTO :a
       FROM Sample.Person
       )
```

&sql 指示文は、括弧で囲まれた有効な埋め込み SQL 文を含む必要があります。SQL 文は、自由にフォーマットして利用できますが、SQL では、空白と新規の行は無視されます。スタジオは、&sql 指示文を認識し、SQL 認識カラー表示機能を使用して SQL コードの文をカラー表示します。

マクロ・プリプロセッサが &sql 指示文を検出すると、SQL クエリ・プロセッサに引用符付きの SQL 文を渡します。クエリ・プロセッサは、クエリの実行に必要なコードを (ObjectScript の INT 形式で) 返します。その後、マクロ・プリプロセッサは &sql 指示文をこのコード (あるいは、コードを含むラベルへの呼び出し) に置換します。スタジオでは、クラスまたはルーチン用に作成された INT コードを調べることで、生成されたコードを表示できます ([**表示**] メニューの [**他のコードを表示**] オプションを使用します)。

&sql 指示文に無効な埋め込み SQL 文が含まれる場合、マクロ・プリプロセッサはコンパイル・エラーを生成します。無効な SQL 文には、構文エラーがある場合や、コンパイル時に存在しないテーブルや列を参照している場合があります。"埋め込み SQL コードの検証" を参照してください。

&sql 指示文には、括弧内の任意の場所に SQL スタイルのコメントを含めることができ、SQL コードが含まれない場合やコメント文のみが含まれる場合もあります。&sql 指示文に、SQL コードが含まれておらず、コメント文のみが含まれている場合、その指示文は空命令として解析され、SQLCODE 変数は定義されません。

ObjectScript

```
NEW SQLCODE
WRITE !,"Entering Embedded SQL"
&sql()
WRITE !,"Leaving Embedded SQL"
```

ObjectScript

```
NEW SQLCODE
WRITE !,"Entering Embedded SQL"
&sql(/* SELECT Name INTO :a FROM Sample.Person */)
WRITE !,"Leaving Embedded SQL"
```

7.2.2 &sql の代替構文

複雑な埋め込み SQL プログラムには、複数の &sql 指示文 (入れ子になった &sql 指示文など) が含まれることがあるため、以下の代替構文形式が用意されています。

- `##sql(...)` : この指示文は、機能面では &sql と等価です。コードをわかりやすくするために代替構文が用意されています。ただし、マーカ構文を含めることはできません。
- `&sql<marker>(...)<reversemarker>` : この指示文を使用すると、複数の &sql 指示文を、それぞれユーザが選択したマーカ文字またはマーカ文字列で特定して指定できます。このマーカ構文については、以下のセクションで説明します。

7.2.3 &sql のマーカ構文

具体的な &sql 指示文を、ユーザ定義のマーカ構文を使用して特定できます。この構文は、“&sql” と開始の括弧文字の間に指定された文字または文字列で構成されます。また、埋め込み SQL の最後には、閉じ括弧の直後にこのマーカと同じ文字を逆の順序で指定する必要があります。構文は、以下のとおりです。

```
&sql<marker>( SQL statement )<reverse-marker>
```

空白 (スペース、タブ、または改行) は、&sql、marker、および開始の括弧の間、あるいは閉じ括弧と reverse-marker の間には使用できません。

marker には、1 文字または一連の文字を指定できます。また、marker には、以下の句読点文字を含めることはできません。

```
( + - / \ | * )
```

marker には、空白 (スペース、タブ、または改行) を含めることはできません。その他すべての印刷可能文字や文字の組み合わせ (Unicode 文字など) を含めることができます。marker および reverse-marker では、大文字と小文字が区別されます。

対応する reverse-marker には、marker と同じ文字を逆の順序で指定する必要があります。例えば、以下のようになります。`&sqlABC( ...)CBA`。marker に [や { を含める場合は、対応する] や } を reverse-marker に含める必要があります。以下は、有効な &sql marker および reverse-marker の組み合わせの例です。

```
&sql@@( ... )@@   &sql[( ... )]   &sqltest( ... )tset   &sql[Aa{( ... )}aA]
```

マーカの文字または文字列を選択する際には、次の重要な SQL の制限に注意してください。SQL コードでは、リテラル文字列やコメントを含めて、コードのどこにも “`)<reversemarker>`” を含めることはできません。例えば、マーカが “ABC“ だとすると、文字列 “)CBA“ は、埋め込み SQL コードのどこにも指定することはできません。指定した場合、有効なマーカと有効な SQL コードの組み合わせが原因となって、コンパイルが失敗します。したがって、marker の文字または文字列は、競合が起こらないように注意して選択することが重要です。

7.2.4 埋め込み SQL と行オフセット

埋め込み SQL が存在することで、ObjectScript の行オフセットは以下のような影響を受けます。

- 埋め込み SQL は、その時点でのルーチンの INT コード行の総数に (少なくとも) 2 を追加します。そのため、1 行の埋め込み SQL は 3 行としてカウントされ、2 行の埋め込み SQL は 4 行としてカウントされます (それ以降も同様にカウントされます)。他のコードを呼び出す埋め込み SQL は、INT コードにさらに多くの行数を追加する可能性があります。

  コメントのみを含むダミーの埋め込み SQL 文 (例 : `&sql( /* for future use */)`) は、2 INT コード行としてカウントされます。

- 埋め込み SQL に含まれるすべての行は、コメント行と空白行を含めて、行オフセットとしてカウントされます。

INT コード行は、^ROUTINE グローバルを使用して表示できます。

7.3 埋め込み SQL のコード

埋め込み SQL で SQL コードを記述する際には、以下の点を考慮してください。

- 単純な (非カーソル) 埋め込み SQL 文
- スキーマ名の解析
- リテラル・データ値
- %List および日付/時刻のデータ値のデータ・フォーマット
- 特権チェック

埋め込み SQL からデータ値をエクスポートするために使用されるホスト変数については、この章の後半で説明します。

7.3.1 単純な SQL 文

単純な SQL 文 (1 つの埋め込み SQL 文) を使用して、以下のようなさまざまな演算を実行できます。

- INSERT、UPDATE、INSERT OR UPDATE、および DELETE 文
- DDL 文
- GRANT と REVOKE 文
- 1 行のみを返す (もしくは、返された先頭の行のみが必要な場合) SELECT 文

単純な SQL 文は、非カーソル・ベース SQL 文とも呼ばれます。カーソル・ベース埋め込み SQL については、この章で後述します。

例えば、以下の文は、ID 番号が 43 の Patient の名前のみを検索します。

ObjectScript

```
&sql(SELECT Name INTO :name
   FROM Patient
   WHERE %ID = 43)
```

複数の行を返すことのできる、クエリの単純文を使用する場合、最初の行のみが返されます。

ObjectScript

```
&sql(SELECT Name INTO :name
   FROM Patient
   WHERE Age = 43)
```

クエリによっては、実際にどの行が最初に返されるかの保証はありません。

埋め込み SQL のコンパイル時に、INTO 節の出力ホスト変数は NULL 文字列に設定されます。このため、単純な埋め込み SQL 文は、出力ホスト変数にアクセスする前に、SQLCODE=100 (クエリからデータが返されない)、または SQLCODE=0 (正常実行) をテストする必要があります。

7.3.2 スキーマ・ネームの解析

テーブル名、ビュー名、またはストアド・プロシージャ名は、修飾されている (スキーマ名を指定) か、未修飾 (スキーマ名の指定なし) かのいずれかです。名前がスキーマ名を指定しない場合、InterSystems IRIS はスキーマ名を以下のように解決します。

- データ定義：未修飾名の解決にはシステム全体の既定のスキーマが使用されます。この既定のスキーマが存在しない場合、スキーマおよび対応するクラス・パッケージが InterSystems IRIS によって作成されます。すべてのデータ定義文でシステム全体の既定のスキーマが使用されます。データ定義文では、#import および #sqlcompile path マクロ・プリプロセッサ指示文は無視されます。
- データ管理：埋め込み SQL 文を含むクラスまたはルーチンに対して有効な #sqlcompile path および #import マクロ・プリプロセッサ指示文のいずれかまたは両方によって指定されたスキーマ検索パスが使用されます。#import および #sqlcompile path 指示文は、異なる機能を持つ使用可能なスキーマ名の互いに独立したリストです。どちらかまたは両方を使用して、未修飾のテーブル、ビュー、またはストアド・プロシージャ名にスキーマ名を指定できます。スキーマ検索パスが指定されていない場合、InterSystems IRIS はシステム全体の既定のスキーマ名を使用します。

スキーマの詳細は、"クラスの定義と使用" の "パッケージ" の章を参照してください。

7.3.3 リテラル値

埋め込み SQL クエリには、リテラル値 (文字列、数字、日付) が含まれる場合があります。文字列は一重引用符 (') で囲む必要があります。(InterSystems SQL では、二重引用符は区切り識別子を指定します。)

ObjectScript

```
&sql(SELECT 'Employee (' || Name || ')' INTO :name
     FROM Sample.Employee)
        IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
        ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
WRITE name
```

数値は (引用符で囲まずに) 直接使用できます。リテラルの数値およびタイムスタンプ値には、InterSystems IRIS がこれらのリテラル値をフィールド値と比較する前に、"簡略な正規化" が行われます。以下の例では、+0050.000 は 50 に正規化されます。

ObjectScript

```
&sql(SELECT Name,Age INTO :name,:age
     FROM Sample.Person
     WHERE Age = +0050.000)
       IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
       ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
WRITE name," age=",age
```

算術、関数、および特殊変数式は、以下のように指定できます。

ObjectScript

```
&sql(DECLARE C1 CURSOR FOR
     SELECT Name,Age-65,$HOROLOG INTO :name,:retire,:today
     FROM Sample.Person
     WHERE Age > 60
     ORDER BY Age,Name)
&sql(OPEN C1)
    QUIT:(SQLCODE'=0)
&sql(FETCH C1)
WHILE (SQLCODE = 0) {
   WRITE $ZDATE(today)," ",name," has ",retire," eligibility years",!
  &sql(FETCH C1) }
&sql(CLOSE C1)
```

また、リテラル値は、入力ホスト変数を使用して入力することもできます。入力ホスト数値にも、“簡略な正規化“ が行われます。詳細は、この章の “ホスト変数” セクションを参照してください。

埋め込み SQL では、## で始まるいくつかの文字シーケンスは文字列リテラル内での使用が許可されず、##lit を使用して指定する必要があります。この文字シーケンスは、##;、##beginlit、##expression(、##function(、##quote(、##stripq(、および ##unique(です。例えば、以下の例は失敗します。

ObjectScript

```
WRITE "Embedded SQL test",!
&sql(SELECT 'the sequence ##unique( is restricted' INTO :x)
WRITE x
```

以下の回避策は成功します。

ObjectScript

```
WRITE "Embedded SQL test",!
&sql(SELECT 'the sequence ##lit(##unique() is restricted' INTO :x)
WRITE x
```

7.3.4 データ形式

埋め込み SQL では、データ値は “論理モード”、つまり SQL クエリ・プロセッサによって使用されるネイティブ形式の値です。LogicalToODBC 変換も LogicalToDisplay 変換も定義されていない文字列、整数やその他のデータ型に対しては何の影響も与えません。データ形式は %List データ、および %Date と %Time データ型に影響します。

%List データ型は、出力不能なリスト・エンコーディング文字で開始する要素値として論理モードで表示されます。WRITE コマンドは、これらの値を連結された要素として表示します。例えば、Sample.Person の FavoriteColors フィールドには、$LISTBUILD('Red','Black') のように %List データ型が含まれます。埋め込み SQL の場合、これは論理モードでは 12 文字の長さの RedBlack として表示されます。表示モードでは、Red Black として表示されます。ODBC モードでは、Red,Black となります。詳細は、以下の例を参照してください。

ObjectScript

```
&sql(DECLARE C1 CURSOR FOR
     SELECT TOP 10 FavoriteColors INTO :colors
     FROM Sample.Person WHERE FavoriteColors IS NOT NULL)
&sql(OPEN C1)
    QUIT:(SQLCODE'=0)
&sql(FETCH C1)
WHILE (SQLCODE = 0) {
   WRITE $LENGTH(colors),": ",colors,!
  &sql(FETCH C1) }
&sql(CLOSE C1)
```

InterSystems IRIS の %Date および %Time データ型は、論理形式として InterSystems IRIS の内部日付表示 ($HOROLOG 形式) を使用します。%Date データ型は、論理モードでは INTEGER 型、表示モードでは VARCHAR データ型、ODBC モードでは DATE データ型を返します。%TimeStamp データ型は、論理、表示、および ODBC 形式に対して、ODBC 日付形式 (YYYY-MM-DD HH:MM:SS) を使用します。

例えば、以下のクラス定義について考えてみます。

Class Definition

```
Class MyApp.Patient Extends %Persistent
{
/// Patient name
Property Name As %String(MAXLEN = 50);

/// Date of birth
Property DOB As %Date;

/// Date and time of last visit
Property LastVisit As %TimeStamp;
}
```

このテーブルに対する単純な SQL クエリは、論理モードで値を返します。例えば、以下のクエリを考えてみます。

ObjectScript

```
 &sql(SELECT Name, DOB, LastVisit
        INTO :name, :dob, :visit
         FROM Patient
        WHERE %ID = :id)
```

このクエリは、ホスト変数の name、dob、および visit の 3 つのプロパティに対する論理値を返します。

ホスト変数	値
name	"Weiss,Blanche"
dob	44051
visit	"2001-03-15 11:11:00"

dob は、$HOROLOG 形式で表されています。$ZDATETIME 関数を使用して、これを表示形式に変換できます。

ObjectScript

```
 SET dob = 44051
 WRITE $ZDT(dob,3),!
```

WHERE 節でも、同様です。例えば、任意の誕生日の Patient (患者) を検索するには、WHERE 節の論理値を使用します。

ObjectScript

```
 &sql(SELECT Name INTO :name
        FROM Patient
        WHERE DOB = 43023)
```

または、ホスト変数を使用します。

ObjectScript

```
 SET dob = $ZDH("01/02/1999",1)

 &sql(SELECT Name INTO :name
        FROM Patient
        WHERE DOB = :dob)
```

この場合、$ZDATEH 関数を使用して、表示形式の日付を論理 $HOROLOG 対応に変換します。

7.3.5 特権チェック

埋め込み SQL は、SQL 特権チェックを実行しません。特権の割り当てに関係なく、すべてのテーブル、ビュー、列にアクセスして、どんな操作でも実行できます。これは、埋め込み SQL を使用するアプリケーションが、埋め込み SQL 文を使用する前に特権を確認していると見なされるためです。

埋め込み SQLで InterSystems SQL の %CHECKPRIV 文を使用して、現在の特権を確認できます。

詳細は、このドキュメントの ″SQL のユーザ、ロール、および特権″ の章を参照してください。

7.4 ホスト変数

ホスト変数は、埋め込み SQL との間でリテラル値を受け渡しするローカル変数です。一般に、ホスト変数は、ローカル変数の値を埋め込み SQL への入力値として渡したり、SQL クエリ結果の値を埋め込み SQL クエリからの出力ホスト変数として渡すために使用されます。

ホスト変数を、スキーマ名、テーブル名、フィールド名、カーソル名などの SQL 識別子を指定するために使用することはできません。ホスト変数を、SQL キーワードを指定するために使用することはできません。

- 出力ホスト変数は、埋め込み SQL でのみ使用されます。出力ホスト変数は INTO 節で指定されます。これは、埋め込み SQL でのみサポートされる SQL クエリの節です。埋め込み SQL をコンパイルすると、すべての INTO 節の変数が NULL 文字列 (’’) に初期化されます。
- 入力ホスト変数は、埋め込み SQL またはダイナミック SQL のどちらでも使用できます。ダイナミック SQL では、″?″ 入力パラメータを使用してリテラルを SQL 文に入力することもできます。この “?” 構文は、埋め込み SQL では使用できません。

埋め込み SQL 内では、入力ホスト変数は、リテラル値を使用できる任意の場所で使用できます。出力ホスト変数は、SELECT 文または FETCH 文の INTO 節を使用して指定されます。

注釈　SQL の NULL が ObjectScript に出力された場合、その SQL の NULL は、ObjectScript の空文字列 (″″)、つまり長さゼロの文字列で表されます。″NULL および未定義ホスト変数″ を参照してください。

変数またはプロパティ参照をホスト変数として使用するには、その前にコロン (:) を付けます。埋め込み InterSystems SQL 内のホスト変数は以下のいずれかにできます。

- ObjectScript の 1 つ以上のローカル変数 (:myvar など)。コンマ区切りのリストとして指定されます。ローカル変数は整形可能で、添え字を含めることができます。すべてのローカル変数と同様、大文字と小文字が区別され、Unicode 文字を含めることができます。
- ObjectScript の 1 つのローカル変数配列 (:myvars() など)。ローカル変数配列は、1 つのテーブル (結合テーブルやビューではない) からのフィールド値のみを受け取ることができます。詳細は、後述の “列番号を添え字とするホスト変数” を参照してください。
- :oref.Prop などのオブジェクト参照。Prop はプロパティ名であり、先頭に % 文字を使用してもしなくてもかまいません。これは、単純なプロパティでも多次元配列プロパティでもかまいません (oref.Prop(1) など)。:i%Prop や :i%%Data などのインスタンス変数として指定することができます。プロパティ名は、`:Person."Home City"` のように区切ることができます。区切られたプロパティ名は、区切り識別子のサポートが非アクティブ化されている場合でも使用できます。多次元プロパティには、:i%Prop() や :m%Prop() などのホスト変数参照を含めることができます。オブジェクト参照ホスト変数には、`:Person.Address.City` のような任意のレベル数のドット構文を含めることができます。

  oref.Prop がプロシージャ・ブロック・メソッドの中でホスト変数として使用される場合、oref.Prop 参照全体ではなく oref 変数が自動的に PublicList に追加され、NEW が実行されます。

ホスト変数内の二重引用符は、区切り識別子ではなくリテラル文字列を指定します (`:request.GetValueAt("PID:SetIDPID")`、`:request.GetValueAt("PID:PatientName(1).FamilyName")` など)。

ホスト変数は、ObjectScript プロシージャの PublicList 変数リストにリストし、NEW コマンドを使用して再初期化する必要があります。埋め込み SQL で使用されるすべてのホスト変数をコメント文内にもリストするように InterSystems IRIS を構成できます。これについては、"InterSystems SQL の使用法" の "コメント" のセクションを参照してください。

ホスト変数の値は、以下のように動作します。

- 入力ホスト変数は、SQL 文コードで変更されることはありません。埋め込み SQL を実行した後でも、その元の値が保持されます。ただし、入力ホスト変数の値には、SQL 文コードに提供される前に "簡略な正規化" が行われます。つまり、有効な数値から、先頭と末尾にあるゼロ、先頭にある単一の + 記号、および末尾にある小数点が取り除かれます。タイムスタンプ値からは、末尾にあるスペース、秒の小数部の末尾にあるゼロ、および (秒の小数部がない場合は) 末尾にある小数点が取り除かれます。
- INTO 節で指定される出力ホスト変数は、クエリのコンパイル時に定義されます。参照しても <UNDEFINED> エラーが発生しないように、NULL 文字列に設定されます。SQLCODE=0 の場合、ホスト変数の値は実際の値のみを表します。DECLARE ... SELECT ...INTO 文では、INTO 節の出力ホスト変数を、2 つの FETCH 呼び出し間で変更しないでください。変更すると、クエリが予期せぬ結果となる可能性があります。

出力ホスト変数を処理する前に、SQLCODE 値をチェックする必要があります。出力ホスト変数値は、SQLCODE=0 の場合にのみ使用してください。

ホスト変数のコンマ区切りリストを INTO 節で使用する場合は、select-items (フィールド、集約関数、スカラ関数、算術式、リテラル) の数と同じ数のホスト変数を指定する必要があります。ホスト変数の数が多すぎたり少なすぎたりすると、コンパイル時に SQLCODE -76 というカーディナリティ・エラーが発生します。

このことが一般に問題となるのは、埋め込み SQL で SELECT * を使用している場合です。例えば、`SELECT * FROM Sample.Person` が有効となるのは、コンマ区切りリストのホスト変数の数が 15 の場合のみです (15 は非表示でない列の正確な数であり、この数には、テーブル定義に応じて、システムによって生成される RowID (ID) 列が含まれる場合と含まれない場合があります)。この列の数は、"インターシステムズ・クラス・リファレンス" に記載されているプロパティの数と単純に一致するわけではない場合もあることに注意してください。

列数は変化する可能性があるため、個別のホスト変数の INTO 節リストと共に `SELECT *` を指定することは通常は推奨されません。`SELECT *` を使用している場合は、通常、以下に示すようなホスト変数の添え字付き配列の使用が推奨されます。

ObjectScript

```
  NEW SQLCODE
&sql(SELECT %ID,* INTO :tflds() FROM Sample.Person )
      IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
      ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
 FOR i=0:1:25 {
     IF $DATA(tflds(i)) {
     WRITE "field ",i," = ",tflds(i),! }
   }
```

この例では、RowID が非表示であるかどうかに関係なく、%ID を使用して RowID をフィールド番号 1 として返します。この例では、フィールド番号の添え字は連続的なシーケンスではないことに注目してください。すなわち、一部のフィールドは非表示になっており、スキップされます。NULL を含むフィールドは、空の文字列値でリストされます。ホスト変数配列の使用法は、後述の "列番号を添え字とするホスト変数" を参照してください。

埋め込み SQL を終了するときに SQLCODE 値を直ちにチェックできるので、プログラミングをするうえで便利です。出力ホスト変数値は、SQLCODE=0 の場合にのみ使用してください。

7.4.1 ホスト変数の例

以下の ObjectScript の例では、埋め込み SQL 文は、出力ホスト変数を使用して、名前と自宅住所を SQL クエリから ObjectScript に返します。

ObjectScript

```
&sql(SELECT Name,Home_State
     INTO :CName,:CAddr
     FROM Sample.Person)
     IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
     ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
   WRITE !,"Name is: ",CName
   WRITE !,"State is: ",CAddr
```

この埋め込み SQL では、ホスト変数 `:CName` および `:CAddr` を指定する INTO 節を使用して、選択された顧客の名前と州名をそれぞれローカル変数 CName と CAddr に返しています。

以下の例では、添え字付きのローカル変数を使用して同じ操作を実行します。

ObjectScript

```
&sql(SELECT Name,Home_State
     INTO :CInfo(1),:CInfo(2)
     FROM Sample.Person)
     IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
     ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
   WRITE !,"Name is: ",CInfo(1)
   WRITE !,"State is: ",CInfo(2)
```

これらのホスト変数は、ユーザが指定した添え字を持つ単純なローカル変数です（`:CInfo(1)`）。ただし、添え字を省略すると（`:CInfo()`）、以下に示すように、InterSystems IRIS は SqlColumnNumber を使用して、ホスト変数の添え字付き配列に値を移入します。

以下の ObjectScript の例では、埋め込み SQL 文は、(WHERE 節の) 入力ホスト変数と (INTO 節の) 出力ホスト変数の両方を使用します。

ObjectScript

```
SET minval = 10000
SET maxval = 50000
&sql(SELECT Name,Salary INTO :outname, :outsalary
     FROM MyApp.Employee
     WHERE Salary > :minval AND Salary < :maxval)
      IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
      ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
   WRITE !,"Name is: ",outname
   WRITE !,"Salary is: ",outsalary
```

以下の例では、入力ホスト変数に“簡略な正規化“を行います。InterSystems IRIS では、入力変数の値は文字列として扱われ、正規化は行われませんが、埋め込み SQL では、この数字が 65 に正規化されて WHERE 節で等値比較が実行されます。

ObjectScript

```
SET x="+065.000"
&sql(SELECT Name,Age
     INTO :a,:b
     FROM Sample.Person
     WHERE Age=:x)
      IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
      ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
WRITE !,"Input value is: ",x
WRITE !,"Name value is: ",a
WRITE !,"Age value is: ",b
```

以下の ObjectScript の例では、埋め込み SQL 文は、ホスト変数としてオブジェクト・プロパティを使用します。

ObjectScript

```
&sql(SELECT Name, Title INTO :obj.Name, :obj.Title
     FROM MyApp.Employee
     WHERE %ID = :id )
```

この場合、obj は、変更可能なプロパティ Name および Title を持つオブジェクトへの、有効な参照である必要があります。クエリに INTO 文が含まれているときにデータが返されない場合 (つまり SQLCODE が 100 の場合)、クエリの実行によりホスト変数の値が変更されている可能性があります。

7.4.2 列番号を添え字とするホスト変数

FROM 節に含まれるテーブルが 1 つの場合は、そのテーブルから選択されるフィールドのために添え字付きホスト変数を指定できます。例えば、論理配列 `:myvar()` などです。このローカル配列の値は、各フィールドの SqlColumnNumber を数値の添え字として使用して生成されます。この SqlColumnNumber は、select-list の順番ではなく、テーブル定義の列番号を表しています。(ビューのフィールドのために添え字付きホスト変数を使用することはできません)。

ホスト変数配列は、最下位レベルの添え字が省略されているローカル配列である必要があります。したがって、`:myvar()`、`:myvar(5,)`、および `:myvar(5,2,)` はすべて、有効なホスト変数の添え字付き配列です。

- ホスト変数の添え字付き配列は、INSERT、UPDATE、または INSERT OR UPDATE 文の VALUES 節で入力のために使用できます。ホスト変数配列が INSERT 文または UPDATE 文で使用された場合は、コンパイル時ではなく実行時に更新される列を定義できます。INSERT および UPDATE の使用法については、"InterSystems SQL リファレンス" でそれらのコマンドを参照してください。
- ホスト変数の添え字付き配列は、SELECT または DECLARE 文の INTO 節で出力のために使用できます。SELECT での添え字付き配列の使用法は、以下の例で説明します。

以下の例では、SELECT は、Cdata 配列に指定のフィールドの値を追加します。Cdata() の要素は、SELECT 要素ではなく、テーブル列定義に対応します。したがって、Name フィールド、Age フィールド、および DOB フィールドはそれぞれ、Sample.Person の列 6、列 2、および列 3 です。

ObjectScript

```
&sql(SELECT Name,Age,DOB
     INTO :Cdata()
     FROM Sample.Person)
       IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
       ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
   WRITE !,"Name is: ",Cdata(6)
   WRITE !,"Age is: ",Cdata(2)
   WRITE !,"DOB is: ",$ZDATE(Cdata(3),1)
```

以下の例では、添え字付き配列のホスト変数を使用して、行のフィールド値をすべて返します。

ObjectScript

```
&sql(SELECT * INTO :Allfields()
     FROM Sample.Person)
       IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
       ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
   SET x=1
   WHILE x '="" {
   WRITE !,x," field is ",Allfields(x)
   SET x=$ORDER(Allfields(x))
   }
```

この WHILE ループは、単純な `x=x+1` ではなく、$ORDER を使用してインクリメントされることに注意してください。これは、Sample.Person などの多くのテーブルで、非表示の列が存在する可能性があるためです。このような場合、列番号が不連続になります。

SELECT リストにテーブルのフィールドではない項目 (式や矢印構文のフィールドなど) が含まれている場合には、INTO 節に、コンマ区切りの配列ではないホスト変数も含める必要があります。以下の例では、定義されたテーブル列に対応

する値を返す添え字付き配列のホスト変数と、定義されたテーブル列に対応しない値を返すホスト変数を組み合わせて使用しています。

ObjectScript

```
&sql(SELECT Name,Home_City,{fn NOW},Age,($HOROLOG-DOB)/365.25,Home_State
     INTO :Allfields(),:timestmp('now'),:exactage
     FROM Sample.Person)
       IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
       ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
   SET x=$ORDER(Allfields(""))
   WHILE x '="" {
   WRITE !,x," field is ",Allfields(x)
   SET x=$ORDER(Allfields(x)) }
   WRITE !,"date & time now is ",timestmp("now")
   WRITE !,"exact age is ",exactage
```

配列でないホスト変数は、列でない SELECT 項目と、数および順番が一致している必要があります。

添え字付き配列として使用するホスト変数には、以下の制限が適用されます。

- 添え字付きリストは、選択するすべてのフィールドが、FROM 節の同じ 1 つのテーブルに存在する場合にのみ使用できます。これは、複数のテーブルからフィールドを選択すると、SqlColumnNumber の値が重複する可能性があるためです。
- 添え字付きリストは、テーブルからフィールドを選択する場合にのみ使用できます。式や集計フィールドを対象として使用することはできません。これは、これらの select-list 項目には SqlColumnNumber の値がないからです。

ホスト変数配列の使用法の詳細は、"InterSystems SQL リファレンス" の "INTO 節" を参照してください。

7.4.3 NULL および未定義ホスト変数

定義されていない入力ホスト変数を指定した場合、埋め込み SQL はその値を NULL として処理します。

ObjectScript

```
NEW x
&sql(SELECT Home_State,:x
     INTO :a,:b
     FROM Sample.Person)
       IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
       ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
  WRITE !,"The length of Home_State is: ",$LENGTH(a)
  WRITE !,"The length of x is: ",$LENGTH(b)
```

SQL の NULL は、ObjectScript の "" 文字列 (長さゼロの文字列) に相当します。

埋め込み SQL をコンパイルすると、すべての INTO 節の出力ホスト変数が ObjectScript "" 文字列 (長さゼロの文字列) として定義されます。ホスト変数に NULL を出力する場合、埋め込み SQL では、その値は ObjectScript の "" 文字列 (長さゼロの文字列) として処理されます。例えば、Sample.Person 内のいくつかのレコードに NULL の Spouse フィールドがあるとします。以下のクエリを実行します。

ObjectScript

```
&sql(SELECT Name,Spouse
   INTO :name, :spouse
   FROM Sample.Person
   WHERE Spouse IS NULL)
       IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
       ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
   WRITE !,"Name: ",name," of length ",$LENGTH(name)," defined: ",$DATA(name)
   WRITE !,"Spouse: ",spouse," of length ",$LENGTH(spouse)," defined: ",$DATA(spouse)
```

ホスト変数 spouse は、NULL 値を表す "" (長さゼロの文字列) に設定されます。このため、ObjectScript $DATA 関数を使用して SQL フィールドが NULL かどうかを判断することはできません。NULL 値の SQL フィールドの出力ホスト変数を渡すと、$DATA は true を返します (変数が定義されます)。

テーブルのフィールドに長さゼロの SQL 文字列 ('') が含まれる場合 (アプリケーションが明示的にフィールドを SQL 文字列 '' に設定する場合など)、ホスト変数に、特別なマーカ値 $CHAR(0) (ASCII の 0 の文字を 1 つのみ含む、長さ 1 の文字列) が含まれます。これが、ObjectScript での長さゼロの SQL 文字列の表現です。長さゼロの SQL 文字列は使用しないことを強くお勧めします。

以下の例では、SQL の NULL と長さゼロの SQL 文字列からのホスト変数の出力を比較します。

ObjectScript

```
 &sql(SELECT '',Spouse
    INTO :zls, :spouse
    FROM Sample.Person
    WHERE Spouse IS NULL)
        IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
        ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
    WRITE "In ObjectScript"
    WRITE !,"ZLS is of length ",$LENGTH(zls)," defined: ",$DATA(zls)
       /* Length=1, Defined=1 */
    WRITE !,"NULL is of length ",$LENGTH(spouse)," defined: ",$DATA(spouse)
       /* Length=0, Defined=1 */
```

このホスト変数の NULL の動作は、サーバ・ベースのクエリ (埋め込み SQL とダイナミック SQL) に固有のものであることに注意してください。ODBC および JDBC では、NULL 値は ODBC または JDBC のインタフェースを使用して明示的に指定されます。

7.4.4 ホスト変数の有効性

- 入力ホスト変数は、埋め込み SQL で変更されることはありません。
- 出力ホスト変数は、SQLCODE = 0 のときに埋め込み SQL の後でのみ有効性が保証されます。

例えば、OutVal の以下の使用は、有効性が保証されません。

ObjectScript

```
InvalidExample
   SET InVal = "1234"
   SET OutVal = "None"
   &sql(SELECT Name
        INTO :OutVal
        FROM Sample.Person
        WHERE %ID=:InVal)
   IF OutVal="None" {           ; Improper Use
   WRITE !,"No data returned"
   WRITE !,"SQLCODE=",SQLCODE }
   ELSE {
   WRITE !,"Name is: ",OutVal }
```

埋め込み SQL の実行前に設定された OutVal の値は、埋め込み SQL から返された後に IF コマンドで参照できません。

代わりに、この例は SQLCODE 変数を使用して、以下のようなコードにします。

ObjectScript

```
ValidExample
   SET InVal = "1234"
   &sql(SELECT Name
        INTO :OutVal
        FROM Sample.Person
        WHERE %ID=:InVal)
   IF SQLCODE'=0 { SET OutVal="None"
      IF OutVal="None" {
      WRITE !,"No data returned"
      WRITE !,"SQLCODE=",SQLCODE } }
   ELSE {
   WRITE !,"Name is: ",OutVal }
```

埋め込み SQL では、出力行が正常に取得されたことを示す場合に SQLCODE 変数に 0 が設定されます。SQLCODE 値の 100 は、SELECT 条件に一致する行が見つからないことを示します。SQLCODE の負の数値は、SQL エラーの状態を示します。

7.4.5 ホスト変数とプロシージャ・ブロック

埋め込み SQL をプロシージャ・ブロックの中に記述する場合は、入力および出力のすべてのホスト変数をパブリックにする必要があります。そのためには、プロシージャ・ブロック先頭の PUBLIC セクションでその変数を宣言するか、その変数の名前の先頭に % 文字を使用します (これで自動的にパブリックになります)。ただし、ユーザ定義の % ホスト変数は自動的にパブリックになりますが、自動的に NEW は実行されないことに注意してください。このような変数に対する NEW は、必要に応じてユーザの責任で実行してください。%ROWCOUNT、%ROWID、%msg など一部の SQL % 変数は、"埋め込み SQL の変数" に説明されているように、自動的にパブリックになり、かつ自動的に NEW が実行されます。SQLCODE をパブリックとして宣言する必要があります。SQLCODE 変数の詳細は、"埋め込み SQL の変数" を参照してください。

以下のプロシージャ・ブロックの例では、ホスト変数 zip、city、および state と、SQLCODE 変数を PUBLIC として宣言しています。SQL システム変数 %ROWCOUNT、%ROWID、および %msg は、名前の先頭に % 文字が使用されているので既にパブリックになっています。その後、このプロシージャ・コードでは、SQLCODE、その他の SQL システム変数、および state ローカル変数に対して NEW コマンドを実行しています。

ObjectScript

```
UpdateTest(zip,city)
  [SQLCODE,zip,city,state] PUBLIC {
  NEW SQLCODE,%ROWCOUNT,%ROWID,%msg,state
  SET state="MA"
  &sql(UPDATE Sample.Person
       SET Home_City = :city, Home_State = :state
       WHERE Home_Zip = :zip)
        IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
        QUIT %ROWCOUNT
  }
```

7.5 SQL カーソル

カーソルはデータを指しているポインタであり、埋め込み SQL プログラムはカーソルを使用して、ポインタが指しているレコードに対する操作を実行できます。カーソルを使用すると、埋め込み SQL は結果セットを反復処理できます。埋め込み SQL はカーソルを使用して、複数のレコードからデータを返すクエリを実行できます。埋め込み SQL はカーソルを使用して、複数のレコードを更新または削除することもできます。

まず DECLARE 文を使用して SQL カーソルを宣言して、そのカーソルに名前を付ける必要があります。この DECLARE 文では、そのカーソルが指すレコードを指定する SELECT 文を記述します。次に、このカーソル名を OPEN cursor 文で指定します。次に、FETCH cursor 文を繰り返し発行して、SELECT 結果セットを反復処理します。次に、CLOSE cursor 文を発行します。

- カーソル・ベースのクエリでは、DECLARE cursorname CURSOR FOR SELECT 文を使用してレコードが選択されて、(必要に応じて) 選択列の値が出力ホスト変数内に返されます。FETCH 文は結果セットを反復処理して、これらの変数を使用して選択された列の値を返します。
- カーソル・ベースの DELETE 文や UPDATE 文では、DECLARE cursorname CURSOR FOR SELECT 文を使用して操作対象のレコードが選択されます。出力ホスト変数は指定されません。FETCH 文は、結果セットを反復処理します。DELETE 文や UPDATE 文に含まれている WHERE CURRENT OF 節によって、選択されたレコードに対して操作を実行するために現在のカーソル位置が特定されます。カーソル・ベースの DELETE および UPDATE の詳細は、"InterSystems SQL リファレンス" の "WHERE CURRENT OF" のページを参照してください。

カーソルは、複数のメソッドにまたがることができません。このため、同一クラス・メソッド内で 1 つのカーソルを宣言し、開き、フェッチして、終了する必要があります。.CSP ファイルから生成されたクラスなど、クラスとメソッドを生成するすべてのコードでこれを考慮することが重要です。

以下の例では、カーソルを使用してクエリを実行し、主デバイスへ結果を表示します。

ObjectScript

```
&sql(DECLARE C1 CURSOR FOR
   SELECT %ID,Name
   INTO :id, :name
   FROM Sample.Person
   WHERE Name %STARTSWITH 'A'
   ORDER BY Name
)

&sql(OPEN C1)
     QUIT:(SQLCODE'=0)
&sql(FETCH C1)

While (SQLCODE = 0) {
    Write id, ":  ", name,!
   &sql(FETCH C1)
}

&sql(CLOSE C1)
```

この例は、以下を実行します。

1. これは、カーソル C1 を宣言し、Name の順に並べられた一連の Person 行を返します。
2. カーソルをオープンします。
3. データの最後に達するまで、カーソルの FETCH を呼び出します。FETCH 呼び出しの後、さらにフェッチするデータがある場合は SQLCODE 変数は 0 に設定されます。FETCH への各呼び出しの後、返り値は DECLARE 文の INTO 節により指定されたホスト変数にコピーされます。
4. カーソルをクローズします。

7.5.1 DECLARE カーソル文

DECLARE 文は、カーソル名と、カーソルを定義する SQL SELECT 文の両方を指定します。また、DECLARE 文は、そのカーソルを使用する文より前のルーチンにある必要があります。

カーソル名は、大文字と小文字を区別します。

カーソル名は、クラスまたはルーチンの中で一意である必要があります。そのため、再帰的に呼び出されるルーチンには、カーソル宣言を含めることができません。このような状況では、ダイナミック SQL を使用するほうが望ましいことがあります。

以下の例では、MyCursor という名前のカーソルを宣言します。

ObjectScript

```
&sql(DECLARE MyCursor CURSOR FOR
   SELECT Name, DOB
   FROM Sample.Person
   WHERE Home_State = :state
   ORDER BY Name
   )
```

DECLARE 文には、オプションの INTO 節が含まれていることがあります。この INTO 節では、カーソルが走査されたときにデータを受け取るローカル・ホスト変数の名前を指定します。例えば、上記の例に INTO 節を追加すると、以下のようになります。

ObjectScript

```
&sql(DECLARE MyCursor CURSOR FOR
   SELECT Name, DOB
   INTO :name, :dob
   FROM Sample.Person
   WHERE Home_State = :state
   ORDER BY Name
   )
```

INTO 節には、ホスト変数のコンマ区切りリスト、単一のホスト変数配列、またはそれら両方を組み合わせて含めることができます。コンマ区切りリストとして指定する場合、INTO 節のホスト変数の数は、カーソルの SELECT リスト内の列数と完全に一致する必要があります。一致していない場合は、文のコンパイル時に“カーディナリティ・ミスマッチ”エラーを受け取ることになります。

DECLARE 文に INTO 節が含まれない場合、INTO 節は FETCH 文の中に表す必要があります。INTO 節を FETCH 文ではなく DECLARE 文で指定すると、パフォーマンスが多少向上する可能性があります。

DECLARE は宣言であり実行文ではないため、SQLCODE 変数を設定したり削除したりすることはできません。

指定されたカーソルが既に宣言されている場合、コンパイルは失敗し、SQLCODE -52 エラー（ ）が表示されます。

DECLARE 文を実行しても、SELECT 文はコンパイルされません。SELECT 文は、OPEN 文の初回実行時にコンパイルされます。埋め込み SQL はルーチンのコンパイル時ではなく、SQL 実行時（ランタイム）にコンパイルされます。

7.5.2 OPEN カーソル文

OPEN 文は、カーソルより後の部分を実行するために、カーソルを作成します。

ObjectScript

```
&sql(OPEN MyCursor)
```

OPEN 文を実行すると、DECLARE 文で見つかった埋め込み SQL コードがコンパイルされ、最適化されたクエリ・プランが作成され、クエリ・キャッシュが生成されます。OPEN を実行すると（SQL 実行時に）、リソースが見つからないエラー（未定義のテーブルやフィールドなど）が発行されます。

OPEN 呼び出しに成功すると、SQLCODE 変数が 0 に設定されます。

最初に OPEN 呼び出しを実行せずに、カーソルからデータを FETCH することはできません。

7.5.3 FETCH カーソル文

FETCH 文は、（カーソル・クエリで定義されているように）カーソルの次の行のデータをフェッチします。

ObjectScript

```
&sql(FETCH MyCursor)
```

FETCH 呼び出しの前にカーソルの DECLARE と OPEN を実行する必要があります。

FETCH 文には、INTO 節が含まれていることがあります。この INTO 節では、カーソルが走査されたときにデータを受け取るローカル・ホスト変数の名前を指定します。例えば、上記の例に INTO 節を追加すると、以下のようになります。

ObjectScript

```
&sql(FETCH MyCursor INTO :a, :b)
```

INTO 節には、ホスト変数のコンマ区切りリスト、単一のホスト変数配列、またはそれら両方を組み合わせて含めることができます。コンマ区切りリストとして指定する場合、INTO 節のホスト変数の数は、カーソルの SELECT リスト内の列数と

完全に一致する必要があります。一致していない場合は、文のコンパイル時に SQLCODE -76 “カーディナリティ・ミスマッチ”・エラーを受け取ることになります。

通常、INTO 節は FETCH 文ではなく DECLARE 文で指定されます。DECLARE 文と FETCH 文両方の SELECT クエリに INTO 節が含まれる場合、DECLARE 文により指定されたホスト変数のみが設定されます。FETCH 文にのみ INTO 節が含まれる場合、FETCH 文により指定されたホスト変数が設定されます。

FETCH がデータを取得する場合、SQLCODE 変数は 0 に設定されます。フェッチするデータがない場合 (または、これ以上データがない場合)、SQLCODE は 100 に設定されます (これ以上データがないことを意味します)。ホスト変数値は、SQLCODE=0 の場合にのみ使用してください。

クエリによっては、最初の FETCH 呼び出しが別のタスク (一時的なデータ構造内の値の並べ替えなど) を実行する場合もあります。

7.5.4 CLOSE カーソル文

CLOSE 文は、カーソルの実行を終了します。

ObjectScript

```
&sql(CLOSE MyCursor)
```

CLOSE 文は、クエリの実行に使用したテンポラリ・ストレージをクリーンアップします。CLOSE 呼び出しに失敗したプログラムでは、リソース・リークが起こる可能性があります (テンポラリ・データベースの IRISTEMP の不必要な増加など)。

CLOSE の呼び出しに成功すると、SQLCODE 変数が 0 に設定されます。そのため、カーソルを閉じる前に、最後の FETCH が SQLCODE を 0 を設定するかまたは 100 に設定するかを調べる必要があります。

7.6 埋め込み SQL の変数

以下の変数には、埋め込み SQL で特殊な使用方法があります。これらのローカル変数名では、大文字と小文字が区別されます。プロセス開始時には、これらの変数は未定義です。これらの変数は埋め込み SQL の操作によって設定されます。これらの変数は、SET コマンドを使用して直接設定することも、NEW コマンドを使用して未定義状態にリセットすることもできます。すべてのローカル変数と同様に、値は、プロセスが継続している間、または NEW を使用して未定義状態に設定されるか別の値に設定されるまで保持されます。例えば、一部の埋め込み SQL 操作については、それらの操作が正常に実行されても %ROWID は設定されません。これらの操作の実行後は、%ROWID は定義されないか、前の値に設定されたままになります。

- %msg
- %ROWCOUNT
- %ROWID
- SQLCODE

これらのローカル変数はダイナミック SQL によって設定されません。(SQL シェルと管理ポータルの SQL インタフェースによってダイナミック SQL が実行されることに注意してください。)代わりに、ダイナミック SQL では対応するオブジェクト・プロパティが設定されます。

次の ObjectScript 特殊変数は、埋め込み SQL で使用されます。これらの特殊変数名では、大文字と小文字が区別されません。プロセス開始時には、これらの変数は何らかの値に初期化されます。これらの変数は埋め込み SQL の操作によって設定されます。これらの変数は、SET コマンドや NEW コマンドを使用して直接設定することはできません。

- $TLEVEL

- $USERNAME

InterSystems IRIS では、埋め込み SQL の処理中に、定義済みの InterSystems IRIS 埋め込み SQL インタフェースの中でこれらのどの変数でも設定できます。

クラス・メソッド (ProcedureBlock=ON) に埋め込み SQL を記述した場合、これらのすべての変数が自動的に PublicList に配置され、SQL 文で使用する SQLCODE、%ROWID、%ROWCOUNT、%msg、およびすべての非 % 変数に NEW 操作が実行されます。メソッドからの参照またはメソッドへの参照により、それらの変数を渡すことができます。参照により渡される変数は、クラス・メソッド・プロシージャ・ブロック内での自動的な NEW 操作が行われません。

ルーチンに埋め込み SQL を記述した場合、プログラミングの段階で、埋め込み SQL を呼び出す前に %msg、%ROWCOUNT、%ROWID、および SQLCODE の各変数を NEW コマンドで新規作成しておく必要があります。NEW を使用してこれらの変数を新規作成すると、これらの変数の以前の設定に対する干渉が防止されます。<FRAMESTACK> エラーを防ぐために、繰り返しサイクル内でこの NEW 操作を実行しないでください。

7.6.1 %msg

システム定義のエラー・メッセージ文字列を含む変数。InterSystems SQL は、SQLCODE を負の整数 (エラーを示す) に設定した場合にのみ、%msg を設定します。SQLCODE が 0 または 100 に設定されている場合は、%msg 変数は以前の値のまま変更されません。

この動作は、ダイナミック SQL の対応する %Message プロパティとは異なります。このプロパティは現在エラーがない場合に空の文字列に設定されます。

場合により、特定の SQLCODE エラー・コードを、SQLCODE を生成したさまざまな条件を記述して、複数の %msg 文字列に関連付けることができます。%msg はユーザ定義のメッセージ文字列を取ることもできます。これは、一般的には、トリガ・コードにより %ok=0 が明示的に設定されたときに、トリガからユーザ定義メッセージを発行して、トリガを中止するために使用します。

SQL コードの実行時に、エラー・メッセージ文字列は、そのプロセスに使用される NLS 言語で生成されます。SQL コードは異なる NLS 言語環境でコンパイルされる可能性があり、メッセージは実行時の NLS 環境に従って生成されます。″$SYS.NLS.Locale.Language″ を参照してください。

7.6.2 %ROWCOUNT

特定の文によって影響される行の数を示す整数カウンタ。

- INSERT、UPDATE、INSERT OR UPDATE、および DELETE は %ROWCOUNT を影響される行の数に設定します。明示的な値が指定された INSERT コマンドによって影響を与えることができるのは 1 行のみであるため、このコマンドによって %ROWCOUNT は 0 または 1 に設定されます。INSERT クエリ結果、UPDATE、または DELETE は複数の行に影響を与えることができるため、%ROWCOUNT は 0 または正の整数に設定される可能性があります。
- TRUNCATE TABLE は、削除された行数や行が削除されたかどうかに関係なく、常に %ROWCOUNT を -1 に設定します。このため、削除された実際の行数を特定するには、TRUNCATE TABLE の前にテーブルに対して COUNT(*) を実行するか、TRUNCATE TABLE ではなく DELETE を使用してテーブル内のすべての行を削除します。
- カーソルが宣言されていない SELECT は単一の行しか処理できないため、単純な SELECT の実行は常に %ROWCOUNT を 1 (取得された選択条件に一致する行が 1 行) または 0 (選択条件に一致する行なし) のどちらかに設定します。
- DECLARE cursorname CURSOR FOR SELECT によって %ROWCOUNT は初期化されません。%ROWCOUNT は SELECT の実行後に変更されず、OPEN cursorname の実行後も変更されないままです。FETCH が初めて正常に実行されると、%ROWCOUNT が設定されます。クエリ選択条件に一致する行がない場合、FETCH は %ROWCOUNT=0 に設定します。FETCH がクエリ選択条件に一致する行を取得した場合、%ROWCOUNT=1 に設定されます。行を取得する後続の各 FETCH は、%ROWCOUNT をインクリメントします。CLOSE 時、または FETCH が SQLCODE

100 (データがない、またはこれ以上データがない) を発行すると、%ROWCOUNT には検出された行の総数が含まれます。

この SELECT の動作は、ダイナミック SQL の対応する %ROWCOUNT プロパティとは異なります。このプロパティはクエリ実行の完了時に 0 に設定され、プログラムがクエリによって返された結果セットの繰り返し処理を行ったときにのみインクリメントされます。

SELECT クエリが集約関数のみを返す場合、すべての FETCH で %ROWCOUNT=1 が設定されます。最初の FETCH は、テーブルにデータがない場合でも、常に %SQLCODE=0 で完了します。後続の FETCH は %SQLCODE=100 で完了し、%ROWCOUNT=1 を設定します。

以下の埋め込み SQL の例は、カーソルを宣言し、FETCH を使用してテーブル内の各行を取り出します。データの最後に達すると (SQLCODE=100)、%ROWCOUNT には、取得された行数が含まれます。

ObjectScript

```
   SET name="LastName,FirstName",state="##"
   &sql(DECLARE EmpCursor CURSOR FOR
        SELECT Name, Home_State
        INTO :name,:state FROM Sample.Person
        WHERE Home_State %STARTSWITH 'M')
   WRITE !,"BEFORE: Name=",name," State=",state
   &sql(OPEN EmpCursor)
      QUIT:(SQLCODE'=0)
   FOR { &sql(FETCH EmpCursor)
        QUIT:SQLCODE
        WRITE !,"Row fetch count: ",%ROWCOUNT
        WRITE " Name=",name," State=",state
 }
   WRITE !,"Final Fetch SQLCODE: ",SQLCODE
   &sql(CLOSE EmpCursor)
   WRITE !,"AFTER: Name=",name," State=",state
   WRITE !,"Total rows fetched: ",%ROWCOUNT
```

以下の埋め込み SQL の例では、UPDATE が実行され、変更によって影響される行の数が設定されます。

ObjectScript

```
 &sql(UPDATE MyApp.Employee
     SET Salary = (Salary * 1.1)
     WHERE Salary < 50000)
        IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
  WRITE "Employees: ", %ROWCOUNT,!
```

所定のプロセス内の埋め込み SQL 文はすべて、%ROWCOUNT 変数を変更することに注意してください。%ROWCOUNT によって指定される値が必要な場合は、必ずその値を取得してからその他の埋め込み SQL 文を実行してください。埋め込み SQL を呼び出す方法によっては、埋め込み SQL の実行に入る前に NEW コマンドで変数 %ROWCOUNT を新規作成しておく必要があります。

%ROWCOUNT の値は、トランザクションの明示的なロール・バックによる影響を受けません。例えば、以下はロール・バックされているにもかかわらず、変更があったことをレポートします。

ObjectScript

```
 TSTART // start an explicit transaction
  NEW SQLCODE,%ROWCOUNT,%ROWID
 &sql(UPDATE MyApp.Employee
     SET Salary = (Salary * 1.1)
     WHERE Salary < 50000)
        IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}

 TROLLBACK // force a rollback; this will NOT modify %ROWCOUNT
 Write "Employees: ", %ROWCOUNT,!
```

暗黙のトランザクション (UPDATE が制約のチェックに失敗した場合など) は、%ROWCOUNT により反映されます。

7.6.3 %ROWID

プロセスを初期化したときに、%ROWID は未定義です。NEW %ROWID コマンドを発行すると、%ROWID は未定義状態にリセットされます。%ROWID は、以下で説明している埋め込み SQL 操作によって設定されます。その操作が正常に実行されなかった場合や、正常に実行されたがどの行も取得または変更されなかった場合は、%ROWID の値は以前の値のまま変更されません。すなわち、未定義のままか、以前の埋め込み SQL 操作によって設定された値のままとなります。このため、それぞれの埋め込み SQL 操作の前に NEW %ROWID を実行することが重要です。

%ROWID は、以下の操作の影響を受ける最後の行の RowID に設定されます。

- INSERT、UPDATE、INSERT OR UPDATE、または DELETE：単一行を対象にした操作の後は、%ROWID 変数には、挿入、更新、または削除されたレコードにシステムによって割り当てられた RowID (オブジェクト ID) の値が含まれています。複数行を対象にした操作の後は、%ROWID 変数には、最後に挿入、更新、または削除されたレコードにシステムによって割り当てられた RowID (オブジェクト ID) の値が含まれています。どのレコードも挿入、更新、または削除されていない場合は、%ROWID 変数の値は変更されません。TRUNCATE TABLE は %ROWID を設定しません。
- カーソル・ベースの SELECT：DECLARE cursorname CURSOR 文と OPEN cursorname 文では %ROWID は初期化されないため、%ROWID の値は以前の値から変更されません。FETCH が初めて正常に実行されると、%ROWID が設定されます。行を取得する後続の各 FETCH は、%ROWID を現在の RowID 値にリセットします。FETCH は、更新可能なカーソルの行を取得すると、%ROWID を設定します。更新可能なカーソルとは、最上位の FROM 節にテーブル名か更新可能なビュー名のいずれかの要素が 1 つだけ含まれるカーソルのことです。カーソルが更新可能でない場合、%ROWID は変更されません。クエリ選択条件と一致する行がない場合、FETCH は以前の %ROWID 値 (ある場合) を変更しません。CLOSE 時、または FETCH が SQLCODE 100 (データがない、またはこれ以上データがない) を発行すると、%ROWID には取得された最後の行の RowID が含まれます。

  カーソル・ベースの SELECT で DISTINCT キーワードまたは GROUP BY 節を使用すると、%ROWID は設定されません。%ROWID 値に以前の値がある場合、%ROWID 値はその値から変更されません。

  集約関数を含むカーソル・ベースの SELECT は、集約関数値のみを返す場合、%ROWID を設定しません。フィールド値と集約関数値の両方を返す場合、すべての FETCH の %ROWID 値は、クエリによって返された最後の行の RowID に設定されます。
- カーソルが宣言されていない SELECT は、%ROWID を設定しません。%ROWID は単純な SELECT 文の完了時に未変更のままとなります。

ダイナミック SQL では、対応する %ROWID プロパティは最後に挿入、更新、または削除されたレコードの RowID 値を返します。ダイナミック SQL では、SELECT クエリの実行時に **%ROWID** プロパティの値は返されません。

以下のメソッド呼び出しを使用することによって、ObjectScript から現在の %ROWID を取得できます。

ObjectScript

```
WRITE $SYSTEM.SQL.GetROWID()
```

INSERT、UPDATE、DELETE、TRUNCATE TABLE、またはカーソル・ベースの SELECT 操作の後に、LAST_IDENTITY という SQL 関数は、直前に変更されたレコードの IDENTITY フィールドの値を返します。テーブルに IDENTITY フィールドがない場合、この関数は、直前に変更されたレコードの RowID を返します。

7.6.4 SQLCODE

埋め込み SQL クエリを実行した後、出力ホスト変数を処理する前に SQLCODE をチェックする必要があります。

SQLCODE=0 の場合、クエリは正常に完了し、データを返しています。出力ホスト変数にはフィールド値が含まれています。

SQLCODE=100 の場合、クエリは正常に完了していますが、出力ホスト変数値が異なることがあります。以下のいずれかです。

- クエリは 1 つ以上のデータ行を返した後 (SQLCODE=0)、データの最後に達しました (SQLCODE=100)。この場合、出力ホスト変数は、返された最後の行のフィールド値に設定されます。%ROWCOUNT>0 です。
- クエリはデータを返しませんでした。この場合、出力ホスト変数は NULL 文字列に設定されます。%ROWCOUNT=0 です。

クエリが集約関数のみを返す場合、テーブルにデータがない場合でも、最初の FETCH は常に SQLCODE=0 および %ROWCOUNT=1 で完了します。2 番目の FETCH は、SQLCODE=100 および %ROWCOUNT=1 で完了します。テーブルにデータがない場合や、クエリ条件と一致するデータがない場合、クエリは出力ホスト変数を適宜 0 または空の文字列に設定します。

SQLCODE が負数の場合、クエリは失敗し、エラー状態になっています。これらのエラー・コードのリストおよび詳細は、"InterSystems IRIS エラー・リファレンス" の "SQLCODE 値とエラー・メッセージ" の章を参照してください。

埋め込み SQL を呼び出す方法によっては、埋め込み SQL の実行に入る前に NEW コマンドで変数 SQLCODE を新規作成しておく必要があります。トリガ・コード内で、SQLCODE をゼロ以外の値に設定すると、自動的に %ok=0 が設定され、トリガ処理が中止されてロール・バックされます。

ダイナミック SQL では、対応する %SQLCODE プロパティは SQL エラー・コードの値を返します。

7.6.5 $TLEVEL

トランザクション・レベルのカウンタ。InterSystems SQL は $TLEVEL を 0 に初期化します。現在のトランザクションがない場合、$TLEVEL は 0 です。

- 最初の START TRANSACTION は、$TLEVEL を 1 に設定します。追加の START TRANSACTION 文は、$TLEVEL に影響を与えません。
- 各 SAVEPOINT 文 は $TLEVEL を 1 インクリメントします。
- ROLLBACK TO SAVEPOINT pointname 文は、$TLEVEL をデクリメントします。デクリメントの量は、セーブポイントによって決定します。
- COMMIT は $TLEVEL を 0 にリセットします。
- ROLLBACK は $TLEVEL を 0 にリセットします。

また、%INTRANSACTION 文を使用すると、トランザクションが処理中かどうかを判定できます。

$TLEVEL は、ObjectScript トランザクション・コマンドによって設定することもできます。詳細は、"ObjectScript リファレンス" の "$TLEVEL" 特殊変数を参照してください。

7.6.6 $USERNAME

SQL ユーザ名は、InterSystems IRIS ユーザ名と同じであり、ObjectScript の $USERNAME 特殊変数に格納されています。ユーザ名は、システム全体の既定のスキーマとして、またはスキーマ検索パス内の要素として使用できます。

7.7 永続クラスのメソッドの埋め込み SQL

以下の例では、クラス・メソッドとインスタンス・メソッド (どちらにも埋め込み SQL が含まれます) を含む永続クラスを示します。

Class Definition

```
Class Sample.MyClass Extends %Persistent [DdlAllowed]
 {
 ClassMethod NameInitial(Myval As %String) As %String [SqlProc]
  {
     &sql(SELECT Name INTO :n FROM Sample.Stuff WHERE Name %STARTSWITH :Myval)
        IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE  RETURN %msg}
        ELSEIF SQLCODE=100 {WRITE "Query returns no results"  RETURN}
   WRITE "Hello "  RETURN n
  }
 Method CountRows() As %Integer
  {
   &sql(SELECT COUNT(*) INTO :count FROM Sample.Stuff)
        IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE  RETURN %msg}
        ELSEIF SQLCODE=100 {WRITE "Query returns no results"  RETURN}
   WRITE "Number of rows is "  RETURN count
  }
 }
```

クラス・メソッドは、以下のように呼び出します。

ObjectScript

```
  WRITE ##class(Sample.MyClass).NameInitial("G")
```

インスタンス・メソッドは、以下のように呼び出します。

ObjectScript

```
  SET x=##class(Sample.MyClass).%New()
  WRITE x.CountRows()
```

これらのメソッドを正常にコンパイルするのに、テーブルやフィールドなどの SQL エンティティが存在する必要はありません。実行時に SQL エンティティの存在チェックが実行されるため、埋め込み SQL メソッドには SQLCODE テスト・ロジックが含まれている必要があります。

コードを実行することなく、埋め込み SQL で指定された SQL エンティティの存在をテストすることができます。これについては、"埋め込み SQL コードの検証" を参照してください。

7.8 埋め込み SQL コードの検証

次の 2 つの方法で、コードを実行せずに埋め込み SQL コードを検証できます。

- `/compileembedded=1` 修飾子を使用して、埋め込み SQL コードを含むルーチンをコンパイルします。
- $SYSTEM.OBJ.GenerateEmbedded() メソッドを使用して、複数の埋め込み SQL ルーチンをコンパイルします。
- 管理ポータルの SQL インタフェースの **[プラン表示]** オプションを使用して、埋め込み SQL コードをテストします。

7.8.1 /compileembedded 修飾子でのコンパイル

$SYSTEM.OBJ クラスのコンパイル・クラス・メソッドを使用し、qspec 引数で `/compileembedded=1` 修飾子を指定して、埋め込み SQL コードを検証できます。`/Compileembedded` のデフォルトは 0 です。

- $SYSTEM.OBJ.Compile() は、指定したクラスおよびそのクラス内のすべてのルーチンをコンパイルします。
- $SYSTEM.OBJ.CompileList() は、指定したクラスのリストおよびそれらのクラス内のすべてのルーチンをコンパイルします。
- $SYSTEM.OBJ.CompilePackage() は、指定したパッケージ (スキーマ) 内のすべてのクラス/ルーチンをコンパイルします。

- $SYSTEM.OBJ.CompileAll() は、現在のネームスペースのすべてのクラス/ルーチンをコンパイルします。
- $SYSTEM.OBJ.CompileAllNamespaces() は、すべてのネームスペースのすべてのクラス/ルーチンをコンパイルします。

ターミナルから SetQualifiers() メソッドを使用して、既定で `/compileembedded=1` 修飾子の使用を指定できます。

Terminal

```
USER>DO $SYSTEM.OBJ.SetQualifiers("/compileembedded=1")    /* sets /compileembedded for current namespace
 */
```

Terminal

```
USER>DO $SYSTEM.OBJ.SetQualifiers("/compileembedded=1",1)  /* sets /compileembedded for all namespaces
 */
```

`/compileembedded` を含む qspec 修飾子のリストを表示するには、以下を呼び出します。

Terminal

```
USER>DO $SYSTEM.OBJ.ShowQualifiers()
```

既定以外の修飾子設定は、ShowQualifiers() 表示の最後に表示されます。

スタジオで、[ツール]→[オプション]→[コンパイラ]→[フラグと最適化]→[フラグ] フィールドに移動して、`/compileembedded=1` 修飾子を設定できます。これは、"スタジオ・オプションの設定" で説明されています。

7.8.2 [プラン表示] を使用したテスト

管理ポータルの SQL インタフェースを使用して、コードを実行することなく、埋め込み SQL コードを検証できます。この処理では、SQL 構文の検証と指定された SQL エンティティの存在チェックの両方が行われます。

管理ポータルの **[システム・エクスプローラ]** オプションで、**[SQL]** オプションを選択して [クエリ実行] コード領域を表示します。

1. 埋め込み SQL クエリを入力します。例：`SELECT Name INTO :n FROM Sample.MyTest` または `DECLARE MyCursor CURSOR FOR SELECT Name,Age INTO :n,:a FROM Sample.MyTest WHERE Age > 21 FOR READ ONLY`。
2. **[プラン表示]** ボタンを押して、コードを確認します。コードが有効な場合、**[プラン表示]** にクエリ・プランが表示されます。コードが無効な場合、**[プラン表示]** に SQLCODE エラー値とメッセージが表示されます。

   INTO 節がない場合、INTO 節は FETCH 文で指定される可能性があるため、**[プラン表示]** の検証でエラーは表示されません。INTO 節にエラーが含まれるか、INTO 節が間違った場所にある場合、**[プラン表示]** で該当するエラーが発行されます。

[実行] ボタンを使用して、埋め込み SQL コードを実行することはできません。

7.9 埋め込み SQL の監査

InterSystems IRIS では、埋め込み SQL 文の監査をオプションでサポートしています。埋め込み SQL の監査は、以下の 2 つの要件が満たされる場合に実行されます。

1. %System/%SQL/EmbeddedStatement システム監査イベントがシステム全体で有効化されている場合。既定では、このシステム監査イベントは有効化されていません。有効にするには、管理ポータルに移動し、**[システム管理]**、**[セキュリティ]**、**[監査]**、**[システムイベントを構成]** の順に選択します。
2. 埋め込み SQL 文を含むルーチンには、#sqlcompile audit マクロ・プリプロセッサ指示文が含まれている必要があります。この指示文が ON に設定されている場合は、コンパイルされたルーチン内でその指示文に続く埋め込み SQL 文が実行時に監査されます。

監査の情報は、監査データベースに記録されます。監査データベースを表示するには、管理ポータルに移動し、**[システム管理]**、**[セキュリティ]**、**[監査]**、**[監査データベースの閲覧]** の順に選択します。EmbeddedStatement に **[イベント名]** フィルタを設定して、**[監査データベースの閲覧]** を埋め込み SQL 文に制限することができます。監査データベースには、埋め込み SQL 文のタイプを指定する時間 (ローカル・タイムスタンプ)、ユーザ、PID (プロセス Id)、および説明がリストされます (**SQL SELECT Statement** など)。

イベントの **[詳細]** リンクを選択することで、**[イベントデータ]** などの追加情報をリストできます。イベント・データには、実行された SQL 文と、文の入力引数の値が含まれます。以下に例を示します。

```
SELECT TOP :n Name,ColorPreference INTO :name,:color FROM Sample.Stuff WHERE Name %STARTSWITH :letter
Parameter values:
n=5
letter="F"
```

InterSystems IRIS では、ダイナミック SQL 文 (**Event Name**=DynamicStatement) および ODBC および JDBC 文 (**Event Name**=XDBCStatement) の監査もサポートしています。

8

ダイナミック SQL の使用

この章では、InterSystems IRIS® データ・プラットフォームから実行時に作成され、実行されるクエリおよびその他の SQL 文であるダイナミック SQL について説明します。

この章では、ダイナミック SQL の推奨実装である、**%SQL.Statement** クラスを使用したダイナミック SQL プログラミングについて説明します。この章および弊社のドキュメント全体にわたって、ダイナミック SQL に関するすべての文は、**%SQL.Statement** の実装を指しています。

8.1 ダイナミック SQL の概要

ダイナミック SQL は、実行時に作成され、実行される SQL 文です。ダイナミック SQL では、SQL コマンドの作成と実行は別個の操作です。ダイナミック SQL により、InterSystems IRIS でも ODBC アプリケーションや JDBC アプリケーションと似た方法でプログラムを作成できます (データベース・エンジンと同じプロセス・コンテキストで SQL 文を実行している場合は除きます)。ダイナミック SQL は ObjectScript プログラムから呼び出されます。

ダイナミック SQL クエリは、コンパイル時ではなく、プログラム実行時に作成されます。つまり、コンパイラはコンパイル時にエラーのチェックを実行できず、プリプロセッサ・マクロをダイナミック SQL 内で使用することはできません。また、実行プログラムはユーザやその他の入力に応答して、専用のダイナミック SQL クエリを生成できます。

ダイナミック SQL を使用して SQL クエリを実行できます。また、他の SQL 文の発行にも使用できます。この章の例では、SELECT クエリを実行します。CREATE TABLE、INSERT、UPDATE、DELETE、または CALL を呼び出すダイナミック SQL プログラムの例については、"InterSystems SQL リファレンス" でこれらのコマンドを参照してください。

ダイナミック SQL は、InterSystems IRIS SQL シェルの実行、InterSystems IRIS 管理ポータルの [クエリ実行] インタフェース、SQL コードのインポート・メソッド、およびデータのインポート/エクスポート・ユーティリティで使用されます。

ダイナミック SQL (およびそれを使用するアプリケーション) の最大行サイズは、3,641,144 文字です。

8.1.1 ダイナミック SQL と埋め込み SQL

ダイナミック SQL と埋め込み SQL の相違点は以下のとおりです。

- ダイナミック SQL クエリの初回実行ではクエリ用のインライン・コードを生成しないため、埋め込み SQL よりもやや非効率的です。ただし、どちらもクエリ・キャッシュをサポートしているため、ダイナミック SQL も埋め込み SQL も再実行はその初回のクエリ実行よりはるかに高速です。
- ダイナミック SQL は、クエリに入力されるリテラル値を 2 つの形で受け付けることができます。1 つは、“?” 文字を使用して指定された入力パラメータであり、もう 1 つは入力ホスト変数です (:var など)。埋め込み SQL では、入力/出力ホスト変数が使用されます (:var など)。

- ダイナミック SQL の出力値は、結果セット・オブジェクト (つまり Data プロパティ) の API を使用して取得します。埋め込み SQL は、SELECT 文の INTO 節でホスト変数 (:var など) を使用して、値を出力します。
- ダイナミック SQL では、%SQLCODE、%Message、%ROWCOUNT、および %ROWID というオブジェクト・プロパティが設定されます。埋め込み SQL では、対応するローカル変数である SQLCODE、%msg、%ROWCOUNT、および %ROWID が設定されます。ダイナミック SQL では SELECT クエリに対して %ROWID が設定されませんが、埋め込み SQL では、カーソル・ベースの SELECT クエリに対して %ROWID が設定されます。
- ダイナミック SQL により、クエリのメタデータ (列の数や名前など) を簡単に見ることができます。
- ダイナミック SQL では SQL 特権の確認が既定で行われるので、テーブルやフィールドなどにアクセスするときやこれらを変更するときは、適切な特権が必要です。埋め込み SQL では、SQL 特権の確認は行われません。詳細は、SQL "%CHECKPRIV" 文を参照してください。
- ダイナミック SQL では、プライベート・クラス・メソッドにアクセスできません。既存のクラス・メソッドにアクセスするには、そのメソッドをパブリックにする必要があります。これは SQL の一般的な制限です。ただし、埋め込み SQL の場合は、埋め込み SQL の操作自体が同じクラスのメソッドなので、この制限を回避できます。

ダイナミック SQL と埋め込み SQL は同じデータ表示 (既定では論理モードですが、変更できます)、および NULL 処理を使用します。

8.2 %SQL.Statement クラス

ダイナミック SQL の推奨インタフェースは、%SQL.Statement クラスです。ダイナミック SQL 文を作成および実行するには、%SQL.Statement のインスタンスを使用します。ダイナミック SQL 文の実行結果は、%SQL.StatementResult クラスのインスタンスである SQL 文の結果オブジェクトになります。SQL 文の結果オブジェクトは、単一値、結果セット、またはコンテキスト・オブジェクトのいずれかです。いずれの場合も、結果オブジェクトは標準インタフェースをサポートします。それぞれの結果オブジェクトは %SQLCODE や %Message などの結果オブジェクト・プロパティを初期化します。これらのプロパティの設定値は、発行された SQL 文によって異なります。SELECT 文が正常に実行された場合、オブジェクトは、結果セット (具体的には %SQL.StatementResult のインスタンス) であり、予想される結果セットの機能をサポートします。

次の ObjectScript コードでは、ダイナミック SQL クエリを作成および実行します。

ObjectScript

```
/* Simple %SQL.Statement example */
set myquery = "SELECT TOP 5 Name,DOB FROM Sample.Person"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(myquery)
 if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
set rset = tStatement.%Execute()
do rset.%Display()
write !,"End of data"
```

この章の例では、%SQL.Statement クラスおよび %SQL.StatementResult クラスに関連付けられているメソッドを使用します。

8.3 オブジェクト・インスタンスの作成

%New() クラス・メソッドを使用して、%SQL.Statement クラスのインスタンスを作成できます。

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New()
```

この時点で結果セット・オブジェクトは SQL 文を作成できる状態です。**%SQL.Statement** クラスのインスタンスを作成すると、そのインスタンスを使用して、複数のダイナミック SQL クエリの発行や、INSERT、UPDATE、または DELETE の各操作の実行が可能です。

%New() では、コンマで区切られた 3 つのオプション・パラメータを次の順序で指定できます。

1. %SelectMode：データ入力とデータ表示に使用するモードを指定します。
2. %SchemaPath：未修飾のテーブル名のスキーマ名を指定するために使用する検索パスを指定します。
3. %Dialect：Transact-SQL (TSQL) Sybase 言語または MSSQL 言語を指定します。既定は IRIS (InterSystems SQL) です。

%ObjectSelectMode プロパティもありますが、これは %New() パラメータとして設定できません。**%ObjectSelectMode** は、フィールドのデータ型のバインディングを関連するオブジェクト・プロパティに指定します。

以下の ObjectScript の例では、%SelectMode に 2 (表示モード) を指定し、%SchemaPath に既定のスキーマとして "Sample" を指定しています。

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New(2,"Sample")
```

以下の ObjectScript の例では、%SelectMode には値を指定せず (該当の位置にはコンマのみが記述されています)、%SchemaPath には、3 つのスキーマ名を記述したスキーマ検索パスを指定しています。

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New(,"MyTests,Sample,Cinema")
```

8.3.1 %SelectMode プロパティ

%SelectMode プロパティは、0 = 論理 (既定値)、1 = ODBC、2 = 表示のいずれかのモードを指定します。これらのモードは、データ値を入力および表示する方法を指定します。このモードは、通常、日時の値や、%List データ (エンコードされたリストを含む文字列) の表示に最もよく使用されます。データは論理モードで格納されます。

SELECT クエリは、**%SelectMode** 値を使用してデータ表示に使用する形式を決定します。

INSERT または UPDATE 操作は、**%SelectMode** 値を使用してデータ入力に使用できる形式を決定します。

%SelectMode は、データ表示のために使用されます。SQL 文は、内部的に論理モードで実行されます。例えば、ORDER BY 節は、**%SelectMode** の設定に関係なく、レコードをその論理値に基づいて並べ替えます。SQL 関数は、**%SelectMode** の設定に関係なく、論理値を使用します。SQLPROC として投影されたメソッドも論理モードで実行されます。SQL 文で関数として呼び出された SQL ルーチンは、論理形式で関数値を返す必要があります。

- SELECT クエリでは、**%SelectMode** により、データの表示に使用する形式を指定します。**%SelectMode** を ODBC または表示に設定すると、比較述語の値の指定に使用されるデータ形式にも影響します。述語の値には、**%SelectMode** 形式で指定する必要があるものや、**%SelectMode** に関係なく、論理形式で指定する必要があるものがあります。詳細は、"InterSystems SQL リファレンス" の "述語の概要" を参照してください。
  - **%SelectMode=1** の Time データ型データ (ODBC) では秒の小数部を表示できます。これは実際の ODBC 時間とは異なります。InterSystems IRIS Time データ型は秒の小数部をサポートします。対応する ODBC TIME データ型 (TIME_STRUCT 標準ヘッダ定義) では秒の小数部はサポートされません。ODBC TIME データ型は、指定された時間値を整数秒に切り捨てます。ADO DotNet および JDBC では、この制限はありません。

  - **%SelectMode=0** (論理) の %List データ型では、内部ストレージ値は表示されません。これは、%List データが出力不能文字を使用してエンコードされるためです。代わりに、ダイナミック SQL は %List データ値を $LISTBUILD 文として表示します。例えば、`$lb("White","Green")` のようになります。使用例は、"%Print() メソッド" を参照してください。**%SelectMode=1** の %List データ型 (ODBC) は、リスト要素をコンマで区切って表示します。この要素区切り文字は、CollectionOdbcDelimiter パラメータとして指定されます。**%SelectMode=2** の %List データ型 (表示) は、リスト要素を `$CHAR(10,13)` (改行、キャリッジ・リターン) で区切って表示します。この要素区切り文字は、CollectionDisplayDelimiter パラメータとして指定されます。

- INSERT または UPDATE 操作の場合、**%SelectMode** は論理ストレージ形式に変換される入力データの形式を指定します。このデータ変換を行うには、INSERT または UPDATE の実行時に表示または ODBC **%SelectMode** が使用されるように、SQL コードが RUNTIME 選択モード (既定) を使用してコンパイルされている必要があります。日時に使用できる入力値については、DATE データ型および TIME データ型を参照してください。詳細は、"InterSystems SQL リファレンス" の "INSERT" または "UPDATE" 文を参照してください。

%SelectMode は、以下の 2 つの例に示すように、%New() クラス・メソッドの最初のパラメータとして指定するか、または直接設定できます。

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New(2)
```

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New()
set tStatement.%SelectMode=2
```

以下の例は %SelectMode の現在の値を返します。

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New()
write !,"default select mode=",tStatement.%SelectMode
set tStatement.%SelectMode=2
write !,"set select mode=",tStatement.%SelectMode
```

現在のプロセスで既定の SelectMode 設定を確認できますが、そのためには、$SYSTEM.SQL.Util.GetOption("SelectMode") メソッドを使用します。現在のプロセスで既定の SelectMode 設定を変更できますが、そのためには、$SYSTEM.SQL.Util.SetOption("SelectMode",n) メソッドを使用します。n には、0 = 論理、1 = ODBC、または 2 = 表示のいずれかを指定します。%SelectMode による設定は、現在のオブジェクト・インスタンスの既定の設定よりも優先して適用されます。SelectMode プロセスの既定の設定は変更されません。

SelectMode オプションの詳細は、このドキュメントの “InterSystems IRIS SQL の基礎” の章にある “データ表示オプション” を参照してください。

8.3.2 %SchemaPath プロパティ

%SchemaPath プロパティは、未修飾のテーブル名、ビュー名、またはストアド・プロシージャ名のスキーマ名の指定に使用する検索パスを指定します。スキーマ検索パスは、SELECT、CALL、INSERT、TRUNCATE TABLE などのデータ管理操作に使用され、DROP TABLE などのデータ定義操作では無視されます。

検索パスは、単独のスキーマ名を引用符で囲んで指定するか、複数のスキーマ名をコンマで区切って記述したリストを引用符で囲んで指定します。このリストに記述したスキーマは左から右に検索されます。一致するテーブル名、ビュー名、またはストアド・プロシージャ名が検出されるまで、指定した各スキーマが検索されます。スキーマは指定された順序で検索されるため、あいまいなテーブル名は検出されません。検索対象となるのは、現在のネームスペースに存在するスキーマ名のみです。

スキーマ検索パスには、リテラルのスキーマ名と、CURRENT_PATH、CURRENT_SCHEMA、および DEFAULT_SCHEMA の各キーワードの両方を記述できます。

- CURRENT_PATH は、前もって %SchemaPath プロパティで定義した現在のスキーマ検索パスを示します。これは、通常、既存のスキーマ検索パスの先頭または末尾にスキーマを付加するときに使用します。
- CURRENT_SCHEMA は、%SQL.Statement 呼び出しがクラス・メソッド内から行われた場合の、現在のスキーマのコンテナのクラス名を指定します。クラスのメソッドで #sqlcompile path マクロ指示文を定義している場合、CURRENT_SCHEMA は現在のクラス・パッケージにマップされたスキーマになります。そうでない場合には、CURRENT_SCHEMA は DEFAULT_SCHEMA と同じです。
- DEFAULT_SCHEMA は、システム全体の既定のスキーマを指定します。このキーワードを使用すると、リストされている他のスキーマを検索する前に、スキーマ検索パス内の項目としてシステム全体の既定のスキーマを検索できます。パスに指定されているすべてのスキーマを検索して一致が見つからなかった場合、システム全体の既定のスキーマは常に、スキーマ検索パスの検索後に検索されます。

%SchemaPath は、InterSystems IRIS が一致テーブル名を最初に探す場所です。%SchemaPath が指定されていない場合、または一致するテーブル名が含まれるスキーマを表示しない場合、InterSystems IRIS はシステム全体の既定のスキーマを使用します。

スキーマ検索パスを指定できますが、そのためには、%SchemaPath プロパティを指定するか、%New() クラス・メソッドの 2 つ目のパラメータを指定します。以下に 2 つの例を示します。

ObjectScript

```
set path="MyTests,Sample,Cinema"
set tStatement = ##class(%SQL.Statement).%New(,path)
```

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New()
set tStatement.%SchemaPath="MyTests,Sample,Cinema"
```

%SchemaPath は、このキーワードを使用する %Prepare() メソッドより前であればどの場所で設定してもかまいません。

以下の例は %SchemaPath の現在の値を返します。

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New()
write !,"default path=",tStatement.%SchemaPath
set tStatement.%SchemaPath="MyTests,Sample,Cinema"
 write !,"set path=",tStatement.%SchemaPath
```

%ClassPath() メソッドを使用して、%SchemaPath に、特定のクラス名に対して定義された検索パスを設定することができます。

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New()
set tStatement.%SchemaPath=tStatement.%ClassPath("Sample.Person")
write tStatement.%SchemaPath
```

8.3.3 %Dialect プロパティ

%Dialect プロパティは、SQL 文の言語を指定します。Sybase、MSSQL、または IRIS (InterSystems SQL) を指定できます。Sybase または MSSQL 設定によって、指定した Transact-SQL 言語を使用して SQL 文が処理されます。

Sybase 言語および MSSQL 言語が言語内でサポートする SQL 文は限定されています。サポートされる文は、SELECT、INSERT、UPDATE、DELETE、および EXECUTE です。また、CREATE TABLE 文は、永続的なテーブルに対してはサポートされますが、一時テーブルに対してはサポートされません。CREATE VIEW はサポートされます。CREATE TRIGGER

と DROP TRIGGER もサポートされます。ただし、この実装では、CREATE TRIGGER 文が部分的に成功してもクラス・コンパイルに対しては失敗する場合には、トランザクション・ロールバックはサポートされません。CREATE PROCEDURE と CREATE FUNCTION はサポートされます。

Sybase 言語および MSSQL 言語は、IF 制御フロー文をサポートします。このコマンドは IRIS (InterSystems SQL) 言語ではサポートされていません。

既定は、空の文字列 ("") で表される、または "IRIS" として指定される InterSystems SQL です。

%Dialect は、以下の 3 つの例に示すように、%New() クラス・メソッドの 3 番目のパラメータとして指定するか、プロパティとして直接設定するか、またはメソッドを使用して設定することができます。

%New() クラス・メソッドで %Dialect を設定する：

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New(,,"Sybase")
write "language mode set to=",tStatement.%Dialect
```

%Dialect プロパティを直接設定する：

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New()
set defaultdialect=tStatement.%Dialect
write "default language mode=",defaultdialect,!
set tStatement.%Dialect="Sybase"
write "language mode set to=",tStatement.%Dialect,!
set tStatement.%Dialect="IRIS"
 write "language mode reset to default=",tStatement.%Dialect,!
```

エラー・ステータスを返す %DialectSet() インスタンス・メソッドを使用して %Dialect プロパティを設定する：

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New()
set tStatus = tStatement.%DialectSet("Sybase")
  if tStatus'=1 {write "%DialectSet failed:" do $System.Status.DisplayError(tStatus) quit}
write "language mode set to=",tStatement.%Dialect
```

%DialectSet() メソッドは %Status 値を返します。成功の場合、1 のステータスを返します。失敗の場合、0 の後にエンコードされたエラー情報が続くオブジェクト式を返します。このため、失敗には tStatus=0 テストは実行できません。失敗には $$$ISOK(tStatus)=0 macro テストを実行できます。

8.3.4 %ObjectSelectMode プロパティ

%ObjectSelectMode プロパティはブーリアン値です。%ObjectSelectMode=0 (既定値) の場合、SELECT リストのすべての列が、結果セットでリテラル・タイプのプロパティに結合されます。%ObjectSelectMode= 1 の場合、SELECT リストの列は、関連付けたプロパティ定義で定義したタイプのプロパティに結合されます。

%ObjectSelectMode では、タイプ・クラスがスウィズル可能クラスである列を、SELECT 文から生成された結果セット・クラスでどのように定義するかを指定できます。%ObjectSelectMode=0 の場合、スウィズル可能列に対応するプロパティは、結果セットにおいて、SQL テーブルの RowID タイプに対応する単純なリテラル・タイプとして定義されます。%ObjectSelectMode=1 の場合、そのプロパティは、その列の宣言されているタイプで定義されます。つまり、結果セットのプロパティにアクセスするとスウィズリングがトリガされます。

%ObjectSelectMode は、%New() のパラメータとしては設定できません。

以下の例は、%ObjectSelectMode の既定値を返し、%ObjectSelectMode を設定したうえで、その新しい %ObjectSelectMode 値を返します。

ObjectScript

```
set myquery = "SELECT TOP 5 %ID AS MyID,Name,Age FROM Sample.Person"
set tStatement = ##class(%SQL.Statement).%New()
write !,"default ObjectSelectMode=",tStatement.%ObjectSelectMode
set tStatement.%ObjectSelectMode=1
write !,"set ObjectSelectMode=",tStatement.%ObjectSelectMode
```

%ObjectSelectMode=1 は主に、フィールド名プロパティを使用して結果セットから値を返す際に使用します。これについては、この章の “結果セットからの特定値の返送” セクションの ″フィールド名プロパティ″ に、例を用いた詳しい説明があります。

%ObjectSelectMode=1 は、SELECT リスト内のフィールドがコレクション・プロパティにリンクされている場合に使用できます。%ObjectSelectMode はコレクションをスウィズルします。%SelectMode=1 または 2 の場合、スウィズリングの前にコレクションのシリアル値が論理モード形式に変換されます。結果として得られる oref は、完全なコレクション・インタフェースをサポートします。

8.4 SQL 文の作成

SQL 文の作成では、その文を検証し、以降の実行に向けて準備して、その SQL 文のメタデータを生成します。

%SQL.Statement クラスを使用して SQL 文を作成する方法は 3 つあります。

- %Prepare() : 後続の %Execute() に対して SQL 文 (クエリなど) を作成します。
- %PrepareClassQuery() : 既存のクエリへの呼び出し文を作成します。呼び出し文を作成したら、後続の %Execute() を使用してこのクエリを実行できます。
- %ExecDirect() : SQL 文を作成および実行します。%ExecDirect() については、“SQL 文の実行” に説明があります。
- %ExecDirectNoPriv() : SQL 文を作成および実行しますが、特権は確認しません。%ExecDirectNoPriv() は、″“SQL 文の実行”″ を参照してください。

$SYSTEM.SQL.Prepare() メソッドを使用することで、オブジェクト・インスタンスを作成することなく SQL 文を作成することもできます。以下のターミナル例では、Prepare() メソッドが使用されています。

Terminal

```
USER>set topnum=5
USER>set prep=$SYSTEM.SQL.Prepare("SELECT TOP :topnum Name,Age FROM Sample.Person WHERE Age=?")

USER>do prep.%Display()
```

SQL 文の作成によって、クエリ・キャッシュが作成されます。クエリ・キャッシュを使用すると、SQL 文を再作成する必要なしに同じ SQL クエリを複数回実行できます。クエリ・キャッシュは、任意のプロセスによって 1 回以上実行することができます。異なる入力パラメータ値を使用して実行することができます。

SQL 文を準備するたびに、InterSystems IRIS はクエリ・キャッシュを検索して、同じ SQL 文が既に準備されキャッシュされているかどうかを判別します (2 つの SQL 文で、リテラルおよび入力パラメータの値だけが異なる場合、これらの文は ″同一″ と見なされます)。準備済みの文がクエリ・キャッシュ内に存在しない場合、InterSystems IRIS はクエリ・キャッシュを作成します。準備済みの文が既にクエリ・キャッシュ内に存在する場合は、新しいクエリ・キャッシュは作成されません。このため、ループ構造内で準備文をコード化しないことが重要です。

8.4.1 %Prepare()

SQL 文の作成には、**%SQL.Statement** クラスの %Prepare() インスタンス・メソッドを使用できます。%Prepare() メソッドは、最初の引数として SQL 文を取ります。次の例に示すように、この引数には、引用符付きの文字列、または引用符付きの文字列に解決される変数を指定できます。

ObjectScript

```
set qStatus = tStatement.%Prepare("SELECT Name,Age FROM Sample.Person")
```

次の例に示すように、参照渡しの添え字付き配列を使用することによって、より複雑なクエリを指定できます。

ObjectScript

```
set myquery = 3
set myquery(1) = "SELECT %ID AS id, Name, DOB, Home_State"
set myquery(2) = "FROM Person WHERE Age > 80"
set myquery(3) = "ORDER BY 2"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(.myquery)
```

クエリには、重複するフィールド名とフィールド名エイリアスを含めることができます。

%Prepare() に渡されるクエリには、次の例で示すように入力ホスト変数を含めることができます。

ObjectScript

```
set minage = 80
set myquery = 3
set myquery(1) = "SELECT %ID AS id, Name, DOB, Home_State"
set myquery(2) = "FROM Person WHERE Age > :minage"
set myquery(3) = "ORDER BY 2"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(.myquery)
```

InterSystems IRIS は SQL 文の実行時に、それぞれの入力ホスト変数を定義済みのリテラル値に置換します。ただし、このコードがメソッドとして呼び出された場合、minage 変数を Public にする必要があります。既定では、メソッドは ProcedureBlocks です。これは、メソッド (%Prepare() など) がその呼び出し元によって定義された変数を認識できないことを意味します。この既定をオーバーライドするには、クラスを [Not ProcedureBlock] として指定して、メソッドを [ProcedureBlock = 0] として指定するか、または [PublicList = minage] を指定します。

注釈　プログラム記述時には、入力変数を SQL コードに挿入する前に、その変数に適切な値が含まれていること必ず確認することをお勧めします。

? 入力パラメータを使用して、クエリにリテラル値を渡すこともできます。InterSystems IRIS は、%Execute() メソッドに渡された対応するパラメータ値を使用して、それぞれの ? 入力パラメータをリテラル値に置換します。%Prepare() の後に、%GetImplementationDetails() メソッドを使用して、クエリ内の ? 入力パラメータと入力ホスト変数をリスト表示できます。

%Prepare() メソッドは %Status 値を返します。成功の場合、1 のステータスを返します (クエリ文字列が有効で、参照したテーブルが現在のネームスペース内に存在しています)。失敗の場合、0 の後にエンコードされたエラー情報が続くオブジェクト式を返します。このため、失敗には `status=0` テストは実行できません。失敗には `$$$ISOK(status)=0` macro テストを実行できます。

%Prepare() メソッドは、先に定義されている **%SchemaPath** プロパティを使用して、修飾されていない名前を解決します。

注釈　ダイナミック SQL のパフォーマンスは、できる限り完全修飾名を使用することによって大幅に向上させることができます。

SQL 文で入力パラメータを指定するには、“？ ” 文字を使用します。

ObjectScript

```
set myquery="SELECT TOP ? Name,Age FROM Sample.Person WHERE Age > ?"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(myquery)
```

クエリの実行時に、%Execute() インスタンス・メソッドの各 "?" 入力パラメータに値を指定します。入力パラメータは、リテラル値を受け取るか、評価結果がリテラル値になる式を受け取る必要があります。入力パラメータは、フィールド名値やフィールド名エイリアスを受け取ることはできません。入力パラメータは、PUBLIC として宣言し、SELECT 文がそれを直接参照できるようにする必要があります。

クエリには、フィールドのエイリアスを含めることができます。この場合、Data プロパティは、フィールド名でなく、エイリアスを使用してデータにアクセスします。

ダイナミック SQL では、SELECT 文に限らず、%Prepare() インスタンス・メソッドを使用して、CALL、INSERT、UPDATE、DELETE などの他の SQL 文も作成できます。

現在作成している文の情報を表示するには、次の例に示すように、%Display() インスタンス・メソッドを使用します。

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New(,"Sample")
set myquery = 3
set myquery(1) = "SELECT TOP ? Name,DOB,Home_State"
set myquery(2) = "FROM Person"
set myquery(3) = "WHERE Age > 60 AND Age < 65"
set qStatus = tStatement.%Prepare(.myquery)
 if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
do tStatement.%Display()
write !,"End of %Prepare display"
```

この情報は、実装クラス、引数 (リテラル値または ? 入力パラメータによる実際の引数のコンマ区切りのリスト)、および文のテキストで構成されます。

%Prepare() はオプションの 2 番目の引数として `checkPriv` を取ります。この引数は、InterSystems IRIS で文に対する特権が確認されるかどうかを指定する論理値です。`checkPriv` が 0 の場合、特権は確認されません。特権の確認を無効化することで、ダイナミック・クエリの実行をアプリケーションで詳しく制御できるようになりますが、セキュリティ・リスクは高くなります。既定値は 1 で、この場合は特権が確認されます。以下に例を示します。

ObjectScript

```
set statement = ##class(%SQL.Statement).%New()
set status =statement.%Prepare("DELETE FROM T",0) // No privileges checked

set statement2 = ##class(%SQL.Statement).%New()
set status =statement2.%Prepare("DELETE FROM T") // Privilege is checked
```

8.4.2 %PrepareClassQuery()

既存の SQL クエリを使用する場合は、%SQL.Statement クラスの %PrepareClassQuery() インスタンス・メソッドを使用できます。%PrepareClassQuery() メソッドは、既存のクエリのクラス名およびクエリ名の 2 つのパラメータを取ります。次の例に示すように、両方の引数に、引用符付きの文字列、または引用符付きの文字列に解決される変数を指定できます。

ObjectScript

```
set qStatus = tStatement.%PrepareClassQuery("User.queryDocTest","DocTest")
```

%PrepareClassQuery() メソッドは %Status 値を返します。成功の場合、1 のステータスを返します。失敗の場合、0 の後にエンコードされたエラー情報が続くオブジェクト式を返します。このため、失敗には `qStatus=0` テストは実行できません。失敗には `$$$ISOK(qStatus)=0` macro テストを実行できます。

%PrepareClassQuery() メソッドは、先に定義されている %SchemaPath プロパティを使用して、修飾されていない名前を解決します。

%PrepareClassQuery() は、実行に CALL 文を使用します。このため、実行されたクラス・クエリには、SqlProc パラメータが存在する必要があります。

以下の例では、%PrepareClassQuery() は Sample.Person クラスで定義されている ByName クエリを呼び出しています。このとき、文字列を渡すことで、その文字列値で始まる名前を返すように制限しています。

ObjectScript

```
  set statemt=##class(%SQL.Statement).%New()
  set cqStatus=statemt.%PrepareClassQuery("Sample.Person","ByName")
    if cqStatus'=1 {write "%PrepareClassQuery failed:" do $System.Status.DisplayError(cqStatus) quit}
  set rset=statemt.%Execute("L")
  do rset.%Display()
```

以下は、%PrepareClassQuery() を使用して既存のクエリを呼び出す例です。

ObjectScript

```
   set tStatement=##class(%SQL.Statement).%New()
   set cqStatus=tStatement.%PrepareClassQuery("%SYS.GlobalQuery","Size")
     if cqStatus'=1 {write "%PrepareClassQuery failed:" do $System.Status.DisplayError(cqStatus) quit}

   set install=$SYSTEM.Util.DataDirectory()
   set rset=tStatement.%Execute(install_"mgr\User")
   do rset.%Display()
```

以下の例では、CREATE QUERY 文を準備する %Prepare() を示してから、このクラス・クエリを呼び出す %PrepareClassQuery() を示します。

ObjectScript

```
  /* Creating the Query */
  set query=4
  set query(1)="CREATE QUERY DocTest() SELECTMODE RUNTIME PROCEDURE "
  set query(2)="BEGIN "
  set query(3)="SELECT TOP 5 Name,Home_State FROM Sample.Person ; "
  set query(4)="END"

  set statement = ##class(%SQL.Statement).%New()
  set qStatus = statement.%Prepare(.query)
  if qStatus '= 1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}
  write !,"Created a query",!

  /* Calling the Query */
  write !,"Calling a class query..."
  set cqStatus = statement.%PrepareClassQuery("User.queryDocTest","DocTest")
  if cqStatus '= 1 {write "%PrepareClassQuery failed:" do $SYSTEM.Status.DisplayError(cqStatus) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  write "Query data:",!,!
  while rset.%Next()
  {
    do rset.%Print()
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,"End of data."

  /* Deleting the Query */
  &sql(DROP QUERY DocTest)
  if SQLCODE < 0 {write !,"Error deleting query:", SQLCODE, " ", %msg quit}
  write !,"Deleted the query."
```

格納されたクエリにより取得されるデータの行を表示するには、この例で示すように %Print() メソッドを使用できます。格納されたクエリにより取得された特定の列データを表示するには、%Get("fieldname") メソッドまたは %GetData(colnum) メソッドのいずれかを使用する必要があります。"結果セットの繰り返し処理" を参照してください。

引数を受け入れるようにクエリが定義されている場合、"?" 文字を使用して、SQL 文で入力パラメータを指定できます。" " クエリの実行時に、%Execute() メソッドの各 "?" 入力パラメータに値を指定します。入力パラメータは、PUBLIC として宣言し、SELECT 文がそれを直接参照できるようにする必要があります。

現在作成しているクエリの情報を表示するには、次の例に示すように、%Display() メソッドを使用します。

ObjectScript

```
  /* Creating the Query */
  set myquery=4
  set myquery(1)="CREATE QUERY DocTest() SELECTMODE RUNTIME PROCEDURE "
  set myquery(2)="BEGIN "
  set myquery(3)="SELECT TOP 5 Name,Home_State FROM Sample.Person ; "
  set myquery(4)="END"

  set statement = ##class(%SQL.Statement).%New()
  set qStatus = statement.%Prepare(.query)
  if qStatus '= 1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}
  write !,"Created a query",!

  /* Preparing and Displying Info about the Query */
  write !,"Preparing a class query..."
  set cqStatus = statement.%PrepareClassQuery("User.queryDocTest","DocTest")
  if cqStatus '= 1 {write "%PrepareClassQuery failed:" do $SYSTEM.Status.DisplayError(cqStatus) quit}

  do statement.%Display()
  write !,"End of %Prepare display"

  /* Deleting the Query */
  &sql(DROP QUERY DocTest)
  if SQLCODE < 0 {write !,"Error Deleting query:",SQLCODE," ",%msg  quit }
  write !,"Deleted the query"
```

この情報は、実装クラス、引数 (リテラル値または ? 入力パラメータによる実際の引数のコンマ区切りのリスト)、および文のテキストで構成されます。

詳細は、"クラスの定義と使用" の "クラス・クエリの定義と使用" を参照してください。

8.4.3 正常な作成の結果

正常に作成できると (%Prepare()、%PrepareClassQuery()、または %ExecDirect())、%SQL.Statement の %Display() インスタンス・メソッドまたは %GetImplementationDetails() インスタンス・メソッドを呼び出して、現在作成されている文の詳細を返すことができます。次に、例を示します。

%Display() :

ObjectScript

```
  set myquery = "SELECT TOP 5 Name,Age FROM Sample.Person WHERE Age > 21"
  set tStatement = ##class(%SQL.Statement).%New()
  set qStatus = tStatement.%Prepare(myquery)
   if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
  do tStatement.%Display()
  set rset = tStatement.%Execute()
```

%GetImplementationDetails() :

ObjectScript

```
set myquery = "SELECT TOP ? Name,Age FROM Sample.Person WHERE Age > 21 AND Name=:fname"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(myquery)
 if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
set bool = tStatement.%GetImplementationDetails(.pclassname,.ptext,.pargs)
 if bool=1 {write "Implementation class= ",pclassname,!
           write "Statement text= ",ptext,!
           write "Arguments= ",$listtostring(pargs),!  }  // returns "?,?,c,21,v,fname
 else {write "%GetImplementationDetails() failed",!}
set rset = tStatement.%Execute()
```

これらのメソッドは、以下の情報を提供します。

- **実装クラス**：クエリ・キャッシュに対応するクラス名。例：`%sqlcq.SAMPLES.cls49`。
- **引数**：指定した順序での、クエリ引数のリスト。リテラル置換を抑制するために引数が二重括弧で囲まれている場合、その引数は引数リストには含まれません。

  %Display() の場合、コンマ区切りのクエリ引数リストが表示されます。それぞれの引数は、リテラル値、入力ホスト変数の名前 (コロンなし)、または入力パラメータの疑問符 (?) のいずれかです。引数がない場合、この項目には <<none>> が表示されます。複数の値を指定する述語 (IN や %INLIST など) は、各値を個別の引数として表示します。

  %GetImplementationDetails() は、クエリ引数を %List 構造として返します。それぞれの引数は、タイプ要素と値要素のペアで表されます。タイプ c (定数) の後ろにリテラル値が続き、タイプ v (変数) の後ろに入力ホスト変数の名前 (コロンなし) が続きます。タイプ ? は入力パラメータで、その後ろに 2 つ目の疑問符が続きます。引数がない場合、引数リストは空の文字列です。複数の値を指定する述語 (IN や %INLIST など) は、各値をタイプと値の個別のペアとして表示します。
- **文テキスト**：クエリ・テキスト (指定されたとおり)。大文字小文字は保持され、ホスト変数および入力パラメータは記述されたとおりに表示され、既定のスキーマは表示されません。例えば、%Prepare() の場合 `SELECT TOP :n Name FROM Clients`。例えば、%PrepareClassQuery() の場合、`call Sample.SP_Sample_By_Name(?)`。

作成済みのクエリに関して生成されるその他のメタデータ情報については、"SQL メタデータ" を参照してください。

8.4.4 preparse() メソッド

preparse() メソッドを使用することで、SQL クエリを作成しなくても、クエリ引数の %List 構造を返すことができます。クエリ引数は、%GetImplementationDetails() と同じ形式で返されます。

preparse() メソッドは、クエリ・テキストも返します。ただし、指定されたとおりにクエリ・テキストを返す %Display() や %GetImplementationDetails() とは異なり、preparse() メソッドは各クエリ引数を ? 文字に置換し、コメントを削除し、空白を正規化します。既定のスキーマ名は返しません。以下の例で、preparse() メソッドは解析バージョンのクエリ・テキストおよびクエリ引数の %List 構造を返します。

ObjectScript

```
set myq=2
set myq(1)="SELECT TOP ? Name /* first name */, Age "
set myq(2)="FROM Sample.MyTable WHERE Name='Fred' AND Age > :years -- end of query"
do ##class(%SQL.Statement).preparse(.myq,.stripped,.args)
write "preparsed query text: ",stripped,!
write "arguments list: ",$listtostring(args)
```

8.5 SQL 文の実行

%SQL.Statement クラスを使用して SQL 文を実行する方法は 2 つあります。

- %Execute() : %Prepare() または %PrepareClassQuery() を使用して作成済みの SQL 文を実行します。
- %ExecDirect() : SQL 文を作成し、実行します。
- %ExecDirectNoPriv() : SQL 文を作成および実行しますが、特権は確認しません。

$SYSTEM.SQL.Execute() メソッドを使用することで、オブジェクト・インスタンスを作成することなく SQL 文を実行することもできます。このメソッドは、SQL 文の作成と実行の両方を行います。また、クエリ・キャッシュを作成します。以下のターミナル例では、Execute() メソッドが使用されています。

```
USER>set topnum=5
USER>set rset=$SYSTEM.SQL.Execute("SELECT TOP :topnum Name,Age FROM Sample.Person")

USER>do rset.%Display()
```

8.5.1 %Execute()

クエリを作成した後は、**%SQL.Statement** クラスの %Execute() インスタンス・メソッドを呼び出すことで、そのクエリを実行できます。SELECT 以外の文の場合は、INSERT の実行などの目的の操作を %Execute() で呼び出すことができます。SELECT クエリの場合、以降の検索とデータ取得に使用する結果セットを %Execute() で生成できます。例えば以下のようにします。

ObjectScript

```
set rset = tStatement.%Execute()
```

%Execute() メソッドは、すべての SQL 文に対して **%SQL.StatementResult** クラス・プロパティ %SQLCODE および %Message を設定します。文の実行が成功すると、%SQLCODE が 0 に設定されます。これは、文によって結果が問題なく取得されたことを示しているわけではありません。同様に、文によって結果が取得されていない場合、%Execute() では %SQLCODE が 100 に設定されません。%Next() メソッドの使用などによって一度に 1 行の結果を取得すると、結果が確認され、続いて %SQLCODE が 0、100、または負数のエラー値に設定されます。

%Execute() は、以下のようにその他の **%SQL.StatementResult** プロパティを設定します。

- INSERT、UPDATE、INSERT OR UPDATE、DELETE、および TRUNCATE TABLE 文は、%ROWCOUNT を操作の影響を受ける行の数に設定します。TRUNCATE TABLE では、削除される実際の行数は特定できず、%ROWCOUNT は -1 に設定されます。

  INSERT、UPDATE、INSERT OR UPDATE、および DELETE は、%ROWID を最後に挿入、更新、または削除されたレコードの RowID 値に設定します。操作でレコードが挿入、更新、または削除されなかった場合、%ROWID は定義されないか、前の値に設定されたままになります。TRUNCATE TABLE は %ROWID を設定しません。

- SELECT 文は、結果セットの作成時に %ROWCOUNT プロパティを 0 に設定します。**%ROWCOUNT** は、プログラムが結果セットのコンテンツを繰り返し処理を行うときに、(例えば %Next() メソッドを使用して) インクリメントされます。%Next() は、行上に配置されている場合は 1 を返し、最後の行の後 (結果セットの最後) に配置されている場合は 0 を返します。カーソルが最後の行の後に配置されている場合、**%ROWCOUNT** の値は結果セットに含まれる行数を示します。

  SELECT クエリが集約関数のみを返す場合、すべての %Next() で %ROWCOUNT=1 が設定されます。最初の %Next() では、テーブルにデータがない場合でも、常に %SQLCODE=0 が設定されます。後続の %Next() では %SQLCODE=100 および %ROWCOUNT=1 が設定されます。

  また、SELECT は %CurrentResult および %ResultColumnCount も設定します。SELECT は、**%ROWID** を設定しません。

ZWRITE を使用して、すべての **%SQL.StatementResult** クラス・プロパティの値を返すことができます。

詳細は、このドキュメントの“埋め込み SQL の使用法”の章の対応する SQL システム変数の説明を参照してください。%Dialect を Sybase または MSSQL に設定して TSQL コードを実行した場合、エラーはその SQL 言語の標準プロトコルと、InterSystems IRIS の **%SQLCODE** プロパティおよび **%Message** プロパティの両方で報告されます。

8.5.1.1 入力パラメータを持つ %Execute()

%Execute() メソッドは、作成された SQL 文の入力パラメータ (“?” に示される) に対応する 1 つ以上のパラメータを取得できます。%Execute() パラメータは、SQL 文内で“?”文字が現れる順序に対応します。最初のパラメータは最初の“?”に対応し、2 番目のパラメータは、2 番目の“?”に対応し、以下同様に続きます。複数の %Execute() パラメータは、コンマで区切られます。プレースホルダとしてコンマを指定することによって、そのパラメータ値を省略できます。%Execute() パラメータの数は、“?”入力パラメータ数と一致していなければなりません。%Execute() のパラメータの数が、対応する“?”入力パラメータの数と一致しない場合は、**%SQLCODE** プロパティが SQLCODE -400 エラーに設定された状態でメソッドの実行に失敗します。

入力パラメータを使用して、リテラル値や式を SELECT リストや他のクエリ節 (TOP 節や WHERE 節など) に渡すことができます。入力パラメータを使用して、列名や列名エイリアスを SELECT リストや他のクエリ節に渡すことはできません。

明示的な %Execute() パラメータとして指定した場合の入力パラメータの最大数は 255です。可変長配列 %Execute(vals...) を使用して指定した場合の入力パラメータの最大数は 380です。

Prepare の後に、Prepare の引数メタデータを使用して、? 入力パラメータの数と必要なデータ型を返すことができます。%GetImplementationDetails() メソッドを使用して、作成済みクエリ内の ? 入力パラメータのリスト、および ? 入力パラメータをコンテキストで示したクエリ・テキストを返すことができます。

次の ObjectScript の例では、2 つの入力パラメータを持つクエリを実行します。%Execute() メソッドで入力パラメータ値 (21 と 26) を指定しています。

ObjectScript

```
SET tStatement = ##class(%SQL.Statement).%New(1)
SET tStatement.%SchemaPath = "MyTests,Sample,Cinema"
SET myquery=2
SET myquery(1)="SELECT Name,DOB,Age FROM Person"
SET myquery(2)="WHERE Age > ? AND Age < ? ORDER BY Age"
SET qStatus = tStatement.%Prepare(.myquery)
  IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
SET rset = tStatement.%Execute(21,26)
WRITE !,"Execute OK: SQLCODE=",rset.%SQLCODE,!!
DO rset.%Display()
WRITE !,"End of data: SQLCODE=",rset.%SQLCODE
```

次の ObjectScript の例では、同じクエリを実行します。%Execute() メソッドの仮パラメータ・リストは、可変長配列 (dynd...) を使用して、不確定数の入力パラメータ値を指定できます。この場合は、dynd 配列の添え字です。dynd 変数は 2 に設定されていますが、これは、2 つの添え字値を示しています。

ObjectScript

```
set tStatement = ##class(%SQL.Statement).%New(1)
set tStatement.%SchemaPath = "MyTests,Sample,Cinema"
set myquery=2
set myquery(1)="SELECT Name,DOB,Age FROM Person"
set myquery(2)="WHERE Age > ? AND Age < ? ORDER BY Age"
set dynd=2,dynd(1)=21,dynd(2)=26
set qStatus = tStatement.%Prepare(.myquery)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
set rset = tStatement.%Execute(dynd...)
write !,"Execute OK: SQLCODE=",rset.%SQLCODE,!!
do rset.%Display()
write !,"End of data: SQLCODE=",rset.%SQLCODE
```

作成された 1 つの結果セットに対して、複数の %Execute() 操作を発行できます。これにより、クエリを複数回、それぞれ異なる入力パラメータ値を指定して実行することが可能になります。次の例に示すように、複数の %Execute() 操作の間で結果セットを閉じる必要はありません。

ObjectScript

```
set myquery="SELECT Name,SSN,Age FROM Sample.Person WHERE Name %STARTSWITH ?"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(myquery)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
set rset = tStatement.%Execute("A")
do rset.%Display()
write !,"End of A data",!!
set rset = tStatement.%Execute("B")
do rset.%Display()
write !,"End of B data"
```

8.5.1.2 Try/Catch の使用による %Execute エラーの処理

TRY ブロック構造内でダイナミック SQL を実行して、関連付けられた CATCH ブロック例外ハンドラに実行時エラーを渡すことができます。%Execute() エラーの場合は、**%Exception.SQL** クラスを使用して例外インスタンスを作成してから、それを CATCH 例外ハンドラに THROW できます。

以下の例では、%Execute() エラーが発生したときに SQL 例外インスタンスを作成します。この場合は、? 入力パラメータの数 (1) と %Execute() のパラメータの数 (3) とでカーディナリティが一致しないというエラーが起こります。これにより、**%SQLCODE** プロパティと **%Message** プロパティの値 (Code および Data として) が CATCH 例外ハンドラにスローされます。この例外ハンドラは、%IsA() インスタンス・メソッドを使用して例外タイプをテストし、%Execute() エラーを表示します。

ObjectScript

```
try {
set myquery = "SELECT TOP ? Name,DOB FROM Sample.Person"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(myquery)
   if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
set rset = tStatement.%Execute(7,9,4)
   if rset.%SQLCODE=0 { write !,"Executed query",! }
   else { set badSQL=##class(%Exception.SQL).%New(,rset.%SQLCODE,,rset.%Message)
          throw badSQL }
do rset.%Display()
write !,"End of data"
return
}
catch exp { write "In the catch block",!
            if 1=exp.%IsA("%Exception.SQL") {
              write "SQLCODE: ",exp.Code,!
              write "Message: ",exp.Data,! }
            else { write "Not an SQL exception",! }
            return
}
```

8.5.2 %ExecDirect()

%SQL.Statement クラスには、クエリの作成と実行の両方を単一の操作で処理する %ExecDirect() クラス・メソッドがあります。これは、指定されたクエリ (%Prepare() など) または既存のクラス・クエリ (%PrepareClassQuery() など) を作成できます。

%ExecDirect() は、指定されたクエリを作成して実行します。

ObjectScript

```
set myquery=2
set myquery(1)="SELECT Name,Age FROM Sample.Person"
set myquery(2)="WHERE Age > 21 AND Age < 30 ORDER BY Age"
set rset = ##class(%SQL.Statement).%ExecDirect(,.myquery)
  if rset.%SQLCODE=0 { write !,"ExecDirect OK",!! }
  else { write !,"ExecDirect SQLCODE=",rset.%SQLCODE,!,rset.%Message  quit}
do rset.%Display()
write !,"End of data: SQLCODE=",rset.%SQLCODE
```

%ExecDirect() は、既存のクラス・クエリを作成して実行します。

ObjectScript

```
set mycallq = "?=CALL Sample.PersonSets('A','NH')"
set rset = ##class(%SQL.Statement).%ExecDirect(,mycallq)
  if rset.%SQLCODE=0 { write !,"ExecDirect OK",!! }
  else { write !,"ExecDirect SQLCODE=",rset.%SQLCODE,!,rset.%Message  quit}
do rset.%Display()
write !,"End of data: SQLCODE=",rset.%SQLCODE
```

次の例に示すように、%ExecDirect() クラス・メソッドの 3 番目以降のパラメータとして入力パラメータの値を指定できます。

ObjectScript

```
set myquery=2
set myquery(1)="SELECT Name,Age FROM Sample.Person"
set myquery(2)="WHERE Age > ? AND Age < ? ORDER BY Age"
set rset = ##class(%SQL.Statement).%ExecDirect(,.myquery,12,20)
  if rset.%SQLCODE'=0 {write !,"1st ExecDirect SQLCODE=",rset.%SQLCODE,!,rset.%Message  quit}
do rset.%Display()
write !,"End of teen data",!!
set rset2 = ##class(%SQL.Statement).%ExecDirect(,.myquery,19,30)
  if rset2.%SQLCODE'=0 {write !,"2nd ExecDirect SQLCODE=",rset2.%SQLCODE,!,rset2.%Message  quit}
do rset2.%Display()
write !,"End of twenties data"
```

%ExecDirect() の入力パラメータは、SQL 文内で “?” 文字が現れる順序に対応します。すなわち、3 番目のパラメータは最初の “?” に対応し、4 番目のパラメータは、2 番目の “?” に対応し、以下同様に続きます。プレースホルダとしてコンマを指定することによって、そのパラメータ値を省略できます。%ExecDirect() の入力パラメータの数が、対応する “?” 入力パラメータの数より少ない場合、既定値が存在すればその既定値が使用されます。

次の例では、最初の %ExecDirect() では、3 つの “?” 入力パラメータのすべてを指定しています。2 番目の %ExecDirect() では、2 番目の ? 入力パラメータのみを指定し、1 番目と 3 番目のパラメータは省略しています。3 番目の入力パラメータには、Sample.PersonSets() の既定値 (’MA’) が使用されます。

ObjectScript

```
set mycall = "?=CALL Sample.PersonSets(?,?)"
set rset = ##class(%SQL.Statement).%ExecDirect(,mycall,"","A","NH")
  if rset.%SQLCODE'=0 {write !,"1st ExecDirect SQLCODE=",rset.%SQLCODE,!,rset.%Message  quit}
do rset.%Display()
write !,"End of A people data",!!
set rset2 = ##class(%SQL.Statement).%ExecDirect(,mycall,,"B")
  if rset2.%SQLCODE'=0 {write !,"2nd ExecDirect SQLCODE=",rset2.%SQLCODE,!,rset2.%Message  quit}
do rset2.%Display()
write !,"End of B people data"
```

%ExecDirect() は、**%SQL.Statement** の %Display() インスタンス・メソッドまたは %GetImplementationDetails() インスタンス・メソッドを呼び出して、現在作成されている文の詳細を返すことができます。%ExecDirect() は、指定クエリでも既存のクラス・クエリでも作成して実行できるため、作成されているクエリの種類を判別するには、%GetImplementationDetails() の pStatementType パラメータを使用できます。

ObjectScript

```
set mycall = "?=CALL Sample.PersonSets('A',?)"
set rset = ##class(%SQL.Statement).%ExecDirect(tStatement,mycall,,"NH")
  if rset.%SQLCODE'=0 {write !,"ExecDirect SQLCODE=",rset.%SQLCODE,!,rset.%Message  quit}
set bool = tStatement.%GetImplementationDetails(.pclassname,.ptext,.pargs,.pStatementType)
if bool=1 {if pStatementType=1 {write "Type= specified query",!}
           elseif pStatementType=45 {write "Type= existing class query",!}
           write "Implementation class= ",pclassname,!
           write "Statement text= ",ptext,!
           write "Arguments= ",$listtostring(pargs),!!  }
else {write "%GetImplementationDetails() failed"}
do rset.%Display()
write !,"End of data"
```

詳細は、″正常な作成の結果″ を参照してください。

8.5.3 %ExecDirectNoPriv()

%SQL.Statement クラスには、%ExecDirect() のようにクエリの作成と実行の両方を単一の操作で処理する %ExecDirectNoPriv() クラス・メソッドがあります。%ExecDirectNoPriv() では、クエリの準備中に文に対する特権確認も無効になります。特権の確認を無効化することで、ダイナミック・クエリの実行をアプリケーションで詳しく制御できるようになりますが、セキュリティ・リスクは高くなります。

8.6 完全な結果セットの返送

%Execute() または %ExecDirect() のいずれかで文を実行すると、%SQL.StatementResult インタフェースを実装するオブジェクトが返されます。このオブジェクトは、単一値、結果セット、または CALL 文から返されたコンテキスト・オブジェクトのいずれかです。

8.6.1 %Display() メソッド

次の例に示すように、%SQL.StatementResult クラスの %Display() インスタンス・メソッドを呼び出すことで、結果セット（結果オブジェクトの内容）全体を表示できます。

ObjectScript

```
do rset.%Display()
```

%Display() メソッドは %Status 値を返さない点に注意してください。

%Display() によるクエリ結果セットの表示の最後には、“影響を受けた行数 5” のように行数が表示されます（これは、%Display() が結果セットを繰り返し処理した後の %ROWCOUNT 値です）。%Display() では、行数を通知するこの文の後には改行が発行されません。

%Display() には 2 つのオプションの引数があります。

- 区切り文字：データ列とデータ・ヘッダの間に挿入される文字列。結果セット列の間、ヘッダまたはデータ値の直前に表示されます。既定では、区切り文字はありません。省略されている場合は、Column Alignment フラグの前にプレースホルダのコンマを指定します。
- Column Alignment：データ列とデータ・ヘッダの間の空白の算出方法を指定する整数フラグ。使用可能なオプションは以下のとおりです。
  - 0：結果セット・ヘッダ/データ列は、標準の区切り文字（タブ）に基づいて配置されます。これが既定値です。
  - 1：結果セット・ヘッダ/データ列は、列ヘッダの長さと標準の区切り文字（タブ）に基づいて配置されます。
  - 2：結果セット・ヘッダ/データ列は、列データ・プロパティの精度/長さと標準の区切り文字（タブ）に基づいて配置されます。

8.6.2 %DisplayFormatted() メソッド

%Display() を呼び出すのではなく、%SQL.StatementResult クラスの %DisplayFormatted() インスタンス・メソッドを呼び出すことで、結果セットの内容の再フォーマットおよび生成されたファイルへのリダイレクトができます。

文字列オプション %DisplayFormatted("HTML") または対応する整数コード %DisplayFormatted(1) を指定することで、結果セットの形式を指定できます。InterSystems IRIS は、指定されたタイプのファイルを生成し、適切なファイル名拡張子を付けます。次の表は、指定できるオプションと生成できるファイルを示します。

文字列オプション	整数コード	生成されるファイルの拡張子
"XML"	0	.xml
"HTML"	1	.html
"PDF"	2	.pdf
"TXT"	99	.txt
"CSV"	100	.csv

生成された CSV ファイル内の値は、コンマではなく、タブで区切られます。

他の数値または文字列を指定すると、%DisplayFormatted() はテキスト・ファイル (.txt) を生成します。テキスト・ファイルは、行カウントで終わります (“5 Rows(s) Affected” など)。他の形式は行カウントを含みません。

結果セットのファイル名を指定することも、省略することもできます。

- 宛先ファイルを指定した場合 (例えば、`%DisplayFormatted(99,"myresults")`)、この名前と該当する接尾辞 (ファイル名拡張子) が付いたファイルが、mgr ディレクトリの現在のネームスペースのサブディレクトリ内に生成されます。例えば、C:¥InterSystems¥IRIS¥mgr¥user¥myresults.txt です。その接尾辞が付いた指定のファイルが既に存在する場合、そのファイルは新しいデータで上書きされます。
- 宛先ファイルを指定しない場合 (例えば、`%DisplayFormatted(99)`)、ランダムに生成された名前と該当する接尾辞 (ファイル名拡張子) が付いたファイルが、mgr ディレクトリの Temp サブディレクトリ内に生成されます。例えば、C:¥InterSystems¥IRIS¥mgr¥Temp¥w4FR2gM7tX2Fjs.txt です。クエリが実行されるたびに、新しい宛先ファイルが生成されます。

以下の例は、Windows のファイル名を示しています。InterSystems IRIS では、その他のオペレーティング・システムの同等の場所がサポートされます。

指定したファイルを開くことができない場合、この操作は 30 秒後にタイムアウトしてエラー・メッセージが表示されます。これは、指定されたディレクトリ (ファイル・フォルダ) に対する書き込み権限をユーザが持っていない場合に多く発生します。

データを指定された形式でレンダリングできない場合、宛先ファイルは作成されますが、結果セット・データは書き込まれません。代わりに、適切なメッセージが宛先ファイルに書き込まれます。例えば、ストリーム・フィールドの OID には、XML および HTML の特殊な書式設定文字と競合する文字が含まれます。この XML および HTML のストリーム・フィールドの問題は、ストリーム・フィールドで XMLELEMENT 関数を使用することで解決できます (例 : `SELECT Name,XMLELEMENT("Para",Notes)`)。

必要に応じて、指定された形式変換の実行時に %DisplayFormatted() によって使用される、変換テーブルの名前を指定できます。

結果セット・シーケンス内に複数の結果セットがある場合は、各結果セットの内容は、それぞれ固有のファイルに書き込まれます。

オプションの 3 番目の %DisplayFormatted() 引数は、メッセージを別個の結果セットに格納することを指定します。正常に完了すると、以下のようなメッセージが返されます。

```
Message
21 row(s) affected.
```

以下の Windows の例では、2 つの PDF (整数コード 2) 結果セット・ファイルが C:¥InterSystems¥IRIS¥mgr¥user¥ 内に作成されます。メッセージ用の mess 結果セットを作成してから、%Display() を使用してターミナルにメッセージを表示します。

ObjectScript

```
set $NAMESPACE="USER"
set myquery=2
set myquery(1)="SELECT Name,Age FROM Sample.Person"
set myquery(2)="WHERE Age > ? AND Age < ? ORDER BY Age"
set rset = ##class(%SQL.Statement).%ExecDirect(,.myquery,12,20)
  if rset.%SQLCODE'=0 {write !,"1st ExecDirect SQLCODE=",rset.%SQLCODE,!,rset.%Message  quit}
do rset.%DisplayFormatted(2,"Teenagers",.mess)
do mess.%Display()
write !,"End of teen data",!!
set rset2 = ##class(%SQL.Statement).%ExecDirect(,.myquery,19,30)
  if rset2.%SQLCODE'=0 {write !,"2nd ExecDirect SQLCODE=",rset2.%SQLCODE,!,rset2.%Message  quit}
do rset2.%DisplayFormatted(2,"Twenties",.mess)
do mess.%Display()
write !,"End of twenties data"
```

8.6.3 結果セットのページ付け

ビュー ID (%VID) を使用して、結果セットのページ付けを行うことができます。次の例は、結果セットからページを返します。各ページに 5 行ずつ格納されています。

ObjectScript

```
set q1="SELECT %VID AS RSRow,* FROM "
set q2="(SELECT Name,Home_State FROM Sample.Person WHERE Home_State %STARTSWITH 'M') "
set q3="WHERE %VID BETWEEN ? AND ?"
set myquery = q1_q2_q3
set tStatement = ##class(%SQL.Statement).%New()
set qStatus=tStatement.%Prepare(myquery)
    if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
for i=1:5:25 {
    write !!,"Next Page",!
    set rset=tStatement.%Execute(i,i+4)
    do rset.%Display()
    }
```

結果セットから行 (レコード) のグループを返す別の方法については、"%GetRows()" を参照してください。

8.7 結果セットからの特定値の返送

クエリ結果セットから特定の値を返すには、一度に 1 行ずつ繰り返し結果セットを処理する必要があります。結果セットを繰り返し処理するには、%Next() インスタンス・メソッドを使用します そして、%Print() メソッドを使用して現在の行全体の結果を表示するか、または現在の行内で指定列の値を取得することができます。

%Next() メソッドは、クエリの結果の次の行のデータを取り出し、そのデータを結果セット・オブジェクトの Data プロパティに入れます。%Next() は以下のいずれかの値を返します。

- %Next() = 1 ― クエリ結果のいずれかの行にカーソルが配置されています。
- %Next() = 0 ― 最後の行の後にカーソルが達して、これ以上返す行がないこと、またはクエリから行が返されなかったことを示しています。

%Next() の呼び出しで 1 が返されるたびに、結果セットの %ROWCOUNT プロパティ値に 1 が加算されます。カーソルが最後の行の後に達している場合 (%Next() が 0 を返す場合)、%ROWCOUNT は結果セットにある行の数を示します。

%Next() を呼び出すたびに、結果セットの %SQLCODE プロパティも更新されます。更新後の %SQLCODE 値は、取得した結果によって以下のように異なります。

- %SQLCODE = 0 ― %Next() で結果の行を正常に取得しました。
- %SQLCODE = 100 ― %Next() で行が取得されませんでした。クエリから何の結果も返されなかったか、カーソルが最後の行の後に達していて、これ以上取得する行がありません。

- %SQLCODE < 0 ― %Next() で行の取得に失敗しました。%Next() によって、%SQLCODE には取得が失敗する原因となったエラーの SQLCODE が設定されます。また、結果セットの %Message プロパティにはエラー・メッセージのテキストが設定されます。%Next() をループで反復して呼び出す場合は、エラーが発生しても通知されない状況に対処するために、%SQLCODE に負数値が設定されていないか確認し、%SQLCODE のエラー値と %Message のエラー・メッセージのテキストを表示します。以下に例を示します。

  ObjectScript

  ```
  while rset.%Next()
  {
    write "%Next succeeded."
  }
  if (rset.%SQLCODE < 0)
  {
    write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
    quit
  }
  ```

SELECT クエリから集約関数のみが返されると、すべての %Next() で %ROWCOUNT=1 が設定されます。最初の %Next() で 1 が返され、テーブルにデータがない場合でも、%SQLCODE=0 と %ROWCOUNT=1 が設定されます。以降の %Next() では 0 が返され、%SQLCODE=100 と %ROWCOUNT=1 が設定されます。

結果セットから行が取得されると、その行のデータを以下の方法で表示できます。

- rset.%Print() ：クエリ結果セットから、現在行のすべてのデータ値を返します。
- rset.%GetRow() and rset.GetRows() ：クエリ結果セットから、行のデータ値を、エンコードされたリスト構造の要素として返します。
- rset.name ：クエリ結果セットから、プロパティ名、フィールド名、エイリアス・プロパティ名、またはエイリアス・フィールド名によりデータ値を返します。
- rset.%Get("fieldname") ：クエリ結果セットまたは格納クエリのいずれかから、フィールド名またはエイリアス・フィールド名によりデータ値を返します。
- %GetData() ：クエリ結果セットまたは格納クエリのいずれかから、列番号によりデータ値を返します。

8.7.1 %Print() メソッド

%Print() インスタンス・メソッドは、結果セットにある現在のレコードを取得します。既定では、%Print() は、データ・フィールド値の間に空白の区切り文字を挿入します。%Print() は、レコードの最初のフィールド値の前や最後のフィールド値の後には空白を挿入しません。レコードの末尾では改行を発行します。データ・フィールド値に既に空白が含まれている場合、そのフィールド値は、区切り文字と区別するために引用符で囲まれます。例えば、%Print() が都市名を返す場合は、次のように返します。`Chicago "New York" Boston Atlanta "Los Angeles" "Salt Lake City" Washington`。%Print() は、結果セットにフィールドが 1 つしかない場合など、%Print() 区切り文字が使用されていないときでも、データ値の一部として区切り文字が含まれているフィールド値を引用符で囲みます。

%Print() では、フィールド値の間に配置する別の区切り文字を指定するオプションのパラメータを指定できます。別の区切り文字を指定すると、空白を含むデータ文字列の引用符はオーバーライドされます。この %Print() 区切り文字には 1 つ以上の文字を使用できます。区切り文字は、引用符で囲んだ文字列として指定します。一般に、%Print() 区切り文字には、結果セットのデータでは見かけない文字や文字列を指定することが推奨されます。ただし、結果セットのフィールド値に %Print() 区切り文字 (または文字列) が含まれる場合、そのフィールド値は、区切り文字と区別するために、引用符で囲んで返されます。

結果セットのフィールド値に改行文字が含まれる場合、フィールド値は引用符で区切って返されます。

以下の ObjectScript の例では、この区切り文字として "^|^" を指定したうえで %Print() を使用してクエリ結果セットの各レコードを表示し、この結果セットを繰り返し処理しています。%Print() で、要素のエンコードされたリストである FavoriteColors フィールドからどのようにデータが表示されるかに注意してください。

ObjectScript

```
  set q1="SELECT TOP 5 Name,DOB,Home_State,FavoriteColors "
  set q2="FROM Sample.Person WHERE FavoriteColors IS NOT NULL"
  set query = q1_q2
  set statement = ##class(%SQL.Statement).%New()

  set status = statement.%Prepare(query)
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
    write "Row count ",rset.%ROWCOUNT,!
    do rset.%Print("^|^")
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,"End of data"
  write !,"Total row count=",rset.%ROWCOUNT
```

以下の例では、区切り文字を含むフィールド値が引用符で囲んで返される様子を示します。この例では、大文字の A がフィールド区切り文字として使用されています。そのため、大文字 A のリテラルを含むフィールド値 (名前、番地、または州の省略形) は、引用符で区切られて返されます。

ObjectScript

```
  set query = "SELECT TOP 25 Name,Home_Street,Home_State,Age FROM Sample.Person"
  set statement = ##class(%SQL.Statement).%New()

  set status = statement.%Prepare(query)
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
     do rset.%Print("A")
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,"End of data"
  write !,"Total row count=",rset.%ROWCOUNT
```

8.7.2 %GetRow() および %GetRows() メソッド

%GetRow() インスタンス・メソッドは、結果セットから、現在の行 (レコード) をフィールド値の要素のエンコードされたリストとして取得します。

ObjectScript

```
  set myquery = "SELECT TOP 17 %ID,Name,Age FROM Sample.Person"
  set tStatement = ##class(%SQL.Statement).%New()
  set qStatus = tStatement.%Prepare(myquery)
    if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
  set rset = tStatement.%Execute()
  for {set x=rset.%GetRow(.row,.status)
        if x=1 {write $listtostring(row," | "),! }
        else {write !,"End of data"
              write !,"Total row count=",rset.%ROWCOUNT
              return }
      }
```

%GetRows() インスタンス・メソッドは、結果セットから、指定されたサイズの行 (レコード) のグループを取得します。各行は、フィールド値の要素のエンコードされたリストとして返されます。

以下の例では、結果セットの最初の行、6 番目の行、および 11 番目の行を返します。この例では、%GetRows() の最初のパラメータ (5) は、%GetRows() が連続する 5 行のグループを取得することを指定します。5 行のグループを正常に取得した場合、%GetRows() は 1 を返します。.rows パラメータは、参照によってこれら 5 行の添え字付き配列を渡すため、rows(1) は 5 行の各セットの最初の行 (行 1、6、11) を返します。rows(2) を指定すると、行 2、7、12 が返されます。

ObjectScript

```
set myquery = "SELECT TOP 17 %ID,Name,Age FROM Sample.Person"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(myquery)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
set rset = tStatement.%Execute()
for {set x=rset.%GetRows(5,.rows,.status)
      if x=1 {write $listtostring(rows(1)," | "),! }
      else {write !,"End of data"
            write !,"Total row count=",rset.%ROWCOUNT
            return }
     }
```

添え字によって個々の行を取得する代わりに、ZWRITE rows コマンドを使用して、取得した配列のすべての添え字を返すことができます。上の例で、ZWRITE rows は結果セットの 16 番目と 17 番目の行は返さないことに注意してください。これらの行は、5 行の最後のグループが取得された後の、残りの部分であるためです。

8.7.3 rset.name プロパティ

InterSystems IRIS は結果セットを生成するときに、その結果セット内のそれぞれのフィールド名とフィールド名エイリアスに対応する固有のプロパティが含まれた結果セット・クラスを作成します。

rset.name プロパティを使用すると、プロパティ名、フィールド名、プロパティ名エイリアス、またはフィールド名エイリアスでデータ値を返すことができます。

- プロパティ名 ： フィールド・エイリアスが定義されていない場合、フィールド・プロパティ名は rset.PropName と指定します。結果セットのフィールド・プロパティ名は、テーブル定義クラス内の対応するプロパティ名から取られます。
- フィールド名 ： フィールド・エイリアスが定義されていない場合、フィールド名 (またはプロパティ名) は rset."fieldname" と指定します。これはテーブル定義で指定されている SqlFieldName です。InterSystems IRIS はこのフィールド名を使用して、対応するプロパティ名を見つけます。多くの場合、プロパティ名とフィールド名 (SqlFieldName) は同一となります。
- エイリアス・プロパティ名 : フィールド・エイリアスが定義されている場合、エイリアス・プロパティ名は rset.AliasProp と指定します。エイリアス・プロパティ名は、SELECT 文の列名エイリアスから生成されます。定義されたエイリアスがあるフィールドのフィールド・プロパティ名は指定できません。
- エイリアス名 ： フィールド・エイリアスが定義されている場合、このエイリアス名 (またはエイリアス・プロパティ名) は rset."alias" と指定します。これは SELECT 文内の列名エイリアスです。定義されたエイリアスがあるフィールドのフィールド名は指定できません。
- 集約、式、またはサブクエリ : InterSystems IRIS はこれらの select-items に Aggregate_n、Expression_n、または Subquery_n というフィールド名を割り当てます (整数 n は、クエリで指定された select-item リストの順序に対応しています)。これらの select-item の値を取得するには、フィールド名 (rset."SubQuery_7" : 大文字と小文字の区別なし)、対応するプロパティ名 (rset.Subquery7 : 大文字と小文字の区別あり)、またはユーザ定義のフィールド名エイリアスを使用します。rset.%GetData(n) を使用して、select-item のシーケンス番号のみを指定することもできます。

プロパティ名を指定するときには、大文字/小文字を正しく使用する必要があります。フィールド名を指定するときには、大文字/小文字を区別する必要はありません。

プロパティ名を使用した rset.name のこの呼び出しは、以下のような結果になります。

- 大文字/小文字：プロパティ名は大文字/小文字を区別します。フィールド名は、大文字と小文字が区別されません。ダイナミック SQL が、指定されたフィールドまたはエイリアスの名前と、対応するプロパティ名との間の大文字/小文字の相違を自動的に解決します。ただし、大文字/小文字の解決には時間がかかります。パフォーマンスを最大化するには、プロパティ名またはエイリアスの大文字/小文字を正しく指定する必要があります。
- 非英数字：プロパティ名には英数字のみを含めることができます（先頭の % 文字は除く）。対応する SQL フィールド名またはフィールド名エイリアスに非英数字が含まれている場合（例えば `Last_Name` など）、以下のいずれかのようにすることができます。
  - フィールド名を引用符で囲んで指定します。例えば、`rset."Last_Name"` などとします。このような区切り文字の使用においては、区切り識別子を有効とする必要はありません。大文字/小文字の解決が実行されます。
  - 対応するプロパティ名を、非英数字を削除して指定します。例えば、`rset.LastName`（または `rset."LastName"`）などとします。プロパティ名には大文字/小文字を正しく指定する必要があります。
- % プロパティ名：一般に、% 文字で始まるプロパティ名は、システムによる使用のために予約されています。フィールド・プロパティ名またはエイリアスが % 文字で始まり、その名前がシステム定義のプロパティと競合する場合、システム定義のプロパティが返されます。例えば、`SELECT Notes AS %Message` の場合、`rset.%Message` を呼び出すと、Notes フィールドの値ではなく、文の結果クラスに対して定義されている %Message プロパティが返されます。フィールド値が返されるようにするには、rset.%Get(″%Message″) を使用できます。
- 列エイリアス：エイリアスが指定されている場合、ダイナミック SQL は、フィールド名またはフィールド・プロパティ名とではなく、必ずエイリアスとマッチングします。例えば、`SELECT Name AS Last_Name` の場合、データは `rset.Name` を使用してではなく、`rset.LastName` または `rset."Last_Name"` を使用してのみ取得できます。
- 重複名：複数の名前が同一のプロパティ名に解決されると、それらは重複します。重複名は、単一テーブル内の同じフィールドを複数参照するか、単一テーブル内の複数のフィールドをエイリアス参照するか、または複数のテーブル内の複数のフィールドを参照する可能性があります。例えば、`SELECT p.DOB,e.DOB` は 2 つの重複名を指定していますが、これらの名前は複数のテーブル内のフィールドを参照しています。

  SELECT 文に同じフィールド名またはフィールド名エイリアスの複数のインスタンスが含まれている場合、`rset.PropName` または `rset."fieldname"` は必ず、SELECT 文で指定された最初のインスタンスを返します。例えば、`SELECT c.Name,p.Name FROM Sample.Person AS p,Sample.Company AS c` の場合に、`rset.Name` を使用すると、会社名フィールドのデータを取得します。`SELECT c.Name,p.Name AS Name FROM Sample.Person AS p,Sample.Company AS c` の場合、`rset."name"` を使用したときも、会社名フィールドのデータを取得します。クエリ内に重複する `Name` フィールドがある場合、フィールド名（`Name`）の最後の文字は、一意のプロパティ名を作成するための文字（複数の文字の場合あり）に置換されます。このため、クエリ内の重複する `Name` フィールド名には、対応する一意のプロパティ名があり、これは `Nam0`（最初の重複）から始まり `Nam9` まで続き、さらに大文字の付いた `NamA` から `NamZ` へと続きます。

%Prepare() を使用して作成されたユーザ指定クエリでは、それ自体でプロパティ名を使用できます。%PrepareClassQuery() を使用して作成された格納クエリでは、%Get(″fieldname″) メソッドを使用する必要があります。

以下の例では、プロパティ名で指定された 3 つのフィールドの値を返します。その内の 2 つのフィールド値はプロパティ名によるもの、3 つ目のフィールド値はエイリアス・プロパティ名によるものとなります。これらの場合、指定されたプロパティ名はフィールド名またはフィールド・エイリアスと同一になります。

ObjectScript

```
  set query = "SELECT TOP 5 Name,DOB AS bdate,FavoriteColors FROM Sample.Person"
  set statement = ##class(%SQL.Statement).%New(1)

  set status = statement.%Prepare(query)
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
    write "Row count ",rset.%ROWCOUNT,!
    write rset.Name
    write " prefers ",rset.FavoriteColors
    write ", Birth date ",rset.bdate,!!
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,"End of data"
  write !,"Total row count=",rset.%ROWCOUNT
```

上記の例で、返されるフィールドの 1 つは FavoriteColors フィールドです。これには %List データが含まれます。このデータを表示するため、%New(1) クラス・メソッドにより %SelectMode プロパティ・パラメータが 1 (ODBC) に設定され、このプログラムが %List データをコンマで区切られた文字列および ODBC 形式の生年月日として表示するようにしています。

以下の例では、Home_State フィールドを返します。プロパティ名にアンダースコア文字を含めることができないため、この例では引用符で区切られたフィールド名 (SqlFieldName) を指定しています ("Home_State")。また、引用符なしで対応する生成プロパティ名を指定することもできます (HomeState)。区切られたフィールド名 ("Home_State") は大文字と小文字が区別されないが、生成されたプロパティ名 (HomeState) では区別されることに注意してください。

ObjectScript

```
  set query = "SELECT TOP 5 Name,Home_State FROM Sample.Person"
  set statement = ##class(%SQL.Statement).%New(2)

  set status = statement.%Prepare(query)
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
     write "Row count ",rset.%ROWCOUNT,!
     write rset.Name
     write " lives in ",rset."Home_State",!
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,"End of data"
  write !,"Total row count=",rset.%ROWCOUNT
```

8.7.3.1 %ObjectSelectMode=1 を使用したフィールド名・プロパティのスウィズリング

以下の例は %ObjectSelectMode=1 で作成したもので、これによりタイプ・クラスがスウィズル可能な型 (永続クラス、シリアル・クラス、またはストリーム・クラス) であるフィールドが、フィールド名プロパティを使用して値を返す際に自動的にスウィズルします。フィールド値のスウィズリングの結果は対応するオブジェクト参照 (oref) となります。InterSystems IRIS では、%Get() または %GetData() メソッドを使用してフィールドにアクセスした場合、スウィズリング処理は実行されません。この例では、`rset.Home` がスウィズルされても、同じフィールドを参照する `rset.%GetData(2)` はスウィズルされません。

ObjectScript

```
  set query = "SELECT TOP 5 Name,Home FROM Sample.Person"
  set statement = ##class(%SQL.Statement).%New(0)
  set statement.%ObjectSelectMode = 1
```

```
  write !,"set ObjectSelectMode=",statement.%ObjectSelectMode,!

  set status = statement.%Prepare(query)
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
     write "Row count: ",rset.%ROWCOUNT,!
     write rset.Name,!
     write " ",rset.Home,!
     write rset.%GetData(1)
     write " ",$listtostring(rset.%GetData(2)),!!
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,"End of data"
  write !,"Total row count=",rset.%ROWCOUNT
```

以下の例では、**%ObjectSelectMode** に 1 を指定して、選択したレコードの Home_State 値を一意のレコード ID (%ID) から取得しています。ここでは、元のクエリで Home_State フィールドが選択されていません。

ObjectScript

```
  set query = "SELECT TOP 5 %ID AS MyID,Name,Age FROM Sample.Person"
  set statement = ##class(%SQL.Statement).%New()
  set statement.%ObjectSelectMode=1

  set status = statement.%Prepare(query)
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
     write rset.Name
     write " Home State:",rset.MyID.Home.State,!
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,"End of data"
  write !,"Total row count=",rset.%ROWCOUNT
```

構成されている場合、スウィズルされるプロパティが定義されていても参照できない場合、<SWIZZLE FAIL> エラーが生成されます。これは参照されるプロパティがディスクから予期せず削除されてしまった場合、もしくは他のプロセスによりロックされた場合に起こり得ます。スウィズル失敗の原因を判別するには、<SWIZZLE FAIL> エラーの直後に %objlasterror を調べて、この %Status の値をデコードします。

既定では、<SWIZZLE FAIL> は構成されていません。`set ^%SYS("ThrowSwizzleError")=1` を設定するか、InterSystems IRIS の管理ポータルを使用することで、この動作をグローバルに設定できます。**[システム管理]** から、**[構成]**、**[SQL およびオブジェクトの設定]**、**[オブジェクト]** の順に選択します。この画面で、<SWIZZLE FAIL> オプションを設定できます。

8.7.4 %Get("fieldname") メソッド

フィールド名またはフィールド名エイリアスでデータ値を返すには、%Get("fieldname") インスタンス・メソッドを使用できます。ダイナミック SQL は、必要に応じて大文字/小文字を解決します。指定したフィールド名またはフィールド名エイリアスが存在しない場合、<PROPERTY DOES NOT EXIST> エラーが生成されます。

以下の例は、クエリ結果セットから、Home_State フィールドの値と Last_Name エイリアスの値を返します。

ObjectScript

```
  set query = "SELECT TOP 5 Home_State,Name AS Last_Name FROM Sample.Person"
  set statement = ##class(%SQL.Statement).%New(2)

  set status = statement.%Prepare(query)
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
    write rset.%Get("Home_State")," : ",rset.%Get("Last_Name"),!
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,"End of data"
  write !,"Total row count=",rset.%ROWCOUNT
```

%PrepareClassQuery() を使用して作成された既存クエリから、フィールド・プロパティ名により個別のデータ項目を取得するには、%Get("fieldname") インスタンス・メソッドを使用する必要があります。フィールド・プロパティ名が存在しない場合、<PROPERTY DOES NOT EXIST> エラーが生成されます。

以下の例では、組み込みのクエリからフィールド・プロパティ名により Nsp (ネームスペース) フィールドを返します。このクエリは既存の格納クエリなので、このフィールド取得には %Get("fieldname") メソッドを使用する必要があります。"Nsp" はプロパティ名なので、大文字と小文字が区別されることに注意してください。

ObjectScript

```
  set statement = ##class(%SQL.Statement).%New(2)
  set status = statement.%PrepareClassQuery("%SYS.Namespace","List")
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
     write "Namespace: ",rset.%Get("Nsp"),!
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,"End of data"
  write !,"Total row count=",rset.%ROWCOUNT
```

重複名 : 複数の名前が同一のプロパティ名に解決されると、それらは重複します。重複名は、同じフィールドを複数参照するか、1 テーブル内の異なるフィールドを参照するか、または異なるテーブル内の複数のフィールドを参照する可能性があります。SELECT 文に同じフィールド名またはフィールド名エイリアスの複数のインスタンスが含まれている場合、%Get("fieldname") は必ず、クエリで指定された重複名の最後のインスタンスを返します。これは、クエリで指定された重複名の最初のインスタンスを返す `rset.PropName` とは逆です。以下に例を示します。

ObjectScript

```
  set query = "SELECT c.Name,p.Name FROM Sample.Person AS p,Sample.Company AS c"
  set statement = ##class(%SQL.Statement).%New()

  set status = statement.%Prepare(query)
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
    write "Prop=",rset.Name," Get=",rset.%Get("Name"),!
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write !,rset.%ROWCOUNT," End of data"
```

8.7.5 %GetData(n) メソッド

%GetData(n) インスタンス・メソッドは、結果セットの整数値列番号でインデックス指定された現在の行に対してデータを返します。%Prepare() を使用して作成された指定クエリ、もしくは %PrepareClassQuery() を使用して作成された格納クエリのいずれかで %GetData(n) を使用できます。

整数 n はクエリ内で指定された select-item リストのシーケンスに対応しています。RowID フィールドには、select-item リストで明示的に指定されていない限り、整数 n の値は示されません。n が、クエリ内の select-item の数より大きいか、0 であるか、または負数である場合、ダイナミック SQL は値を返さず、エラーも発行しません。

%GetData(n) は、特定の重複フィールド名または重複エイリアスを返す唯一の方法です。`rset.Name` は最初の重複を返し、`%Get("Name")` は最後の重複を返します。

ObjectScript

```
  set query = "SELECT TOP 5 Name,SSN,Age FROM Sample.Person"
  set statement = ##class(%SQL.Statement).%New()

  set status = statement.%Prepare(query)
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set rset = statement.%Execute()
  if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

  while rset.%Next()
  {
    write "Years:",rset.%GetData(3)," Name:",rset.%GetData(1),!
  }
  if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}
  write "End of data"
  write !,"Total row count=",rset.%ROWCOUNT
```

8.8 複数の結果セットの返送

CALL 文は、結果セット・シーケンス (RSS) と呼ばれる集合として複数のダイナミック結果セットを返すことができます。

次の例では、%NextResult() メソッドを使用して複数の結果セットを別々に返します。

ObjectScript

```
  set mycall = "CALL Sample.CustomSets()"
  set rset = ##class(%SQL.Statement).%ExecDirect(,mycall)
  if rset.%SQLCODE'=0 {write !,"ExecDirect SQLCODE=",rset.%SQLCODE,!,rset.%Message  quit}

  set rset1 = rset.%NextResult()
  do rset1.%Display()
  write !,"End of 1st Result Set data",!!

  set rset2 = rset.%NextResult()
  do rset2.%Display()
  write !,"End of 2nd Result Set data"
```

8.9 SQL メタデータ

ダイナミック SQL には、以下のメタデータ・タイプが用意されています。

- Prepare の後に、クエリのタイプを表すメタデータ。
- Prepare の後に、クエリ内の select-item を表すメタデータ (Columns および Extended Column Info)。

- Prepare の後に、? パラメータ、:var パラメータ、および定数のクエリ引数を表すメタデータ。(Statement Parameters、Formal Parameters、および Objects)
- Execute の後に、クエリ結果セットを表すメタデータ。

%SQL.StatementMetadata プロパティの値は、Prepare 操作 (%Prepare()、%PrepareClassQuery()、または %ExecDirect()) の後に取得できます。

- %SQL.StatementMetadata のプロパティは、直前の %Prepare() については直接返すことができます。
- %SQL.StatementMetadata のプロパティの oref が含まれる、%SQL.Statement の %Metadata プロパティを返すことができます。これにより、複数の Prepare 操作のメタデータを返すことができます。

SELECT または CALL 文は、このメタデータすべてを返します。INSERT、UPDATE、または DELETE は、文のタイプを示すメタデータと**仮パラメータ**を返します。

8.9.1 文のタイプを示すメタデータ

以下に示すように、%SQL.Statement クラスを使用した Prepare の後に %SQL.StatementMetadata の statementType プロパティを使用して、作成されている SQL 文のタイプを判別できます。この例では、%SQL.Statement の %Metadata プロパティを使用し、2 つの Prepare 操作のメタデータを保持して比較しています。

ObjectScript

```
 set tStatement = ##class(%SQL.Statement).%New()
 set myquery1 = "SELECT TOP ? Name,Age,AVG(Age),CURRENT_DATE FROM Sample.Person"
 set myquery2 = "CALL Sample.SP_Sample_By_Name(?)"
 set qStatus = tStatement.%Prepare(myquery1)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
  set meta1 = tStatement.%Metadata
   set qStatus = tStatement.%Prepare(myquery2)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
  set meta2 = tStatement.%Metadata
write "Statement type query 1: ",meta1.statementType,!
write "Statement type query 2: ",meta2.statementType,!
write "End of metadata"
```

statementType プロパティの Class Reference エントリが、文のタイプを示す整数コードをリストします。一般的なコードは、1 (SELECT クエリ) と 45 (ストアド・クエリに対する CALL) です。

"正常な作成の結果" で説明しているように、%GetImplementationDetails() インスタンス・メソッドを使用しても同じ情報を返すことができます。

クエリの実行後に、結果セットから文のタイプ名 (例えば、SELECT) を返すことができます。

8.9.2 select-item のメタデータ

%SQL.Statement クラスを使用した SELECT または CALL 文の Prepare の後に、クエリに指定されている各 select-item 列に関するメタデータを返すことができます。そのためには、メタデータをすべて表示するか、個々のメタデータ項目を指定します。この列のメタデータには、ODBC データ型情報、クライアント・タイプ、インターシステムズのオブジェクトのプロパティ起源、およびクラス・タイプ情報が含まれます。

以下の例では、直前に作成されたクエリに指定されている列の数が返されます。

ObjectScript

```
set myquery = "SELECT %ID AS id,Name,DOB,Age,AVG(Age),CURRENT_DATE,Home_State FROM Sample.Person"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(myquery)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
write "Number of columns=",tStatement.%Metadata.columnCount,!
write "End of metadata"
```

以下の例では、各 select-item フィールドの列名 (または列のエイリアス)、ODBC データ型、最大データ長 (有効桁数)、およびスケールが返されます。

ObjectScript

```
set $NAMESPACE="SAMPLES"
set myquery=2
set myquery(1)="SELECT Name AS VendorName,LastPayDate,MinPayment,NetDays,"
set myquery(2)="AVG(MinPayment),$HOROLOG,%TABLENAME FROM Sample.Vendor"
set rset = ##class(%SQL.Statement).%New()
set qStatus = rset.%Prepare(.myquery)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
set x=rset.%Metadata.columns.Count()
set x=1
while rset.%Metadata.columns.GetAt(x) {
  set column=rset.%Metadata.columns.GetAt(x)
  write !,x," ",column.colName," is data type ",column.ODBCType
  write " with a size of ",column.precision," and scale = ",column.scale
  set x=x+1 }
write !,"End of metadata"
```

以下の例では、**%SQL.StatementMetadata** の %Display() インスタンス・メソッドを使用して、列のメタデータすべてが表示されます。

ObjectScript

```
set query = "SELECT %ID AS id,Name,DOB,Age,AVG(Age),CURRENT_DATE,Home_State FROM Sample.Person"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(query)
if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
do tStatement.%Metadata.%Display()
write !,"End of metadata"
```

これは、選択したフィールドの 2 つのテーブル・リストを返します。最初の列のメタデータ・テーブルには、以下のような列定義情報が示されます。

表示ヘッダ	%SQL.StatementColumn プロパティ	説明
列名	colName	列の SQL 名。列にエイリアスが指定されている場合は、フィールド名ではなく列エイリアスがここに示されます。名前とエイリアスは、12 文字に切り捨てられます。 式、集計、リテラル、ホスト変数、またはサブクエリの場合、割り当てられている “Expression_n”、“Aggregate_n”、“Literal_n”、“HostVar_n”、または “Subquery_n” ラベルがリストされます (n は SELECT 項目のシーケンス番号)。式、集計、リテラル、ホスト変数、またはサブクエリにエイリアスを割り当てていた場合は、そのエイリアスがここに示されます。
タイプ	ODBCType	ODBC データ型の整数コード。このコードについては、″InterSystems SQL リファレンス″ の ″データ型″ リファレンス・ページの ″データ型の整数コード″ セクションに一覧があります。これらの ODBC データ型コードは、CType データ型コードとは異なる点に注意してください。
Prec	precision	文字の精度または最大長。TIME データ型の精度およびスケールのメタデータについては、″日付、時刻、PosixTime、およびタイムスタンプのデータ型″ を参照してください。
スケール	scale	小数点以下の最大桁数。整数または非数値の場合は 0 を返します。TIME データ型の精度およびスケールのメタデータについては、″日付、時刻、PosixTime、およびタイムスタンプのデータ型″ を参照してください。
ヌル	isNullable	列に NULL を許可しないか (0)、許可するか (1) を示す整数値。RowID は 0 を返します。SELECT 項目が集計またはサブクエリであり、その結果が NULL になる可能性がある場合、または NULL リテラルを指定する場合、この項目は 1 に設定されます。SELECT 項目が式またはホスト変数の場合、この項目は 2 に設定されます (特定できません)。
ラベル	label	列名または列エイリアス (列名と同じ)。
テーブル	tableName	SQL テーブル名。テーブルにエイリアスを指定していた場合でも、ここには実際のテーブル名が常に示されます。SELECT 項目が式または集計である場合には、テーブル名は示されません。SELECT 項目がサブクエリである場合、サブクエリ・テーブル名が示されます。
スキーマ	schemaName	テーブルのスキーマ名。スキーマ名が指定されていない場合、システム全体の既定スキーマを返します。SELECT 項目が式または集計である場合には、スキーマ名は示されません。SELECT 項目がサブクエリである場合には、スキーマ名は示されません。
CType	clientType	クライアント・データ型の整数コード。値のリストについては、″%SQL.StatementColumn″ の ″clientType″ プロパティを参照してください。

2 番目の列のメタデータ・テーブルには、より詳細な列情報が示されます。Extended Column Info テーブルの各列には、Y (Yes) または N (No) と指定された 12 のブーリアン・フラグが示されます (SQLRESULTCOL)。

ブーリアン・フラグ	%SQL.StatementColumn プロパティ	説明
1: AutoIncrement	isAutoIncrement	RowID および IDENTITY フィールドは Y を返します。
2: CaseSensitive	isCaseSensitive	%EXACT 照合を持つ文字列データ型フィールドは Y を返します。 %SerialObject 埋め込みオブジェクトを参照するプロパティは Y を返します。
3: Currency	isCurrency	MONEY データ型など、%Library.Currency データ型で定義されたフィールド。
4: ReadOnly	isReadOnly	式、集約、リテラル、HostVar、またはサブクエリは Y を返します。RowID、IDENTITY、および RowVersion フィールドは Y を返します。
5: RowVersion	isRowVersion	RowVersion フィールドは Y を返します。
6: Unique	isUnique	一意の値制約があると定義されたフィールド。RowID および IDENTITY フィールドは Y を返します。
7: Aliased	isAliased	非フィールド選択項目にはエイリアスが指定されます。このため、ユーザが列エイリアスを指定してシステム・エイリアスを置き換えたかどうかに関係なく、式、集計、リテラル、ホスト変数、またはサブクエリは Y を返します。このフラグは、ユーザが指定した列エイリアスには影響されません。
8: Expression	isExpression	式は Y を返します。
9: Hidden	isHidden	テーブルが %PUBLICROWID または SqlRowIdPrivate=0 (既定) を指定して定義されている場合、RowID フィールドは N を返します。そうでない場合、RowID フィールドは Y を返します。%SerialObject 埋め込みオブジェクトを参照するプロパティは Y を返します。
10: Identity	isIdentity	IDENTITY フィールドとして定義されたフィールドは、Y を返します。RowID が非表示でない場合、RowID フィールドは Y を返します。
11: KeyColumn	isKeyColumn	主キー・フィールドまたは外部キー制約の対象として定義されたフィールド。RowID フィールドは Y を返します。

ブーリアン・フラグ	%SQL.StatementColumn プロパティ	説明
12: RowID	isRowId	RowID および IDENTITY フィールドは Y を返します。
13: isList	isList	データ型 %Library.List または %Library.ListOfBinary として定義されたフィールド、あるいはリストまたは配列コレクションであるフィールドは、Y を返します。CType (クライアント・データ型)=6。 $LISTBUILD または $LISTFROMSTRING 関数を使用してリストを生成する式は、Y を返します。

Extended Column Info メタデータ・テーブルには、選択した各フィールドの Column Name (SQL 名または列エイリアス)、Linked Prop (リンクされた永続クラス・プロパティ) および Type Class (データ型クラス) がリストされます。Linked Prop には、永続クラス名 (SQL テーブル名ではなく) とプロパティ名 (列エイリアスではなく) がリストされることに注意してください。

- 通常のテーブル・フィールドの場合 (`SELECT Name FROM Sample.Person`) : Linked Prop=Sample.Person.Name、Type Class=%Library.String。
- テーブルの RowID の場合 (`SELECT %ID FROM Sample.Person`) : Linked Prop= [none]、Type Class=Sample.Person。
- 式、集計、リテラル、ホスト変数、またはサブクエリの場合 (`SELECT COUNT(Name) FROM Sample.Person`) : Linked Prop= [none]、Type Class=%Library.BigIn。
- 参照されている %SerialObject 埋め込みオブジェクト・プロパティの場合 (`SELECT Home_State FROM Sample.Person`) : Linked Prop=Sample.Address.State、Type Class=%Library.String。
- %SerialObject 埋め込みオブジェクトを参照しているフィールドの場合 (`SELECT Home FROM Sample.Person`) : Linked Prop=Sample.Person.Home、Type Class=Sample.Address。

この例では、Sample.Person の Home_State フィールドは、%SerialObject クラス Sample.Address の State プロパティを参照します。

以下の例は、文パラメータでもある 1 つの仮パラメータで呼び出されたストアド・プロシージャのメタデータを返します。

ObjectScript

```
set mysql = "CALL Sample.SP_Sample_By_Name(?)"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(.mysql)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
do tStatement.%Metadata.%Display()
write !,"End of metadata"
```

これは、列 (フィールド) 情報だけでなく、文パラメータ、仮パラメータ、およびオブジェクトの値も返します。

以下の例では、3 つの仮パラメータを使用してメタデータが返されます。これら 3 つのパラメータの 1 つでは、疑問符 (?) が使用されており、これが文パラメータになります。

ObjectScript

```
set mycall = "CALL personsets(?,'MA')"
set tStatement = ##class(%SQL.Statement).%New(0,"sample")
set qStatus = tStatement.%Prepare(mycall)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
do tStatement.%Metadata.%Display()
write !,"End of metadata"
```

このメタデータでは、列情報が返されませんが、文パラメータと仮パラメータのリストには、列名とデータ型が含まれます。

8.9.3 クエリ引数のメタデータ

%SQL.Statement クラスを使用した Prepare の後に、入力パラメータ (疑問符 (?) として指定)、入力ホスト変数 (:varname として指定)、および定数 (リテラル値) のクエリ引数に関するメタデータを返すことができます。以下のメタデータを返すことができます。

- ? パラメータの数 : parameterCount プロパティ
- ? パラメータの ODBC データ型 : %SQL.StatementMetadata の %Display() インスタンス・メソッドの文パラメータのリスト。
- ?、v (:var)、および c (定数) パラメータのリスト : %GetImplementationDetails() インスタンス・メソッド (″正常な作成の結果″ を参照)。
- ?、v (:var)、および c (定数) パラメータの ODBC データ型 : formalParameters プロパティ。

  %SQL.StatementMetadata の %Display() インスタンス・メソッドの仮パラメータ・リスト。
- これらの引数を表すクエリ・テキスト : %GetImplementationDetails() インスタンス・メソッド (″正常な作成の結果″ を参照)。

文メタデータの %Display() メソッドは、文パラメータと仮パラメータをリストします。パラメータごとに、連続したパラメータ番号、ODBC データ型、精度、スケール、NULL 値が許容されるかどうか (2 は、値が常に指定されていることを意味します)、対応するプロパティ名 (colName)、および列タイプが示されます。

一部の ODBC データ型は、負の整数として返されることに注意してください。ODBC データ型整数コードのテーブルについては、″InterSystems SQL リファレンス″ の ″データ型″ リファレンス・ページを参照してください。

以下の例では、クエリ引数 (?、:var、および定数) それぞれの ODBC データ型が順番に返されます。TOP ALL を指定できるため、TOP の引数は、データ型 4 (INTEGER) ではなく 12 (VARCHAR) として返されます。

ObjectScript

```
set myquery = 4
set myquery(1) = "SELECT TOP ? Name,DOB,Age+10 "
set myquery(2) = "FROM Sample.Person"
set myquery(3) = "WHERE %ID BETWEEN :startid :endid AND DOB=?"
set myquery(4) = "ORDER BY $PIECE(Name,',',?)"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(.myquery)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
set prepmeta = tStatement.%Metadata
write "Number of ? parameters=",prepmeta.parameterCount,!
set formalobj = prepmeta.formalParameters
set i=1
while formalobj.GetAt(i) {
   set prop=formalobj.GetAt(i)
   write prop.colName," type= ",prop.ODBCType,!
   set i=i+1 }
write "End of metadata"
```

Execute の後に、引数メタデータをクエリ結果セットのメタデータから取得することはできません。結果セットでは、すべてのパラメータが解決済みです。このため、parameterCount = 0 で、formalParameters にはデータが含まれていません。

8.9.4 クエリ結果セットのメタデータ

%SQL.Statement クラスを使用した Execute の後に、以下を呼び出すことで結果セットのメタデータを返すことができます。

- %SQL.StatementResult クラスのプロパティ。
- %SQL.StatementResult の %GetMetadata() メソッド (%SQL.StatementMetadata クラスのプロパティにアクセスします)。

8.9.4.1 %SQL.StatementResult のプロパティ

Execute クエリ操作の後に、%SQL.StatementResult は以下を返します。

- %StatementType プロパティは、直前に実行された SQL 文に対応する整数コードを返します。この整数コードには、1 = SELECT、2 = INSERT、3 = UPDATE、4 = DELETE または TRUNCATE TABLE、9 = CREATE TABLE、15 = CREATE INDEX、45 = CALL などがあります。すべてのコード値のリストは、"インターシステムズ・クラス・リファレンス" の "%SQL.StatementResult" を参照してください。
- %StatementTypeName 計算プロパティは、%StatementType に基づいて、直前に実行された SQL 文のコマンド名を返します。この名前は大文字で返されます。TRUNCATE TABLE 操作は、DELETE として返されることに注意してください。INSERT OR UPDATE は、更新操作を実行する場合でも、INSERT として返されます。
- %ResultColumnCount プロパティは、結果セット行に含まれる列の数を返します。

以下の例は、これらのプロパティを示しています。

ObjectScript

```
set myquery = "SELECT TOP ? Name,DOB,Age FROM Sample.Person WHERE Age > ?"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(myquery)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
set rset = tStatement.%Execute(10,55)
if rset.%SQLCODE=0 {
write "Statement type=",rset.%StatementType,!
write "Statement name=",rset.%StatementTypeName,!
write "Column count=",rset.%ResultColumnCount,!
write "End of metadata" }
else { write !,"SQLCODE=",rset.%SQLCODE," ",rset.%Message }
```

8.9.4.2 %SQL.StatementResult の %GetMetadata()

Execute の後に、%SQL.StatementResult %GetMetadata() メソッドを使用して、%SQL.StatementMetadata クラスのプロパティにアクセスできます。これらのプロパティは、Prepare の後に %SQL.Statement の %Metadata プロパティを使用してアクセスするものと同じです。

以下の例は、これらのプロパティを示しています。

ObjectScript

```
set myquery=2
set myquery(1)="SELECT Name AS VendorName,LastPayDate,MinPayment,NetDays,"
set myquery(2)="AVG(MinPayment),$HOROLOG,%TABLENAME FROM Sample.Vendor"
set tStatement = ##class(%SQL.Statement).%New()
set qStatus = tStatement.%Prepare(.myquery)
  if qStatus'=1 {write "%Prepare failed:" do $System.Status.DisplayError(qStatus) quit}
  set rset = tStatement.%Execute()
if rset.%SQLCODE=0 {
set rsmeta=rset.%GetMetadata()
set x=rsmeta.columns.Count()
set x=1
while rsmeta.columns.GetAt(x) {
  set column=rsmeta.columns.GetAt(x)
  write !,x," ",column.colName," is data type ",column.ODBCType
  write " with a size of ",column.precision," and scale = ",column.scale
  set x=x+1 }
}
else { write !,"SQLCODE=",rset.%SQLCODE," ",rset.%Message }
write !,"End of metadata"
```

結果セットのメタデータは、引数メタデータを提供しないことに注意してください。これは、Execute 操作がすべてのパラメータを解決するからです。このため、結果セットでは、parameterCount = 0 で、formalParameters にはデータが含まれていません。

8.10 ダイナミック SQL の監査

InterSystems IRIS では、ダイナミック SQL 文の監査をオプションでサポートしています。ダイナミック SQL の監査は、%System/%SQL/DynamicStatement システム監査イベントを有効化しているときに実行されます。既定では、このシステム監査イベントは有効化されていません。

%System/%SQL/DynamicStatement を有効にすると、システム全体で実行されるすべての %SQL.Statement ダイナミック文が自動的に監査されます。監査の情報は、監査データベースに記録されます。

監査データベースを表示するには、管理ポータルに移動し、[**システム管理**]、[**セキュリティ**]、[**監査**]、[**監査データベースの閲覧**] の順に選択します。DynamicStatement に [**イベント名**] フィルタを設定して、[**監査データベースの閲覧**] をダイナミック SQL 文に制限することができます。監査データベースには、イベントの時間 (ローカル・タイムスタンプ)、ユーザ、PID (プロセス ID)、および説明がリストされます。説明では、ダイナミック SQL 文のタイプを指定します。例えば、SQL `SELECT Statement (%SQL.Statement)` や SQL `CREATE VIEW Statement (%SQL.Statement)` です。

イベントの [**詳細**] リンクを選択することで、[**イベントデータ**] などの追加情報をリストできます。イベント・データには、実行された SQL 文と、文の引数の値が含まれます。以下に例を示します。

```
SELECT TOP ? Name , Age FROM Sample . MyTest WHERE Name %STARTSWITH ?
/*#OPTIONS {"DynamicSQLTypeList":",1"} */
Parameter values:
%CallArgs(1)=5
%CallArgs(2)="Fred"
```

文とパラメータが含まれるイベント・データの最大合計長は、3,632,952 文字です。文とパラメータの合計長が 3,632,952 文字を超える場合、イベント・データは切り捨てられます。

InterSystems IRIS では、ODBC および JDBC 文 (Event Name=XDBCStatement) の監査、および埋め込み SQL 文 (Event Name=EmbeddedStatement) の監査もサポートしています。

9

SQL シェル・インタフェースの使用法

SQL 文をテストする 1 つの方法として、SQL シェルを使用して InterSystems IRIS® データ・プラットフォーム・ターミナルから SQL 文を実行する方法があります。このインタラクティブな SQL シェルでは、SQL 文を動的に実行できます。SQL シェルは ダイナミック SQL を使用するので、実行時にクエリが作成および実行されます。SQL シェルは、現在のネームスペース内でリソースにアクセスして操作を実行します。

特に明記のない限り、SQL シェル・コマンドおよび SQL コードでは大文字と小文字は区別されません。

この章で説明する項目は以下のとおりです。

- ターミナル・プロンプト、管理ポータル、またはプログラムから SQL を実行するその他の方法。
- SQL シェルの呼び出し。ターミナルから単一行または複数の SQL 文として SQL を入力および実行します。
- 入力パラメータを使用して、実行時に SQL 文に値をインタラクティブに渡します。
- SQL シェル内から ObjectScript コマンドを実行します。
- 現在のネームスペース内のスキーマ、テーブル、およびビューの参照。
- SQL CALL 文を使用して、SQL ストアド・プロシージャを実行します。
- シェルの RUN コマンドを使用して、SQL スクリプト・ファイルを実行します。
- 番号または割り当てられた名前によって、SQL 文の格納と呼び出しを行います。
- クエリ・キャッシュの削除
- SQL シェルの構成パラメータの設定。
- SQL 文のメタデータの表示。
- 文テキスト、クエリ・キャッシュ名、リテラル引数の表示。
- EXPLAIN または SHOW PLAN を使用したクエリ・プランの表示。
- SQL 文のパフォーマンスのタイミング。
- SQL シェルから Transact-SQL 文を実行します (Sybase または MSSQL)。

9.1 SQL を実行するその他の方法

$SYSTEM.SQL.Execute() メソッドを使用すると、SQL シェルを呼び出さずにターミナル・コマンド行から SQL コードを 1 行実行できます。以下の例では、ターミナル・プロンプトからこのメソッドを使用する方法を示します。

```
USER>SET result=$SYSTEM.SQL.Execute("SELECT TOP 5 name,dob,ssn FROM Sample.Person")

USER>DO result.%Display()

USER>SET result=$SYSTEM.SQL.Execute("CALL Sample.PersonSets('M','MA')")

USER>DO result.%Display()
```

SQL 文にエラーがある場合は、Execute() メソッドは正常に完了し、%Display() メソッドは以下のようなエラー情報を返します。

```
USER>DO result.%Display()

[SQLCODE: <-29>:<Field not found in the applicable tables>]
[%msg: < Field 'GAME' not found in the applicable tables^ SELECT TOP ? game ,>]
0 Rows Affected
USER>
```

Execute() メソッドは、オプションのパラメータ SelectMode、Dialect、および ObjectSelectMode も提供します。

InterSystems IRIS は、このドキュメントの他の章で説明するように、SQL コードを記述し実行する、他の多くの方法をサポートしています。これには、以下のものがあります。

- 埋め込み SQL：ObjectScript コードに埋め込んだ SQL コードです。
- ダイナミック SQL：**%SQL.Statement** クラス・メソッドを使用して、ObjectScript コードから SQL 文を実行します。
- 管理ポータルの SQL インタフェース：InterSystems IRIS 管理ポータルから **[クエリ実行]** インタフェースを使用してダイナミック SQL を実行します。

9.2 SQL シェルの呼び出し

以下のように、$SYSTEM.SQL.Shell() メソッドを使用して、ターミナルのプロンプトから SQL シェルを呼び出すことができます。

ObjectScript

```
DO $SYSTEM.SQL.Shell()
```

または、以下のように、**%SQL.Shell** クラスを使用して、インスタンス化されたインスタンスとして SQL シェルを呼び出すことができます。

ObjectScript

```
DO ##class(%SQL.Shell).%Go("IRIS")
```

または

ObjectScript

```
SET sqlsh=##class(%SQL.Shell).%New()
DO sqlsh.%Go("IRIS")
```

呼び出す方法に関係なく、SQL シェルは、以下のように表示される SQL シェル・プロンプトを返します。

```
[SQL]termprompt>>
```

[SQL] は、SQL シェル内にいることを示すリテラルです。termprompt は、構成されているターミナル・プロンプトです。>> は、SQL コマンド行を示すリテラルです。既定では、SQL シェル・プロンプトは [SQL]*nsp*>> のように表示されます。nsp は、現在のネームスペースの名前です。

このプロンプトでは、以下のシェル・モードのいずれも使用できます。

- 単一行モード：プロンプトで、SQL コードを 1 行入力します。SQL 文を終了するには、Enter キーを押します。既定では、SQL コードが作成および実行されます (即時実行モードと呼ばれます)。クエリの結果セットは、ターミナル画面に表示されます。その他の SQL 文の場合、SQLCODE と行カウント値がターミナル画面に表示されます。
- 複数行モード：プロンプトで、Enter キーを押します。これで、複数行モードになります。SQL コードを複数行入力できます。新しい行のプロンプトではそれぞれ、行番号が示されます (空白行によって行番号がインクリメントされることはありません)。複数行の SQL 文を終了させるには、「GO」と入力し、Enter キーを押します。既定では、いずれの場合も、SQL コードが作成および実行されます。クエリの結果セットは、ターミナル画面に表示されます。その他の SQL 文の場合、SQLCODE と行カウント値がターミナル画面に表示されます。

  複数行モードでは、以下のコマンドが用意されています。これらのコマンドを複数行プロンプトに入力し、Enter キーを押します。L または LIST は、これまでに入力されたすべての SQL コードをリストします。C または CLEAR は、これまでに入力されたすべての SQL コードを削除します。C n または CLEAR n (n は行番号を示す整数) は、SQL コードの特定の行を削除します。G または GO は、SQL コードを作成および実行し、単一行モードに戻ります。Q または QUIT は、これまでに入力されたすべての SQL コードを削除し、単一行モードに戻ります。これらのコマンドでは、大文字と小文字は区別されません。コマンドを発行しても、次の複数行プロンプトで行番号がインクリメントされることはありません。複数行プロンプトで「?」と入力すると、これらの複数行コマンドがリストされます。

SQL 文を作成する場合、SQL シェルでは最初に文が検証されます。この検証では、指定したテーブルが現在のネームスペースに存在しているか、および指定したフィールドがテーブル内に存在しているかどうかの確認が行われます。存在しない場合、適切な SQLCODE が表示されます。

SQL シェルでは SQL 特権が確認されるので、テーブルやフィールドなどへのアクセスや変更のための適切な特権が必要です。詳細は、このドキュメントの“SQL のユーザ、ロール、および特権”の章を参照してください。

文が有効で適切な特権を持っている場合は、SQL シェルに SQL 文が表示され、連続番号が割り当てられます。この番号はターミナル・セッションが存続している限り、ネームスペースの変更や SQL シェルの終了と再開にかかわりなく連続的に割り当てられます。SQL 文に割り当てられた番号によって、以下のように、既存の SQL 文を呼び出すことができます。

使用可能なすべての SQL シェル・コマンドのリストを表示するには、SQL プロンプトで ? を入力します。

SQL シェルのセッションを終了し、ターミナルのプロンプトに戻るには、SQL プロンプトに Q または QUIT コマンドか、E または EXIT コマンドを入力します。SQL シェル・コマンドでは、大文字と小文字が区別されません。Ctrl-C コマンドは、SQL シェルでは無効になっています。

既定のパラメータ設定を使用した SQL シェルのセッションの例を以下に示します。

```
USER>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]USER>>SELECT TOP 5 Name,Home_State FROM Sample.Person ORDER BY Home_State
1. SELECT TOP 5 Name,Home_State FROM Sample.Person ORDER BY Home_State

Name                Home_State
Djokovic,Josephine W.   AK
Klingman,Aviel P.       AK
Quine, Sam X.           AK
Xiang,Robert C.         AL
Roentgen,Alexandria Q.  AR
```

```
5 Row(s) Affected
---------------------------------------------
[SQL]USER>>SELECT GETDATE()
2. SELECT GETDATE()

Expression_1
2009-09-29 11:41:42

1 Row(s) Affected
---------------------------------------------
[SQL]USER>>QUIT

USER>
```

既定のパラメータ設定を使用した、複数行の SQL シェルのセッションを以下に示します。

```
USER>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
---------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]USER>>   << entering multiline statement mode >>
        1>>SELECT TOP 5
        2>>Name,Home_State
        3>>FROM Sample.Person
        4>>ORDER BY Home_State
        5>>GO

1. SELECT TOP 5
   Name,Home_State
   FROM Sample.Person
   ORDER BY Home_State

Name               Home_State
Djokovic,Josephine W.  AK
Klingman,Aviel P.      AK
Quine, Sam X.          AK
Xiang,Robert C.        AL
Roentgen,Alexandria Q. AR

5 Row(s) Affected
---------------------------------------------
[SQL]USER>>
```

9.2.1 GO コマンド

SQL シェルの GO コマンドは、最後の SQL 文を実行します。単一行モードでは、GO は最後に実行した SQL 文を再実行します。複数行モードでは、GO コマンドは、複数の SQL 文を実行して複数行モードを終了するために使用されます。単一行モードでの後続の GO は、前の複数の SQL 文を再実行します。

9.2.2 入力パラメータ

SQL シェルでは、“?” 文字を使用した入力パラメータを SQL 文に使用できます。SQL 文を実行するたびに、それらの入力パラメータの値を指定するように求められます。それらの値は、SQL 文に各 “?” 文字が記述されているのと同じ順で指定する必要があります。最初のプロンプトは最初の “?” に値を渡し、2 番目のプロンプトは 2 番目の “?” に値を渡し、以下同様の対応となります。

入力パラメータの個数に制限はありません。入力パラメータを使用して、TOP 節および WHERE 節に値を渡したり、SELECT リストに式を渡したりできますが、SELECT リストに列名を渡すことはできません。

ホスト変数は、入力パラメータ値として指定できます。入力パラメータの入力を求められたら、コロン (:) で始まる値を指定します。この値には、パブリック変数、ObjectScript 特殊変数、数値リテラル、または式を指定できます。次に、“is this a literal (Y/N)?” というプロンプトが表示されます。このプロンプトで N (No) を指定すると (または、何も指定しないで Enter キーを押すと)、入力値はホスト変数と解析されます。例えば、`:myval` はローカル変数 myval の値、`:^myval` はグローバル変数 ^myval の値、`:$HOROLOG` は特殊変数 $HOROLOG の値、`:3` は数字の 3、`:10-3` は数字の 7 と解析されます。このプロンプトで Y (Yes) を指定すると、先頭のコロンを含む入力値が、リテラルとして入力パラメータに渡されます。

9.2.3 ObjectScript コマンドの実行

SQL シェル内では、ObjectScript コマンドを発行できます。例えば、SET $NAMESPACE コマンドを使用して、InterSystems IRIS ネームスペースを、参照する SQL テーブルまたはストアド・プロシージャが含まれるネームスペースに変更する場合などです。SQL シェルで ! コマンドまたは OBJ コマンドを使用して、1 つ以上の ObjectScript コマンドで構成される ObjectScript コマンド行を発行できます (OBJ は OBJECTSCRIPT の省略形です)。!、OBJ、および OBJECTSCRIPT コマンドは同義語です。これらのコマンドの使用法を以下の例に示します。

```
%SYS>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]%SYS>>! SET oldns=$NAMESPACE SET $NAMESPACE="USER" WRITE "changed the namespace"
changed the namespace
[SQL]USER>>OBJ SET $NAMESPACE=oldns WRITE "reverted to old namespace"
reverted to old namespace
[SQL]%SYS>>
```

コマンド行で OBJ コマンドに続く残りの部分は、ObjectScript コードとして扱われます。! と ObjectScript コマンド行の間にスペースは必要ありません。OBJ コマンドは、SQL シェル単一行モードまたは SQL シェル複数行モードで指定することができます。以下の例では、USER ネームスペースで定義されたテーブルに対して SELECT クエリを実行します。

```
%SYS>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]%SYS>>  << entering multiline statement mode >>
        1>>OBJ SET $NAMESPACE="USER"

        1>>SELECT TOP 5 Name,Home_State
        2>>FROM Sample.Person
        3>>GO
   /* SQL query results */
[SQL]USER>>
```

OBJ 文が SQL 行カウントを進めることはありません。

SQL シェル複数行モードでは、OBJ コマンドは改行時に実行されますが、SQL 文は GO を指定するまで発行されません。したがって、以下の例は、機能的には前述の例と同じです。

```
%SYS>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]%SYS>>  << entering multiline statement mode >>
        1>>SELECT TOP 5 Name,Home_State
        2>>FROM Sample.Person
        3>>OBJ SET $NAMESPACE="USER" WRITE "changed namespace"
changed namespace
        3>>GO
   /* SQL query results */
[SQL]USER>>
```

以下の例では、OBJ コマンドを使用してホスト変数を定義します。

```
USER>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]USER>>  << entering multiline statement mode >>
        1>>SELECT TOP :n Name,Home_State
        2>>FROM Sample.Person
        3>>OBJ SET n=5

        3>>GO
```

9.2.4 ネームスペースの参照

SQL シェルでは、現在のネームスペースに定義されているか現在のネームスペースからアクセス可能なスキーマ、テーブル、およびビューを表示する BROWSE コマンドがサポートされます。この表示は、複数レベルのプロンプトで構成されています。前のプロンプト・レベルに戻るには、プロンプトで Return (Enter) キーを押します。名前は、大文字と小文字を区別します。

1. SQL シェル・プロンプトで「BROWSE」と入力すると、現在のネームスペース内のスキーマがリストされます。
2. Schema: プロンプトで、スキーマを名前または番号で選択します。これにより、そのスキーマ内のテーブルとビューがリストされます。
3. Table/View: プロンプトで、テーブル (T) またはビュー (V) を名前または番号で選択します。これにより、テーブルの情報が表示されてから、オプションのリストが表示されます。
4. Option: プロンプトで、オプションを番号で選択します。このオプションを使用して、テーブルに定義されているフィールドまたはマップをリストできます。

   オプション 1 (名前別のフィールド) またはオプション 2 (番号別のフィールド) を指定して、Field: プロンプトを表示します。オプション 3 (マップ) を指定すると、Map: プロンプトが表示されます。
5. Field: プロンプトで、フィールドを番号または名前で選択するか、* を指定してすべてのフィールドをリストします。これにより、詳細なフィールド情報が表示されます。

   Map: プロンプトで、マップを番号または名前で選択するか、* を指定してすべてのマップをリストします。これにより、詳細なマップ情報が表示されます。

9.2.5 CALL コマンド

SQL シェルを使用して、以下の例のように、SQL CALL 文を発行してSQL ストアド・プロシージャを呼び出すことができます。

```
[SQL]USER>>CALL Sample.PersonSets('G','NY')
```

指定したストアド・プロシージャが現在のネームスペースに存在しない場合、SQL シェルは SQLCODE -428 エラーを発行します。

入力パラメータの指定がストアド・プロシージャで定義された数よりも多い場合、SQL シェルは SQLCODE -370 エラーを発行します。パラメータの値をストアド・プロシージャに指定するには、リテラル ('string')、ホスト変数 (:var)、および入力パラメータ (?) を任意に組み合わせて使用することができます。

- 以下の例に示すように、CALL 文では ホスト変数 を使用することができます。

  ```
  [SQL]USER>>OBJ SET a="G",b="NY"
  [SQL]USER>>CALL Sample.PersonSets(:a,:b)
  ```

- 以下の例に示すように、CALL 文では入力パラメータ (“?” 文字) を使用することができます。

  ```
  [SQL]USER>>CALL Sample.PersonSets(?,?)
  ```

  CALL 文が実行される際には、SQL シェルによって、これらの各入力パラメータの値を指定するように求められます。

9.2.6 SQL スクリプト・ファイルの実行

SQL シェルの RUN コマンドは、SQL スクリプト・ファイルを実行します。スクリプト・ファイルのタイプは、DIALECT の設定によって決まります。DIALECT の既定は IRIS (InterSystems SQL) です。詳細は、この章で後述する "RUN コマンド" を参照してください。

9.3 SQL 文の格納と呼び出し

9.3.1 番号による呼び出し

SQL シェルでは自動的に、ターミナル・セッション中に正常に発行された各 SQL 文がローカル・キャッシュに格納され、それぞれに連続した番号が割り当てられます。この番号は、現在のターミナル・プロセスにおける既存の SQL 文の呼び出しに使用されます。SQL シェルでは、正常完了した SQL 文にのみ番号が割り当てられます。SQL 文の作成時にエラーが発生した場合、番号は割り当てられません。これらの番号の割り当ては、ネームスペースに固有ではありません。以下は、番号コマンドによる使用可能な呼び出しです。

- # : # を使用すると、キャッシュされた既存のすべての SQL 文とそれらに割り当てられている番号をリスト表示できます。
- #n : 既存の SQL 文は、SQL シェル・プロンプトで #n を指定することで、呼び出して実行することができます。n は SQL シェルが文に割り当てた整数を表します。
- #0 : 直前に作成された SQL 文は、SQL シェル・プロンプトで #0 コマンドを指定することで、呼び出して実行することができます。#0 は、直前に作成された SQL 文を呼び出しますが、それが最後に実行された SQL 文であるとは限りません。そのため、SQL 文の呼び出しと実行は、#0 によってどの SQL 文が呼び出されるかには影響しません。

番号を使用して SQL 文を呼び出す際は、その文に新しい番号は割り当てられません。SQL シェルでは、ターミナル・セッションの継続中に番号が連続的に割り当てられます。SQL シェルをいったん終了してから再開したり、ネームスペースを変更したりしても、番号の割り当てや既に割り当てられている番号の有効性には影響しません。

すべての番号割り当てを削除するには、#CLEAR を使用して、表示されたプロンプトでこのアクションを確認します。この操作により、それまでに割り当てられていたすべての番号割り当てが削除され、番号割り当てが 1 から再開されます。

9.3.2 名前による呼び出し

オプションとして、SQL 文に名前を割り当て、その名前で文を呼び出すことができます。この名前は、現在のユーザのすべてのターミナル・プロセスで発行された既存の SQL 文の呼び出しに使用されます。名前によって SQL 文の保存と呼び出しを行うには、以下の 2 つの方法があります。

- SAVEGLOBAL を使用してグローバルに保存し、OPEN を使用してグローバルから呼び出す。
- SAVE を使用してファイルに保存し、LOAD を使用してファイルから呼び出す。

9.3.2.1 グローバルへの保存

直前の SQL 文にグローバル名を割り当てるには、SQL シェル・コマンド SAVEGLOBAL name を使用します。このコマンドは、SG name と省略できます。この後は、SQL シェル・コマンド OPEN name を使用して、グローバルから SQL 文を呼び出すことができます。EXECUTEMODE が IMMEDIATE の場合、SQL シェルは文の呼び出しと実行の両方を行います。EXECUTEMODE が DEFERRED の場合、文は作成されますが、GO コマンドを指定するまで実行されません。

OPEN name を使用して SQL 文をグローバル名で呼び出すたびに、SQL シェルによって新しい番号が文に割り当てられます。番号による呼び出しでは、古い番号と新しい番号の両方が有効です。

name には、空白スペース文字以外のすべての表示可能文字を使用できます。name では、大文字と小文字が区別されます。name は、任意の長さにできます。name は現在のネームスペースに固有です。同一の SQL 文を別の名前で複数回保存できます。この場合、保存されたすべての名前が有効になります。既に割り当てられている名前を使用して SQL 文を保存しようとすると、SQL シェルによって、既存の名前を上書きして新しい SQL 文にその名前を再割り当てするかどうかの確認が求められます。

グローバル名は、現在のネームスペースに対して割り当てられます。SQL シェルの L（または LIST）コマンドを使用すると、現在のネームスペースに割り当てられているすべてのグローバル名をリスト表示できます。割り当てられた name は、現在のユーザのすべてのターミナル・プロセスで使用できます。割り当てられた name は、それを作成したターミナル・プロセスが終了しても存続します。name の割り当てがない場合、LIST は“No statements saved”というメッセージを返します。

グローバルの name の割り当てを削除するには、CLEAR name を使用します。現在のネームスペースのすべてのグローバルの name の割り当てを削除するには、CLEAR を使用し、表示されたプロンプトでこのアクションを確認します。

9.3.2.2 ファイルへの保存

直前の SQL 文にファイル名を割り当てるには、SQL シェル・コマンド SAVE name を使用します。この後は、SQL シェル・コマンド LOAD name を使用して、SQL 文を呼び出すことができます。EXECUTEMODE が IMMEDIATE の場合、SQL シェルは文の呼び出しと実行の両方を行います。LOAD name を使用して SQL 文をファイル名で呼び出すたびに、SQL シェルによって新しい番号が文に割り当てられます。番号による呼び出しでは、古い番号と新しい番号の両方が有効です。

name には、空白スペース文字以外のすべての表示可能文字を使用できます。name では、大文字と小文字が区別されます。name は、任意の長さにできます。name は現在のネームスペースに固有です。同一の SQL 文を別の名前で複数回保存できます。この場合、保存されたすべての名前が有効になります。既に割り当てられている名前を使用して SQL 文を保存しようとすると、SQL シェルによって、既存の名前を上書きして新しい SQL 文にその名前を再割り当てするかどうかの確認が求められます。

名前は、現在のネームスペースに対して定義されます。割り当てられた name は、現在のユーザのすべてのターミナル・プロセスで使用できます。割り当てられた name は、それを作成したターミナル・プロセスが終了しても存続します。

9.4 クエリ・キャッシュの削除

SQL シェルは PURGE（省略形は P）コマンドを指定して、現在のネームスペースにあるすべてのクエリ・キャッシュを削除します。このコマンドは、SQL シェルを使用して生成されたクエリ・キャッシュだけでなく、ネームスペース内のすべてのクエリ・キャッシュを削除します。

$SYSTEM.SQL.Purge() メソッドおよび管理ポータルの [アクション] ドロップダウン・リストのオプションには、選択したクエリ・キャッシュのみの削除や、ネームスペースにあるすべてのクエリ・キャッシュの削除をはじめとする詳細なオプションが用意されています。

9.5 SQL シェルの構成

- 管理ポータルを使用して、SQL シェルの既定値をシステム全体に構成できます。
- SQL シェルのパラメータを使用して、個々の SQL シェルを構成できます。SQL シェルのパラメータを変更すると、SQL シェルの現在の呼び出しについてのシステム全体の既定値がオーバーライドされます。システム全体の SQL シェルの既定値は変更されません。

以下は、指定可能な SQL シェル構成オプション、対応するシェルのパラメータ、および既定の設定を示しています。

管理ポータルのシェルの構成	シェルのパラメータ	既定値
[選択モード]	selectmode	[論理]
[SQL 言語] (TSQL)	dialect (TSQL)	[IRIS]
[スキーマ検索パス]	path	なし
[結果列の配置]	colalign	[区切り文字]
[コマンド接頭語] (TSQL)	commandprefix (TSQL)	なし
[結果出力表示モード]	displaymode	[現在のデバイス]
[表示パス]	displaypath	なし
[表示ファイル]	displayfile	なし
[表示ファイル変換テーブル]	displaytranslatetable	なし
[エコー・モード]	echo	[オン]
[実行モード]	executemode	[即時]
[メッセージ・モード]	messages	[オン]
[実行を許可する IF 条件]		[1]
	log	[オフ]

ラベルの付いたパラメータ (TSQL) は主に、Transact-SQL コードである Sybase または MSSQL を SQL シェルから実行するために使用されます。これらについては、この章の最後の “Transact-SQL のサポート” のセクションで説明します。

9.5.1 SQL シェルのシステム全体の既定値の構成

管理ポータルに進み、[**システム管理**]、[**構成**]、[**SQL およびオブジェクトの設定**]、[SQL] の順に選択します。[**SQL シェル**] タブを選択します。システム全体に適用されている SQL シェルの現在の既定の設定を確認して設定します。

1 つまたは複数の構成設定を変更した場合、画面左上の管理ポータルのパスのすぐ後にアスタリスク (*) が表示され、変更したことが示されます。例えば、`System > Configuration > SQL *` のようになります。[**保存**] ボタンを押して、変更内容を受け入れます。変更が有効になり、アスタリスクは消えます。

9.5.2 SQL シェルのパラメータの構成

SQL シェルの構成パラメータは、現在のターミナル・プロセスにおける現在の SQL シェルの呼び出しに固有です。設定は複数のネームスペースにわたって適用されます。ただし、SQL シェルを終了すると、SQL シェルのパラメータはすべてシステム全体の既定値にリセットされます。InterSystems IRIS ではシステム既定値が用意されていますが、以下で説明するように、SET SAVE を使用して、現在のプロセスに別の既定値を設定できます。

SQL シェルの SET コマンド (引数なし) では、以下の例に示すように、現在のシェル構成パラメータが表示されます。この例では、SET によりシステム既定値が表示されますが、これらは SQL シェルを呼び出した場合に設定される値です。

```
[SQL]USER>>SET

commandprefix = ""
dialect = IRIS
displayfile =
displaymode = currentdevice
displaypath =
displaytranslatetable =
echo = on
executemode = immediate
log = off
messages = on
path = SQLUser
selectmode = logical
[SQL]USER>>
```

単一の構成パラメータの現在の設定を表示するには、SET param を指定します。例えば、SET SELECTMODE では、現在の selectmode 設定が返されます。

SQL シェルの SET コマンドを使用して、シェル構成パラメータを設定することができます。設定値は SQL シェルを呼び出している間維持されます。SQL シェルを呼び出すたびに、パラメータは既定値にリセットされます。SET では、以下の構文形式のうちのいずれかを使用できます。

```
SET param value SET param = value
```

param と value のどちらも、大文字と小文字が区別されません。スペースは許可されていますが、等号の前後には不要です。

SQL シェルの SET SAVE コマンドでは、現在のシェル構成パラメータの設定がユーザの既定値として保存されます。これらの既定値は、現在のプロセスから後続の SQL シェルのすべての呼び出しに適用されます。また、これらは SQL シェルの既定値として、ユーザが起動するターミナル・プロセスで今後呼び出されるすべての SQL シェルにも適用されます。これは明確にリセットされるまで保持されます。SET SAVE を使用しても、現在実行中の SQL シェルの呼び出しには影響しません。

SQL シェルの SET CLEAR コマンドでは、現在のプロセスに対するシェル構成パラメータの現在の設定がクリアされます (システムの既定値にリセットされます)。InterSystems IRIS では、現在のプロセス、または現在のユーザによって呼び出されるすべての新しいターミナル・プロセスによる後続の SQL シェルの呼び出しに、この既定値へのリセットが適用されます。SET CLEAR は、現在実行中の SQL シェルの呼び出しには影響しません。

SET SAVE も SET CLEAR も、管理ポータルを使用して構成および表示されるシステム全体の SQL シェルの既定設定を変更することはありません。

9.5.3 COLALIGN の設定

SET COLALIGN を使用して、クエリ結果セット・データと列ヘッダの表示に使用する空白形式を指定できます。使用可能なオプションは以下のとおりです。

- 区切り文字：結果セット・ヘッダ/データ列は、標準の区切り文字 (タブ) に基づいて配置されます。これが既定値です。
- ヘッダ ：結果セット・ヘッダ/データ列は、列ヘッダの長さと標準の区切り文字 (タブ) に基づいて配置されます。
- データ：結果セット・ヘッダ/データ列は、列データ・プロパティの精度/長さと標準の区切り文字 (タブ) に基づいて配置されます。

詳細は、“ダイナミック SQL の使用法” の章の ″%Display() メソッド″ を参照してください。

9.5.4 DISPLAYMODE および DISPLAYTRANSLATETABLE の設定

以下の例に示すように、SET DISPLAYMODE を使用して、クエリ・データの表示に使用する形式を指定できます。

```
USER>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]USER>>SET DISPLAYMODE XML

displaymode = xml
[SQL]USER>>
```

DISPLAYMODE の既定値は CURRENTDEVICE です。この既定値では、クエリ・データが TXT 形式でターミナルに表示されます。SET DISPLAYMODE = CUR を指定して、CURRENTDEVICE の既定値をリストアすることができます。

その他の使用可能なオプションは、TXT、HTML、PDF、XML、および CSV です。形式の選択によってファイル・タイプが決まります。InterSystems IRIS は、そのタイプのファイルを作成し、ファイルにクエリ・データを書き込み、可能であれば、そのクエリ・データ・ファイルを表示するための適切なプログラムを起動します。TXT 以外のすべてのオプションで、結果セットのメッセージを記録するために、2 つ目のファイルが作成されます。既定では、SQL シェルにより、それらのファイルが InterSystems IRIS の mgr¥Temp¥ ディレクトリに作成され、ランダムに生成されたファイル名が適切なファイル・タイプの接尾語を伴って割り当てられます。生成されたメッセージ・ファイル名は、追加された文字列 “Messages” を除いて、データ・ファイルと同じ名前になります。HTML、PDF、および XML オプションの場合、Messages ファイルには、クエリ・データ・ファイルと同じファイル・タイプ接尾語が付きます。CSV オプションの場合は、Messages ファイルには TXT ファイル・タイプ接尾語が付きます。

以下は DISPLAYMODE = TXT の場合に生成されたファイルの例です。

```
C:\InterSystems\IRIS\mgr\Temp\sGm7qLdVZn5VbA.txt
C:\InterSystems\IRIS\mgr\Temp\sGm7qLdVZn5VbAMessages.txt
```

クエリを実行するたびに、SQL シェルではランダムに生成されたファイル名を持つファイルの新規ペアが作成されます。

DISPLAYMODE が TXT または CSV の場合は、形式変換の実行時に、適用する変換テーブルの名前を指定することもできます。SET DISPLAYTRANSLATE または SET DISPLAYTRANSLATETABLE のいずれかを指定できます。変換テーブル名の値では、大文字と小文字が区別されます。

DISPLAYMODE が CURRENTDEVICE 以外の値に設定された場合、制御文字を含むクエリ結果セットのデータがあると、警告メッセージが生成されます。一般的に、クエリ結果セットのデータに制御文字が出現するのは、論理モードの場合のみです。例えば、論理モードで表示された場合には、リスト構造のデータに制御文字が含まれます。このため、DISPLAYMODE を CURRENTDEVICE 以外の値に設定する場合は、SELECTMODE も DISPLAY または ODBC に設定することをお勧めします。

9.5.4.1 DISPLAYFILE および DISPLAYPATH の設定

DISPLAYMODE の値が CURRENTDEVICE 以外に設定されていると、DISPLAYFILE および DISPLAYPATH パラメータを使用してターゲット・ファイルの場所を指定することができます。

- DISPLAYFILE の場合、このパラメータを接尾語なしの簡単なファイル名に設定します。例えば、SET DISPLAYFILE = myfile のようになります。また、このパラメータを部分修飾されたパスに設定することもできます。InterSystems IRIS は、これを DISPLAYPATH の値または既定のディレクトリに追加します (必要に応じてサブディレクトリを作成します)。例えば、SET DISPLAYFILE = mydir¥myfile のようになります。DISPLAYPATH が設定されている場合、指定されたディレクトリにこのファイル名でファイルが作成されます。DISPLAYPATH が設定されていない場合、InterSystems IRIS の mgr¥Temp¥ ディレクトリにこのファイル名でファイルが作成されます。
- DISPLAYPATH の場合、このパラメータを既存の完全修飾されたディレクトリ・パス構造に設定します。このディレクトリ・パス構造の末尾には、お使いのオペレーティング システム・プラットフォームに応じてスラッシュ (“/”) または円記号 (“¥”) を付加する必要があります。DISPLAYFILE が設定されている場合、このディレクトリに DISPLAYFILE 名

でファイルが作成されます。DISPLAYFILE が設定されていない場合、このディレクトリにランダムに生成された名前でファイルが作成されます。DISPLAYPATH ディレクトリが存在しない場合、InterSystems IRIS は DISPLAYPATH および DISPLAYFILE 設定を無視し、代わりに、既定のディレクトリと既定のランダム生成ファイル名を使用します。

必要に応じて、DISPLAYPATH 値の末尾にスラッシュ（または円記号）を自動的に付加したり、DISPLAYFILE 値の先頭からスラッシュ（または円記号）を自動的に削除して、有効な完全修飾のディレクトリ・パスが作成されます。

以下の例では、DISPLAYMODE、DISPLAYFILE、および DISPLAYPATH を設定します。

```
[SQL]USER>>SET DISPLAYMODE XML

displaymode = xml
[SQL]USER>>SET DISPLAYFILE = myfile

displayfile = myfile
[SQL]USER>>SET DISPLAYPATH = C:\temp\mydir\

displaypath = C:\temp\mydir\
[SQL]USER>>
```

クエリを実行すると、SQL シェルにより以下のファイルが生成されます。最初のファイルにはクエリ・データが含まれます。2 番目のファイルにはクエリの実行により生じたあらゆるメッセージが含まれます。

```
C:\temp\mydir\myfile.xml
C:\temp\mydir\myfileMessages.xml
```

DISPLAYFILE も DISPLAYPATH も指定しない場合、InterSystems IRIS インストール先の **Mgr¥Temp¥** ディレクトリ（**C:¥InterSystems¥IRIS¥Mgr¥Temp¥** など）にランダム生成ファイル名でファイルが作成されます。

DISPLAYMODE が CURRENTDEVICE に設定されていない場合、DISPLAYFILE を設定してクエリを実行するたびに、命名されたファイルおよび関連メッセージ・ファイルのあらゆる既存データが新規のクエリ・データにより置換されます。DISPLAYFILE 未設定でクエリを実行するたびに、SQL シェルではランダムに生成されたファイル名で新規ファイルが作成され、かつ新規の関連メッセージ・ファイルも作成されます。

DISPLAYMODE が CURRENTDEVICE に設定される場合、DISPLAYFILE および DISPLAYPATH パラメータは影響を受けません。

9.5.5 EXECUTEMODE の設定

SQL シェルでは、SQL 文の即時実行と遅延実行がサポートされます。即時実行では、Enter キーを押すと、指定された SQL 文の作成と実行が行われます。遅延実行では、Enter キーを押したときに文の作成は行われますが、SQL プロンプトで GO を指定するまで実行されません。

使用可能なオプションは、`SET EXECUTEMODE IMMEDIATE`（既定）、`SET EXECUTEMODE DEFERRED`、および現在のモード設定を表示する `SET EXECUTEMODE` です。以下の例は、実行モードを設定します。

```
USER>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]USER>>SET EXECUTEMODE DEFERRED

Executemode = deferred
[SQL]USER>>
```

遅延実行を使用すると、複数の SQL クエリを作成し、名前または番号で呼び出して実行できます。作成済みの SQL 文を実行するには、必要な文を（適切なネームスペースから）呼び出して、GO を指定します。

以下の例は、遅延モードでの 3 つのクエリの作成を示しています。最初の 2 つは保存され、呼び出し名が割り当てられます。3 つ目には名前は割り当てられませんが、番号で呼び出すことができます。

```
[SQL]USER>>SELECT TOP 5 Name,Home_State FROM Sample.Person
1. SELECT TOP 5 Name,Home_State FROM Sample.Person
[SQL]USER>>SAVE 5sample
Query saved as: 5sample
[SQL]USER>>SELECT TOP 5 Name,Home_State FROM Sample.Person ORDER BY Home_State
2. SELECT TOP 5 Name,Home_State FROM Sample.Person ORDER BY Home_State
[SQL]USER>>SAVE 5ordered
Query saved as: 5ordered
[SQL]USER>>SELECT Name,Home_State FROM Sample.Person ORDER BY Home_State
3. SELECT Name,Home_State FROM Sample.Person ORDER BY Home_State
[SQL]USER>>
```

以下は、上記の例で定義された 2 つのクエリを遅延モードで実行する例です。この例では、クエリの 1 つは名前で呼び出され (呼び出し時に SQL シェルで新しい番号が割り当てられます)、もう 1 つは番号で呼び出されます。

```
[SQL]USER>>OPEN 5ordered
SELECT TOP 5 Name,Home_State FROM Sample.Person ORDER BY Home_State
4. SELECT TOP 5 Name,Home_State FROM Sample.Person ORDER BY Home_State
---------------------------------------------
[SQL]USER>>GO

Name               Home_State
Djokovic,Josephine W.  AK
Klingman,Aviel P.      AK
Quine, Sam X.          AK
Xiang,Robert C.        AL
Roentgen,Alexandria Q. AR

5 Row(s) Affected
---------------------------------------------
[SQL]USER>>#3
SELECT Name,Home_State FROM Sample.Person ORDER BY Home_State
3. SELECT Name,Home_State FROM Sample.Person ORDER BY Home_State
---------------------------------------------
[SQL]USER>>GO
.
.
.
```

9.5.6 ECHO の設定

SET ECHO を使用して、クエリ結果を SQL シェルにエコーするかどうかを指定できます。SET ECHO=OFF と指定すると、クエリが作成され、クエリ・キャッシュが定義され、クエリが実行されます。ターミナルにクエリ結果は表示されません。これを以下の例で示しています。

```
[SQL]USER>>set echo=off

echo = off
[SQL]USER>>SELECT Name,Age FROM Sample.MyTest
4.      SELECT Name,Age FROM Sample.MyTest


statement prepare time(s)/globals/cmds/disk: 0.0002s/5/155/0ms
          execute time(s)/globals/cmds/disk: 0.0001s/0/105/0ms
                            cached query class: %sqlcq.USER.cls3
---------------------------------------------------------------------------
[SQL]USER>>
```

SET ECHO=ON (既定) と指定すると、クエリ結果がターミナルに表示されます。これを以下の例で示しています。

```
[SQL]USER>>set echo=on

echo = on
[SQL]USER>>SELECT Name,Age FROM Sample.MyTest
5.      SELECT Name,Age FROM Sample.MyTest

Name    Age
Fred Flintstone 41
Wilma Flintstone        38
Barney Rubble   40
Betty Rubble    42

4 Rows(s) Affected
statement prepare time(s)/globals/cmds/disk: 0.0002s/5/155/0ms
          execute time(s)/globals/cmds/disk: 0.0002s/5/719/0ms
                         cached query class: %sqlcq.USER.cls3
---------------------------------------------------------------------------
[SQL]USER>>
```

SET ECHO は、DISPLAYMODE=CURRENTDEVICE (既定) の場合にのみ有効です。

SET ECHO と SET MESSAGES は、ターミナルに表示されるものを指定します。これらはクエリの作成と実行には影響を与えません。SET MESSAGES=OFF および SET ECHO=OFF の場合、クエリが作成され、クエリ・キャッシュが作成され、クエリ実行によってクエリ結果セットが作成されますが、ターミナルには何も返されません。

9.5.7 MESSAGES の設定

SET MESSAGES を使用して、クエリ・エラー・メッセージを表示するかどうか (成功した場合)、またはクエリ実行情報を表示するかどうか (成功した場合) を指定できます。

- クエリ実行が失敗した場合：SET MESSAGES=OFF と指定している場合、ターミナルには何も表示されません。SET MESSAGES=ON (既定) と指定している場合、クエリ・エラー・メッセージが表示されます。例：`ERROR #5540: SQLCODE: -30 Message: Table 'SAMPLE.NOTABLE' not found`。
- クエリ実行が成功した場合：SET MESSAGES=OFF と指定している場合、クエリ結果と行 `n Rows(s) Affected` のみが表示されます。SET MESSAGES=ON (既定) を指定している場合、クエリ結果と行 `n Rows(s) Affected` に続いて、文作成のメトリック、文実行のメトリック、および生成されたクエリ・キャッシュの名前が表示されます。

  作成および実行のメトリックは、経過時間 (秒の小数部単位)、グローバル参照の合計数、実行されたコマンドの合計数、およびディスク読み取り待ち時間 (ミリ秒単位) で測定されます。

SET MESSAGES=ON の場合に表示される情報が、DISPLAYMODE の設定によって変更されることはありません。一部の DISPLAYMODE オプションでは、クエリ結果セット・ファイルとメッセージ・ファイルの両方が作成されます。このメッセージ・ファイルには、SET MESSAGES=ON の場合にターミナルに表示されるクエリ作成および実行のメッセージではなく、結果セットのメッセージが含まれます。

SET MESSAGES と SET ECHO は、ターミナルに表示されるものを指定します。これらはクエリの作成と実行には影響を与えません。SET MESSAGES=OFF および SET ECHO=OFF の場合、正常なクエリが作成され、クエリ・キャッシュが作成され、クエリ実行によってクエリ結果セットが作成されますが、ターミナルには何も返されません。

9.5.8 LOG の設定

SET LOG を使用して、SQL シェルのアクティビティのログをファイルに記録するかどうかを指定できます。使用可能なオプションは以下のとおりです。

- SET LOG OFF：既定値です。InterSystems IRIS は、現在の SQL シェルのアクティビティのログを記録しません。
- SET LOG ON：InterSystems IRIS は、SQL シェルのアクティビティのログを既定のログ・ファイルに記録します。
- SET LOG pathname：InterSystems IRIS は、SQL シェルのアクティビティのログを、pathname で指定されたファイルに記録します。

SET LOG ON は、IRIS¥mgr¥namespace にログ・ファイルを作成します (namespace は、プロセスの現在のネームスペースの名前を表します)。この既定のログ・ファイル名は xsqlnnnn.log です (nnnn は、現在のプロセスのプロセス ID (pid) 番号を表します)。

既定では、ログ・ファイルは、現在のプロセスおよび現在のネームスペースに固有です。SQL シェルのアクティビティのログを、複数のプロセスまたは複数のネームスペースから同じログに記録するには、プロセスまたはネームスペースごとに SET LOG pathname を指定する際に、同じ pathname を使用します。

ログ・ファイルは、一時停止したり再開したりすることができます。ログ・ファイルを作成したら、SET LOG OFF によって、そのログ・ファイルへの書き込みが一時停止されます。また、SET LOG ON によって、既定のログ・ファイルへの書き込みが再開されます。ログ記録の再開時に、`Log restarted: date time` がログ・ファイルに書き込まれます。SET LOG ON では常に、既定のログ・ファイルが有効になります。したがって、指定した pathname ログ・ファイルへの書き込みを一時停止した場合は、再開時に SET LOG pathname を指定する必要があります。

ログ・ファイルの有効化によって、ターミナルに表示される SQL シェルのアクティビティのコピーが作成されますが、SQL シェルのターミナル出力にはリダイレクトされません。SQL シェルのログには、SQL の実行に失敗した場合は SQL エラーが記録され、SQL の実行に成功した場合はその SQL コードと結果として得られる行数が記録されます。SQL シェルのログには、結果セットのデータは記録されません。

ログが既に有効な場合は、SET LOG ON を指定しても効果はありません。その場合は、SET LOG pathname を指定すると、現在のログが一時停止され、pathname で指定されたログが有効になります。

9.5.9 PATH の設定

SET PATH schema を使用して、スキーマ検索パスを設定できます。SQL はこれを使用して、未修飾テーブル名に適切なスキーマ名を指定します。以下の例で示しているように、schema は、単一のスキーマ名でも、複数のスキーマ名のコンマ区切りリストでもかまいません。

```
[SQL]USER>>SET PATH cinema,sample,user
```

引数なしで SET PATH を使用すると、現在のスキーマ検索パスが削除され、システム全体の既定のスキーマ名に戻ります。

SET PATH schema が指定されていない場合、または指定されたスキーマでテーブルが見つからない場合、SQL シェルではシステム全体の既定のスキーマ名が使用されます。スキーマ検索パスの詳細は、″ObjectScript の使用法″ の “ObjectScript マクロとマクロ・プリプロセッサ” の章で #sqlcompile path マクロを参照してください。

9.5.10 SELECTMODE の設定

SET SELECTMODE を使用して、クエリ・データの表示に使用するモードを指定できます。

```
USER>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]USER>>SET SELECTMODE DISPLAY

selectmode = display
[SQL]USER>>
```

使用可能なオプションは、DISPLAY、LOGICAL、および ODBC です。既定は LOGICAL です。現在のモードを決定するには、値なしで SET SELECTMODE を指定します。

```
[SQL]USER>>SET SELECTMODE

selectmode = logical
[SQL]USER>>
```

%List データは出力不能文字を使用してエンコードされます。そのため、selectmode=logical の場合、SQL シェル は %List データ値を $LISTBUILD 文として表示します。例えば、$lb("White","Green") のようになります。Time データ型は秒の小数部をサポートします。そのため、selectmode=odbc の場合、SQL シェルは秒の小数部を表示します。これは ODBC 標準に対応しません。実際の ODBC TIME データ型では小数部が切り捨てられます。

SelectMode オプションの詳細は、このドキュメントの“InterSystems IRIS SQL の基礎”の章にある“データ表示オプション”を参照してください。

また、SET SELECTMODE を使用して、入力データを表示形式から論理格納形式に変換するか指定します。このデータ変換を行うには、SQL コードが RUNTIME 選択モードを使用してコンパイルされている必要があります実行時には、SET SELECTMODE は LOGICAL (既定) に設定する必要があります。詳細は、″InterSystems SQL リファレンス″ の ″INSERT″ または ″UPDATE″ 文を参照してください。

9.6 SQL のメタデータ、クエリ・プラン、およびパフォーマンス・メトリック

9.6.1 メタデータの表示

SQL シェルでは、現在のクエリに関するメタデータ情報を表示するための、M または METADATA コマンドをサポートしています。

結果セット項目ごとに、このコマンドは次のメタデータをリストします： Column Name (SQL フィールド名)、Type (ODBC データ型整数コード)、Prec (精度 または 最大長)、Scale (最大小数桁数)、Null (ブーリアン: 1=NULL を許可、0=NULL を許可しない)、Label (ヘッダ・ラベル。″列エイリアス″ を参照)、Table (SQL テーブル名)、Schema (スキーマ名)、CType (クライアント・データ型。%SQL.StatementColumn clientType プロパティを参照)。

詳細は、“ダイナミック SQL の使用法”の章の ″select-item のメタデータ″ を参照してください。

InterSystems SQL シェル・コマンドの詳細は、SQL プロンプトで ? を入力するか、″インターシステムズ・クラスリファレンス″ の ″%SYSTEM.SQL.Shell()″ を参照してください。

9.6.2 SHOW STATEMENT

クエリを実行してから、SHOW STATEMENT または SHOW ST を発行することにより、作成済みの SQL 文を表示できます。既定では、クエリを実行する必要があります。executemode=deferred を設定し、クエリを発行してから SHOW STATEMENT SQL シェル・コマンドを発行することで、クエリの実行を回避することができます。

SHOW STATEMENT 情報は、実装クラス (クエリ・キャッシュ名)、引数 (TOP 節および WHERE 節のリテラル値など、実際の引数値のコンマ区切りリスト)、および文のテキスト (SQL コマンドのリテラル・テキスト、大文字/小文字および引数値を含む) で構成されます。

9.6.3 EXPLAIN と SHOW PLAN

SQL クエリのクエリ・プランは、2 つの方法で表示できます。どちらの方法でも、必要に応じて代替クエリ・プランを表示できます。

- EXPLAIN： SELECT クエリの前に EXPLAIN コマンドを置きます。例を以下に示します。

```
SQL]USER>>EXPLAIN SELECT Name FROM Sample.MyTable WHERE Name='Fred Rogers'
```

- SHOW PLAN：クエリを発行してから、SHOW PLAN シェル・コマンドを発行します。例を以下に示します。

```
SQL]USER>>SELECT Name FROM Sample.MyTable WHERE Name='Fred Rogers'
SQL]USER>>SHOW PLAN
```

EXPLAIN SQL コマンドは、クエリを実行せずに、指定された SELECT クエリに関するクエリ・プラン情報を表示します。EXPLAIN ALT は、代替クエリ・プランを表示できます。EXPLAIN STAT は、パフォーマンス統計とクエリ・プランを返します。EXPLAIN は、SELECT クエリのクエリ・プランを返すためにのみ使用できます。クエリ操作を実行する INSERT、UPDATE、DELETE 文など、他のコマンドのクエリ・プランは返しません。

SHOW PLAN SQL シェル・コマンドを使用すると、SQL シェルによって正常に発行された最後のクエリのクエリ・プラン情報を表示できます。SHOW PLAN は、SELECT、INSERT、UPDATE、DELETE を含め、クエリ操作を実行する任意の SQL コマンドに使用できます。既定では、クエリを実行する必要があります。executemode=deferred を設定し、クエリを発行してから、次のいずれかの SQL シェル・コマンドを発行することで、クエリの実行を回避することができます。

- SHOW PLAN、SHOW PL（または、単なる SHOW）は、現在のクエリに関するクエリ・プラン情報を表示します。クエリ・プランは、クエリのデバッグおよびパフォーマンスの最適化に使用できます。クエリに対するインデックスやコスト値の使用など、クエリの実行方法を指定します。クエリ・プランは、SELECT、DECLARE、非カーソルの UPDATE や DELETE、および INSERT...SELECT の文で返すことができます。このコマンドには、V（VERBOSE）オプションがあります。
- SHOW PLANALT を使用して、現在のクエリの代替表示プランを表示します。このコマンドには、V（VERBOSE）オプションがあります。詳細は、"SQL 最適化ガイド" の "代替表示プラン" を参照してください。

InterSystems SQL シェル・コマンドの詳細は、SQL プロンプトで？を入力するか、"インターシステムズ・クラス・リファレンス" の "%SYSTEM.SQL.Shell()" を参照してください。

$SYSTEM.SQL.Explain() メソッドを使用して、ObjectScript からクエリ・プランを生成できます。詳細は、"SQL 最適化ガイド" の "クエリ実行プラン" を参照してください。

クエリ・プランの解釈の詳細は、"SQL 最適化ガイド" の "SQL クエリ・プランの解釈" を参照してください。

9.6.4 SQL シェルのパフォーマンス

SQL 文が正常に実行された後に、SQL シェルには 4 つの文準備値（times(s)/globals/cmds/disk）と 4 つの文実行値（times(s)/globals/cmds/disk）が表示されます。

- statement **prepare time**（文の準備時間）とは、ダイナミック文の準備にかかった時間です。これには、文の生成とコンパイルにかかった時間が含まれます。また、文のキャッシュで文を検索するためにかかった時間も含まれます。したがって、文が実行されてから、番号または名前で呼び出された場合、呼び出された文にかかった準備時間はほぼゼロになります。また、文が準備され実行されてから、GO コマンドの発行によって再実行された場合、再実行にかかった準備時間はゼロになります。
- elapsed **execute time**（経過した実行時間）とは、%Execute() の呼び出しを起点として、%Display() から応答が返されるまでに経過した時間です。パラメータ値が入力されるまでの待機時間は含まれません。

文の **globals** はグローバル参照の数、**cmds** は実行された SQL コマンドの数、**disk** はディスク待ち時間（ミリ秒単位）です。SQL シェルでは、準備操作と実行操作について別々の数が保持されます。

これらのパフォーマンス値が表示されるのは、DISPLAYMODE が currentdevice に設定され、MESSAGES が ON に設定されている場合のみです。これらは、SQL シェルの既定の設定です。

9.7 Transact-SQL のサポート

既定で、SQL シェルは InterSystems SQL コードを実行します。ただし、SQL シェルは Sybase コードまたは MSSQL コードの実行に使用できます。

9.7.1 DIALECT の設定

既定で、SQL シェルはコードを InterSystems SQL として解析します。SET DIALECT を使用して、Sybase コードまたは MSSQL コードを実行するように SQL シェルを構成することができます。現在の言語を変更するには、SET DIALECT で Sybase、MSSQL、または IRIS に設定します。既定値は Dialect=IRIS です。これらの SET DIALECT オプションでは、大文字と小文字は区別されません。

以下は、SQL シェルから MSSQL プログラムを実行する例です。

```
USER>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]USER>>SET DIALECT MSSQL

dialect = MSSQL
[SQL]USER>>SELECT TOP 5 name + '-' + ssn FROM Sample.Person
1.      SELECT TOP 5 name + '-' + ssn FROM Sample.Person

Expression_1
Zweifelhofer,Maria H.-559-20-7648
Vonnegut,Bill A.-552-41-2071
Clinton,Terry E.-757-30-8013
Bachman,Peter U.-775-59-3756
Avery,Emily N.-833-18-9563

5 Rows(s) Affected
statement prepare time: 0.2894s, elapsed execute time: 0.0467s.
---------------------------------------------------------------------------
[SQL]USER>>
```

Sybase 言語および MSSQL 言語が言語内でサポートする SQL 文は限定されています。サポートされる文は、SELECT、INSERT、UPDATE、および DELETE です。また、CREATE TABLE 文は、永続的なテーブルに対してはサポートされますが、一時テーブルに対してはサポートされません。CREATE VIEW はサポートされます。CREATE TRIGGER と DROP TRIGGER もサポートされます。ただし、この実装では、CREATE TRIGGER 文が部分的に成功してもクラス・コンパイルに対しては失敗する場合には、トランザクション・ロールバックはサポートされません。CREATE PROCEDURE と CREATE FUNCTION はサポートされます。

9.7.2 COMMANDPREFIX の設定

SET COMMANDPREFIX を使用して、後続の SQL シェル・コマンドに付加する必要のある接頭語 (通常は 1 文字) を指定することができます。この接頭語は、SQL シェル・プロンプトから発行された SQL 文には使用されません。この接頭語の目的は、SQL シェル・コマンドと SQL コード文をはっきりと区別することにあります。例えば、SET は SQL シェル・コマンドであり、Sybase および MSSQL における SQL コード文でもあります。

既定では、コマンド接頭語はありません。コマンド接頭語を設定するには、SET COMMANDPREFIX=prefix を使用します (prefix は引用符なしで指定してください)。また、コマンド接頭語がない状態に戻すには、SET COMMANDPREFIX="" を使用します。以下の例は、設定、使用、および元に戻されるコマンド接頭語の / (スラッシュ文字) の例を示しています。

```
USER>DO $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------------

The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
```

```
[SQL]USER>>SET COMMANDPREFIX=/

commandprefix = /
[SQL]USER>>/SET LOG=ON

log = xsql4148.log
[SQL]USER>>  << entering multiline statement mode >>
        1>>SELECT TOP 3 Name,Age
        2>>FROM Sample.Person
        3>>/GO
9.      SELECT TOP 3 Name,Age
        FROM Sample.Person

Name    Age
Frith,Jose M.   13
Finn,William D. 15
Ximines,Uma Y.  44

3 Rows(s) Affected
statement prepare time: 0.0010s, elapsed execute time: 0.0014s.
---------------------------------------------------------------------------
[SQL]USER>>/SET COMMANDPREFIX

commandprefix = /
[SQL]USER>>/SET COMMANDPREFIX=""

commandprefix = ""
[SQL]USER>>SET COMMANDPREFIX

commandprefix =
[SQL]USER>>
```

コマンド接頭語を設定する場合は、すべての SQL シェル・コマンドに使用する必要がありますが、?、#、および GO は例外です。これら 3 つの SQL シェル・コマンドは、コマンド接頭語があってもなくても発行することができます。

SQL シェルでは、SET コマンドまたは SET COMMANDPREFIX コマンドを発行すると、SQL シェルの初期化の一部として、また ? コマンドのオプション表示の最後に、現在のコマンド接頭語が表示されます。

9.7.3 RUN コマンド

SQL シェルの RUN コマンドは、SQL スクリプト・ファイルを実行します。RUN コマンドを発行する前に SET DIALECT を使用して、IRIS (InterSystems SQL)、Sybase (Sybase TSQL)、MSSQL (Microsoft SQL) のいずれかを指定する必要があります。既定の言語は IRIS です。`RUN scriptname` を呼び出すことも、単に `RUN` を呼び出してスクリプト・ファイル名の入力を求められるようにすることもできます。

RUN は、スクリプト・ファイルをロードし、ファイルに含まれるそれぞれの文を作成して実行します。スクリプト・ファイル内の文は、通常 GO 行またはセミコロン (;) で区切る必要があります。RUN コマンドによって、区切り文字を指定するように求められます。

SQL スクリプト・ファイルの結果は、現在のデバイスに表示されます。また、オプションでログ・ファイルに記録することもできます。必要に応じて、作成に失敗した文が格納されるファイルを生成することもできます。

RUN コマンドは、以下の例に示すように、これらのオプションを指定するように求めるプロンプトを返します。

```
[SQL]USER>>SET DIALECT=Sybase

dialect = Sybase
[SQL]USER>>RUN

Enter the name of the SQL script file to run: SybaseTest

Enter the file name that will contain a log of statements, results and errors (.log): SyTest.log
     SyTest.log

Many script files contain statements not supported by IRIS SQL.
Would you like to log the statements not supported to a file so they
can be dealt with manually, if applicable?   Y=> y
Enter the file name in which to record non-supported statements (_Unsupported.log): SyTest_Unsupported.log


Please enter the end-of-statement delimiter (Default is 'GO'):  GO=>

Pause how many seconds after error?   5 => 3
```

```
Sybase Conversion Utility (v3)
Reading source from file:
Statements, results and messages will be logged to: SyTest.log
.
.
.
```

9.7.4 TSQL の例

以下の SQL シェルの例は、AvgAge という Sybase プロシージャを作成します。そのプロシージャが、Sybase EXEC コマンドを使用して実行されます。その後、言語が InterSystems IRIS に変更され、InterSystems SQL の CALL コマンドを使用して同じプロシージャが実行されます。

```
[SQL]USER>>SET DIALECT Sybase

dialect = Sybase
[SQL]USER>>  << entering multiline statement mode >>
        1>>CREATE PROCEDURE AvgAge
        2>>AS SELECT AVG(Age) FROM Sample.Person
        3>>GO
12.     CREATE PROCEDURE AvgAge
        AS SELECT AVG(Age) FROM Sample.Person


statement prepare time: 0.1114s, elapsed execute time: 0.4364s.
---------------------------------------------------------------------------
[SQL]USER>>EXEC AvgAge
13.     EXEC AvgAge



Dumping result #1
Aggregate_1
44.35

1 Rows(s) Affected
statement prepare time: 0.0956s, elapsed execute time: 1.1761s.
---------------------------------------------------------------------------

[SQL]USER>>SET DIALECT=IRIS

dialect = IRIS
[SQL]USER>>CALL AvgAge()
14.     CALL AvgAge()



Dumping result #1
Aggregate_1
44.35

1 Rows(s) Affected
statement prepare time: 0.0418s, elapsed execute time: 0.0040s.
---------------------------------------------------------------------------
[SQL]USER>>
```

10

管理ポータルの SQL インタフェースの使用法

この章では、InterSystems IRIS® データ・プラットフォームの管理ポータルから SQL 操作を実行する方法について説明します。管理ポータル・インタフェースはダイナミック SQL を使用するので、実行時にクエリが作成および実行されます。管理ポータル・インタフェースの目的は、小規模なデータ・セットの SQL コードの開発とテストを支援することです。プロダクション環境で SQL を実行するためのインタフェースを目的としたものではありません。

また、管理ポータルには、SQL を構成するための各種オプションも用意されています。詳細は、"システム管理ガイド" にリストされている SQL およびオブジェクトの設定ページを参照してください。

管理ポータルの使用に関する一般的な情報については、左上にある [**ヘルプ**] ボタンを選択します。右上隅にある [**担当者**] ボタンを使用すると、管理ポータルから InterSystems ソフトウェアに関する問題をインターシステムズのサポート窓口 (WRC) に報告できます。SQL パフォーマンスに関する問題を WRC に報告するには、この章の "ツール" のセクションを参照してください。

10.1 管理ポータルの SQL 機能

InterSystems IRIS では、InterSystems IRIS 管理ポータルから SQL ツールを使用してデータを検証および操作できます。これを始める起点は、管理ポータルの [**システム・エクスプローラ**] オプションです。ここから、[**SQL**] オプションを選択します。ここには SQL インタフェースが表示され、以下のことができます。

- SQL クエリ実行 – SQL コマンドを作成して実行します。既存のテーブルおよびデータに対して SQL クエリを実行し、テーブルを作成したり、テーブル・データを挿入、更新、または削除できます。テキスト・ボックスに SQL コード (SELECT、INSERT、UPDATE、DELETE、CREATE TABLE、およびその他の SQL 文) を直接記述したり、テキスト・ボックスに SQL 履歴から文を取得したり、テキスト・ボックスにテーブルをドラッグ・アンド・ドロップしてクエリ (SELECT 文) を生成したり、クエリ・ビルダ・インタフェースを使用してクエリ (SELECT 文) を構成したりすることができます。
- スキーマ・コンテンツのフィルタ処理 – 画面の左側に、現在のネームスペースに対する SQL スキーマ、またはこれらのスキーマのフィルタ処理済みサブセットを、各スキーマのテーブル、ビュー、プロシージャ、およびクエリ・キャッシュと共に表示します。個々のテーブル、ビュー、プロシージャ、またはクエリ・キャッシュを選択して、その [カタログの詳細] を表示できます。
- ウィザード – ウィザードを実行し、データのインポート、データのエクスポート、またはデータの移行を実行します。ウィザードを実行して、テーブルまたはビューにリンク、またはストアド・プロシージャにリンクします。
- アクション – ビューの定義、テーブル定義の詳細の出力、テーブルのチューニングの実行やインデックスの再構築によるクエリのパフォーマンスの向上、不要なクエリ・キャッシュの削除や不要なテーブル、ビュー、プロシージャ定義の削除によるクリーン・アップを実行します。
- テーブルを開く – 表示モードで、テーブル内の現在のデータを表示します。これは通常、テーブル内の完全なデータではありません。列内のレコードの数とデータの長さの両方が、表示を管理しやすくするために制限されます。

- ツール ー [SQL 実行時統計]、[インデックス分析]、[代替表示プラン]、[レポート生成]、および [レポートのインポート] のうちのいずれかのツールを実行します。
- ドキュメント ー SQL エラーコードのリストおよび SQL 予約語のリストを表示できます。テーブルを選択すると、クラス・ドキュメント (テーブルのクラス・リファレンス・ページ) を表示できます。

10.1.1 ネームスペースの選択

すべての SQL 操作は、特定のネームスペース内で行われます。したがって、最初に SQL インタフェース・ページの上部にある **[切り替え]** オプションをクリックして、使用するネームスペースを指定する必要があります。これにより、利用可能なネームスペースのリストが表示され、その中から選択できます。

管理ポータルの既定のネームスペースを設定できます。管理ポータルで、**[システム管理]**、**[セキュリティ]**、**[ユーザ]** の順に選択します。目的のユーザの名前をクリックします。これにより、ユーザ定義を編集できるようになります。[一般] タブで、ドロップダウン・リストから **[開始ネームスペース]** を選択します。**[保存]** をクリックします。開始ネームスペースが選択されていない場合は、%SYS が既定で使用されます。

10.1.2 ユーザ・カスタマイズ

管理ポータルの多くの SQL 操作は、ユーザごとに自動的にカスタマイズされます。[クエリ実行] タブまたは [SQL 文] タブでフィルタ、最大、モードなどのオプションを設定した場合、このユーザ指定値は将来使用するために保持されます。同じユーザが管理ポータルを有効化すると、そのユーザの前の設定が表示されます。InterSystems IRIS を再起動すると、すべてのオプションが既定値に戻ります。

ネームスペースの選択はカスタマイズされません。ユーザ定義の実行開始ネームスペースに戻ります。

フィルタ・オプションの使用の詳細は、"スキーマ・コンテンツのフィルタ処理" を参照してください。

10.2 SQL クエリの実行

管理ポータルで、**[システム・エクスプローラ]**、**[SQL]** の順に選択します。ページ上部の **[切り替え]** オプションを使ってネームスペースを選択します。利用可能なネームスペースのリストが表示されます。SQL クエリを実行するには、以下の 3 つのオプションがあります。

- [クエリ実行] : SQL コマンドを作成して実行します。SQL コマンドには、SELECT クエリ、あるいは InterSystems SQL の DDL 文または DML 文を入力できます。文は、実行時に InterSystems IRIS サーバで検証されます。
- [履歴を表示] : 前に実行した SQL 文を呼び出し、それを再実行するか、編集してから実行します。正常に実行されなかった文を含め、実行されたすべての文がリストされます。
- **[クエリ・ビルダ]** : SQL クエリ・ビルダを起動します (これは、SELECT 文を作成することのみを目的としています)。SQL クエリ・ビルダ内で、テーブル、列、WHERE 節の述語、および他のクエリ・コンポーネントを選択して、SQL SELECT クエリを作成します。その後、**[クエリ実行]** をクリックしてクエリを実行できます。

10.2.1 SQL 文の作成

[クエリ実行] テキスト・ボックスにより、SELECT クエリや CALL クエリを記述できますが、それだけでなく、CREATE TABLE などの DDL 文や、INSERT、UPDATE、DELETE などの DML 文を含めて、大半の SQL 文も記述できます。

以下を使用して、**[クエリ実行]** テキスト・ボックスで SQL コードを指定できます。

- テキスト・ボックスに SQL コードを入力 (または貼り付け) ます。SQL コードの領域では、SQL テキストはカラー表示されず、構文や存在の検証も行われません。ただし、自動スペル検証は行われます。[X] アイコンを使用して、テキスト・ボックスの内容を消去できます。
- [履歴を表示] リストを使用して、前の SQL 文を選択します。選択した文がテキスト・ボックスにコピーされます。文を実行すると、この文は [履歴を表示] リストの一番上に移動します。[履歴を表示] には、実行に失敗した文も含め、以前に実行されたすべての文がリストされます。
- テーブルのドラッグ・アンド・ドロップを使用して、テキスト・ボックスで SQL コードを作成します。
- **[クエリ実行]** テキスト・ボックスではなく **[クエリ・ビルダ]** を使用して、SELECT クエリを指定および実行できます。**[クエリ・ビルダ]** を使用して実行された SELECT クエリは、**[クエリ実行]** には表示されず、**[履歴を表示]** にはリストされません。

[クエリ実行] テキスト・ボックス内の SQL コードには、以下を含めることができます。

- ? 入力パラメータ。`TOP ?` や `WHERE Age BETWEEN ? AND ?` のような入力パラメータを指定すると、**[実行]** ボタンに **[クエリのパラメータ値を入力してください]** ウィンドウが表示されます。このウィンドウには、クエリに指定した順序でパラメータごとの入力フィールドが表示されます。? 入力パラメータの詳細は、このドキュメントの“ダイナミック SQL の使用法”の章の ″SQL 文の実行″ を参照してください。
- 空白文字。複数の空白スペース、1 つまたは複数の改行を指定できます。Tab キーは無効にされます。コードを SQL コードの領域にコピーすると、既存のタブは単一の空白スペースに変換されます。改行および複数の空白スペースは保持されません。
- コメント。SQL コードの領域では、1 行および複数行のコメントがサポートされています。コメントは保持され、[履歴を表示] 表示に示されます。コメントは [プラン表示] の [文テキスト] 表示またはクエリ・キャッシュには表示されません。
- 複数の結果セットを返すクエリ。

テキスト・ボックスで SQL コードを作成したら、**[プラン表示]** ボタンをクリックすれば、SQL コードを実行せずに SQL コードをチェックすることができます。コードが有効な場合、**[プラン表示]** にクエリ・プランが表示されます。コードが無効な場合、**[プラン表示]** に SQLCODE エラー値とメッセージが表示されます。**[プラン表示]** ボタンを使用して、最後に実行した SQL コードについても、この情報を表示できます。

SQL コードを実行するには、**[実行]** ボタンをクリックします。

10.2.1.1 テーブルのドラッグ・アンド・ドロップ

画面の左側にある [テーブル] リスト (または [ビュー] リスト) からテーブル (またはビュー) をドラッグし、それを **[クエリ実行]** テキスト・ボックスにドロップして、クエリを生成できます。これにより、テーブル内の非表示でないすべてのフィールドの select-item リストが指定された SELECT、およびテーブルを指定する FROM 節が生成されます。その後、さらにこのクエリを変更し、**[実行]** ボタンを使用して実行できます。

画面の左側にある [プロシージャ] リストから、プロシージャ名をドラッグ・アンド・ドロップすることもできます。

10.2.2 クエリ・オプションの実行

SQL 実行インタフェースには、以下のオプションがあります。

- SELECT の [選択モード] ドロップダウン・リストでは、クエリでデータ値の提供 (WHERE 節内など) とクエリ結果セットへのデータ値の表示に使用される形式を指定します。可能なオプションは、**[表示モード]** (既定)、**[ODBC モード]**、および **[論理モード]** です。これらのオプションの詳細は、このドキュメントの“InterSystems IRIS の基礎”の章にある“データ表示オプション”を参照してください。

  INSERT または UPDATE の [選択モード] ドロップダウン・リストでは、入力データを表示形式から論理格納形式に変換するかどうかを指定できます。このデータ変換を行うには、SQL コードが RUNTIME 選択モードを使用してコン

パイルされている必要があります実行時には、[選択モード]ドロップダウン・リストは **[論理モード]** に設定する必要があります。詳細は、"InterSystems SQL リファレンス" の "INSERT" または "UPDATE" 文を参照してください。

[選択モード] は、InterSystems IRIS 日時や ObjectScript %List 構造データなど、論理格納形式が目的の表示形式 (表示または ODBC) とは異なるデータ型に対して特に有用です。

- **[最大]** フィールドでは、クエリから返すデータの最大行数を制限できます。0 を含む任意の正の整数に設定できます。**[最大]** を設定すると、明示的に変更するまで、セッションの期間中はその値がすべてのクエリに使用されます。既定値は 1000 です。最大値は 100,000 であり、値を入力していない (**[最大]** を NULL に設定した) 場合、100,000 より大きな値を入力した場合、または数値以外の値を入力した場合は、これが既定値となります。TOP 節を使用することで、返されるデータの行数を制限することもできます。**[最大]** は、DELETE など、他の SQL 文には影響を与えません。

[詳細] オプションをクリックすると、SQL 実行インタフェースには以下の追加オプションが表示されます。

- **[言語]**：SQL コードの言語。選択可能な値は、IRIS、Sybase、および MSSQL です。既定値は IRIS です。Sybase と MSSQL については、"Transact-SQL (TSQL) 移行ガイド" を参照してください。選択した言語が、管理ポータルに次回アクセスするときのユーザ・カスタマイズされた既定値になります。

  [DIALECT] が Sybase または MSSQL である場合、**[クエリ実行]** テキスト・ボックスには複数の SQL コマンドを指定できます。すべての挿入、削除、および更新コマンドが最初に実行された後、指定された順に SELECT コマンドが実行されます。複数の結果セットは別のタブで返されます。

- **[行番号]**：結果セットの表示に各行の行番号を含めるかどうかを指定するチェック・ボックス。行番号は、結果セット内の各行に連続的に割り当てられる整数です。これは返された行の単純な番号であり、RowID や %VID とは一致しません。行番号列の見出し名は # です。既定では、行番号は表示されません。

- **[フォアグラウンドでクエリを実行]**：フォアグラウンドでクエリを実行するかどうかを指定するチェック・ボックスです。簡単なクエリは、バックグラウンドよりもフォアグラウンドで実行する方がはるかに高速であることが普通です。一方、長時間を要するクエリをフォアグラウンドで実行すると、その実行中に管理ポータルが応答しなくなることがあります。既定では、バックグラウンドですべてのクエリが実行されます。

これらのオプションはすべて、ユーザ・カスタマイズされます。

10.2.3 [プラン表示] ボタン

[プラン表示] ボタンをクリックすると、ページのテキスト・ボックスに表示されているクエリの現在のクエリ・プランの相対コスト (オーバーヘッド) を含む文テキストとクエリ・プランが表示されます。[プラン表示] は、**[クエリ実行]** または **[履歴を表示]** インタフェースのいずれかから起動できます。クエリ・プランは、クエリが作成 (コンパイル) されたとき、つまり、クエリを記述して **[プラン表示]** ボタンを選択したときに生成されます。クエリ・プランを表示するために、クエリを実行する必要はありません。**[プラン表示]** では、無効なクエリで起動すると SQLCODE とエラー・メッセージが表示されます。

10.2.4 SQL 文の結果

[クエリ実行] テキスト・ボックスで SQL コードを記述したら、**[実行]** ボタンをクリックしてそのコードを実行できます。SQL 文が正常に実行されると、結果がコード・ウィンドウの下に表示されます。SQL コードが失敗すると、エラー・メッセージがコード・ウィンドウの下に (赤色で) 表示されます。**[プラン表示]** ボタンを押すと、SQLCODE エラーとエラー・メッセージが表示されます。

[クエリ実行] では、SQL コード実行はバックグラウンド・プロセスとして実行されます。コードの実行中、**[実行]** ボタンは **[キャンセル]** ボタンに置き換わります。これにより、長時間実行されているクエリの実行をキャンセルできます。

10.2.4.1 クエリ・データの表示

結果セットはテーブルとして返されます。その際、[行番号] ボックスにチェックが付けられている場合は、最初の列 (#) として行カウンタが表示されます。残りの列は、指定した順に表示されます。RowID (ID フィールド) は、表示される場合も

非表示の場合もあります。各列は列名（または、指定した場合は列のエイリアス）で識別されます。集計、式、サブクエリ、ホスト変数、またはリテラルの SELECT 項目は、列エイリアスによって（指定されている場合）、または Aggregate_、Expression_、Subquery_、HostVar_、または Literal_ という文字列のすぐ後に SELECT 項目のシーケンス番号（既定では）が続く文字列によって識別されます。

行または列にデータがない（NULL）場合は、結果セットに空白のテーブル・セルが表示されます。空の文字列リテラルを指定すると、HostVar_ フィールドが空白のテーブル・セルと共に表示されます。NULL を指定すると、Literal_ フィールドが空白のテーブル・セルと共に表示されます。

選択したフィールドが日付、時刻、タイムスタンプ、または %List エンコードされたフィールドの場合、表示される値は表示モードによって異なります。

以下の表示機能は、管理ポータルの SQL インタフェース [**クエリ実行**] の結果表示および [**テーブルを開く**] のデータ表示に固有です。

- データ型が %Stream.GlobalCharacter のストリーム・フィールドには、実際のデータ（最大 100 文字）が文字列として表示されます。ストリーム・フィールド内のデータが 100 文字より多い場合は、データの最初の 100 文字の後に、他にもデータがあることを示す省略記号（...）が表示されます。
- データ型が %Stream.GlobalBinary のストリーム・フィールドは <binary> として表示されます。
- 文字列データ・フィールドでは、必要に応じて行を折り返して、実際のデータをすべて表示します。
- 整数フィールドは、結果テーブル・セル内で右寄せされます。RowID、数値、および他のすべてのフィールドは左寄せされます。

同じクエリをダイナミック SQL コード、SQL シェル、または埋め込み SQL コードを使用して実行した場合、これらの結果表示機能は使用されません。

指定したクエリが 1 つ以上の結果セットを返した場合、[**クエリ実行**] ではこれらの結果セットが [**結果 #1**]、[**結果 #2**] などの名前のタブとして表示されます。

10.2.4.2 クエリ実行メトリック

成功した場合、[**クエリ実行**] には、パフォーマンス情報とクエリ・キャッシュのルーチン名が表示されます。表示する結果データがある場合は、パフォーマンス情報の下に表示されます。実行情報には、[ ]、[ ]、クエリ・キャッシュ名を示す [ ]、およびクエリを最後に実行した時間のタイムスタンプを示す [ ] が含まれます。

- [ ]：操作が成功した場合、CREATE TABLE などの DDL 文では 0 が表示されます。INSERT、UPDATE、DELETE などの DML 文では、影響を受ける行の数が表示されます。

  SELECT では、結果セットとして返される行の数が表示されます。返される行の数は [**最大**] の設定によって制御されます。これは、選択可能だった行数よりも少ない場合があります。複数の結果セットの場合は、結果セットごとの行数が / 文字で区切られてリストされます。1 つ以上の集約関数を指定し、フィールドを選択しないクエリでは、FROM 節のテーブルに行が含まれない場合でも、常に 1 が表示され、式、サブクエリ、集約関数の結果が返されます。集約関数を指定せず、行を選択しないクエリでは、FROM 節のテーブルを参照しない式とサブクエリのみをクエリで指定した場合でも、常に 0 が表示され、結果は返されません。FROM 節のないクエリでは、常に 1 が表示され、式、サブクエリ、集約関数の結果が返されます。

- [ ]：経過時間（秒の小数部単位）、グローバル参照の合計数、実行されたコマンドの合計数、およびディスク読み取り待ち時間（ミリ秒単位）で測定されます。クエリにクエリ・キャッシュがある場合、これらのパフォーマンス・メトリックはクエリ・キャッシュの実行に関するものになります。このため、クエリの初回実行のパフォーマンス・メトリックは、以降の実行よりもかなり高くなります。指定したクエリが複数の結果セットを返す場合、これらのパフォーマンス・メトリックはすべてのクエリの合計値です。

  これらのパフォーマンス・メトリックを詳しく分析するには、MONLBL（行単位の監視ユーティリティ）を実行し、アスタリスク・ワイルドカードを %sqlcq* のように使用して、ルーチン名を指定します。"^%SYS.MONLBL を使用したルーチン・パフォーマンスの検証" を参照してください。

- [ ]：自動的に生成されたクエリ・キャッシュ・クラス名。例えば、`%sqlcq.USER.cls2` では、USER ネームスペースの 2 番目のクエリ・キャッシュを示しています。各新規クエリには、次に続く整数を伴った新規クエリ・キャッシュ名が割り当てられます。このクエリ・キャッシュ名をクリックして、クエリ・キャッシュに関する情報を表示できます。さらに、その [表示プラン] を表示したり、クエリ・キャッシュを実行したりするためのリンクも表示されます。(DDL 文は、"キャッシュされない SQL コマンド" を参照してください。)

  管理ポータルを閉じたり、InterSystems IRIS を停止しても、クエリ・キャッシュが削除されたり、クエリ・キャッシュの番号の割り付けがリセットされたりすることはありません。現在のネームスペースからクエリ・キャッシュを削除するには、%SYSTEM.SQL.Purge() メソッドを呼び出します。

  すべての SQL 文でクエリ・キャッシュが生成されるわけではありません。リテラル置換値 (TOP 節の値および述語のリテラルなど) を除いて、既存のクエリ・キャッシュと同じクエリでは、新しいクエリ・キャッシュは作成されません。DDL 文や特権割り当て文など、一部の SQL 文はキャッシュされません。CREATE TABLE など、クエリ以外のSQL 文ではクエリ・キャッシュ名も表示されます。ただし、このクエリ・キャッシュ名は作成された後すぐに削除され、同じクエリ・キャッシュ名が次の SQL 文 (クエリまたはクエリ以外) によって再使用されます。

- 凍結状態：クエリ・プランが凍結されている場合、クエリ・キャッシュ・クラス名の後に凍結プランの状態が (括弧で囲んで) 表示されます。例：`(Frozen/Explicit)`。クエリ・プランが凍結されていない場合、ここには何も表示されません。

- [ ]：**[クエリ実行]** (または他の SQL 操作) が最後に実行された日時。このタイムスタンプは、同じクエリが繰り返し実行される場合でも、クエリが実行されるたびにリセットされます。

- また、実行に成功すると、**[印刷]** リンクが表示されます。リンクをクリックすると [クエリ印刷] ウィンドウが表示され、クエリ・テキストおよびクエリ結果セットを印刷するかファイルにエクスポートするかを選択できます。クリック可能な **[クエリ]** と **[結果]** を切り替えて、クエリ・テキストまたはクエリ結果セットを表示または非表示にすることができます。表示されるクエリ結果セットには、ネームスペース名、結果セット・データおよび行数、タイムスタンプ、クエリ・キャッシュ名が含まれます (タイムスタンプは、[クエリ印刷] ウィンドウが開かれた時間で、クエリが実行された時間ではありません)。[クエリ印刷] ウィンドウの **[印刷]** ボタンをクリックすると、[クエリ印刷] ウィンドウのスクリーンショットが印刷されます。[ファイルにエクスポート] チェック・ボックスには、エクスポート・ファイルの形式 (xml、hdml、pdf、txt、csv) およびエクスポート・ファイルのパス名を指定するオプションが表示されます。**[エクスポート]** オプションでは **[クエリ]** と **[結果]** の切り替えは無視され、常に結果セット・データ (既定では `exportQuery.pdf`) と行数 (既定では `exportQueryMessages.pdf`) のみがエクスポートされます。クエリ・テキスト、ネームスペース、タイムスタンプ、クエリ・キャッシュ名は含まれません。

失敗した場合は、**[クエリ実行]** にエラー・メッセージが表示されます。[プラン表示] ボタンをクリックすると、対応する SQLCODE エラーの値およびメッセージを表示できます。

10.2.5 履歴の表示

現在のセッション中に実行された既存の SQL 文をリストするには、**[履歴を表示]** をクリックします。[履歴を表示] には、このインタフェースから呼び出したすべての SQL 文 (正常に実行された文と、実行に失敗した文の両方) がリストされます。既定では、SQL 文は実行時間順にリストされ、最後に実行した文がリストの先頭に表示されます。任意の列見出しをクリックして、SQL 文を列の値別に昇順または降順で並べ替えることができます。**[履歴を表示]** リストから SQL 文を実行すると、その [実行時間] (ローカルの日付とタイムスタンプ) が更新され、その [カウント] (実行された回数) がインクリメントされます。

[履歴を表示] リストをフィルタ処理するには、**[フィルタ]** ボックスに文字列を指定してから Tab キーを押します。文字列を含む履歴項目のみが、更新されたリストに含まれます。フィルタ文字列は、[SQL文] 列 (テーブル名など) または [実行時間] 列 (データなど) で見つかった文字列のいずれかとなります。フィルタ文字列では、大文字と小文字は区別されません。フィルタ文字列は、明示的に変更するまで有効です。

[履歴を表示] から SQL 文を選択してその文を変更および実行できます。これにより、その文が **[クエリ実行]** テキスト・ボックスに表示されるようになります。**[クエリ実行]** で、SQL コードを変更してから **[実行]** をクリックします。**[履歴を表示]** から取得した SQL 文を変更すると、その文は新規文として **[履歴を表示]** に保存されます。これには、大文字と小文字、

空白、またはコメントの変更など、実行に影響しない変更が含まれます。空白は **[履歴を表示]** には表示されませんが、SQL 文が **[履歴を表示]** から取得されるときには保持されます。

未変更の SQL 文は **[履歴を表示]** リストから直接実行（再実行）できます。これを行うには、[履歴を表示] リスト内のその SQL 文の右にある **[実行]** ボタンをクリックします。

[プラン]、**[印刷]**、**[削除]** のボタンを選択することもできます。**[印刷]** ボタンをクリックすると [クエリ印刷] ウィンドウが表示され、クエリ・テキストおよびクエリ結果セットを印刷するかファイルにエクスポートするかを選択できます。**[削除]** ボタンは、履歴から SQL 文を削除します。**[履歴を表示]** リストの最後にある **[すべて削除]** ボタンは、履歴内のすべての SQL 文を削除します。

[履歴を表示] リストは、クエリ・キャッシュのリストと異なる点に注意してください。**[履歴を表示]** は、実行中に失敗したものも含めて現在のセッションから呼び出された SQL 文をリストします。

10.2.6 その他の SQL インタフェース

InterSystems IRIS は、このドキュメントの他の章で説明するように、SQL コードを記述し実行する、他の多くの方法をサポートしています。これには、以下のものがあります。

- 埋め込み SQL：ObjectScript コードに埋め込んだ SQL コードです。
- ダイナミック SQL：**%SQL.Statement** クラス・メソッド（または他の結果セット用クラス・メソッド）を使用して、ObjectScript コードから SQL 文を実行します。
- SQL シェル：SQL シェル・インタフェースを使用して、ターミナルからダイナミック SQLを実行します。

10.3 スキーマ・コンテンツのフィルタ処理

管理ポータルの左側にある SQL インタフェースでは、スキーマ（またはフィルタ・パターンに一致する複数のスキーマ）のコンテンツを表示できます。

1. SQL インタフェース・ページの上部にある **[切り替え]** オプションをクリックして、使用するネームスペースを指定します。これにより、利用可能なネームスペースのリストが表示され、その中から選択できます。
2. **[フィルタ]** を適用するか、**[スキーマ]** ドロップダウン・リストからスキーマを選択します。

   [フィルタ] フィールドを使用して、検索パターンを入力してリストをフィルタ処理できます。スキーマ、またはスキーマ（1 つまたは複数）内のテーブル、ビュー、またはプロシージャ名（項目）をフィルタ処理できます。検索パターンは、スキーマの名前、ドット（.）、および項目の名前で構成されます。それぞれの名前は、リテラルやワイルドカードの組み合わせで構成されます。リテラルでは、大文字と小文字が区別されません。ワイルドカードは以下のとおりです。

   - アスタリスク（*）は、任意のタイプの 0 文字以上の文字を意味します。
   - アンダースコア（_）は、任意の型の任意の 1 文字を意味します。
   - アポストロフィ（'）の否定の接頭語は、“ではない“（以外のすべて）を意味します。
   - 円記号（¥）エスケープ文字：¥_ は、リテラルのアンダースコア文字を意味します。

   例えば、`S*` は、先頭文字が S のすべてのスキーマを返します。`S*.Person` は、先頭文字が S のすべてのスキーマ内の、すべての Person 項目を返します。`*.Person*` は、すべてのスキーマ内の、先頭文字が Person のすべての項目を返します。検索パターンのコンマ区切りリストを使用して、リストされたパターンのいずれかを満たすすべての項目を選択できます（OR ロジック）。例えば、`*.Person*,*.Employee*` は、すべてのスキーマ内の、すべての Person 項目および Employee 項目を選択します。

   [フィルタ] 検索パターンを適用するには、更新ボタンをクリックするか、Tab キーを押します。

[フィルタ] 検索パターンは、明示的に変更するまで有効です。[フィルタ] フィールドの右にある “x” ボタンを使用すると、検索パターンがクリアされます。

3. [スキーマ] ドロップダウン・リストからスキーマを選択すると、前のすべての [フィルタ] 検索パターンがオーバーライドされてリセットされ、1 つのスキーマのために選択されます。[フィルタ] 検索パターンを指定すると、前のすべての [スキーマ] がオーバーライドされます。
4. 必要に応じ、[適用先]ドロップダウン・リストを使用して、リストする項目のカテゴリをテーブル、ビュー、プロシージャ、クエリ・キャッシュ、または上記のすべてから指定します。既定は [すべて] です。“[適用先]“ ドロップダウン・リストで指定したカテゴリは、[フィルタ] または [スキーマ] によって制限されます。“[適用先]“ で指定されていないカテゴリは、引き続き、ネームスペース内のそのカテゴリ・タイプのすべての項目をリストします。
5. オプションで、[システム] チェック・ボックスにチェックを付けて、システム項目 (名前の先頭文字が % の項目) を含めます。既定では、システム項目は含まれません。
6. 指定された [スキーマ] または指定された [フィルタ] 検索パターンの項目をリストするには、カテゴリのリストを展開します。リストを展開する場合、項目が含まれていないカテゴリは展開されません。
7. 展開されたリスト内の項目をクリックすると、SQL インタフェースの右側にその [カタログの詳細] が表示されます。

   選択した項目が [テーブル] または [プロシージャ] の場合は、[カタログの詳細] の [クラス名] 情報に、対応する Class Reference ドキュメントへのリンクが表示されます。

フィルタ設定はユーザ・カスタマイズされ、そのユーザが将来使用するために保持されます。

10.3.1 [参照] タブ

[参照] タブには、ネームスペース内のすべてのスキーマ、またはネームスペース内のスキーマのフィルタ処理済みサブセットを簡単に表示できる便利な方法が用意されています。[すべてのスキーマの表示] または [フィルタを使用してスキーマを表示] を選択できます。これらを選択すると、管理ポータル の SQL インタフェースの左側で指定されているフィルタが適用されます。[スキーマ名] 見出しをクリックして、そのスキーマをアルファベットの昇順または降順でリストできます。

リストされた各スキーマには、それが関連付けられているテーブル、ビュー、プロシージャ、およびクエリ (クエリ・キャッシュ) へのリンクがあります。スキーマにそのタイプの項目がない場合は、そのスキーマ・リスト列にハイフン (名前付きリンクではない) が表示されます。これにより、スキーマのコンテンツに関する情報を簡単に入手できます。

テーブル、ビュー、プロシージャ、またはクエリのリンクをクリックすると、それらの項目に関する基本情報のテーブルが表示されます。テーブルの見出しをクリックして、リストを列の値の昇順または降順で並べ替えることができます。[プロシージャ] テーブルには、管理ポータルの SQL インタフェースの左側にある [プロシージャ] 設定に関係なく、必ずエクステント・プロシージャが含まれています。

[カタログの詳細] タブを使用すると、個々のテーブル、ビュー、プロシージャ、およびクエリ・キャッシュの詳細情報を入手できます。[参照] タブから [テーブル] または [ビュー] を選択しても、そのテーブル用の [テーブルを開く] リンクはアクティブになりません。

10.4 カタログの詳細

管理ポータルでは、テーブル、ビュー、プロシージャ、およびクエリ・キャッシュごとに、[カタログの詳細] 情報が示されます。管理ポータルの SQL インタフェースのスキーマ・コンテンツのフィルタ処理 (左側) コンポーネントを使用すると、[カタログの詳細] に表示する個々の項目を選択できます。

10.4.1 テーブルのカタログの詳細

テーブルごとに以下の [カタログの詳細] オプションが用意されています。

- [テーブル情報]：[テーブルタイプ] (TABLE、GLOBAL TEMPORARY、または SYSTEM TABLE (システム・テーブルは [システム] チェック・ボックスにチェックが付いている場合にのみ表示されます))、所有者名、最終コンパイルのタイムスタンプ、外部のブーリアン値と読み取り専用のブーリアン値、[クラス名]、エクステント・サイズ、子テーブルと親テーブル (関連する場合) のいずれかまたは両方の名前および他のテーブル (関連する場合) への 1 つ以上の [リファレンス] フィールド、既定のストレージ・クラス %Storage.Persistent を使用するかどうか、ビットマップ・インデックスをサポートしているかどうか、RowID フィールド名、RowId の基礎となっているフィールドのリスト (関連する場合)、およびテーブルがシャード化されているかどうか。明示的なシャード・キーが存在する場合、シャード・キー・フィールドが表示されます。

  [クラス名] は、"インターシステムズ・クラス・リファレンス" ドキュメント内の対応するエントリへのリンクです。**[クラス名]** は、"識別子とクラスのエンティティ名" で説明されているように、句読点文字を削除してテーブル名から導出された一意の package.class 名です。

  [リファレンス] が [テーブル情報] に表示されるのは、現在のテーブル内のフィールドから別のテーブルへの 1 つまたは複数の参照がある場合のみです。このような別のテーブルへの参照は、参照されるテーブルの [テーブル情報] へのリンクとしてリストされます。

  [シャード化]：テーブルがシャード・マスタ・テーブルの場合、[テーブル情報] にシャード・ローカル・クラスおよびシャード・ローカル・テーブルの名前が、"インターシステムズ・クラス・リファレンス" ドキュメント内の対応するエントリへのリンクと共に表示されます。テーブルがシャード・ローカル・テーブルの場合、[テーブル情報] にシャード・マスタ・クラスおよびシャード・マスタ・テーブルの名前が、"インターシステムズ・クラス・リファレンス" ドキュメント内の対応するエントリへのリンクと共に表示されます。シャード・ローカル・テーブルは、[システム] チェック・ボックスにチェックが付いている場合にのみ表示されます。

  このオプションには、**[テーブルを開くときにロードする行の数]** 用の変更可能な値も用意されています。これは、[テーブルを開く] で表示する行の最大数を設定します。設定可能な範囲は 1 ～ 10,000 です。既定値は 100 です。管理ポータルでは、設定可能な範囲外の値が有効な値に修正されます。0 は 100 に修正され、小数は次に大きな整数へ切り上げられます。10,000 を超える数値は 10,000 に修正されます。

- [フィールド]：テーブル内のフィールド (列) のリスト。表示される項目は、[フィールド名]、[データタイプ]、[カラム#]、[必須]、[ユニーク]、[照合]、[隠し]、[最大長]、[最大値]、[最小値]、[ストリーム]、[コンテナ]、[xDBCタイプ]、[参照先]、[バージョン列]、[選択性]、[外れ値の選択性]、および [平均フィールドサイズ] です。

- [マップ/インデックス]：テーブルに定義されているインデックスのリスト。[インデックス名]、[SQL マップ名]、[列]、[タイプ]、[ブロックカウント]、[マップを継承？]、および [グローバル] の各項目が表示されます。

  [インデックス名] はインデックス・プロパティ名で、プロパティの名前付け規約に従います。SQL インデックス名から生成される際、SQL インデックス名の句読点文字 (アンダースコアなど) は削除されます。**[SQL マップ名]** は、インデックスの SQL 名です。生成される **[SQL マップ名]** は制約名と同じで、同じ名前付け規約に従います (後述)。**[列]** は、インデックスに指定されるフィールドまたはフィールドのコンマ区切りリストを指定します。例 `$$SQLUPPER({Sample.People.Name})` のように、インデックスの照合タイプ および schema.table.field 完全参照を指定する場合もあります。**[タイプ]** は、ビットマップ・エクステント、データ/マスタ、インデックス (標準インデックス)、ビットマップ、またはビットスライス・インデックス、および一意制約のいずれかです。**[ブロックカウント]** には、総数、およびその総数の測定方法の両方が含まれています。この測定方法は、クラス作成者による明示的な設定 ([定義済み])、TuneTableによる計算 ([測定済み])、またはクラス・コンパイラによる推測 ([推測済み]) です。**[継承したマップ?]**) が [はい] の場合は、このマップはスーパークラスから継承されています。**[グローバル]** は、インデックス・データを含む添え字付きグローバルの名前です。インデックス・グローバルの名前付け規約は、"インデックスのグローバル名" を参照してください。このグローバル名を ZWRITE に渡すことで、インデックス・データを表示できます。

  このオプションには、インデックスを再構築するための各インデックスのリンクも用意されています。

- [トリガ]：テーブルに定義されているトリガのリスト。表示される項目は、[トリガ名]、[時間イベント]、[順序]、および [コード] です。

- [制約]：テーブルのフィールドに対する制約のリスト。表示される項目は、[制約名]、[制約タイプ]、[制約データ] (括弧内にリストされたフィールド名) です。制約には、主キー、外部キー、一意制約が含まれます。主キーは定義

上、一意であり、1 回のみリストされます。このオプションでは、制約を制約名別にリストします。複数のフィールドに関係がある制約は、コンポーネント・フィールドのコンマ区切りリストを表示する制約データを使用して、1 回のみリストされます。制約タイプは、UNIQUE、PRIMARY KEY、暗黙的な PRIMARY KEY、FOREIGN KEY、または暗黙的な FOREIGN KEY です。

INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE を呼び出して制約をリストすることもできます。この場合は、制約をフィールド名別にリストします。以下の例では、UNIQUE、PRIMARY KEY、FOREIGN KEY、および CHECK のすべての制約のフィールド名と制約名を返します。

SQL

```
SELECT Column_Name,Constraint_Name FROM INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE WHERE
TABLE_SCHEMA='Sample' AND TABLE_NAME='Person'
```

テーブルが %PUBLICROWID を指定して定義され、明示的な主キーが定義されていない場合、制約タイプ `Implicit PRIMARY KEY` および制約名 `RowIDField_As_PKey` で RowID フィールドが表示されます。

明示的な制約の場合、制約名は以下のように生成されます。

- フィールド定義で指定された制約：例えば、`FullName VARCHAR(48) UNIQUE` または `FullName VARCHAR(48) PRIMARY KEY`。フィールドの制約名値は構文 `TABLENAME_CTYPE#` で生成される値です。ここで、CTYPE は UNIQUE、PKEY、または FKEY で、# は名前のない制約にテーブル定義で指定された順序で割り当てられる連続する整数です。例えば、FullName に MyTest テーブルで 2 番目の名前のない一意の制約 (ID フィールドは除く) がある場合、FullName に生成される制約名は `MYTEST_UNIQUE2` です。FullName が主キーで、MyTest テーブルで 3 番目の名前のない制約が指定されている (ID フィールドは除く) 場合、FullName に生成される制約名は `MYTEST_PKEY3` です。
- CONSTRAINT キーワード名前付き制約節：例えば、`CONSTRAINT UFullName UNIQUE(FirstName,LastName)` または `CONSTRAINT PKName PRIMARY KEY(FullName)`。制約名は指定された一意の制約名です。例えば、MyTest テーブル内の FirstName と LastName それぞれが制約名 `UFullName` を持ち、FullName は制約名 `PKName` を持ちます。
- 名前なし制約節：例えば、`UNIQUE(FirstName,LastName)` または `PRIMARY KEY (FullName)`。制約名値は構文 `TABLENAMECType#` で生成される値です。ここで、CType は Unique、PKey、または FKey で、# は名前のない制約にテーブル定義で指定された順序で割り当てられる連続する整数です。例えば、FirstName と LastName に MyTest テーブルで 2 番目の名前のない一意の制約 (ID フィールドは除く) がある場合、FirstName と LastName に生成される制約名は `MYTESTUnique2` です。FullName が主キーで、MyTest テーブルで 3 番目の名前のない制約 (ID フィールドは除く) が指定されている場合、FullName に生成される制約名は `MYTESTPKey3` です。(大文字/小文字が混在し、アンダースコアがないことに注意してください。)

フィールドが複数の一意制約に関係している場合、制約名ごとに別個にリストされます。

- [クエリキャッシュ]：テーブルのクエリ・キャッシュのリスト。表示される項目は、[ルーチン名]、[クエリ文字列]、[作成日時]、[ソース]、および [クエリタイプ] です。
- **[テーブルの SQL 文]**：このテーブル用に生成された SQL 文のリスト。ネームスペース全体の SQL 文の表示と同じ情報です。

10.4.2 ビューのカタログの詳細

管理ポータルの SQL インタフェースでは、ビュー、プロシージャ、およびクエリ・キャッシュの **[カタログの詳細]** も用意されています。

ビューごとに以下の **[カタログの詳細]** オプションが用意されています。

- 情報表示：**[所有者]** 名、**[最終コンパイル]** タイムスタンプ。このタイムスタンプは、**[ビュー編集]** リンクを使用し、変更を保存すると更新されます。

[読込専用で定義されています] および **[ビューは更新可能です]** ブーリアン：ビュー定義に WITH READ ONLY が含まれる場合、これらはそれぞれ 1 と 0 に設定されます。それ以外の場合、ビューが単一テーブルから定義されている場合は 0 と 1 に設定され、ビューが結合テーブルから定義されている場合は 0 と 0 に設定されます。このオプションは、**[ビュー編集]** リンクを使用して変更できます。

[クラス名] は、"識別子とクラスのエンティティ名" で説明されているように、句読点文字を削除してビュー名から導出された一意の package.class 名です。

[チェックオプション] は、ビュー定義に WITH CHECK OPTION 節が含まれる場合にのみリストされます。LOCAL または CASCADED を指定できます。このオプションは、**[ビュー編集]** リンクを使用して変更できます。

[クラスタイプ] は VIEW です。これは、ビュー定義を編集するための **[ビュー編集]** リンクを提供します。

[テキストの表示] は、ビューの定義に使用する SELECT 文です。ビュー定義は、**[ビュー編集]** リンクを使用して変更できます。

フィールドのリストには、[フィールド名]、[データ型]、[MAXLEN パラメータ]、[MAXVAL パラメータ]、[MINVAL パラメータ]、[BLOB] (%Stream.GlobalCharacter または %Stream.GlobalBinary フィールド)、[長さ]、[精度]、および [スケール] が含まれます。

- **[ビューの SQL 文]**：このビュー用に生成された SQL 文のリストです。ネームスペース全体の SQL 文の表示と同じ情報です。

10.4.3 ストアド・プロシージャのカタログの詳細

プロシージャごとに以下の **[カタログの詳細]** オプションが用意されています。

- ストアド・プロシージャ情報：

  [クラス名] は、クラス名の先頭に型識別子 (‘func’、‘meth’、‘proc’、‘query’) を追加し (例えば、SQL 関数 MyProc は funcMyProc になります)、"識別子とクラスのエンティティ名" で説明されているように、句読点文字を削除して、プロシージャ名から導出された一意の package.class 名です。**[クラスのドキュメント]** は、"インターシステムズ・クラスリファレンス" 内の対応するエントリへのリンクです。**[プロシージャタイプ]** (関数など)。**[メソッドまたはクエリ名]** は、生成されるクラス・メソッドまたはクラス・クエリの名前です。この名前は、"識別子とクラスのエンティティ名" での説明のとおりに生成されます。**[プロシージャ実行]** リンクは、プロシージャをインタラクティブに実行するオプションを提供します。

- **[ストアドプロシージャの SQL 文]**：このストアド・プロシージャ用に生成された SQL 文のリストです。ネームスペース全体の SQL 文の表示と同じ情報です。

10.4.4 クエリ・キャッシュのカタログの詳細

[クエリキャッシュ] には、クエリの全テキスト、クエリ実行プランを表示するためのオプション、および該当クエリ・キャッシュをインタラクティブに実行するためのオプションが用意されています。

10.5 ウィザード

- **[データインポートウィザード]** ― テキスト・ファイルから InterSystems IRIS クラスにデータをインポートするウィザードを実行します。
- **[データエクスポートウィザード]** ― InterSystems IRIS クラスからテキスト・ファイルにデータをエクスポートするウィザードを実行します。

- **[データ移行ウィザード]** – 外部ソースからデータを移行し、InterSystems IRIS クラス定義を作成してそれを格納するウィザードを実行します。
- **[リンクテーブルウィザード]** – ネイティブの InterSystems IRIS データであるかのように外部ソースのテーブルまたはビューにリンクするウィザードを実行します。
- **[リンクプロシージャウィザード]** – 外部ソース内のプロシージャにリンクするウィザードを実行します。

10.6 アクション

- **[ビュー作成]** – ビューを作成するためのページを表示します。このオプションの使用法は、このドキュメントの“ビューの定義と使用”の章で説明しています。
- **[カタログ印刷]** – テーブル定義に関する完全な情報を印刷できます。**[カタログ印刷]** をクリックすると、印刷プレビューが表示されます。この印刷プレビューのインデックス、トリガ、制約をクリックすると、カタログの出力から情報を追加したり除外することができます。
- **[クエリキャッシュ削除]** – クエリ・キャッシュを削除する 3 つのオプション (現在のネームスペースのすべてのクエリ・キャッシュの削除、指定されたテーブルのすべてのクエリ・キャッシュの削除、選択したクエリ・キャッシュのみ削除) があります。
- **[テーブル・チューニング情報]** – 選択したテーブルに対してテーブルチューニング機能を実行します。これは現在のデータに対する各テーブル列の選択性を計算します。選択性の値が 1 の場合は、一意として定義されている (すべてが一意のデータ値を持つ) 列を示します。選択性の値が 1.0000% の場合は、現在のすべてのデータ値が一意の値である、一意として定義されていない列を示します。1.0000% よりも大きいパーセントは、現在のデータの列の重複する値の相対的な数を示します。これらの選択性の値を使用すると、定義するインデックスとインデックスの使用法を決定してパフォーマンスを最適化できます。
- **[スキーマ内の全テーブルをチューニング]** – 現在のネームスペース内の指定されたスキーマに属するすべてのテーブルに対してテーブルのチューニング機能を実行します。
- **[テーブルのインデックスを再構築]** – 指定されたテーブルのすべてのインデックスを再作成します。
- **[この項目を削除]** – 指定されたテーブル定義、ビュー定義、プロシージャ、またはクエリ・キャッシュをドロップ (削除) します。この操作を実行するには、適切な権限を持っている必要があります。**[削除]** は、テーブル・クラスの定義に [DdlAllowed] が含まれている場合を除き、永続クラスを定義して作成したテーブルでは使用できません。使用すると、操作は SQLCODE -300 エラーで失敗し、%msg が“`DDL schema.tablename`”に設定されます。対応する永続クラスにサブクラス (派生クラス) がある場合、テーブルに対して **[削除]** は使用できません。操作は SQLCODE -300 エラーで失敗し、%msg が″ `'Schema.tablename' DDL` ″に設定されます。

  クラスがリンク・テーブルとして定義されている場合は、そのリンク・テーブル・クラスが DdlAllowed として定義されていない場合でも、Drop アクションによりローカル・システム上のそのリンク・テーブルが削除されます。Drop では、このリンクが参照する、サーバ上の実際のテーブルは削除されません。
- **[すべての文のエクスポート]** – 現在のネームスペース内のすべての SQL 文をエクスポートします。SQL 文は XML 形式でエクスポートされます。ファイルにエクスポートするか、ブラウザ表示ページにエクスポートするかを選択できます。
- **[文のインポート]** – XML ファイルから現在のネームスペースに SQL 文をインポートします。

10.7 テーブルを開く

管理ポータルの SQL インタフェースの左側にあるテーブルまたはビューを選択すると、そのテーブルまたはビューの [カタログの詳細] が表示されます。ページ上部の **[テーブルを開く]** リンクもアクティブになります。**[テーブルを開く]** は、テーブル内の実際の (または、ビューを介してアクセスする) データを表示します。このデータは、表示形式で示されます。

既定では、データの最初の 100 行が表示されます。この既定値を変更するには、[カタログの詳細] タブの [テーブル情報] にある **[テーブルを開いたときにロードする列の数]** を設定します。値をロードする行数よりもテーブル内にある行が多い場合は、データ表示の下部に [. . .] インジケータが表示されます。値をロードする行数よりもテーブル内にある行が少ない場合は、データ表示の下部に [ ] インジケータが表示されます。

データ型が %Stream.GlobalCharacter の列には、実際のデータ (最大 100 文字) が文字列として表示されます。最初の 100 文字を超えるデータは、省略記号 (. . .) によって示されます。

データ型が %Stream.GlobalBinary の列は <binary> として表示されます。

10.8 ツール

[システム・エクスプローラ]→[SQL]→**[ツール]** ドロップダウン リストから、以下のツールにアクセスできます。これらのツールは、**[システム・エクスプローラ]**→**[ツール]**→**[SQL パフォーマンス・ツール]** からアクセスできるものと同じです。

- [SQL 実行時統計] : 指定したクエリの SQL 実行時統計を生成するためのユーザ・インタフェース。
- [インデックス分析] : 指定したスキーマに関してさまざまなタイプのインデックス分析を収集するためのユーザ・インタフェース。
- [代替表示プラン] : 指定したクエリの代替表示プランを生成するためのユーザ・インタフェース。
- レポート生成は、SQL クエリ・パフォーマンス・レポートをインターシステムズ WRC (インターシステムズのカスタマ・サポート窓口) に送信するために使用します。このレポート・ツールを使用するには、最初に WRC から WRC 追跡番号を取得する必要があります。
- **[レポートのインポート]** は、既存の WRC レポートをファイル名でインポートするために使用します。インターシステムズでの使用専用。

11
テーブルの定義

この章では、InterSystems SQL でテーブルを作成する方法を説明します。

11.1 テーブル名とスキーマ名

テーブルは、テーブルの定義 (CREATE TABLE を使用) またはテーブルに投影される永続クラスの定義によって作成できます。

- DDL : InterSystems IRIS® データ・プラットフォームでは、CREATE TABLE に指定したテーブル名を使用して、対応する永続クラス名が生成され、指定したスキーマ名を使用して、対応するパッケージ名が生成されます。
- クラス定義 : InterSystems IRIS® データ・プラットフォームでは、永続クラス名を使用して対応するテーブル名が生成され、パッケージ名を使用して、対応するスキーマ名が生成されます。

これら 2 つの名前の対応は、以下の理由により、同一ではない場合があります。

- 永続クラスおよび SQL テーブルは、異なる名前付け規約に従います。適用される有効な文字と長さの要件が異なります。スキーマ名とテーブル名では大文字と小文字が区別されませんが、パッケージ名とクラス名では大文字と小文字が区別されます。システムによって、指定された有効な名前が対応する有効な名前に自動的に変換され、生成される名前は必ず一意になります。
- 既定で、永続クラス名と対応する SQL テーブル名は一致します。SqlTableName クラス・キーワードを使用して、異なる SQL テーブル名を指定できます。
- 既定のスキーマ名は、既定のパッケージ名と一致しない場合があります。未修飾の SQL テーブル名または永続クラス名を指定すると、システムによって既定のスキーマ名またはパッケージ名が指定されます。初期の既定のスキーマ名は SQLUser で、初期の既定のパッケージ名は User です。

11.2 スキーマ名

テーブル名、ビュー名、ストアド・プロシージャ名は修飾する (`schema.name`) ことも、修飾しない (`name`) ことも可能です。

- スキーマ名を指定する場合 (修飾名)、指定されたテーブル、ビュー、またはストアド・プロシージャがそのスキーマに割り当てられます。スキーマが存在しない場合、InterSystems SQL はスキーマを作成し、テーブル、ビュー、またはストアド・プロシージャをそのスキーマに割り当てます。

- スキーマ名を指定しない場合（未修飾名）、InterSystems SQL は既定のスキーマ名またはスキーマ検索パスを使用してスキーマを以下のように割り当てます。

このセクションでは、以下のトピックについて説明します。

- スキーマの名前付けに関する考慮事項
- 予約スキーマ名
- 既定のスキーマ名
- CURRENT_USER を使用したスキーマの指定
- スキーマ検索パス
- プラットフォーム固有のスキーマ名を含める
- スキーマのリスト

11.2.1 スキーマの名前付けに関する考慮事項

スキーマ名は、識別子の規約に従い、英数字以外の文字の使用法に関して多くの考慮事項があります。スキーマ名を区切り識別子として指定することはできません。“USER” またはその他の SQL 予約語をスキーマ名として指定しようとすると、SQLCODE -312 エラーとなります。INFORMATION_SCHEMA スキーマ名および対応する INFORMATION.SCHEMA パッケージ名は、すべてのネームスペースで予約されています。このスキーマおよびパッケージ内で、テーブルまたはクラスを作成しないでください。

CREATE TABLE などの create 操作を実行する場合に、まだ存在していないスキーマを指定すると、InterSystems IRIS によって新しいスキーマが作成されます。InterSystems IRIS はスキーマ名を使用して、対応するパッケージ名を生成します。スキーマの名前付け規約と対応するパッケージの名前付け規約は異なるため、ユーザは名前を変換する際の英数字以外の文字に関する考慮事項に注意する必要があります。これらの名前変換の考慮事項はテーブルと同じではありません。

- 最初の文字：
  - %（パーセント）：% をスキーマ名の最初の文字として指定すると、対応するパッケージがシステム・パッケージであり、そのすべてのクラスがシステム・クラスであることを示します。この使用には、適切な特権が必要です。そうでないと、<PROTECT> エラーを示す %msg を含む SQLCODE -400 エラーが出力されます。
  - _（アンダースコア）：スキーマ名の最初の文字がアンダースコア文字である場合、この文字は対応するパッケージ名内で小文字の “u” に置き換えられます。例えば、スキーマ名 _MySchema は、パッケージ名 uMySchema を生成します。
- 後続の文字：
  - _（アンダースコア）：スキーマ名の最初の文字以外の任意の文字がアンダースコア文字である場合、この文字は対応するパッケージ名内でピリオド（.）に置き換えられます。ピリオドはクラス区切り文字であるため、アンダースコアはスキーマをパッケージとサブパッケージに分割します。したがって、My_Schema はパッケージ Schema を含むパッケージ My を生成します（My.Schema）。
  - @、#、$ 文字：スキーマ名にこれらの文字のいずれかが含まれる場合、これらの文字は対応するパッケージ名から削除されます。これらの文字を削除することで重複したパッケージ名が生成される場合、削除されたパッケージ名がさらに変更されます。削除されたスキーマ名の最後の文字が連続した整数（0 で開始）で置き換えられ、一意のパッケージ名が生成されます。したがって、My@#$Schema はパッケージ MySchema を生成し、以降に作成される My#$Schema は、パッケージ MySchem0 を生成します。同じルールはクラス名に対応するテーブル名にも適用されます。

11.2.2 予約スキーマ名

INFORMATION_SCHEMA スキーマ名および対応する INFORMATION.SCHEMA パッケージ名は、すべてのネームスペースで予約されています。このスキーマおよびパッケージ内で、テーブルまたはクラスを作成しないでください。

IRIS_Shard スキーマ名は、すべてのネームスペースで予約されています。このスキーマ内で、テーブル、ビュー、またはプロシージャを作成しないでください。IRIS_Shard スキーマに格納される項目は、カタログ・クエリでも、INFORMATION_SCHEMA クエリでも表示されません。

11.2.3 既定のスキーマ名

- テーブル、ビュー、トリガ、またはストアド・プロシージャの作成や削除などの DDL 操作を実行する場合、未修飾名として既定のスキーマ名が指定されます。スキーマ検索パスの値は無視されます。
- 既存のテーブル、ビュー、またはストアド・プロシージャにアクセスして、SELECT、CALL、INSERT、UPDATE、DELETE などの DML 操作を実行する場合、未修飾名としてスキーマ検索パス (指定されている場合) からのスキーマ名が指定されます。スキーマ検索パスが指定されていない場合、またはスキーマ検索パスを使用して名前の付いた項目が見つからない場合、既定のスキーマ名が指定されます。

初期設定では、すべてのネームスペース (システム全体) に同じ既定のスキーマ名を使用します。すべてのネームスペースに同じ既定のスキーマ名を設定するか、現在のネームスペースに既定のスキーマ名を設定できます。

未修飾名を使用してテーブルまたはその他の項目を作成すると、InterSystems IRIS によってその項目に既定のスキーマ名と対応する永続クラス・パッケージ名が割り当てられます。名前の付いたスキーマまたは既定のスキーマが存在しない場合、InterSystems IRIS によってスキーマ (およびパッケージ) が作成され、作成した項目がそのスキーマに割り当てられます。スキーマの最後の項目を削除すると、InterSystems IRIS によって、そのスキーマ (およびパッケージ) が削除されます。スキーマ名の解析に関する以下の説明は、テーブル名、ビュー名、およびストアド・プロシージャ名にも当てはまります。

初期のシステム全体の既定の SQL スキーマ名は、SQLUser です。対応する永続クラス・パッケージ名は User です。このため、未修飾テーブル名 Employee か修飾付きテーブル名 SQLUser.Employee のいずれかによって、クラス User.Employee が生成されます。

USER は予約語であるため、スキーマ名を User (または他の SQL 予約語) にして修飾名を指定しようとすると、SQLCODE -1 エラーが返されます。

現在の既定のスキーマ名を返すには、$SYSTEM.SQL.Schema.Default() メソッドを呼び出します。

または、以下のプリプロセッサ・マクロを使用します。

ObjectScript

```
#include %occConstant
  WRITE $$$DefSchema
```

以下のいずれかを使用して、既定のスキーマ名を変更できます。

- 管理ポータルに移動します。[**システム管理**] から、[**構成**]、[**SQL およびオブジェクトの設定**]、[**SQL**] の順に選択します。この画面で、[**既定のスキーマ**] の現在のシステム全体の設定を表示および編集できます。このオプションでは、システム全体の既定のスキーマ名が設定されます。このシステム全体の設定は、現在のネームスペースの SetDefault() メソッド値によってオーバーライドできます。
- $SYSTEM.SQL.Schema.SetDefault() メソッド。既定では、このメソッドはシステム全体の既定のスキーマ名を設定します。ただし、ブーリアンの 3 番目の引数 = 1 を設定することで、現在のネームスペースのみの既定のスキーマを

設定できます。ネームスペースごとに異なる既定のスキーマ名が設定されている場合、$SYSTEM.SQL.CurrentSettings() メソッドは現在のネームスペースの既定のスキーマ名を返します。

注意　デフォルト SQL スキーマ名を変更すると、システム上のすべてのネームスペースのすべてのクエリ・キャッシュが自動的に削除されます。デフォルト・スキーマ名を変更すると、未修飾のテーブル、ビュー、またはストアド・プロシージャ名を含むすべてのクエリの意味が変更されることになります。デフォルト SQL スキーマ名は InterSystems IRIS のインストール時に設定し、その後は変更しないことを強くお勧めします。

スキーマ名は、対応するクラス・パッケージ名の生成に使用されます。これらの名前は名前付け規約が異なるため、同一ではない場合があります。

システム全体の**既定のスキーマ**として SQL 予約語を設定することで、SQL 予約語と同じ名前でスキーマを作成できますが、これは推奨されません。クラスの命名の一意性に関する規約に従って、`User` という名前の既定のスキーマにより、対応するクラス・パッケージ名 `Use0` が生成されます。

11.2.3.1 _CURRENT_USER キーワード

- システム全体の既定のスキーマ名として使用：_CURRENT_USER を既定のスキーマ名として指定すると、現在ログインしているプロセスのユーザ名が既定のスキーマ名として割り当てられます。_CURRENT_USER の値は、ObjectScript 特殊変数値 $USERNAME の最初の部分です。$USERNAME が名前とシステム・アドレスで構成される場合 (Deborah@TestSys)、_CURRENT_USER には名前の部分のみが含まれます。つまり、_CURRENT_USER は、同じ既定のスキーマ名を複数のユーザに割り当てることができます。プロセスがログインしていない場合、_CURRENT_USER では SQLUser が既定のスキーマ名として指定されます。

  _CURRENT_USER/name (name は任意の文字列) を既定のスキーマ名として指定すると、現在ログインしているプロセスのユーザ名が既定のスキーマ名として割り当てられます。プロセスがログインしていない場合、name が既定のスキーマ名として使用されます。例えば、_CURRENT_USER/HMO は、プロセスがログインしていないときは既定のスキーマ名として HMO を使用します。

  $SYSTEM.SQL.Schema.SetDefault() で、`"_CURRENT_USER"` を引用符付き文字列として指定します。

- DDL コマンドのスキーマ名として使用：_CURRENT_USER を DDL 文で明示的なスキーマ名として指定すると、これは、現在のシステム全体の既定のスキーマに置き換えられます。例えば、システム全体の既定のスキーマが SQLUser の場合、コマンド `DROP TABLE _CURRENT_USER.OldTable` では、SQLUser.OldTable が削除されることになります。これは、システム全体の既定のスキーマを使用する必要があることを明示的に示すために名前を修飾する便利な方法です。これは、未修飾名の指定と機能的に同じです。このキーワードを DML 文で使用することはできません。

11.2.4 スキーマ検索パス

DML 操作のために既存のテーブル (またはビューかストアド・プロシージャ) にアクセスする場合、未修飾名としてスキーマ検索パスからのスキーマ名が指定されます。スキーマは、指定された順序で検索され、最初の一致が返されます。検索パスに指定されているスキーマで一致が見つからない場合、または検索パスが存在しない場合、既定のスキーマ名が使用されます (#import マクロ指示文では別の検索方法が使用され、既定のスキーマ名に "フォールスルー" することはありません)。

- 埋め込み SQL では、#sqlcompile path マクロ指示文または #import マクロ指示文を使用して、InterSystems IRIS で未修飾名の解決に使用するスキーマ検索パスを指定できます。#sqlcompile path は、最初に検出された一致で未修飾名を解決します。#import は、検索パス内にリストされているすべてのスキーマに対して一致がちょうど 1 つある場合に未修飾名を解決します。
- 以下の例では、2 つのスキーマ名を記述して検索パスを指定しています。

ObjectScript

```
#sqlcompile path=Customers,Employees
```

詳細は、"ObjectScript の使用法" の "ObjectScript マクロとマクロ・プリプロセッサ" を参照してください。

- ダイナミック SQL では、%SchemaPath プロパティを使用して、InterSystems IRIS で未修飾テーブル名の解決に使用するスキーマ検索パスを指定できます。%SchemaPath プロパティを直接指定することも、このプロパティを %SQL.Statement の %New() メソッドの 2 つ目のパラメータとして指定することもできます。以下の例では、2 つのスキーマ名を記述して検索パスを指定しています。

  ObjectScript

  ```
  SET tStatement = ##class(%SQL.Statement).%New(0,"Customers,Employees")
  ```

  詳細は、"InterSystems SQL の使用法" の "ダイナミック SQL の使用法" を参照してください。

- SQL シェルでは、SQL シェル構成パラメータ PATH を使用して、InterSystems IRIS で未修飾テーブル名の解決に使用するスキーマ検索パスを指定できます。

スキーマ検索パスで指定されているスキーマのどれとも未修飾名が一致しなかったり既定のスキーマ名とも未修飾名が一致しない場合、SQLCODE -30 エラー (例えば、"SQLCODE: -30 : 'PEOPLE' : CUSTOMERS,EMPLOYEES,SQLUSER") が発行されます。

11.2.5 プラットフォーム固有のスキーマ名を含める

Mac で ODBC ベースのクエリを作成し、Microsoft Query を使用して Microsoft Excel から実行する場合、使用可能なテーブルのリストからテーブルを選択すると、生成されたクエリにはテーブルのスキーマ (クラスのパッケージと同等のもの) は含まれません。例えば、Sample スキーマから Person テーブルのすべての行を返すよう選択すると、生成されるクエリは以下のようになります。

SQL

```
SELECT * FROM Person
```

InterSystems IRIS では未修飾のテーブル名は SQLUser スキーマにあるものとして解釈されるため、この文は、失敗するか、間違ったテーブルからデータを返します。これを修正するには、[SQL ビュー] タブで、必要なスキーマを明示的に参照するようにクエリを編集します。クエリは以下のようになります。

```
SELECT * FROM Sample.Person
```

11.2.6 スキーマのリスト

INFORMATION.SCHEMA.SCHEMATA 永続クラスは、現在のネームスペース内のすべてのスキーマをリストします。

以下の例では、現在のネームスペース内のすべての非システム・スキーマの名前が返されます。

SQL

```
SELECT SCHEMA_NAME
FROM INFORMATION_SCHEMA.SCHEMATA WHERE NOT SCHEMA_NAME %STARTSWITH '%'
```

管理ポータルの左側にある SQL インタフェースでは、スキーマ (またはフィルタ・パターンに一致する複数のスキーマ) のコンテンツを表示できます。詳細は、"スキーマ・コンテンツのフィルタ処理" を参照してください。

11.3 テーブル名

各テーブルには、スキーマ内で一意の名前が付けられています。テーブルには SQL テーブル名および対応する永続クラス名の両方があります。これらの名前は、許可されている文字、大文字と小文字の区別、および最大長が異なります。SQL CREATE TABLE コマンドを使用して定義されている場合は、識別子の規約に従った SQL テーブル名を指定します。対応する永続クラス名がシステムで生成されます。永続クラス定義として定義されている場合は、英数字のみを含む名前を指定する必要があります。この名前が、大文字小文字を区別する永続クラス名および (既定では) 対応する大文字と小文字を区別しない SQL テーブル名の両方に使用されます。オプションの SqlTableName クラス・キーワードを使用して、別の SQL テーブル名を指定できます。

CREATE TABLE コマンドを使用してテーブルを作成する場合、InterSystems IRIS はテーブル名を使用して対応する永続クラス名を生成します。テーブルの名前付け規約と対応するクラスの名前付け規約は異なるため、ユーザは名前を変換する際の英数字以外の文字に関する考慮事項に注意する必要があります。

- 最初の文字：
  - % (パーセント)：テーブル名の最初の文字として % が予約されているため、避ける必要があります (″識別子″を参照)。指定した場合、% 文字は対応する永続クラス名から削除されます。
  - _ (アンダースコア)：テーブル名の最初の文字がアンダースコア文字である場合、この文字は対応する永続クラス名から削除されます。例えば、テーブル名 _MyTable は、クラス名 MyTable を生成します。
  - 数値：テーブル名の最初の文字を数値にすることはできません。テーブル名の最初の文字が句読点文字の場合、2 番目の文字に数字を指定することはできません。これにより、SQLCODE -400 エラーが発生し、生成される %msg の値は“エラー #5053：クラス名 'schema.name' が無効です”になります (句読点文字なし)。例えば、指定したテーブル名が _7A の場合、生成される %msg は“エラー #5053: クラス名 'User.7A' が無効です”になります。

- 後続の文字：
  - 文字：テーブル名には、最低でも 1 文字を含める必要があります。テーブル名の先頭の文字または最初の句読点に続く文字は、数字以外の文字にする必要があります。$ZNAME テストに合格した文字は、有効な文字です。$ZNAME 文字検証はロケールによって異なります (識別子には句読点文字を含めることができるため、$ZNAME を使用して SQL 識別子を検証することはできません)。
  - _ (アンダースコア)、@、#、$ 文字：テーブル名にこれらの文字のいずれかが含まれる場合、これらの文字は対応するクラス名から削除され、一意の永続クラス名が生成されます。生成されたクラス名には句読点が含まれないため、句読点のみが異なるテーブル名の作成はお勧めできません。

- テーブル名は、そのスキーマ内で重複しないようにする必要があります。既存テーブルと大文字/小文字区別のみが異なる名前でテーブルを作成しようとすると、SQLCODE -201 エラーが生成されます。

  ただし、同じスキーマ内のビューおよびテーブルには、同じ名前を指定できません。これを実行しようとすると、SQLCODE -201 エラーが返されます。

  $SYSTEM.SQL.Schema.TableExists() メソッドを使用して、テーブル名が既に存在するかどうかを確認できます。$SYSTEM.SQL.Schema.ViewExists() メソッドを使用して、ビュー名が既に存在するかどうかを確認できます。これらのメソッドは、テーブル名またはビュー名に対応するクラス名も返します。管理ポータルの SQL インタフェース [カタログの詳細] の **[テーブル情報]** オプションでは、選択した SQL テーブル名に対応するクラス名が表示されます。

“USER“ またはその他の SQL 予約語をスキーマ名として指定しようとすると、SQLCODE -312 エラーとなります。SQL 予約語をテーブル名またはスキーマ名として指定するには、名前を区切り文字付き識別子として指定します。区切り文字付き識別子を使用して英数字以外の文字を含むテーブルまたはスキーマ名を指定した場合、InterSystems IRIS は対応するクラスまたはパッケージ名を生成する際にこれらの英数字以外の文字を削除します。

以下のテーブル名の長さの制限が適用されます。

- 一意性：InterSystems IRIS は永続クラス名の最初の 189 文字で一意性チェックを実行します。対応する SQL テーブル名は 189 文字を超える長さになる可能性がありますが、英数字以外の文字を取り除くと、この 189 文字の制限内で一意である必要があります。InterSystems IRIS はパッケージ名の最初の 189 文字で一意性チェックを実行します。
- 推奨される最大長：一般に、テーブル名は 128 文字を超えることはできません。テーブル名は 96 文字よりも大幅に長くすることができますが、最初の 96 の英数文字が異なるようにテーブル名を作成すると処理がはるかに容易になります。
- 合計最大長：パッケージ名とその永続クラス名（同時に追加する場合）は、220 文字を超えることはできません。これには、既定のスキーマ（パッケージ）名（スキーマ名が指定されていない場合）およびパッケージ名とクラス名を区切るドット文字が含まれます。テーブル名が対応する永続クラス名に変換されるときに 220 文字を超える文字が削除される場合は、スキーマ名とテーブル名の合計を 220 文字より長くすることができます。

テーブル名の詳細は、"InterSystems SQL リファレンス" の "CREATE TABLE" コマンドを参照してください。クラスの詳細は、"クラスの定義と使用" の "クラスの定義とコンパイル" を参照してください。

11.4 RowID フィールド

SQL では、すべてのレコードは RowID という一意の整数値によって識別されます。InterSystems SQL では、RowID フィールドを指定する必要はありません。テーブルを作成して希望のデータ・フィールドを指定するとき、RowID フィールドが自動的に作成されます。この RowID は内部的に使用されますが、クラス・プロパティにマップはされません。既定では、永続クラスが SQL テーブルに投影される場合にのみ、表示されます。この投影されたテーブルでは、追加の RowID フィールドが表示されます。既定では、このフィールドは "ID" と名付けられ、列 1 に割り当てられます。

既定では、テーブルにデータが入力されるときに、InterSystems IRIS は、このフィールドに 1 から始まる連続した正の整数を割り当てます。RowID データ型は BIGINT (%Library.BigInt) です。RowID 用に生成される値には、以下の制約があります。各値は一意です。NULL 値は許可されていません。照合は EXACT です。既定では、値を変更することはできません。

既定では、InterSystems IRIS は、このフィールドに "ID" という名前を付けます。ただし、このフィールド名は予約されていません。RowID フィールド名は、テーブルがコンパイルされるたびに再設定されます。ユーザが "ID" という名前のフィールドを定義している場合、テーブルがコンパイルされると、InterSystems IRIS は RowID に "ID1" という名前を付けます。例えば、その後でユーザが ALTER TABLE を使用して "ID1" という名前のフィールドを定義すると、テーブルのコンパイルで RowID の名前が "ID2" に変更されます（それ以降も同様に動作します）。永続クラス定義では、Sql-RowIdName クラス・キーワードを使用して、このクラスが投影されるテーブルの RowID フィールド名を直接指定できます。これらの理由により、名前で RowID フィールドを参照することは避ける必要があります。

InterSystems SQL では、どのようなフィールド名が RowID に割り当てられていても常に RowID 値を返す、%ID 疑似列名（エイリアス）を提供しています。（InterSystems TSQL では、同じ働きをする $IDENTITY 疑似列名を提供しています。）

ALTER TABLE では、RowID フィールド定義の変更や削除はできません。

レコードをテーブルに挿入すると、InterSystems IRIS は各レコードに整数の ID 値を割り当てます。RowID の値は必ず増分になります。再使用されることはありません。そのため、レコードが挿入および削除された場合、RowID の値は昇順の数値にはなりますが、数値としての連続性はなくなります。

- 既定では、CREATE TABLE を使用して定義されたテーブルは、複数のプロセスによるテーブルへの迅速な同時入力ができる $SEQUENCE を使用して ID の割り当てを実行します。$SEQUENCE を使用してテーブルにデータを入力する際、一連の RowID 値がプロセスに割り当てられ、その後、プロセスがそれらを連続的に割り当てます。同時プロセスがそれぞれ独自の割り当てられたシーケンスを使用して RowID を割り当てるため、複数のプロセスによって挿入されたレコードが挿入順であると見なすことはできません。

SetOption() メソッドの DDLUseSequence オプションを設定すると、InterSystems IRIS が $INCREMENT を使用して ID 割り当てを実行するように構成できます。現在の設定を確認するには、$SYSTEM.SQL.CurrentSettings() メソッドを呼び出します。

- 既定では、永続クラスを作成して定義したテーブルが、$INCREMENT を使用して ID 割り当てを実行します。永続クラス定義では、IdFunction ストレージ・キーワードを sequence または increment に設定できます (例えば、<IdFunction>sequence</IdFunction>)。

永続クラス定義では、IdLocation ストレージ・キーワード・グローバル (例えば、永続クラス Sample.Person の場合は <IdLocation>^Sample.PersonD</IdLocation>) には、割り当てられた最も大きい RowID カウンタ値が含まれます。(これはレコードに割り当てられた最も大きい整数であり、プロセスに割り当てられた最も大きい整数ではありません。) この RowID カウンタ値は、既存のレコードとは対応しなくなる可能性があることに注意してください。特定の RowID 値を持つレコードが存在しているかどうかを判別するには、テーブルの %ExistsId() メソッドを呼び出します。

RowID カウンタは、TRUNCATE TABLE コマンドによってリセットされます。このカウンタは、DELETE コマンドがテーブル内のすべての行を削除した場合でも、DELETE コマンドによってはリセットされません。テーブルにデータが挿入されていない場合、または TRUNCATE TABLE を使用してすべてのテーブル・データが削除されている場合、IdLocation ストレージ・キーワード・グローバル値は定義されません。

既定では、RowID の値をユーザが変更することはできません。RowID 値を変更しようとすると、SQLCODE -107 エラーが生成されます。この既定値をオーバーライドして RowID 値を変更できるようにすると、取り返しの付かない結果を招くことがあるため、極めて特別な状況でのみ、十分な注意を払って行う必要があります。Config.SQL.AllowRowIDUpdate プロパティにより、RowID の値のユーザによる変更を許可します。

11.4.1 フィールドに基づく RowID

テーブルを投影する永続クラスを定義することで、RowID にフィールドの値またはフィールドの組み合わせの値を設定するよう定義できます。それには、IdKey インデックス・キーワードでインデックスを指定します。例えば、インデックス定義 IdxId On PatientName [IdKey]; を指定して、テーブルに PatientName フィールドの値と同じ値の RowID を設定したり、インデックス定義 IdxId On (PatientName,SSN) [IdKey]; を指定して、PatientName フィールドと SSN フィールドを組み合わせた値の RowID を設定できます。

- フィールドに基づく RowID は、システムで割り当てられる連続する正の整数を取る RowID よりも非効率的です。
- INSERT 時：RowID を構成するフィールドまたはフィールドの組み合わせに指定される値は、一意である必要があります。一意でない値を指定すると、SQLCODE -119 "UNIQUE あるいは PRIMARY KEY 制約が INSERT の一意性チェックに失敗しました" が生成されます。
- UPDATE 時：既定では、RowID を構成する各フィールドの値は変更できません。これらのいずれかのフィールドの値を変更しようとすると、SQLCODE -107 "RowID またはフィールドに基づく RowID を UPDATE できません" が生成されます。

RowID が複数のフィールドに基づく場合、RowID 値は各コンポーネント・フィールドの値を || 演算子で結合した値になります (Ross,Betsy||123-45-6789 など)。InterSystems IRIS は複数のフィールドに基づく RowID の最大長の特定を試み、最大長を特定できない場合、RowID の長さは既定で 512 になります。

詳細は、"主キー" を参照してください。

11.4.2 RowID は非表示か

- CREATE TABLE を使用してテーブルを作成する際、既定では RowID は非表示です。非表示フィールドは SELECT * によって表示されず、PRIVATE です。テーブルの作成時に、%PUBLICROWID キーワードを指定すると、RowID は非表示でなくなり、PUBLIC になります。このオプションの %PUBLICROWID キーワードは、テーブル要素の CREATE TABLE コンマ区切りリスト内の任意の場所で指定できます。ALTER TABLE では指定できません。詳細は、CREATE TABLE のリファレンス・ページの "RowID フィールドと %PUBLICROWID" を参照してください。

- テーブルとして投影する永続クラスを作成する際、既定では RowID は非表示ではありません。SELECT * によって表示され、PUBLIC です。クラス・キーワード SqlRowIdPrivate を指定することで、非表示で PRIVATE の RowID で永続クラスを定義できます。

外部キー参照として使用される RowID はパブリックである必要があります。

既定では、パブリック RowID を持つテーブルをコピー元またはコピー先テーブルとして使用し、INSERT INTO Sample.DupTable SELECT * FROM Sample.SrcTable を使用してデータを重複テーブルにコピーすることはできません。

管理ポータルの SQL インタフェース [**カタログの詳細**] の [フィールド] リストの [**隠し**] 列を使用して、RowID が非表示かどうかを示すことができます。

以下のプログラムを使用して、指定したフィールド (この例では、ID) が非表示かどうかを返すことができます。

ObjectScript

```
SET myquery = "SELECT FIELD_NAME,HIDDEN FROM %Library.SQLCatalog_SQLFields(?) WHERE FIELD_NAME='ID'"

SET tStatement = ##class(%SQL.Statement).%New()
SET qStatus = tStatement.%Prepare(myquery)
 IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
SET rset = tStatement.%Execute()
DO rset.%Display()
WRITE !,"End of data"
```

テーブル内のフィールド名 (非表示および表示) をリストする方法は、“列の名前と番号”を参照してください。

11.5 主キー

InterSystems IRIS では、テーブル内の行を一意に識別する RowID と主キーの 2 つの方法が用意されています。

オプションの主キーは、アプリケーションがテーブル内の行を一意に識別するために使用できる (例えば、結合で)、意味のある値です。主キーは、ユーザ指定のデータ・フィールドまたは複数のデータ・フィールドの組み合わせにできます。主キーの値は一意でなければなりませんが、整数値である必要はありません。RowID は、テーブル内の行を識別するために内部的に使用される整数値です。通常、主キーはアプリケーションによって生成された値であり、RowID は InterSystems IRIS によって生成された一意の整数値です。

RowID フィールドを使用してデータの行にアクセスするために、マスタ・マップが自動的に作成されます。主キー・フィールドを定義すると、主キー・インデックスが自動的に作成されて維持されます。

行を識別するために 2 つの異なるフィールドとインデックスを持つという二重性は、必ずしも良いことではありません。以下の 2 つの方法のいずれかで、単一の行識別子とインデックスに解決することができます。

- アプリケーションで生成された主キー値を IDKEY として使用します。PrimaryKey と IdKey の両方のキーワードを含むクラス定義で主キー・インデックスを識別することで、これを行うことができます (このために PKEY_IS_IDKEY フラグを設定した場合は、DDL からもこの操作を行うことができます)。これにより、主キー・インデックスがテーブルのマスタ・マップになります。この結果、主キーが行のメインの内部アドレスとして使用されます。主キーが複数のフィールドで構成される場合、または主キーの値が整数でない場合、これは非効率です。
- アプリケーションで生成された主キー値は使用しないでください。代わりにシステムで生成された RowID の整数をアプリケーションが使用する主キーとしてアプリケーション内で使用してください (例えば、結合で)。この操作の利点は、整数の RowID が、ビットマップ・インデックスの使用を含め、より効率的な処理に適していることです。

アプリケーションの性質に応じて、単一の行識別子とインデックスに解決したり、アプリケーションで生成された主キーとシステムで生成された RowID に別個のインデックスを設定することができます。

11.6 RowVersion、AutoIncrement、および Serial カウンタ・フィールド

InterSystems SQL は、カウンタ値を自動的にインクリメントするための 3 つの専用データ型をサポートしています。3 つのデータ型はすべて、%Library.BigInt データ型クラスを拡張するサブクラスです。

- %Library.RowVersion：ネームスペース全体のすべての RowVersion テーブルに対する挿入と更新をカウントします。ROWVERSION フィールドを含むテーブルの挿入と更新のみが、このカウンタをインクリメントします。ROWVERSION の値は一意で、変更できません。このネームスペース全体のカウンタがリセットされることはありません。詳細は、"ROWVERSION フィールド" を参照してください。
- %Library.Counter (SERIAL カウンタ・フィールドとも呼ばれます)：テーブルへの挿入をカウントします。既定では、このフィールドは自動的にインクリメントされた整数を受け取ります。ただし、ユーザはこのフィールドにゼロ以外の整数値を指定できます。ユーザは重複値を指定できます。ユーザが指定した値が、システムで指定された最も大きい値よりも大きい場合、自動インクリメント・カウンタが設定され、ユーザが指定した値からインクリメントされます。詳細は、"Serial カウンタ・フィールド" を参照してください。
- %Library.AutoIncrement：テーブルへの挿入をカウントします。既定では、このフィールドは自動的にインクリメントされた整数を受け取ります。ただし、ユーザはこのフィールドにゼロ以外の整数値を指定できます。 ユーザは重複値を指定できます。ユーザ値を指定しても、自動インクリメント・カウンタには影響を与えません。詳細は、"AutoIncrement フィールド" を参照してください。

これら 3 つのフィールドすべて、および IDENTITY フィールドは、以下の例に示すように、AUTO_INCREMENT = YES を返します。

SQL

```
SELECT COLUMN_NAME,AUTO_INCREMENT FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = 'MyTable'
```

11.6.1 RowVersion フィールド

RowVersion フィールドは、行レベルのバージョン管理を提供する、オプションのユーザ定義フィールドです。これにより、ネームスペース全体で各行のデータが変更された順序を把握できます。InterSystems IRIS はネームスペース全体を範囲とするカウンタを維持しており、行データが変更 (挿入、更新、または %Save) されるたびに、そのフィールドに一意の正の増分整数を割り当てます。このカウンタはネームスペース全体を範囲とするため、ROWVERSION フィールドを含む 1 つのテーブルでの操作によって、同じネームスペース内の ROWVERSION フィールドを含む他のすべてのテーブルで使用される ROWVERSION カウンタの増分ポイントが設定されます。

RowVersion フィールドは、フィールド・データ型 ROWVERSION を指定して作成します (%Library.RowVersion)。テーブルごとに指定できる ROWVERSION データ型フィールドは 1 つのみです。複数の ROWVERSION フィールドを含むテーブルを作成しようとすると、5320 コンパイル・エラーが発生します。

このフィールドには任意の名前を付けることができ、どの列位置にも表示できます。ROWVERSION (%Library.RowVersion) データ型は BIGINT (%Library.BigInt) にマッピングされます。

このフィールドは、自動インクリメント・カウンタから 1 で始まる正の整数を受け取ります。このカウンタは、ROWVERSION 対応テーブル内のデータが挿入、更新、または %Save 操作によって変更されると必ずインクリメントされます。インクリメントされた値は、挿入または更新された行の ROWVERSION フィールドに記録されます。

ネームスペースには、RowVersion フィールドがあるテーブルと、そのフィールドがないテーブルを含めることができます。RowVersion フィールドがあるテーブルのデータが変更された場合にのみ、ネームスペース全体のカウンタは増分されます。

テーブルにデータが入力されるときに、InterSystems IRIS は挿入された各行に対して、このフィールドに連続した整数を割り当てます。既にデータが取り込まれているテーブルに ROWVERSION フィールドを追加するために ALTER TABLE を使用する場合、このフィールドは既存のフィールドについては NULL として作成されます。その後テーブルで挿入や更新が実行されると、その行の RowVersion フィールドに連続的な整数が割り当てられます。このフィールドは読み取り専用です。RowVersion 値を変更しようとすると、SQLCODE -138 エラー：″ INSERT/UPDATE ″が生成されます。したがって、RowVersion フィールドは固有であり変更不可であると定義されますが、必須であるまたは NULL でないとは定義されません。

RowVersion の値は必ず増分になります。再使用されることはありません。したがって、挿入と更新により割り当てられる固有の RowVersion 値は、一時的な順序です。削除操作では、このシーケンスから数字が削除されます。そのため、RowVersion 値は数字的に連続していない場合があります。

このカウンタはリセットされません。すべてのテーブル・データを削除しても、RowVersion カウンタはリセットされません。ROWVERSION フィールドを含むネームスペース内のすべてのテーブルを削除しても、このカウンタはリセットされません。

一意キーや主キーに RowVersion フィールドを含めることはできません。RowVersion フィールドは、IDKey インデックスの一部にすることはできません。

シャード・テーブルに RowVersion フィールドを含めることはできません。

RowVersion フィールドは非表示ではありません (これは SELECT * により表示されます)。

これについては以下の、同じネームスペースにある 3 つのテーブルの例で示しています。

1. それぞれ ROWVERSION フィールドがある Table1 と Table3 を作成し、ROWVERSION フィールドがない Table2 を作成します。
2. Table1 に 10 行を挿入します。そうするとこれらの行の ROWVERSION 値により、カウンタは 10 だけ増分されます。カウンタはそれより前には使用されていなかったので、1 から始まり 10 になります。
3. Table2 に 10 行を挿入します。Table2 には ROWVERSION フィールドがないので、カウンタは増分されません。
4. Table1 の 1 行を更新します。そうするとこの行の ROWVERSION 値が反映されて、カウンタは増分されます (この場合は 11 になります)。
5. Table3 に 10 行を挿入します。そうするとこれらの行の ROWVERSION 値により、カウンタは 10 だけ増分されます (12 から始まり 21 になります)。
6. Table1 の 1 行を更新します。そうするとこの行の ROWVERSION 値が反映されて、カウンタは増分されます (この場合は 22 になります)。
7. Table1 の 1 行を削除します。この場合に ROWVERSION カウンタは変更されません。
8. Table3 の 1 行を更新します。そうするとこの行の ROWVERSION 値が反映されて、カウンタは増分されます (この場合は 23 になります)。

11.6.2 Serial カウンタ・フィールド

SERIAL データ型 (永続クラス・テーブル定義の %Library.Counter) を使用して、1 つ以上のオプションの整数カウンタ・フィールドを指定して、テーブルへのレコードの挿入順序を記録することができます。各 Serial カウンタ・フィールドは、独自のカウンタを保持しています。

テーブルに行が挿入されると、Serial カウンタ・フィールドは自動インクリメント・カウンタから正の整数を受け取り、このフィールドに値なし (NULL) または値 0 が指定されます。ただし、ユーザは挿入中にこのフィールドにゼロ以外の整数値を指定して、テーブル・カウンタの既定をオーバーライドできます。

- INSERT がカウンタ・フィールドにゼロ以外の整数値を指定しない場合、カウンタ・フィールドは自動的に正の整数カウンタ値を受け取ります。カウントは 1 から始まります。連続する各値は、このフィールドの最大割り当てカウンタ値から 1 ずつ増加します。

- INSERT がカウンタ・フィールドにゼロ以外の整数値を指定する場合、フィールドはその値を受け取ります。これは、正または負の整数値で、現在のカウンタ値よりも小さくても大きくてもかまいません。また、このフィールドに既に割り当てられている整数にすることもできます。この値が、割り当てられたカウンタ値よりも大きい場合は、自動インクリメント・カウンタのインクリメント開始点をその値に設定します。

カウンタ・フィールドの値を UPDATE しようとすると、SQLCODE -105 エラーが返されます。

このカウンタは、TRUNCATE TABLE コマンドによって 1 にリセットされます。このカウンタは、DELETE コマンドがテーブル内のすべての行を削除した場合でも、DELETE コマンドによってはリセットされません。

シャード・テーブルに Serial カウンタ・フィールドを含めることはできません。

11.6.3 AutoIncrement フィールド

%Library.AutoIncrement データ型 (または `BIGINT AUTO_INCREMENT`) を使用して整数カウンタ・フィールドを指定し、テーブルへのレコードの挿入順序を記録することができます。テーブルごとに指定できる %AutoIncrement データ型フィールドは 1 つのみです。テーブルに行が挿入されると、このフィールドは自動インクリメント・カウンタから正の整数を受け取り、このフィールドに値なし (NULL) または値 0 が指定されます。ただし、ユーザは挿入中にこのフィールドにゼロ以外の整数値を指定して、テーブル・カウンタの既定をオーバーライドできます。

- INSERT がカウンタ・フィールドにゼロ以外の整数値を指定しない場合、カウンタ・フィールドは自動的に正の整数カウンタ値を受け取ります。カウントは 1 から始まります。連続する各値は、このフィールドの最大割り当てカウンタ値から 1 ずつ増加します。
- INSERT がカウンタ・フィールドにゼロ以外の整数値を指定する場合、フィールドはその値を受け取ります。これは、正または負の整数値で、現在のカウンタ値よりも小さくても大きくてもかまいません。また、このフィールドに既に割り当てられている整数にすることもできます。ユーザが割り当てた値は、自動インクリメント・カウンタに影響を与えません。

カウンタ・フィールドの値を UPDATE しようとすると、SQLCODE -105 エラーが返されます。

このカウンタは、TRUNCATE TABLE コマンドによって 1 にリセットされます。このカウンタは、DELETE コマンドがテーブル内のすべての行を削除した場合でも、DELETE コマンドによってはリセットされません。

シャード・テーブルには AutoIncrement フィールドを含めることができます。

11.7 DDL を使用したテーブルの定義

InterSystems SQL で標準の DDL コマンドを使用してテーブルを定義できます。

テーブル 11-1: InterSystems SQL で使用可能な DDL コマンド

ALTER コマンド	CREATE コマンド	DROP コマンド
ALTER TABLE	CREATE TABLE	DROP TABLE
ALTER VIEW	CREATE VIEW	DROP VIEW
	CREATE INDEX	DROP INDEX
	CREATE TRIGGER	DROP TRIGGER

DDL コマンドは、以下のようなさまざまな方法で実行できます。

- ダイナミック SQL を使用

- 埋め込み SQL を使用
- DDL スクリプト・ファイルを使用
- ODBC 呼び出しを使用
- JDBC 呼び出しを使用

11.7.1 埋め込み SQL での DDL の使用

ObjectScript のメソッドまたはルーチンで埋め込み SQL を使用して、DDL コマンドを呼び出すことができます。

例えば、以下のメソッドは Sample.Employee テーブルを生成します。

Class Member

```
ClassMethod CreateTable() As %String
{
 &sql(CREATE TABLE Sample.Employee (
    EMPNUM              INT NOT NULL,
    NAMELAST            CHAR (30) NOT NULL,
    NAMEFIRST           CHAR (30) NOT NULL,
    STARTDATE           TIMESTAMP,
    SALARY              MONEY,
    ACCRUEDVACATION     INT,
    ACCRUEDSICKLEAVE    INT,
    CONSTRAINT EMPLOYEEPK PRIMARY KEY (EMPNUM)))

  IF SQLCODE=0 {WRITE "Table created"  RETURN "Success"}
  ELSEIF SQLCODE=-201 {WRITE "Table already exists"  RETURN SQLCODE}
  ELSE {WRITE "Serious SQL Error, returning SQLCODE"  RETURN SQLCODE_" "_%msg}
 }
```

このメソッドが呼び出されると、Sample.Employee テーブルの生成を試みます (同様に、対応する Sample.Employee クラスの生成も試みます)。これが成功すると、SQLCODE 変数は 0 に設定されます。失敗した場合は、SQLCODE に失敗の原因を示す SQL エラー・コードが含まれます。

このような DDL コマンドでエラーが発生する主な原因は、以下のとおりです。

- SQLCODE -99 (権限違反) : このエラーは、ユーザがその DDL コマンドを実行する特権を持っていないことを示します。通常、これはアプリケーションが現在のユーザが誰であるかを確立していないことに起因します。これは、$SYSTEM.Security.Login() メソッドを使用して、プログラムで実行できます。

  ObjectScript

  ```
  DO $SYSTEM.Security.Login(username,password)
  ```

- SQLCODE -201 (テーブルまたはビュー名がユニークではありません) : 新規のテーブルを生成する際に、既に存在するテーブル名を使用した場合に、このエラーが表示されます。

11.7.2 クラス・メソッドを使用した DDL の実行

ObjectScript では、ダイナミック SQL %SQL.Statement オブジェクトを使用して、ダイナミック SQL を使用した DDL コマンドの作成と実行が可能です。

以下に示す例では、ダイナミック SQL を使用してテーブルを作成するクラス・メソッドを定義しています。

Class Definition

```
Class Sample.NewT
{
ClassMethod DefTable(user As %String,pwd As %String) As %Status [Language=objectscript]
  {
  DO ##class(%SYSTEM.Security).Login(user,pwd)
  SET myddl=2
  SET myddl(1)="CREATE TABLE Sample.MyTest "
  SET myddl(2)="(NAME VARCHAR(30) NOT NULL,SSN VARCHAR(15) NOT NULL)"
  SET tStatement=##class(%SQL.Statement).%New()
  SET tStatus=tStatement.%Prepare(.myddl)
    IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
  SET rset=tStatement.%Execute()
  IF rset.%SQLCODE=0 {WRITE "Created a table"}
  ELSEIF rset.%SQLCODE=-201 {WRITE "table already exists"}
  ELSE {WRITE "Unexpected error SQLCODE=",rset.%SQLCODE}
  }
}
```

このメソッドは、以下のように呼び出します。

ObjectScript

```
DO ##class(Sample.NewT).DefTable("myname","mypassword")
```

埋め込み SQL の例と同様に、現在ログインしているユーザがいない場合、このメソッドは失敗します。

11.7.3 DDL スクリプトのインポートおよび実行によるテーブルの定義

InterSystems SQL DDL スクリプト・ファイルをインポートするには、ターミナル・セッションからインタラクティブに $SYSTEM.SQL.Schema.Run() メソッドを使用するか、バックグラウンド・ジョブとして $SYSTEM.SQL.Schema.ImportDDL("IRIS") メソッドを使用します。このメソッドでは複数の SQL コマンドをインポートおよび実行でき、ユーザは txt スクリプト・ファイルを使用してテーブルおよびビューを定義し、これらにデータを入力することができます。詳細は、このドキュメントの "SQL コードのインポート" の章を参照してください。

他のベンダのリレーショナル・データベースから InterSystems IRIS にテーブルを移行する際、テキスト・ファイル内に 1 つ以上の DDL スクリプトが存在していることがあります。InterSystems IRIS には、そのようなテーブルを InterSystems IRIS にロードする際に役立ついくつかの **%SYSTEM.SQL.Schema** メソッドが用意されています。汎用の ImportDDL() メソッドまたは特定ベンダ用の **%SYSTEM.SQL.Schema** ロード・メソッドを使用できます。ベンダ固有の SQL は、InterSystems SQL に変換されて実行されます。エラーおよびサポートされない機能は、ログ・ファイルに記録されます。詳細は、このドキュメントの “SQL コードのインポート” の章の "コード移行：非 InterSystems SQL のインポート" を参照してください。

例えば、ObjectScript コマンド行から Oracle DDL ファイルをロードするには、以下の手順を実行します。

1. InterSystems IRIS ランチャー・メニューの **[ターミナル]** コマンドを使用して、ターミナル・セッションを開始します。
2. テーブル定義をロードしたいネームスペースに切り替えます。

   ObjectScript

   ```
   SET $namespace = "MYNAMESPACE"
   ```

3. 目的の DDL インポート・メソッドを呼び出します。

   ObjectScript

   ```
   DO $SYSTEM.SQL.Schema.LoadOracle()
   ```

   ターミナルに表示された指示に従います。

11.8 永続クラスの作成によるテーブルの定義

SQL テーブルを定義する標準的な方法は、SQL プロンプト上あるいは JDBC 接続経由または ODBC 接続経由で CREATE TABLE DDL コマンドを実行することですが、VS Code や Studio などの IDE を利用して永続クラス・テーブルの定義を作成することもできます。クラスは、%Persistent として定義するか、%Persistent として定義されているスーパークラスから継承する必要があります。これらのクラスが InterSystems IRIS データベースで保存およびコンパイルされたとき、クラス定義に対応するリレーショナル・テーブルが自動的に投影されます (各クラスはテーブルを表し、各プロパティは列を表します)。1 つのクラス (テーブル) に定義可能なプロパティ (列) の最大数は 1000 です。

例えば、以下では、永続クラス MyApp.Person が定義されます。

Class Definition

```
Class MyApp.Person Extends %Persistent
{
Parameter USEEXTENTSET = 1;

Property Name As %String(MAXLEN=50) [Required];
Property SSN As %String(MAXLEN=15) [InitialExpression = "Unknown"];
Property DateOfBirth As %Date;
Property Sex As %String(MAXLEN=1);

Index BitmapExtent [ Extent, Type = bitmap ];
}
```

このクラスの特徴を以下に示します。

- コンパイルした定義によって、MyApp.Person 永続クラスおよび対応する SQL テーブル Person が MyApp スキーマに作成されます。これらの操作の実行方法の詳細は、"クラスの定義とコンパイル" を参照してください。
- このクラス定義には、パッケージ名 MyApp があります。永続クラスを定義するとき、未指定のパッケージ名は既定で "User" になります。この名前は、既定の SQL スキーマ名である SQLUser に相当します。例えば、テーブル Students を永続クラスとして定義すると、User.Students クラスとそれに対応する SQL schema.table 名 SQLUser.Students が作成されます。
- 永続クラス名 Person が SQL テーブル名として使用されます。別の SQL テーブル名を指定するには SqlTableName クラス・キーワードを使用します。
- USEEXTENTSET クラス・パラメータが定義され、1 に設定されます。このパラメータによって、テーブル・ストレージをまとめた、より効率的なグローバルのセットが作成されます。SQL テーブルに投影されるすべての永続クラスに USEEXTENTSET パラメータを指定することをお勧めします。
- ビットマップ・エクステント・インデックスは、エクステント・セット内のすべての ID のインデックスを作成します。この設定により、カウントやその他の操作がより効率的になります。SQL テーブルに投影されるすべての永続クラスにビットマップ・エクステント・インデックスを作成することをお勧めします。

同じ MyApp.Person テーブルは、DDL 文 CREATE TABLE を使用して、SQL schema.table 名を指定することで定義できます。この SQL 文が正常に実行されると、パッケージ名が MyApp でクラス名が Person の対応する永続クラスが生成されます。

SQL

```
CREATE TABLE MyApp.Person (
    Name VARCHAR(50) NOT NULL,
    SSN VARCHAR(15) DEFAULT 'Unknown',
    DateOfBirth DATE,
    Sex VARCHAR(1)
)
```

DDL コマンドを使用してテーブルを定義するときは、USEEXTENTSET の指定も、ビットマップ・エクステント・インデックスの作成も不要です。InterSystems SQL によって、これらの設定が自動的に適用され、投影された永続クラスに反映されます。

既定では、CREATE TABLE は対応するクラス定義で Final クラス・キーワードを指定します。これは、サブクラスを持てないことを示しています。

データベースのオブジェクト・ビューとリレーショナル・ビューの対応関係は、"既定の SQL プロジェクションの概要" を参照してください。

上記のような永続クラス定義をコンパイルすると、対応するテーブルが作成されますが、このテーブル定義は、SQL DDL コマンド (または管理ポータルの Drop アクション) を使用しても変更や削除ができません。これらのコマンドを使用してこのような操作を実行しようとすると、メッセージ [DDL schema.name] が表示されます。これらの操作を許可するには、そのテーブル・クラス定義で [DdlAllowed] を指定する必要があります。

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
```

11.8.1 プロパティ・パラメータの定義

永続クラスにテーブルを定義すると、そのテーブルの列に、定義したプロパティが投影されます。すべてのプロパティ定義では、そのプロパティの基となっているクラスを指定するデータ型クラスを指定する必要があります。指定したデータ型により、プロパティで使用できる値がそのデータ型に制限されます。テーブルに投影される永続クラスを定義する際、%Library パッケージのクラスを使用して、このデータ型を指定する必要があります。このクラスは、%Library.Datatype として、または %Datatype として指定できます。

多くのデータ型クラスには、使用できるプロパティ値を詳細に定義するためのパラメータが用意されています。これらのパラメータは、個々のデータ型に固有です。より一般的なデータ定義パラメータの一部を以下に示します。

- プロパティ値の制限
- 許可されるプロパティ値
- 一意のプロパティ値
- 計算プロパティ値

データがフィールドに挿入、またはフィールドで更新されると、InterSystems SQL は自動的にデータを検証して、データ型および参照整合性制約を適用します。他の方法でテーブルにデータを入力する場合は、テーブル・データを検証する必要があります。

11.8.1.1 プロパティ値の制限

数値データ型では、MAXVAL および MINVAL パラメータを指定して、使用できる値の範囲を制限できます。定義上、数値データ型にはサポートされる最大値 (正または負) があります。MAXVAL および MINVAL を使用して、使用できる範囲をさらに制限できます。

文字列データ型では、MAXLEN および MINLEN パラメータを指定して、使用できる長さ (文字数) を制限できます。定義上、文字列データ型にはサポートされる最大長があります。MAXLEN および MINLEN を使用して、使用できる範囲をさらに制限できます。MAXLEN パラメータと MINLEN パラメータによって適用される長さ制限は、データベースにデータが格納されている場合にのみ適用されます。クエリでは、データベースにあるすべてのデータが有効であると想定しているからです。具体的には、INSERT を使用してデータベースにデータを追加するとき、または UPDATE を使用してデータベースのデータを編集するとき (または、ObjectScript で %Save() を実行するとき) に、この制限が適用されます。既定では、MAXLEN を超える長さのプロパティ値があると、INSERT の場合は SQLCODE -104、UPDATE の場合は SQLCODE -105 の各検証エラーが発生します。TRUNCATE=1 を指定すると、MAXLEN を超える文字列の値を使用できますが、指定した文字列は MAXLEN の長さに切り詰められます。既定の最大文字列長は 4096 です。この制限値は、CPF の ODBCVarcharMaxlen フィールドで構成できます。

11.8.1.2 許可されるプロパティ値

以下の 2 つの方法で、実際のプロパティ値を制限できます。

- 使用できる値のリスト (VALUELIST および DISPLAYLIST で列挙された値)。
- 使用できる値の一致パターン (PATTERN)。

列挙されるプロパティ値

テーブルを永続クラスとして定義することで、指定した特定の値のみを設定できるプロパティ (列) を定義できます。このためには、VALUELIST パラメータを指定します。VALUELIST (論理ストレージ値のリストを指定) は通常、DISPLAYLIST (対応する表示値のリストを指定) と共に使用されます。どちらのリストも、リスト区切り文字で始まります。複数のデータ型が VALUELIST および DISPLAYLIST を指定できます。次の例では、列挙された値を持つ 2 つのプロパティを定義します。

Class Definition

```
Class Sample.Students Extends %Persistent
{
Parameter USEEXTENTSET = 1;

Property Name As %String(MAXLEN=50) [Required];
Property DateOfBirth As %Date;
Property ChoiceStr As %String(VALUELIST=",0,1,2",DISPLAYLIST=",NO,YES,MAYBE");
Property ChoiceODBCStr As %EnumString(VALUELIST=",0,1,2",DISPLAYLIST=",NO,YES,MAYBE");

Index BitmapExtent [ Extent, Type = bitmap ];
}
```

VALUELIST が指定されている場合、INSERT または UPDATE は VALUELIST にリストされているいずれかの値のみを指定でき、それ以外の場合は値なし (NULL) が指定されます。VALUELIST の有効な値では、大文字と小文字が区別されます。必須のプロパティでは、VALUELIST の値に一致しない値を指定すると、INSERT の場合は SQLCODE -104、UPDATE の場合は SQLCODE -105 の各検証エラーが返されます。必須ではないプロパティでは、VALUELIST の値に一致しない値は NULL 値に変換されます。

ODBC モードで表示した場合、%String および %EnumString データ型は動作が異なります。上の例では、論理モードで表示した場合、ChoiceStr と ChoiceODBCStr は共に VALUELIST 値を表示します。表示モードで表示した場合、ChoiceStr と ChoiceODBCStr は共に DISPLAYLISTT 値を表示します。ODBC モードで表示した場合、ChoiceStr は VALUELIST 値を表示し、ChoiceODBCStr は DISPLAYLIST 値を表示します。

プロパティ値のパターン・マッチング

複数のデータ型が PATTERN パラメータを指定できます。PATTERN は、使用できる値を、指定された ObjectScript パターンと一致する値に制限します。このパターンは、先頭の疑問符を省略した、引用符付きの文字列で指定します。次の例では、パターンを持つプロパティを定義します。

Class Definition

```
Class Sample.Students Extends %Persistent
{
Parameter USEEXTENTSET = 1;

Property Name As %String(MAXLEN=50) [Required];
Property DateOfBirth As %Date;
Property Telephone As %String(PATTERN = "3N1""-""3N1""-""4N");

Index BitmapExtent [ Extent, Type = bitmap ];
}
```

パターンは引用符付きの文字列として指定されるため、パターンで指定されるリテラルは二重引用符で囲む必要があります。パターン・マッチングは、MAXLEN および TRUNCATE の前に適用されます。このため、MAXLEN を超え、切り捨てられる可能性のある文字列のパターンを指定する場合、パターンを“.E”で終了できます (任意のタイプの、無制限の数の末尾の文字)。

PATTERN と一致しない値を指定すると、INSERT の場合は SQLCODE -104、UPDATE の場合は SQLCODE -105 の各検証エラーが発生します。

11.8.1.3 一意のプロパティ値

CREATE TABLE を使用すると、列を UNIQUE として定義できます。これは、すべてのフィールドの値が一意の (重複がない) 値になることを意味します。

テーブルを永続クラスとして定義しても、対応する一意性プロパティ・キーワードはサポートされません。代りに、プロパティとそのプロパティに対する一意のインデックスを定義する必要があります。以下の例では、各レコードに一意の Num 値を指定します。

Class Definition

```
Class Sample.CaveDwellers Extends %Persistent [ DdlAllowed ]
{
Parameter USEEXTENTSET = 1;

Property Num As %Integer;
Property Troglodyte As %String(MAXLEN=50);

Index UniqueNumIdx On Num [ Type=index,Unique ];
Index BitmapExtent [ Extent, Type = bitmap ];
}
```

インデックス名は、プロパティの名前付け規約に従います。オプションの Type キーワードは、インデックス・タイプを指定します。Unique キーワードは、プロパティ (列) を一意として定義します。

一意の値の列は、INSERT OR UPDATE 文を使用する際に必要になります。

11.8.1.4 計算プロパティ値

以下のクラス定義の例は、列 (Birthday) を持つテーブルを定義します。このテーブルでは、DateOfBirth の値が初めて設定されたときに、SqlComputed を使用して列の値が計算され、DateOfBirth の値が更新されたときに、SqlComputeOnChange を使用して列の値が再計算されます。Birthday には、この値の計算日時または再計算日時を記録した最新のタイムスタンプが記述されます。

Class Definition

```
Class Sample.MyStudents Extends %Persistent [DdlAllowed]
{
  Parameter USEEXTENTSET = 1;

  Property Name As %String(MAXLEN=50) [Required];
  Property DateOfBirth As %Date;
  Property Birthday As %String
          [ SqlComputeCode = {SET {Birthday}=$PIECE($ZDATE({DateOfBirth},9),",")_
                              " changed: "_$ZTIMESTAMP},
                              SqlComputed, SqlComputeOnChange = DateOfBirth ];

  Index BitmapExtent [ Extent, Type = bitmap ];
}
```

既存の DateOfBirth 値を指定する DateOfBirth に対して UPDATE を実行しても Birthday の値は再計算されません。

SqlComputeCode プロパティのキーワードには、値の計算に使用される ObjectScript コードが記述されています。*Property*Computation メソッドに計算コードを指定することもできます。この場合、計算するプロパティの名前は *Property* です。このメソッドでは、ObjectScript 以外の言語 (Python など) で計算コードを指定できます。

このクラスでは、AgeComputation メソッドによって DOB (生年月日) プロパティに基づいて Age プロパティが計算されます。cols 入力引数は、%Library.PropertyHelper オブジェクトです。このオブジェクトの getfield メソッドを使用して、他のプロパティを参照できます。

Class/ObjectScript

```
Class Sample.MyStudents Extends %Persistent [ DdlAllowed ]
{
Parameter USEEXTENTSET = 1;

Property Name As %String(MAXLEN = 50) [ Required ];
Property DOB As %Date;
Property Age As %Integer [ Calculated, SqlComputed, SqlComputeOnChange = DOB ];

Index BitmapExtent [ Extent, Type = bitmap ];

ClassMethod AgeComputation(cols As %Library.PropertyHelper) As %Integer
{
    set today = $zdate($horolog,8)
    set bdate = $zdate(cols.getfield("DOB"), 8)
    return $select(bdate = "":"", 1:(today - bdate) \ 10000)
}

}
```

Class/Python

```
Class Sample.MyStudents Extends %Persistent [ DdlAllowed ]
{
Parameter USEEXTENTSET = 1;

Property Name As %String(MAXLEN = 50) [ Required ];
Property DOB As %Date;
Property Age As %Integer [ Calculated, SqlComputed, SqlComputeOnChange = DOB ];

Index BitmapExtent [ Extent, Type = bitmap ];

ClassMethod AgeComputation(cols As %Library.PropertyHelper) As %Integer [ Language = python ]
{
    import datetime as d
    iris_date_offset = d.date(1840,12,31).toordinal()
    bdate = d.date.fromordinal(cols.getfield("DOB") + iris_date_offset).strftime("%Y%m%d")
    today = d.date.today().strftime("%Y%m%d")
    return str((int(today) - int(bdate)) // 10000) if bdate else ""
}

}
```

計算コードの指定に関する詳細は、CREATE TABLE のリファレンス・ページで "計算列" のセクションを参照してください。

クラス・プロパティのキーワードの詳細は、"プロパティの構文とキーワード" を参照してください。

11.8.2 埋め込みオブジェクト (%SerialObject)

プロパティを定義する埋め込みシリアル・オブジェクト・クラスを参照することで、永続テーブルの構造を簡素化できます。例えば、通り、市、都道府県、および郵便番号で構成された住所情報を MyData.Person テーブルに含める場合があります。これらのプロパティを MyData.Person に指定するのではなく、これらのプロパティを定義するシリアル・オブジェクト (%SerialObject) クラスを定義してから、この埋め込みオブジェクトを参照する 1 つの Home プロパティを MyData.Person で指定できます。これを以下のクラス定義で示しています。

Class Definition

```
Class MyData.Person Extends (%Persistent) [ DdlAllowed ]
{  Parameter USEEXTENTSET = 1;
   Property Name As %String(MAXLEN=50);
   Property Home As MyData.Address;
   Property Age As %Integer;
   Index BitmapExtent [ Extent, Type = bitmap ];
}
```

Class Definition

```
Class MyData.Address Extends (%SerialObject)
{  Property Street As %String;
   Property City As %String;
   Property State As %String;
   Property PostalCode As %String;
}
```

シリアル・オブジェクト・プロパティ内のデータに直接アクセスすることはできません。シリアル・オブジェクト・プロパティを参照する永続クラス/テーブルを介してアクセスする必要があります。

- 永続テーブルから個々のシリアル・オブジェクト・プロパティを参照するには、アンダースコアを使用します。例えば、`SELECT Name, Home_State FROM MyData.Person` は州 (State) のシリアル・オブジェクト・プロパティ値を文字列として返します。シリアル・オブジェクト・プロパティ値は、クエリで指定された順序で返されます。
- 永続テーブルからすべてのシリアル・オブジェクト・プロパティを参照するには、参照フィールドを指定します。例えば、`SELECT Home FROM MyData.Person` はすべての MyData.Address プロパティの値を %List 構造として返します。シリアル・オブジェクト・プロパティ値は、シリアル・オブジェクトで指定されている Home_Street、Home_City、Home_State、Home_PostalCode の順序で返されます。管理ポータルの SQL インタフェース [カタログの詳細] では、この参照フィールドは **[コンテナ]** フィールドと呼ばれます。これは **[隠し]** フィールドであるため、`SELECT *` 構文では返されません。
- 永続クラスの `SELECT *` は、入れ子になっているシリアル・オブジェクトを含め、すべてのシリアル・オブジェクト・プロパティを個別に返します。例えば、`SELECT * FROM MyData.Person` は Age、Name、Home_City、Home_PostalCode、Home_State、および Home_Street の値を (この順序で) 返します。Home の %List 構造の値は返しません。シリアル・オブジェクト・プロパティ値は照合順で返されます。`SELECT *` はまず、永続クラスのすべてのフィールドを照合順でリストし (通常、アルファベット順)、その後に、入れ子になっているシリアル・オブジェクト・プロパティを照合順でリストします。

埋め込みシリアル・オブジェクトは、参照する永続テーブルと同じパッケージ内に存在する必要はありません。%Library.SerialObject の SqlCategory (および SqlCategory を明示的に定義していない %SerialObject のすべてのサブクラス) は STRING です。

埋め込みオブジェクトを定義することで、永続テーブルの定義を次のように簡素化できます。

- 永続テーブルに、同じ埋め込みオブジェクト内の異なる複数のレコードを参照する複数のプロパティを含めることができます。例えば、MyData.Person テーブルに、両方とも MyData.Address シリアル・オブジェクト・クラスを参照する、Home プロパティと Office プロパティを含めることができます。
- 複数の永続テーブルが同じ埋め込みオブジェクトのインスタンスを参照できます。例えば、MyData.Person テーブルの Home プロパティと MyData.Employee テーブルの WorkPlace プロパティが、両方とも MyData.Address シリアル・オブジェクト・クラスを参照できます。
- 埋め込みオブジェクトは、別の埋め込みオブジェクトを参照できます。例えば、MyData.Address 埋め込みオブジェクトに、CountryCode プロパティ、AreaCode プロパティ、および PhoneNum プロパティが含まれる MyData.Telephone 埋め込みオブジェクトを参照する Phone プロパティを含めることができます。永続クラスから複数のアンダースコアを使用して、入れ子になったシリアル・オブジェクト・プロパティを参照します (例えば、Home_Phone_AreaCode)。

シリアル・オブジェクト・クラスをコンパイルすると、ストレージ定義にデータ仕様が生成されます。コンパイラはシリアル・オブジェクト・クラス名に "State" という語を付加することで、この仕様にデータ名を割り当てます。したがって、MyData.Address には `<Data name="AddressState">` が割り当てられます。この名前 (この例では AddressState) が既にプロパティ名として使用されている場合、コンパイラは整数を付加して一意のデータ名を作成します (`<Data name="AddressState1">`)。

"クラスの定義と使用" の "シリアル・オブジェクトの概要" を参照してください。

シリアル・オブジェクト・プロパティのインデックス作成の詳細は、"埋め込みオブジェクト (%SerialObject) のプロパティのインデックス作成" を参照してください。

11.8.3 クラス・メソッド

テーブル定義の一部としてクラス・メソッドを指定できます。その例を以下に示します。

Class Definition

```
Class MyApp.Person Extends %Persistent
{
Parameter USEEXTENTSET = 1;
Property Name As %String(MAXLEN=50) [Required];
Property SSN As %String(MAXLEN=15) [InitialExpression = "Unknown"];
Property DateOfBirth As %Date;
Property Sex As %String(MAXLEN=1);
Index BitmapExtent [ Extent, Type = bitmap ];
ClassMethod Numbers() As %Integer [ SqlName = Numbers, SqlProc ]
  {
   QUIT 123
  }
}
```

クラス・メソッドを SQL プロシージャに投影するには `SqlProc` キーワードが必要です。SELECT クエリでは、メソッドに定義した `SqlName` 値を使用してこのメソッドを呼び出すことができます。以下に例を示します。

SQL

```
SELECT Name,SSN,MyApp.Numbers() FROM MyApp.Person
```

11.8.4 永続クラスの作成によるシャード・テーブルの定義

シャード・テーブルとして投影する永続クラスを定義する前に、シャーディング環境を設定する必要があります。その後、シャード永続クラスを定義するには、クラス・キーワード `Sharded=1` を、オプションのシャード関連クラス属性と共に指定します。

注意　　シャード・テーブルは、データのない新しい永続クラスに対してのみ定義します。既存のクラスにシャーディングを適用すると、データにアクセスできなくなる可能性があります。

このクラスは、サンプルのシャード永続クラスと、オプションでシャード関連のクラス属性セットを定義します。

Class Definition

```
Class Sample.MyShardT Extends %Persistent [ Sharded = 1 ]
{
    Parameter DEFAULTCONCURRENCY As BOOLEAN = 0;
    Parameter USEEXTENTSET = 1;
    Index BitmapExtent [ Extent, Type = bitmap ];
}
```

次のクラス属性が適用されます。

- `Sharded = 1` キーワードは、このクラスが投影されるテーブルをシャード・テーブルとして定義します。この設定により、シャーディング・インフラストラクチャで、データ分散を含め、シャード・テーブル・ストレージを管理します。したがって、InterSystems IRIS で生成および維持される既定のストレージ定義をカスタマイズすることはできません。このクラス定義内のストレージ定義に対して行われた変更はすべて無視されます。
- `DEFAULTCONCONCURRENCY` クラス・パラメータは 0（ロックなし）に設定されます。シャード・テーブルには分散する性質があることから、0 のパラメータ値が必要です。この値は、%Open や %OpenId などのオブジェクト・メソッドの既定値として使用されます。したがって、すべてのメソッド呼び出しに並行処理引数を渡す必要はありません。
- `USEEXTENTSET` クラス・パラメータは 1 に設定されます。これにより、テーブル・ストレージは、より効率的なグローバルのセットに編成されます。DDL コマンドを使用してシャード・テーブルを定義する場合、InterSystems SQL はこの設定を自動的に適用します。

- ビットマップ・エクステント・インデックスは、エクステント・セット内のすべての ID のインデックスを作成します。この設定により、カウントやその他の操作がより効率的になります。DDL コマンドを使用してシャード・テーブルを定義する場合、InterSystems SQL はこの設定を自動的に適用します。

その後、シャード・キー・インデックスを定義できます。シャード・テーブルを作成すると、抽象シャード・キー・インデックスが自動的に生成されます。シャード・キー・インデックスは、行が存在するシャードを指定します。シャード・キー・インデックスの定義とシャード・テーブルの作成の詳細は、"シャード・テーブルの作成とデータのロード" を参照してください。

11.8.4.1 シャード・クラスの制限事項

- シャード・クラスではサポートされないクラス・パラメータ : CONNECTION、DEFAULTGLOBAL、DSINTERVAL、DSTIME、IDENTIFIEDBY、OBJJOURNAL。
- シャード・クラスではサポートされないクラス・キーワード : Language、ViewQuery。
- シャード・クラスではサポートされないスーパー・クラス : %Library.IndexBuilder、%DocDB.Document。
- シャード・クラスではサポートされないプロパティ・データ型 : %Library.Text。
- リレーションシップ・プロパティは、シャード・クラスではサポートされていません。
- プロジェクションは、シャード・クラスではサポートされていません。
- "objectscript" 以外の言語のメソッドは、シャード・クラスではサポートされていません。
- タイプが %SQLQuery でないクラス・クエリは、シャード・クラスではサポートされていません。

これらの機能を使用してシャード・クラスをコンパイルしようとすると、コンパイル時エラーが発生します。

11.9 シャード・テーブルの定義

シャード・テーブルを作成するには、3 つの要件があります。

1. ライセンス・キーでシャーディングがサポートされている必要があります。管理ポータルの **[システム管理]→[ライセンス]→[ライセンスキー]** で、現在のライセンスを表示するか、新しいライセンスをアクティブにします。
2. InterSystems IRIS インスタンスでシャーディングを有効にする必要があります。シャーディングを有効にするには、%Admin_Secure 権限が必要です。管理ポータルの **[システム管理]→[構成]→[システム構成]→[シャーディング構成]** で **[シャーディングの有効化]** ボタンを選択します。これにより、シャード・クラスタで現在の InterSystems IRIS インスタンスを使用できるようになります。[ ] または [ ] を選択します。[OK] を押します。InterSystems IRIS インスタンスを再開します。
3. InterSystems IRIS インスタンス上にシャード・クラスタを導入する必要があります。このシャード・クラスタには、シャード・マスタ・ネームスペースが含まれます。現在のネームスペースがシャーディング用に構成されていない場合、シャード・テーブルを定義しようとすると、" #9319: %1 " で失敗します。詳細は、"スケーラビリティ・ガイド" の "シャーディングによるデータ量に応じた InterSystems IRIS の水平方向の拡張" の章にある "API または管理ポータルを使用したクラスタの導入" を参照してください。

次に、シャード・クラスタの一部として定義されているシャード・マスタ・ネームスペース内で、シャード・テーブルを定義できます。CREATE TABLE を使用し、シャード・キーを指定することでシャード・テーブルを定義できます。または、シャード・テーブルに投影される永続クラスを作成できます。

シャード・テーブルの定義の詳細は、"スケーラビリティ・ガイド" の "シャーディングによるデータ量に応じた InterSystems IRIS の水平方向の拡張" の章にある "ターゲット・シャード・テーブルの作成" を参照してください。

11.10 既存のテーブルのクエリによるテーブルの定義

既存のテーブル (テーブルまたはビュー、複数可) に基づいて新しいテーブルを定義して生成できます。これには、クエリと新しいテーブル名を指定します。既存のテーブル名と新しいテーブル名のいずれかまたは両方を修飾付きまたは未修飾にできます。クエリに JOIN 構文を含めることができます。クエリに、新しいテーブルの列名になる列名エイリアスを指定できます。

この操作は、前述したとおり、CREATE TABLE AS SELECT コマンド、または $SYSTEM.SQL.Schema.QueryToTable() メソッドを使用して実行できます。

1. QueryToTable() は、既存のテーブルの DDL 定義をコピーして、指定された新しいテーブル名にこれを割り当てます。これは、data type (データ型)、maxlength (最大長)、minval/maxval (最小/最大有効値) など、クエリに指定されたフィールドの定義をコピーします。既定値、必須の値、一意の値などフィールドのデータ制約はコピーしません。フィールドから別のテーブルへの参照はコピーしません。

   クエリで SELECT * または SELECT %ID を指定すると、元のテーブルの RowID フィールドが、必須ではなく、一意ではない、データ型 integer のデータ・フィールドとしてコピーされます。QueryToTable() は、新しいテーブルの一意の RowID フィールドを生成します。コピーされた RowID を ID と名付けた場合、生成される RowID の名前は ID1 となります。

   QueryToTable() は、この新しいテーブルの対応する永続クラスを作成します。永続クラスは DdlAllowed として定義されます。新しいテーブルの所有者は現在のユーザです。

   ソース・テーブルでの設定に関係なく、新しいテーブルでは、Default Storage = YES (既定のストレージ・クラスを使用する) および Supports Bitmap Indices = YES (ビットマップ・インデックスをサポートする) と定義されます。

   新しいテーブルに作成される唯一のインデックスが IDKEY インデックスです。ビットマップ・エクステント・インデックスは生成されません。コピーされたフィールドのインデックス定義は、新しいテーブルにコピーされません。

2. 次に、QueryToTable() は、クエリによって選択されたフィールドのデータを新しいテーブルに生成します。テーブルのエクステント・サイズを 100,000 に設定します。また、IDKEY ブロック・カウントを見積もります。テーブルのチューニングを実行し、実際のエクステント・サイズとブロック・カウント、および各フィールドの選択性と平均フィールド・サイズの値を設定します。

QueryToTable() は、テーブル定義の作成と新しいテーブルへのデータの生成の両方を実行します。テーブル定義の作成だけを行いたい場合は、クエリの WHERE 節にデータ行なしで選択する条件を指定します。例えば、`WHERE Age < 20 AND Age > 20` のように指定します。

以下の例では、Name フィールドと Age フィールドが Sample.Person からコピーされ、AVG(Age) フィールドが作成されます。これらのフィールド定義は、Sample.Youth という名前の新しいテーブルの作成に使用されます。このメソッドは次に、Age < 21 のレコードに関する Sample.Person のデータを Sample.Youth に生成します。AvgInit フィールドには、テーブルの作成時に選択されたレコードの集約値が含まれます。

ObjectScript

```
 DO $SYSTEM.SQL.Schema.QueryToTable("SELECT Name,Age,AVG(Age) AS AvgInit FROM Sample.Person WHERE Age
< 21","Sample.Youth",1,.errors)
```

11.11 外部テーブル

重要　外部テーブルは、InterSystems IRIS 2023.1 で試験的機能として用意されています。つまり、実稼働環境ではサポートされません。ただし、この機能は十分にテストされており、お客様に大きな価値をもたらすことができると考えています。

インターシステムズでは、この新機能を実際の環境で使用したお客様からのフィードバックをお待ちしています。ご自分の体験を他のユーザと共有する場合、または質問がある場合は、Developer Community にアクセスするか、インターシステムズのサポート窓口 (WRC) までお問い合わせください。

InterSystems SQL では、外部データ・ソースのデータを InterSystems IRIS のインスタンスに投影する外部テーブルを定義できます。このテーブルを使用すると、そのインスタンスに格納されているデータに対するクエリと同様に、このような外部データに対するクエリを実行できます。

11.11.1 外部テーブルの概要

さまざまな理由により、InterSystems IRIS に直接データをロードすることが不可能であるか、合理的ではない場合があります。例として、データ・ファイルがきわめて大きく、InterSystems IRIS テーブルにロードするストレージ・コストに見合うほど頻繁にはクエリが実行されない状況が挙げられます。外部テーブルは、別のシステムで管理されているデータを投影し、InterSystems IRIS のインスタンスで管理および格納しているデータと同様に、投影したデータにクエリおよびアクセスできるようにする機能です。

11.11.2 外部テーブルの作成

外部テーブルを作成する前に、InterSystems IRIS が外部データ・ソースとどのように対話するかを決定するために外部サーバを定義する必要があります。外部サーバを定義すると、外部ソースのデータを表す外部テーブルを 1 つ以上定義できます。そのためには、外部データ・ソースのフィールドを InterSystems IRIS の列にマッピングするうえで必要な、列の名前とタイプなどの詳細情報を指定します。

11.11.2.1 手順 1: 外部サーバを定義する

外部テーブルを定義するには、外部サーバを定義し、使用する外部データ・ラッパを指定しておく必要があります。そのためには、CREATE FOREIGN SERVER コマンドを使用します。

CREATE FOREIGN SERVER コマンドでは、外部データ・ラッパを指定する必要があります。InterSystems IRIS で特定のタイプのデータ・ソースをどのように操作するかを、この外部データ・ラッパで指定します。CREATE FOREIGN SERVER コマンドでは、外部データ・ラッパと、その外部データ・ラッパが必要とするメタデータを指定する必要があります。現在のところ、InterSystems SQL では 2 つの外部データ・ラッパとして CSV と JDBC をサポートしています。CSV 外部データ・ラッパでは、ローカル・ファイル・システムのフォルダへのパスを指定する必要があります。JDBC 外部データ・ラッパでは、外部データベースに接続するための JDBC 接続を指定する必要があります。

外部サーバに定義できる外部テーブルの数に制限はありません。

以下の例は、CSV 外部データ・ラッパを使用する外部サーバの作成方法を示しています。

SQL

```
CREATE FOREIGN SERVER Sample.TestFile FOREIGN DATA WRAPPER CSV HOST '\path\to\file'
```

以下の例は、JDBC 外部データ・ラッパを使用する外部サーバの作成方法を示しています。

SQL

```
CREATE FOREIGN SERVER Sample.PostgresDB FOREIGN DATA WRAPPER JDBC 'postgresConnection'
```

11.11.2.2 手順 2:外部テーブルを定義する

外部サーバを定義すると、CREATE FOREIGN TABLE コマンドを使用して外部テーブルを定義できます。このテーブルの列名は外部ソースのデータと同じ列名にすることができるほか、InterSystems IRIS では別の名前で外部ソースのデータ列を参照することもできます。外部テーブルを作成する構文は、LOAD DATA コマンドに似ています。

SQL

```
CREATE
FOREIGN
TABLE Sample.AccountTeam (
   TeamID BIGINT,
   Name VARCHAR(50),
   CountryCode VARCHAR(10)
) SERVERT Sample.PostgresDB TABLE 'Sample.Teams'
```

データ定義言語 (DDL) 文を使用して外部テーブルを作成すると、ClassType を "view" として、対応するクラスが作成されます。このクラスは手動で編集しないでください。また、外部テーブルは CREATE FOREIGN TABLE コマンドで定義する必要があります。クラス定義を作成することで外部テーブルを作成することはできません。

11.11.3 外部テーブルのクエリ

外部テーブルのクエリは、ネイティブ・テーブルのクエリとまったく同じです。

```
SELECT Name, CountryCode FROM Sample.AccountTeam ORDER BY Name
```

クエリではさらに高度な構文を活用することもできます。

SQL

```
SELECT t.Name, COUNT(m.*)
FROM Sample.AccountManager m JOIN Sample.AccountTeam t
     ON m.TeamID = t.TeamID
WHERE t.CountryCode = 'UK' AND m.Salary > 100000
GROUP BY t.Name
```

InterSystems SQL では、WHERE 節の簡潔な述語が可能な範囲で送信またはプッシュ・ダウンされます。これにより、ネットワーク上で送受信されるデータの量を制限し、リモート・データベースでの最適化を最大限に活用します。ただし、2 つの外部テーブル間での GROUP BY や JOIN のような複雑な節は、外部データの取得を完了した後で InterSystems IRIS によって処理されます。

クエリを発行するユーザには %Gateway_Object:USE 特権が必要です。

注釈　外部テーブルに対するクエリの発行では、その基盤となる Java ベース・エンジンが使用されます。このエンジンでは、サーバ上に Java 仮想マシン (JVM) をインストールした環境が必要です。セットアップ済みの JVM があり、その JVM に PATH 環境変数でアクセスできれば、初めてクエリを発行したときに、InterSystems IRIS で自動的にその JVM を使用して外部言語サーバが起動します。特定の JVM が使用されるように外部言語サーバをカスタマイズする場合、またはリモート・サーバを使用する場合は、"外部サーバ接続の管理" を参照してください。

11.11.4 外部テーブルの削除

外部テーブルを削除するには DROP FOREIGN TABLE コマンドを使用します。

```
DROP FOREIGN TABLE Example.MyForeignTable
```

また、CASCADE オプションを指定して DROP FOREIGN SERVER コマンドを使用すると、外部サーバとそこに定義されているすべての外部テーブルを削除できます。

SQL

```
DROP FOREIGN SERVER Example.PostgresDB CASCADE
```

11.12 テーブルのリスト

INFORMATION.SCHEMA.TABLES 永続クラスは、現在のネームスペース内のすべてのテーブル (およびビュー) に関する情報を表示します。これには、スキーマ名とテーブル名、テーブルの所有者、新しいレコードを挿入できるかどうかなど、さまざまなプロパティが含まれます。TABLETYPE プロパティは、ベース・テーブルなのかビューなのかを示します。

以下の例では、現在のネームスペース内のすべてのテーブルとビューに関するテーブル・タイプ、スキーマ名、テーブル名、および所有者が返されます。

SQL

```
SELECT Table_Type,Table_Schema,Table_Name,Owner FROM INFORMATION_SCHEMA.TABLES
```

INFORMATION.SCHEMA.CONSTRAINTTABLEUSAGE 永続クラスは、現在のネームスペースの各テーブルに定義された主キー (明示的または暗黙的)、外部キー、または一意制約ごとに 1 つの行を表示します。
INFORMATION.SCHEMA.KEYCOLUMNUSAGE は、現在のネームスペースの各テーブルのこれらのいずれかの制約の一部として定義されたフィールドごとに、1 つの行を表示します。

管理ポータルの SQL インタフェースの [カタログの詳細] タブを使用して、単一テーブルに対してほぼ同じ情報を表示できます。

11.13 列の名前と番号のリスト

指定したテーブルのすべての列名 (フィールド名) をリストできる方法として、次の 4 つがあります。

- GetAllColumns() メソッド。これは、非表示の列を含め、すべての列名および列番号をリストします。GetVisibleColumns() メソッドは、非表示でないすべての列をリストします。ID (RowID) フィールドは非表示であることも非表示でないこともあります。x_classname 列は常に非表示です。これは、永続クラスが Final クラス・キーワードを指定して定義されていない限り、自動的に定義されます。
- 管理ポータルの SQL インタフェース ([**システム・エクスプローラ**]→[SQL]) のスキーマ・コンテンツの [カタログの詳細] タブ。これは、すべての列名と列番号 (非表示の列を含む)、およびその他の情報 (データ型、および列が非表示かどうかを示すフラグなど) をリストします。
- `SELECT TOP 0 * FROM` tablename。これは、非表示でないすべての列の名前を列番号順にリストします。非表示の列はどこでも列番号順に表示できるため、非表示でない列の名前を数えることで列番号を特定することはできないことに注意してください。アスタリスク構文の詳細は、"SELECT" コマンドを参照してください。
- INFORMATION.SCHEMA.COLUMNS 永続クラスは、現在のネームスペースの各テーブルまたはビューの、非表示でない各列の行をリストします。INFORMATION.SCHEMA.COLUMNS は、テーブルおよびビューの列の特性をリストするための、多くのプロパティを提供します。非表示のフィールドはカウントされないため、ORDINALPOSITION は列番号と同じではないことに注意してください。GetAllColumns() メソッドは、非表示のフィールドと非表示でないフィールドの両方をカウントします。

  以下の例は、INFORMATION.SCHEMA.COLUMNS を使用していくつかの列のプロパティをリストします。

SQL

```
SELECT TABLE_NAME,COLUMN_NAME,ORDINAL_POSITION,DATA_TYPE,CHARACTER_MAXIMUM_LENGTH,
       COLUMN_DEFAULT,IS_NULLABLE,UNIQUE_COLUMN,PRIMARY_KEY
       FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA='Sample'
```

11.13.1 列の取得メソッド

テーブル内の列の名前を列番号順にリストするには、GetAllColumns() または GetVisibleColumns() メソッドを次のように使用します。

ObjectScript

```
SET stat=##class(%SYSTEM.SQL.Schema).GetAllColumns("Sample.Person",.byname,.bynum)
IF stat=1 {
  SET i=1
  WHILE $DATA(bynum(i)) { WRITE "name is ",bynum(i),"   col num is ",i,!
                          SET i=i+1 }
}
ELSE { WRITE "GetAllColumns() cannot locate specified table" }
```

GetAllColumns() は、非表示の列を含め、定義されているすべての列をリストします。テーブルが埋め込み %SerialObject クラスを参照する場合、GetAllColumns() はまず永続クラスのすべての列をリストし (%SerialObject を参照するプロパティを含め)、次にすべての %SerialObject プロパティをリストします。これを、以下の GetAllColumns() の結果に示します。

```
name is ID   col num is 1
name is Age   col num is 2
name is Home   col num is 3
name is Name   col num is 4
name is x_classname   col num is 5
name is Home_City   col num is 6
name is Home_Phone   col num is 7
name is Home_Phone_AreaCode   col num is 8
name is Home_Phone_Country   col num is 9
name is Home_Phone_TNum   col num is 10
name is Home_PostalCode   col num is 11
name is Home_State   col num is 12
name is Home_Street   col num is 13
```

このメソッドを次のように使用して、指定した列名の列番号を特定することもできます。

ObjectScript

```
SET stat=##class(%SYSTEM.SQL.Schema).GetAllColumns("Sample.Person",.byname)
IF stat=1 {
       WRITE "Home_State is column number ",byname("Home_State"),!  }
ELSE { WRITE "GetAllColumns() cannot locate specified table" }
```

11.14 制約のリスト

INFORMATION.SCHEMA.TABLECONSTRAINTS 永続クラスは、テーブル名、制約タイプ、および制約名をリストします。制約タイプには、UNIQUE、PRIMARY KEY、および FOREIGN KEY があります。テーブル定義で制約の名前を指定しなかった場合、制約名は、テーブル名、制約タイプ、およびテーブルの列番号から生成されます。例えば、MYTABLE_UNIQUE3 のようになります。これを、以下の例に示します。

SQL

```
SELECT Table_Schema,Table_Name,Constraint_Type,Constraint_Name FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
```

INFORMATION.SCHEMA.CONSTRAINTCOLUMNUSAGE 永続クラスは、テーブル名、列名、および制約名をリストします。制約に複数の列が含まれる場合は、列ごとに個々の項目がリストされます。テーブル定義で制約の名前を指定しな

かった場合、制約名は、テーブル名、制約タイプ、およびテーブルの列番号から生成されます。例えば、MYTABLE_UNIQUE3 のようになります。これを、以下の例に示します。

SQL

```
SELECT Table_Schema,Table_Name,Column_Name,Constraint_Name FROM INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE
```

INFORMATION.SCHEMA.REFERENTIALCONSTRAINTS 永続クラスは、参照するテーブルを含む外部キー制約 (CONSTRAINT_SCHEMA、CONSTRAINT_TABLE_NAME)、参照されるテーブル (UNIQUE_CONSTRAINT_SCHEMA、UNIQUE_CONSTRAINT_TABLE)、外部キー名 (CONSTRAINT_NAME)、および UPDATE と DELETE の参照アクション (UPDATE_RULE、DELETE_RULE) とその値 NO ACTION、SET DEFAULT、SET NULL、または CASCADE をリストします。これを、以下の例に示します。

SQL

```
SELECT Constraint_Table_Name,Unique_Constraint_Table,Constraint_Name,Update_Rule,Delete_Rule FROM
INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS
```

12
ビューの定義と使用

ビューは、1 つの SELECT 文、または複数の SELECT 文の UNION を使用して、実行時に 1 つまたは複数の物理テーブルから取得したデータで構成される仮想テーブルです。SELECT は、テーブルまたはその他のビューの任意の組み合わせを指定して、データにアクセスできます。ビューは、物理テーブルの柔軟性とセキュリティ権限をすべて提供するストアド・クエリです。

SELECT クエリで指定されたフィールド、およびビューの SELECT 節で指定された同じフィールドには、同じ Selectivity 値が使用されます。ビューの行の分散がソース・テーブルとは異なる場合があることに注意してください。これはビュー・フィールドの選択性の正確性に影響する可能性があります。

InterSystems IRIS® データ・プラットフォーム上の InterSystems SQL では、ビューを定義し、ビューに対するクエリを実行することができます。すべてのビューは、更新可能にも読み取り専用にもできます。これについてはこの章で後述します。

注釈　読み取り専用としてマウントされたデータベースに格納されているデータでビューを作成することはできません。

ODBC または JDBC ゲートウェイ接続経由でリンクされている Informix テーブルに格納されているデータでビューを作成することはできません。これは、InterSystems IRIS クエリ変換では、このタイプのクエリの場合に FROM 節でサブクエリが使用されるからです。Informix では、FROM 節のサブクエリはサポートされていません。インターシステムズによる Informix SQL のサポートの詳細は、"ISQL 移行ガイド" を参照してください。

12.1 ビューの作成

以下のような方法で、ビューを定義できます。

- SQL 文の CREATE VIEW コマンドを使用します (DDL スクリプトで、または JDBC か ODBC 経由)。
- 管理ポータルの [ビュー作成] インタフェースを使用します。

ビュー名は、修飾、未修飾のどちらでもかまいません。未修飾のビュー名は、単純な識別子です (`MyView`)。修飾されたビュー名は、2 つの単純な識別子であるスキーマ名とビュー名で構成され、ピリオドで区切られています (`MySchema.MyView`)。ビュー名とテーブル名は同じ名前付け規約に従い、未修飾の名前の場合は同じスキーマ名解析を実行します。ただし、同じスキーマ内のビューおよびテーブルには、同じ名前を指定できません。

$SYSTEM.SQL.Schema.ViewExists() メソッドを使用して、ビュー名が既に存在するかどうかを確認できます。このメソッドは、ビューを投影したクラス名も返します。$SYSTEM.SQL.Schema.TableExists() メソッドを使用して、テーブル名が既に存在するかどうかを確認できます。

ビューを利用することで、テーブルの制限されたサブセットを作成できます。以下の埋め込み SQL の例は、このビューでアクセスできる元のテーブルの行 (WHERE 節を使用) と列 (Sample.Person に列が 3 つ以上含まれることを想定) の両方を制限するビューを作成します。

ObjectScript

```
&sql(CREATE VIEW Sample.VSrStaff
     AS SELECT Name AS Vname,Age AS Vage
     FROM Sample.Person WHERE Age>75)
IF SQLCODE=0 {WRITE "Created a view",!}
ELSEIF SQLCODE=-201 {WRITE "View already exists",!}
ELSE {WRITE "Serious SQL problem: ",SQLCODE," ",%msg,! }
```

SQL

```
SELECT * FROM Sample.VSrStaff ORDER BY Vage
```

以下の埋め込み SQL の例は、SalesPeople テーブルに基づくビューを作成し、新しい計算値列 TotalPay を作成します。

ObjectScript

```
&sql(CREATE VIEW Sample.VSalesPay AS
 SELECT Name,(Salary + Commission) AS TotalPay
 FROM Sample.SalesPeople)
IF SQLCODE=0 {WRITE "Created a view",!}
ELSEIF SQLCODE=-201 {WRITE "View already exists",!}
ELSE {WRITE "Serious SQL problem: ",SQLCODE," ",%msg,! }
```

12.1.1 管理ポータルの [ビュー作成] インタフェース

管理ポータルからビューを作成できます。InterSystems IRIS 管理ポータルに移動します。**[システム・エクスプローラ]** から **[SQL]** を選択します。ページ上部の **[切り替え]** オプションを使ってネームスペースを選択します。利用可能なネームスペースのリストが表示されます。ネームスペースを選択したら、**[アクション]** ドロップダウン・リストをクリックし、**[ビューを作成]** をクリックします。

これにより、[ビュー作成] ウィンドウが開き、以下のフィールドが表示されます。

- [スキーマ]: 既存スキーマ内にビューを含めるか、新規スキーマを作成するかを決定できます。既存スキーマを選択する場合は、既存スキーマのドロップダウン・リストが提供されます。新規スキーマを作成する場合は、スキーマ名を入力します。いずれの場合も、スキーマを省略すると、InterSystems IRIS では既定のスキーマ名が使用されます。
- [ビュー名]: 有効なビュー名です。同じスキーマ内でテーブルとビューに同じ名前を使用することはできません。
- [チェックオプション]: オプションは、[読込専用]、[ローカル]、[カスケード] です。
- [to _PUBLIC ビューにすべての権限を付与する]: 選択した場合、すべてのユーザにこのビューの実行権限が付与されます。既定では、すべてのユーザにそのビューへのアクセス権を付与しません。
- [テキスト表示]: 以下の 3 つの方法でテキスト表示を指定できます。
  - [テキスト表示] 領域に SELECT 文を入力します。
  - クエリ・ビルダを使用して SELECT 文を作成してから **[OK]** を押すと、このクエリが [テキスト表示] 領域に読み込まれます。
  - 管理ポータルの SQL インタフェースの左側にあるクエリ・キャッシュ名 (`%sqlcq.USER.cls4` など) を選択してから **[ビュー作成]** を起動すると、このクエリ・キャッシュが [テキスト表示] 領域に読み込まれます。[テキスト表示] 領域では、保存する前に、ホスト変数参照 (疑問符) を実際の値と置換する必要があることに注意してください。

12.1.2 ビューおよび対応するクラス

ビューを定義すると、InterSystems IRIS によって対応するクラスが生成されます。SQL ビュー名を使用し、名前変換ルールに従って対応する一意のクラス名が生成されます。管理ポータルの SQL インタフェースには、このクラス名を含む、既存のビューの **[カタログの詳細]** が表示されます。"ビューのカタログの詳細" を参照してください。

12.2 ビューの変更

管理ポータルの SQL インタフェースでは、既存のビューを選択して、そのビューの **[カタログの詳細]** を表示できます。[カタログの詳細] の **[情報表示]** オプションでは、テキスト表示 (ビューの SELECT 文) を編集するためのインタフェースを提供する **[ビュー編集]** リンクが表示されます。**[チェックオプション]** に [なし]、[読取専用]、[ローカル]、[カスケード] を選択するためのドロップダウン・リストも表示されます。

12.3 更新可能なビュー

更新可能なビューでは、INSERT、UPDATE、および DELETE 演算を実行できます。以下の条件に当てはまる場合のみ、ビューを更新できます。

- ビューのクエリの FROM 節にテーブル参照が 1 つだけ含まれる場合。このテーブル参照は、更新可能な元のテーブルか更新可能なビューと一致する必要があります。
- ビューのクエリの SELECT リスト内の値式が、すべて列参照である場合。
- ビューのクエリが、GROUP BY、HAVING、または SELECT DISTINCT を指定していない場合。
- ビューは、ビューとして投影されたクラス・クエリでない場合。
- ビューのクラスにクラス・パラメータ READONLY=1 が含まれていない場合 (ビュー定義に WITH READ ONLY 節が含まれる場合)。

12.3.1 WITH CHECK オプション

ビューで INSERT または UPDATE 操作を実行すると、本来のビュー・テーブルの一部ではない行が基本テーブルに表れることがあります。これを防ぐために、InterSystems SQL はビュー定義で WITH CHECK OPTION 節をサポートしています。この節は、更新可能なビューでのみ使用できます。

WITH CHECK OPTION 節は、更新可能なビューで INSERT または UPDATE 演算を実行するときに、ビュー定義の WHERE 節に対して得られた行を必ず検証するように指定し、挿入あるいは変更された行が本来のビュー・テーブルの一部になっていることを確認します。

例えば、以下の DDL 文は、GPA (grade point average : 成績の指標) が高い生徒のデータを含む、更新可能な GoodStudent ビューを定義します。

SQL

```
CREATE VIEW GoodStudent AS
    SELECT Name, GPA
      FROM Student
        WHERE GPA > 3.0
    WITH CHECK OPTION
```

ビューに WITH CHECK OPTION が含まれるので、GoodStudent ビューの中で GPA が 3.0 以下の値を持つ行を INSERT または UPDATE しようとしてもできません (それらの行は、“good student” を表すものではないからです)。

WITH CHECK OPTION には、以下の 2 種類があります。

- WITH LOCAL CHECK OPTION は、INSERT または UPDATE 文で指定されたビューの WHERE 節のみがチェックされることを示します。
- WITH CASCADED CHECK OPTION (および WITH CASCADE CHECK OPTION) は、INSERT または UPDATE 文で指定されたビューの WHERE 節とそのビューが元にしている ALL ビューがチェックされることを示します。これは、外観やそれらの定義内の WITH LOCAL CHECK OPTION 節の存在の有無には関係ありません。

WITH CHECK OPTION のみが指定されている場合は、既定は CASCADED です。

UPDATE または INSERT の実行時に、すべての既定値とトリガされた計算フィールドが、下位のテーブルのフィールドのために計算された後で、通常のテーブルの検証 (必須フィールド、データ型の検証、制約など) の前に、WITH CHECK OPTION 条件がチェックされます。

WITH CHECK OPTION の検証が済むと、INSERT または UPDATE が基本テーブル自体で実行されているかのように、INSERT または UPDATE 演算を続けます。すべての制約が確認され、トリガを発生します。

INSERT または UPDATE 文で %NOCHECK オプションが指定されている場合、WITH CHECK OPTION 検証は確認されません。

以下は、WITH CHECK OPTION 検証に関連する 2 つの SQLCODE 値です (INSERT または UPDATE は生成されたビュー・テーブルには存在しない行を生成することがあります)。

- SQLCODE -136 ー ビューの WITH CHECK OPTION 検証が INSERT に失敗しました。
- SQLCODE -137 ー ビューの WITH CHECK OPTION 検証が UPDATE に失敗しました。

12.4 読み取り専用ビュー

読み取り専用ビューでは、INSERT、UPDATE、および DELETE 演算を実行できません。更新可能なビューの基準を満たさないビューは、読み取り専用です。

ビュー定義で WITH READ ONLY 節を指定すると、そのビューは読み取り専用になります。

読み取り専用ビューに INSERT、UPDATE、DELETE 文をコンパイル、作成しようとすると、エラー・コード -35 が生成されます。

12.5 ビュー ID : %VID

InterSystems IRIS は、ビューまたは FROM 節のサブクエリで返される各行に整数のビュー ID (%VID) を割り当てます。テーブルの行 ID 番号と同様、これらのビューの行 ID 番号は、システムによって割り当てられる一意の非 NULL、非ゼロ値で、変更することはできません。この %VID は、通常、ユーザには表示されず、明示的に指定した場合にのみ返されます。これはデータ型 INTEGER として返されます。%VID の値は連続した整数であるため、ビューが順序付きのデータを返す場合はとても有意義なものになります。TOP 節と組み合わせるときには、ビューでは ORDER BY 節以外は使用できません。以下の埋め込み SQL の例では、VSrStaff というビューを作成します。

ObjectScript

```
&sql(CREATE VIEW Sample.VSrStaff
     AS SELECT TOP ALL Name AS Vname,Age AS Vage
     FROM Sample.Person WHERE Age>75
     ORDER BY Name)
IF SQLCODE=0 {WRITE "Created a view",!}
ELSEIF SQLCODE=-201 {WRITE "View already exists",!}
ELSE {WRITE "Serious SQL problem: ",SQLCODE," ",%msg,! }
```

以下の例は、VSrStaffビューで定義されているすべてのデータを（SELECT＊を使用して）返します。また、各行のビューIDも返すように指定しています。テーブルの行 IDとは異なり、ビューの行 IDはアスタリスク構文の使用時には表示されません。表示されるのは、SELECT で明示的に指定された場合のみです。

SQL

```
SELECT *,%VID AS ViewID FROM Sample.VSrStaff
```

%VID を使用すると、以下の例のように、ビューから SELECT で返される行の数をさらに制限することができます。

SQL

```
SELECT *,%VID AS ViewID FROM Sample.VSrStaff WHERE %VID BETWEEN 5 AND 10
```

このように、TOP の代わりに（または TOP に加えて）%VIDを使用すると、クエリで返される行の数を制限できるようになります。一般に、TOP 節は、データ・レコードの小さなサブセットを返すために使用されます。%VID は、小さなサブセットでレコードを返すことで、ほとんどのデータ・レコードまたはすべてのデータ・レコードを返すために使用されます。この機能は、特に Oracle のクエリを移植する場合に便利です（%VID は Oracle の ROWNUM に簡単に対応します）。ただし、以下に示すように TOP と比べると、%VID の使用には、いくつかのパフォーマンス上の制限があることを認識しておく必要があります。

- %VID は、最初の行を返す時間を最適化しません。TOP は、データの最初の行をできる限り短時間で返すように最適化します。%VID は、すべてのデータ・セットを可能な限り短時間で返すように最適化します。
- %VID は、ソートされた結果をクエリで指定した場合、制限付きソート（TOP で実行される特別な最適化）を実行しません。クエリは、まずデータ・セット全体をソートしてから、%VID を使用して、返されるデータ・セットを制限します。TOP はソートの前に適用されるため、SELECT は、制限された行のサブセットだけを対象とした制限付きソートを実行します。

最初の行を返す時間の最適化と制限付きソートの最適化を保持するには、TOP と %VID を組み合わせた FROM 節のサブクエリを使用します。FROM 節のサブクエリで、TOP ALL を使用するのではなく、TOP の値として上限（ここでは 10）を指定します。WHERE 節で、%VID を使用して下限（ここでは >4）を指定します。以下の例は、この方法を使用して前述のビューのクエリと同じ結果を返します。

SQL

```
SELECT *,%VID AS SubQueryID
   FROM (SELECT TOP 10 Name,Age
         FROM Sample.Person
         WHERE Age > 75
         ORDER BY Name)
   WHERE %VID > 4
```

%PARALLEL キーワードを明示的に指定していても、%VID を指定するクエリに対して並列実行を行うことはできません。

12.6 ビューのプロパティのリスト

INFORMATION.SCHEMA.VIEWS 永続クラスは、現在のネームスペース内のすべてのビューに関する情報を表示します。ビュー定義、ビューの所有者、ビューが作成されたときと最後に変更されたときのタイムスタンプなど、さまざまなプロパティが表示されます。これらのプロパティには、ビューが更新可能かどうか、更新可能な場合は、チェック・オプションを指定して定義されているかどうかも含まれます。

埋め込み SQL で指定する場合、INFORMATION.SCHEMA.VIEWS には、#include %occInclude マクロ・プリプロセッサ指示文が必要です。この指示文は、ダイナミック SQL の場合は不要です。

VIEWDEFINITION プロパティ (SqlFieldName = VIEW_DEFINITION) は、現在のネームスペース内のすべてのビューのビュー・フィールド名およびビューのクエリ式を文字列として返します。以下に例を示します。

SQL

```
SELECT View_Definition FROM INFORMATION_SCHEMA.VIEWS
```

`"(vName,vAge) SELECT Name,Age FROM Sample.Person WHERE Age > 21"` のような文字列を返します。管理ポータルの SQL インタフェース [クエリ実行] から発行した場合、ビュー・フィールド・リストの間にスペースが 1 つ入れられ、空白と改行が削除されて、(必要に応じて) 内容が切り捨てられたことを示す省略記号 (...) が付加された文字列を返します。それ以外の場合、このクエリを発行すると、ビュー・フィールド・リストとクエリ・テキストの間に改行が入れられ、ビューのクエリ式に指定された空白が保持されて、ビューごとに最大 1048576 文字の文字列が返され、(必要に応じて) 内容が切り捨てられたことを示す省略記号 (...) が付加されます。

以下の例では、現在のネームスペース内のすべてのビューのビュー名 (Table_Name フィールド) および所有者名が返されます。

SQL

```
SELECT Table_Name,Owner FROM INFORMATION_SCHEMA.VIEWS
```

以下の例では、現在のネームスペース内のすべての非システム・ビューに関するすべての情報が返されます。

SQL

```
SELECT * FROM INFORMATION_SCHEMA.VIEWS WHERE Owner != '_SYSTEM'
```

INFORMATION.SCHEMA.VIEWCOLUMNUSAGE 永続クラスは、現在のネームスペース内の各ビューの、ソース・テーブルのフィールド名を表示します。

SQL

```
SELECT * FROM INFORMATION_SCHEMA.VIEW_COLUMN_USAGE WHERE View_Name='MyView'
```

管理ポータルの SQL インタフェースの [カタログの詳細] タブを使用して、単一ビューに対して INFORMATION.SCHEMA.VIEWS を指定した場合とほぼ同じ情報を表示できます。ビューの [カタログの詳細] には、各ビュー・フィールドの定義 (data type (データ型)、max length (最大長)、minval/maxval (最小/最大有効値) など) が表示され、INFORMATION.SCHEMA ビュー・クラスでは提供されない詳細も含まれています。[カタログの詳細] の [ビュー情報] の表示では、ビュー定義を編集するオプションも用意されています。

12.7 ビューの依存関係のリスト

INFORMATION.SCHEMA.VIEWTABLEUSAGE 永続クラスは、現在のネームスペースのビューのすべておよび依存テーブルを表示します。これを以下の例で示しています。

SQL

```
SELECT View_Schema,View_Name,Table_Schema,Table_Name FROM INFORMATION_SCHEMA.VIEW_TABLE_USAGE
```

%Library.SQLCatalog.SQLViewDependsOn クラス・クエリを呼び出すことで、指定したビューが依存しているテーブルをリストできます。このクラス・クエリに、schema.viewname を指定します。viewname のみを指定すると、システム全体の既定のスキーマ名が使用されます。このクラス・クエリを実行するには、指定したビューに対する特権を呼び出し側が持っている必要があります。これを以下の例で示しています。

ObjectScript

```
  SET statemt=##class(%SQL.Statement).%New()
  SET cqStatus=statemt.%PrepareClassQuery("%Library.SQLCatalog","SQLViewDependsOn")
    IF cqStatus'=1 {WRITE "%PrepareClassQuery failed:" DO $System.Status.DisplayError(cqStatus) QUIT}
  SET rset=statemt.%Execute("vschema.vname")
  DO rset.%Display()
```

この SQLViewDependsOn クエリは、ビューが依存しているテーブルをリストします。テーブル・スキーマに続けてテーブル名がリストされます。呼び出し側に、ビューが依存するテーブルに応じた特権がない場合、そのテーブルとテーブルのスキーマは〈NOT PRIVILEGED〉としてリストされます。これにより、テーブルの特権を持たない呼び出し側は、ビューが依存しているテーブル数を判断できるようになります。ただし、その呼び出し側では、テーブルの名前は判断できません。

13
テーブル間のリレーションシップ

外部キーを定義して、テーブル間の参照整合性を強制できます。外部キー制約を持つテーブルを変更する際に、外部キー制約が確認されます。

13.1 外部キーの定義

InterSystems SQL で外部キーを定義するには、以下のような方法があります。

- 2 つのクラスの間にリレーションシップを定義できます。リレーションシップを定義すると、自動的に SQL に外部キー制約が投影されます。リレーションシップの詳細は、"クラスの定義と使用" を参照してください。
- クラス定義に、明示的に外部キー定義を追加できます (リレーションシップによって定義できない場合)。詳細は、"クラス定義リファレンス" の "外部キー定義" を参照してください。
- CREATE TABLE コマンドまたは ALTER TABLE コマンドを使用して、外部キー定義を追加できます。また、ALTER TABLE コマンドを使用して、外部キーを削除できます。これらのコマンドについては、"InterSystems SQL リファレンス" を参照してください。

外部キー参照として使用される RowID フィールドは、パブリックである必要があります。パブリック (またはプライベート) RowID フィールドを含むテーブルを定義する方法については、"RowID は非表示か" を参照してください。

1 つのテーブル (クラス) のための外部キーの最大数は 400 です。

13.2 外部キーの参照整合性チェック

外部キー制約では、更新または削除時の参照アクションを指定できます。DDL を使用したこの参照アクションの定義については、"CREATE TABLE" の "参照アクション節" を参照してください。テーブルに投影される永続クラスを使用したこの参照アクションの定義については、"クラス定義リファレンス" の "OnDelete" および "OnUpdate" Foreign Key キーワードで定義されています。シャード・テーブルの作成時には、これらの参照アクションを NO ACTION に設定する必要があります。

既定では、InterSystems IRIS® データ・プラットフォームは INSERT、UPDATE、DELETE の各操作に対して、外部キーの参照整合性チェックを実行します。操作が参照整合性に違反する場合、その操作は行われず、その操作によって SQLCODE -121、-122、-123、-124 のいずれかのエラーが発行されます。参照整合性チェックに失敗すると、以下のようなエラーが生成されます。

```
ERROR #5540: SQLCODE: -124 Message: At least 1 Row exists in table 'HealthLanguage.FKey2'
which references key NewIndex1 - Foreign Key Constraint 'NewForeignKey1' (Field 'Pointer1')
failed on referential action of NO ACTION [Execute+5^IRISSql16:USER]
```

他の方法でテーブルにデータを入力する場合、参照整合性制約が適用されない可能性があります。これが心配な場合は、テーブル・データを検証してください。

このチェックをシステム全体で抑制するには、$SYSTEM.SQL.Util.SetOption() メソッドを `SET status=$SYSTEM.SQL.Util.SetOption("FilerRefIntegrity",0,.oldval)` のように使用します。既定値は 1 です (参照整合性制約チェックが実行されます)。現在の設定を確認するには、$SYSTEM.SQL.CurrentSettings() を呼び出します。

既定では、外部キーを持つ行が削除されると、InterSystems IRIS は対応する参照先テーブルの行に対する長期間 (トランザクションが終了するまで) の共有ロックを取得します。これにより、参照している行で DELETE トランザクションが完了するまで、参照先の行を更新または削除できなくなります。これにより、参照先の行が削除され、その後、参照している行の削除がロール・バックされる状況を回避することができます。もし、このような状況が発生した場合、外部キーは存在しない行を参照することになります。外部キーが NoCheck を指定して定義されている場合、または参照している行の DELETE が %NOCHECK または %NOLOCK と共に指定されている場合、このロックは取得されません。

テーブルの定義に永続クラス定義を使用している場合、NoCheck キーワードを指定して外部キーを定義すると、その外部キーの今後のチェックを抑制できます。CREATE TABLE にはこのキーワード・オプションはありません。

特定の操作のチェックを抑制するには、%NOCHECK キーワード・オプションを使用します。

既定では、InterSystems IRIS は次の各操作に対しても、外部キーの参照整合性チェックを実行します。指定されたアクションが参照整合性に違反する場合、このコマンドは実行されません。

- ALTER TABLE DROP COLUMN。
- ALTER TABLE DROP CONSTRAINT。SQLCODE -317 を発行します。外部キーの整合性チェックは、SET OPTION COMPILEMODE=NOCHECK を使用して抑制できます。
- DROP TABLE。SQLCODE -320 を発行します。外部キーの整合性チェックは、SET OPTION COMPILE-MODE=NOCHECK を使用して抑制できます。
- TRUNCATE TABLE (DELETE と同様の考慮事項が適用されます)。
- トリガ・イベント (BEFORE イベントなど)。例えば、DELETE 操作が外部キーの参照整合性に違反するために実行されない場合、BEFORE DELETE トリガは実行されません。

親子リレーションシップでは、子の順序は定義されていません。アプリケーション・コードは特定の順序に依存しないでください。

13.3 親テーブルと子テーブル

このセクションでは、親/子リレーションシップの定義および操作の概要について説明します。詳細は、"クラスの定義と使用" の "リレーションシップの定義と使用" の章を参照してください。

13.3.1 親テーブルと子テーブルの定義

テーブルに投影される永続クラスを定義する際、リレーションシップ・プロパティを使用して、2 つのテーブル間の親子リレーションシップを指定できます。

以下の例では、親テーブルを定義します。

Class Definition

```
Class Sample.Invoice Extends %Persistent
{
Property Buyer As %String(MAXLEN=50) [Required];
Property InvoiceDate As %TimeStamp;
Relationship Pchildren AS Sample.LineItem [ Cardinality = children, Inverse = Cparent ];
}
```

以下の例では、子テーブルを定義します。

Class Definition

```
Class Sample.LineItem Extends %Persistent
{
Property ProductSKU As %String;
Property UnitPrice As %Numeric;
Relationship Cparent AS Sample.Invoice [ Cardinality = parent, Inverse = Pchildren ];
}
```

管理ポータルの SQL インタフェース [カタログの詳細] タブの [**テーブル情報**] で、子テーブルまたは親テーブル (あるいはその両方) の名前を指定します。子テーブルの場合は、`Cparent->Sample.Invoice` など、親テーブルへの参照を指定します。

子テーブル自体が、子テーブルの親になることができます (このような子の子は、"孫" テーブルと呼ばれます)。この場合、[**テーブル情報**] で親テーブルと子テーブル両方の名前を指定します。

13.3.2 親テーブルと子テーブルへのデータの挿入

親テーブルに各レコードを挿入してから、子テーブルに対応するレコードを挿入する必要があります。以下に例を示します。

```
INSERT INTO Sample.Invoice (Buyer,InvoiceDate) VALUES ('Fred',CURRENT_TIMESTAMP)

INSERT INTO Sample.LineItem (Cparent,ProductSKU,UnitPrice) VALUES (1,'45-A7',99.95)
INSERT INTO Sample.LineItem (Cparent,ProductSKU,UnitPrice) VALUES (1,'22-A1',0.75)
```

対応する親レコード ID がない子レコードを挿入しようとすると、SQLCODE -104 エラーが生成され、%msg は、"`'Sample.LineItem'` " のようになります。

子テーブルへの INSERT 操作時には、親テーブル内の対応する行に共有ロックがかかります。この行は、子テーブル行の挿入中はロックされます。その後、ロックは解除されます (トランザクションの終了までロック状態が継続することはありません)。これにより、参照される親の行がこの挿入操作の間に変更されることがなくなります。

13.3.3 親テーブルと子テーブルの識別

埋め込み SQL では、ホスト変数配列を使用して、親テーブルと子テーブルを識別できます。子テーブルでは、ホスト変数配列の添え字 0 は `parentref` の形式で親参照 (Cparent) に設定され、添え字 1 は `parentref||childref` の形式で子レコード ID に設定されています。親テーブル場合、添え字 0 は未定義です。詳細は、以下の例を参照してください。

ObjectScript

```
   KILL tflds,SQLCODE,C1
   &sql(DECLARE C1 CURSOR FOR
        SELECT *,%TABLENAME INTO :tflds(),:tname
        FROM Sample.Invoice)
   &sql(OPEN C1)
        IF SQLCODE<0 {WRITE "Serious SQL Error:",SQLCODE," ",%msg  QUIT}
    &sql(FETCH C1)
       IF SQLCODE=100 {WRITE "The ",tname," table contains no data",!  QUIT}
       WHILE $DATA(tflds(0)) {
                                WRITE tname," is a child table",!,"parent ref: ",tflds(0)," %ID:
",tflds(1),!
                                &sql(FETCH C1)
                                IF SQLCODE=100 {QUIT}
                               }
      IF $DATA(tflds(0))=0 {WRITE tname," is a parent table",!}
    &sql(CLOSE C1)
      IF SQLCODE<0 {WRITE "Error closing cursor:",SQLCODE," ",%msg  QUIT}
```

ObjectScript

```
   KILL tflds,SQLCODE,C1
   &sql(DECLARE C1 CURSOR FOR
        SELECT *,%TABLENAME INTO :tflds(),:tname
        FROM Sample.LineItem)
   &sql(OPEN C1)
       IF SQLCODE<0 {WRITE "Serious SQL Error:",SQLCODE," ",%msg  QUIT}
   &sql(FETCH C1)
       IF SQLCODE=100 {WRITE "The ",tname," table contains no data",! QUIT}
       WHILE $DATA(tflds(0)) {
                                WRITE tname," is a child table",!,"parent ref: ",tflds(0)," %ID:
",tflds(1),!
                                &sql(FETCH C1)
                                IF SQLCODE=100 {QUIT}
                               }
      IF $DATA(tflds(0))=0 {WRITE tname," is a parent table",!}
  &sql(CLOSE C1)
    IF SQLCODE<0 {WRITE "Error closing cursor:",SQLCODE," ",%msg  QUIT}
```

子テーブルの場合、tflds(0) および tflds(1) は以下のような値を返します。

```
parent ref: 1 %ID: 1||1
parent ref: 1 %ID: 1||2
parent ref: 1 %ID: 1||3
parent ref: 1 %ID: 1||9
parent ref: 2 %ID: 2||4
parent ref: 2 %ID: 2||5
parent ref: 2 %ID: 2||6
parent ref: 2 %ID: 2||7
parent ref: 2 %ID: 2||8
```

"孫" テーブルの場合 (子テーブルの子であるテーブル)、tflds(0) および tflds(1) は以下のような値を返します。

```
parent ref: 1||1 %ID: 1||1||1
parent ref: 1||1 %ID: 1||1||7
parent ref: 1||1 %ID: 1||1||8
parent ref: 1||2 %ID: 1||2||2
parent ref: 1||2 %ID: 1||2||3
parent ref: 1||2 %ID: 1||2||4
parent ref: 1||2 %ID: 1||2||5
parent ref: 1||2 %ID: 1||2||6
```

14
トリガの使用法

この章では、InterSystems SQL でトリガを定義する方法を説明します。トリガは、特定の SQL イベントに応答して実行されるコード行です。

14.1 トリガの定義

特定のテーブルにトリガを定義するには、以下のような方法があります。

- SQL テーブルに投影される永続クラス定義にトリガ定義を含めます。例えば、MyApp.Person クラスのこの定義には LogEvent トリガの定義が含まれます。これは、MyApp.Person テーブルへのデータの INSERT が成功するたびに呼び出されます。

  Class/ObjectScript

  ```
  Class MyApp.Person Extends %Persistent [DdlAllowed]
  {
      // ... Class Property Definitions

      Trigger LogEvent [ Event = INSERT, Time = AFTER ]
      {
          // Trigger code to log an event
      }
  }
  ```

  Class/Python

  ```
  Class MyApp.Person Extends %Persistent [DdlAllowed]
  {
      // ... Class Property Definitions

      Trigger LogEvent [ Event = INSERT, Time = AFTER, Language = python ]
      {
          // Trigger code to log an event
      }
  }
  ```

  詳細は、"トリガ定義" を参照してください。

- SQL CREATE TRIGGER コマンドを使用して、トリガを作成します。これにより、対応する永続クラスでトリガ・オブジェクト定義が生成されます。SQL トリガ名は、識別子の名前付け規約に従います。InterSystems IRIS® データ・プラットフォームでは、SQL トリガ名を使用して対応するトリガ・クラス・エンティティ名を生成します。

トリガを作成するには、%CREATE_TRIGGER 管理者レベル特権が必要です。トリガを削除するには、%DROP_TRIGGER 管理者レベル特権が必要です。

1 つのクラスのユーザ定義トリガの最大数は 200 です。

注釈　InterSystems IRIS は、コレクションによって投影されたテーブルのトリガをサポートしていません。ユーザはこのようなトリガを定義することはできず、子テーブルとしてのコレクションの投影では、その基本コレクションに関わるトリガは考慮されません。

InterSystems IRIS では、Security.Roles および Security.Users テーブルを変更するトリガはサポートされません。

14.2 トリガのタイプ

トリガは、以下によって定義します。

- トリガの実行の要因となるイベントのタイプ。トリガには、シングルイベント・トリガとマルチイベント・トリガがあります。シングルイベント・トリガは、指定したテーブルで INSERT、UPDATE、または DELETE のいずれかのイベントが発生したときに実行するように定義します。マルチイベント・トリガは、指定したテーブルで、指定した複数のイベントのいずれか 1 つが発生したときに実行するように定義します。クラス定義または CREATE TRIGGER コマンドを使用して、INSERT/UPDATE、UPDATE/DELETE、または INSERT/UPDATE/DELETE のマルチイベント・トリガを定義できます。イベントのタイプは、クラス定義で、必須の Event トリガ・キーワードによって指定します。
- トリガを実行するタイミング。イベント発生の前 (Before) または後 (After) です。これは、クラス定義で、オプションの Time トリガ・キーワードによって指定します。既定値は Before です。
- 同一のイベントやタイミングに複数のトリガを関連付けることができます。この場合、複数のトリガが起動される順番を Order トリガ・キーワードを使用して制御できます。Order 値が低いトリガから順番に動作します。複数のトリガが同じ Order 値を持つ場合は、動作する順番は不定になります。
- オプションの Foreach トリガ・キーワードを使用すると、よりきめ細かく制御できます。このキーワードは、トリガを行ごとに 1 回起動するか (`Foreach = row`)、行またはオブジェクトにアクセスするたびに 1 回起動するか (`Foreach = row/object`)、または文ごとに 1 回起動するか (`Foreach = statement`) を制御します。Foreach トリガ・キーワードが定義されていないトリガは、行ごとに 1 回起動します。`Foreach = row/object` を使用してトリガが定義されている場合、このドキュメントの後半で説明するように、トリガはオブジェクト・アクセスの特定の時点でも呼び出されます。INFORMATION.SCHEMA.TRIGGERS の ACTIONORIENTATION プロパティを使用して、各トリガの Foreach 値をリスト表示できます。

注釈　Python は、`Foreach = row/object` オプションのみをサポートします。

トリガ・キーワードの完全なリストは、"クラス定義リファレンス" を参照してください。

次の表に、使用できるトリガおよび対応するコールバック・メソッドを示します。

トリガ	対応するコールバック・メソッド
BEFORE INSERT	%OnBeforeSave()
AFTER INSERT	%OnAfterSave()
BEFORE UPDATE	%OnBeforeSave()
AFTER UPDATE	%OnAfterSave()
BEFORE UPDATE OF (指定された列)	なし
AFTER UPDATE OF (指定された列)	なし
BEFORE DELETE	%OnDelete()
AFTER DELETE	%OnAfterDelete()

トリガとコールバック・メソッドは類似していますが、実行されるタイミングが異なる場合があります。

- トリガは、格納されているデータの変更の直前または直後に実行されます。
- コールバック・メソッドは、コールバックのタイプに応じて、%SaveData() または %DeleteData() メソッドの前か後に実行されます。したがって、オブジェクトの前の %OnBeforeSave() メソッドはロックするか、そうでなければ参照アクションによって変更されます。

注釈　トリガが実行されるときに、処理されるテーブル内のプロパティの値をその実行で直接変更することはできません。これは、InterSystems IRIS が、フィールド (プロパティ) 値検証コードの後にトリガ・コードを実行するからです。例えば、処理中の行にある現在のタイムスタンプに LastModified フィールドを設定できません。ただし、トリガ・コードはテーブル内のフィールド値に対して UPDATE を発行できます。UPDATE は、独自のフィールド値検証を実行します。

詳細は、"InterSystems SQL リファレンス" の "CREATE TRIGGER" を参照してください。

14.2.1 AFTER トリガ

AFTER トリガは、INSERT、UPDATE、または DELETE イベントの発生後に実行されます。

- SQLCODE=0 の場合 (イベントが正常に完了した場合)、InterSystems IRIS は AFTER トリガを実行します。
- SQLCODE が負の数値の場合 (イベントが失敗した場合)、InterSystems IRIS は AFTER トリガを実行しません。
- SQLCODE=100 の場合 (挿入、更新、または削除する行が見つからなかった場合)、InterSystems IRIS は AFTER トリガを実行します。

14.2.2 再帰トリガ

トリガの実行は、再帰的になる可能性があります。例えば、テーブル T1 には、テーブル T2 への挿入を実行するトリガがあり、テーブル T2 には、テーブル T1 への挿入を実行するトリガがある場合です。また、テーブル T1 にルーチン/プロシージャを呼び出すトリガがあり、そのルーチン/プロシージャが T1 への挿入を実行する場合にも、再帰が発生する可能性があります。トリガの再帰の処理は、トリガのタイプによって異なります。

- 行レベルおよび行/オブジェクト・レベルのトリガ：InterSystems IRIS では、行レベルおよび行/オブジェクト・レベルのトリガの再帰的な実行が防止されません。トリガの再帰の処理は、プログラマの責任になります。トリガのコードで再帰的な実行を処理していないと、実行時 <FRAMESTACK> エラーが発生する可能性があります。
- 文レベルのトリガ：InterSystems IRIS は、AFTER 文のトリガの再帰的な実行を防止します。InterSystems IRIS は、実行スタック内で AFTER トリガが以前に呼び出されていることを検出すると、AFTER トリガを発行しません。エラーは発行されません。トリガは 2 回目には実行されなくなるだけです。

  InterSystems IRIS では、BEFORE 文のトリガの再帰的な実行が防止されません。BEFORE トリガの再帰の処理は、プログラマの責任になります。BEFORE トリガのコードで再帰的な実行を処理していないと、実行時 <FRAMESTACK> エラーが発生する可能性があります。

14.3 ObjectScript トリガ・コード

各トリガには、トリガされたアクションを実行する 1 行以上のコードが含まれます。このコードは、トリガが定義されたイベントが発生した場合、常に SQL エンジンによって呼び出されます。トリガが CREATE TRIGGER を使用して定義されている場合、このアクション・コードを ObjectScript または SQL で記述できます (InterSystems IRIS は、SQL で記述されたコードをクラス定義の ObjectScript に変換します)。スタジオや VS Code などの IDE で定義したトリガの場合、このアクション・コードは ObjectScrip で記述する必要があります。

トリガのコードはプロシージャとして生成されないため、トリガ内のすべてのローカル変数はパブリック変数となります。システム変数 %ok、%msg、%oper (次のセクションで説明します) を除き、トリガに使用するすべての変数を NEW 文で明示的に宣言する必要があります。これにより、トリガを起動するコードで変数どうしが競合することを防止できます。

14.3.1 %ok、%msg、および %oper システム変数

- %ok：トリガ・コードでのみ使用される変数。トリガ・コードが正常に終了すると、%ok=1 に設定されます。トリガ・コードが失敗すると、%ok=0 に設定されます。トリガ実行時に SQLCODE エラーが発行されると、InterSystems IRIS によって %ok=0 に設定されます。%ok=0 の場合、トリガ・コードの実行は中止され、トリガ操作とトリガを呼び出した操作はロール・バックされます。INSERT または UPDATE トリガ・コードが失敗し、テーブルに外部キー制約が定義されている場合は、InterSystems IRIS によって、外部キー・テーブル内の対応する行のロックが解除されます。

  トリガ・コードによって明示的に %ok=0 に設定できます。これにより、トリガの実行を中止して操作をロール・バックする実行時エラーが生成されます。通常、%ok=0 に設定する前に、トリガ・コードによって、%msg 変数がこのユーザ定義トリガ・コード・エラーを記述するユーザ指定文字列に明示的に設定されます。

  %ok 変数は、非トリガ・コードの SELECT、INSERT、UPDATE、または DELETE 文の完了時に以前の値のまま変更されません。%ok はトリガ・コードの実行によってのみ定義されます。この変数は、トリガ・コードへ参照渡しされて、そこで更新できます。したがって、トリガ・コードの NEW 文で明示的に宣言しないようにする必要があります。
- %msg：トリガ・コードで、%msg 変数に実行時エラーの起きた節を記述する文字列を明示的に設定できます。SQLCODE エラーにより、%msg 変数が設定されます。%ok 変数と同様にトリガ・コードへ参照渡しされるので、トリガ・コードの NEW 文で明示的に宣言しないようにする必要があります。
- %oper：トリガ・コードでのみ使用される変数。トリガ・コードは、変数 %oper を参照できます。この変数には、トリガを発生させるイベントの名前 (INSERT、UPDATE、または DELETE) が含まれます。

14.3.2 {fieldname} 構文

トリガ・コード内で、特別な {fieldname} 構文を使用して、(トリガが関連付けられているテーブルに属するフィールドの) フィールド値を参照できます。例えば、MyApp.Person クラスの LogEvent トリガの以下の定義には、ID フィールドへの参照が {ID} として含まれます。

Class Definition

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
{
    // ... Definitions of other class members

    /// This trigger updates the LogTable after every insert
    Trigger LogEvent [ Event = INSERT, Time = AFTER ]
    {
        // get row id of inserted row
        NEW id,SQLCODE
        SET id = {ID}

        // INSERT value into Log table
        &sql(INSERT INTO LogTable
            (TableName, IDValue)
            VALUES ('MyApp.Person', :id))
        IF SQLCODE<0 {SET baderr="SQLCODE ERROR:"_SQLCODE_" "_%msg
                      SET %ok=0
                    RETURN baderr }

      }
   // ... Definitions of other class members

}
```

この {fieldname} 構文は、単一フィールドをサポートしています。%SerialObject コレクション・プロパティはサポートしていません。例えば、テーブルがプロパティ City を含む、埋め込みのシリアル・オブジェクト・クラス Address を参照する場合、トリガ構文 {Address_City} はフィールドへの有効な参照です。トリガ構文 {Address} はコレクション・プロパティへの参照であり、使用できません。

14.3.3 トリガ・コード内のマクロ

トリガ・コードには、フィールド名を参照するマクロ定義を含めることができます ({fieldname} 構文を使用)。ただし、{fieldname} 構文を使用してフィールド名を参照するマクロに対して #include プリプロセッサ指示文をトリガ・コードで使用している場合、そのフィールド名にはアクセスできません。これは、InterSystems IRIS がトリガ・コード内の {fieldname} 参照を、そのコードがマクロ・プリプロセッサに渡される前に変換するためです。#include ファイルの中で使用している {fieldname} 参照は、トリガ・コードで認識されないので変換されません。

この状況を回避するには、引数を指定してマクロを定義してから、トリガ内のマクロに {fieldname} を渡します。例えば、#include ファイルに、以下のような行を記述できます。

ObjectScript

```
#define dtThrowTrigger(%val) SET x=$GET(%val,"?")
```

次に、{fieldname} 構文を引数として指定して、トリガ内でマクロを呼び出します。

ObjectScript

```
  $$$dtThrowTrigger({%%ID})
```

14.3.4 {name*O}、{name*N}、および {name*C} トリガ・コード構文

UPDATE トリガ・コードでは 3 種類の ObjectScript 構文ショートカットを使用できます。

以下の構文を使用して古い (更新前の) 値を参照できます。

```
{fieldname*O}
```

ここで、fieldname はフィールドの名前で、アスタリスクの後ろの文字はアルファベットの“O”です (Old を意味しています)。INSERT トリガの場合は、{fieldname*O} は必ず空文字列 (””) となります。

以下の構文を使用して新しい (更新後の) 値を参照できます。

```
{fieldname*N}
```

ここで、fieldname はフィールドの名前で、アスタリスクの後ろの文字はアルファベットの“N”です (New を意味しています)。この **{fieldname*N}** 構文は、格納される値を参照するためにのみ使用できます。値を変更するためには使用できません。トリガ・コードに **{fieldname*N}** を設定することはできません。INSERT または UPDATE でのフィールド値の計算は、SqlComputeOnChange などの他の方法で実行する必要があります。

以下の構文を使用して、フィールド値が変更 (更新) されたかどうかをテストできます。

```
{fieldname*C}
```

ここで、fieldname はフィールドの名前で、アスタリスクの後ろの文字はアルファベットの“C”です (Changed を意味しています)。**{fieldname*C}** は、フィールドが変更されているときは 1、変更されていないときは 0 となります。INSERT トリガの場合は、InterSystems IRIS によって **{fieldname*C}** が 1 に設定されます。

ストリーム・プロパティがあるクラスでは、SQL 文 (INSERT または UPDATE) がストリーム・プロパティ自体を挿入または更新しなかった場合は、ストリーム・プロパティ **{Stream*N}** および **{Stream*O}** への SQL トリガ参照では、ストリームの OID が返されます。ただし、SQL 文がストリーム・プロパティを挿入または更新した場合は、**{Stream*O}** は OID のままですが、**{Stream*N}** 値は以下のいずれかに設定されます。

- BEFORE トリガは、UPDATE または INSERT に渡された形式を問わず、ストリーム・フィールドの値を返します。これは、ストリーム・プロパティに入力されたリテラル・データ値か、一時ストリーム・オブジェクトの OREF または OID となります。
- AFTER トリガは、ストリームの ID を **{Stream*N}** 値として返します。これは、InterSystems IRIS によって、そのストリーム・フィールドのためにグローバル ^classnameD 内に格納される ID 値です。この値は、そのストリーム・プロパティの CLASSNAME 型パラメータに基づく適切な ID 形式となります。

ストリーム・プロパティが、InterSystems IRIS オブジェクトを使用して更新される場合は、**{Stream*N}** 値は必ず OID となります。

注釈　シリアル・オブジェクトの配列コレクションによって作成された子テーブルのトリガの場合、トリガ・ロジックはオブジェクト・アクセス/保存で機能しますが、SQL アクセス (INSERT または UPDATE) では機能しません。

14.3.5 その他の ObjectScript トリガ・コード構文

ObjectScript で記述するトリガ・コードには、擬似フィールド参照変数の {%%CLASSNAME}、{%%CLASSNAMEQ}、{%%OPERATION}、{%%TABLENAME}、および {%%ID} を含めることができます。これらの擬似フィールドは、クラスのコンパイル時に特定の値に変換されます。詳細は、″InterSystems SQL リファレンス″ の ″CREATE TRIGGER″ を参照してください。

クラス・メソッドは開いているオブジェクトの有無に依存しないため、クラス・メソッドをトリガ・コード、SQL 計算コード、および SQL マップ定義内から使用できます。トリガ・コード内からクラス・メソッドを呼び出すには、**##class(classname).Methodname()** 構文を使用する必要があります。..Methodname() 構文では、現在開いているオブジェクトが必要なので、この構文は使用できません。

クラス・メソッドの引数として現在の行のフィールドの値を渡すことができますが、クラス・メソッド自体はフィールド構文を使用できません。

14.4 Python トリガ・コード

組み込みの Python を含むトリガ・ブロック内で、そのトリガに関連する属性とメソッドを含む trigger オブジェクトにアクセスできます。これらの属性にはアクセス可能です。これらの詳細は、"トリガのタイプ" を参照してください。

- trigger.type ― 有効な値は "row/object" と "row" です。
- trigger.operation ― 有効な値は "BEFORE"、"UPDATE"、および "DELETE" です。
- trigger.time ― 有効な値は "before" と "after" です。

トリガ・オブジェクトは、テーブル・フィールドにアクセスできる getfield() メソッドも適用します。以下のオプションを指定できます。

- trigger.getfield(fieldName) は、指定されたフィールド fieldName の値を返します。トリガ操作でフィールド値が変更されると、getfield() は新しい値を返します。
- trigger.getfield(fieldName, new) は、ブーリアン値 new を使用して新しいフィールド値と元のフィールド値のどちらを返すかを決定します。
  - new = 1 (既定) の場合、getfield() はトリガ操作で生じた新しい値を返します。このオプションは、INSERT 演算子と UPDATE 演算子に適用されます。
  - new = 0 の場合、getfield() は、トリガ操作の前の値を返します。このオプションは、UPDATE 演算子と DELETE 演算子に適用されます。

この例では、trigger オブジェクトを使用して、Name フィールドの名前変更というトリガ操作の結果をログ・ファイルに書き込みます。Python コードと対応する ObjectScript コードの両方を示します。

Python

```
Trigger LogRename [ Event = UPDATE, Foreach = row/object, Language = python, Time = AFTER ]
{
    with open('C:/temp/log.txt', 'a+') as file:
        file.write("Rename Event Occurred")
        file.write("\nID: " + str(trigger.getfield("ID")))
        file.write("\nOperation: " + trigger.operation)
        file.write("\nOld Name: " + trigger.getfield("Name", 0))
        file.write("\nNew Name: " + trigger.getfield("Name", 1) + "\n\n")
}
```

ObjectScript

```
Trigger LogRename [ Event = UPDATE, Foreach = row/object, Time = AFTER ]
{
    set file = "C:/temp/log.txt"
    open file:("EWA") use file
    write "Rename Event Occurred"
    write !,"ID: "_{ID}
    write !,"Operation: "_{%%OPERATION}
    write !,"Old Name: "_{Name*O}
    write !,"New Name: "_{Name*N},!!
    close file
}
```

14.5 トリガのプル

そのテーブルに対応する DML コマンドが呼び出されると、定義されたトリガが "プル" (実行) されます。

DML コマンドが正常に挿入、更新、または削除した行ごとに、行レベルまたは行/オブジェクト・レベルのトリガがプルされます。

正常に実行される INSERT、UPDATE、または DELETE 文ごとに、文レベルのトリガが 1 回プルされます。文によってテーブル・データの行が実際に変更されるかどうかは関係ありません。

- INSERT 文は、対応する INSERT トリガをプルします。INSERT は、%NOTRIGGER キーワードを指定することにより、この対応するトリガのプルを阻止できます。%NOJOURN キーワードを指定する INSERT は、挿入または対応する INSERT トリガをジャーナリングしません。つまり、挿入イベントでもトリガ・イベントでもロールバックはできません。
- UPDATE 文は、対応する UPDATE トリガをプルします。UPDATE は、%NOTRIGGER キーワードを指定することにより、この対応するトリガのプルを阻止できます。%NOJOURN キーワードを指定する UPDATE は、更新または対応する UPDATE トリガをジャーナリングしません。つまり、更新イベントでもトリガ・イベントでもロールバックはできません。
- INSERT OR UPDATE 文は、実行する DDL 操作のタイプに応じて、対応する INSERT トリガまたは UPDATE トリガをプルします。どちらのタイプのトリガもプルしないようにするには、%NOTRIGGER キーワードを指定します。
- DELETE 文は、対応する DELETE トリガをプルします。DELETE は、%NOTRIGGER キーワードを指定することにより、この対応するトリガのプルを阻止できます。%NOJOURN キーワードを指定する DELETE は、削除または対応する DELETE トリガをジャーナリングしません。つまり、削除イベントでもトリガ・イベントでもロールバックはできません。
- TRUNCATE TABLE 文は、DELETE トリガをプルしません。

既定では、DDL 文および対応するトリガされたアクションはジャーナリングされます。%NOJOURN キーワードは、DDL コマンドとトリガされたアクションの両方のジャーナリングを阻止します。

14.6 トリガとオブジェクト・アクセス

`Foreach = row/object` を使用してトリガが定義されている場合、トリガ定義の Event および Time キーワードに応じて、トリガは以下のようにオブジェクト・アクセスの特定の時点でも呼び出されます。

Event	Time	トリガが呼び出される時点
INSERT	BEFORE	新しいオブジェクトの %Save() の直前
INSERT	AFTER	新しいオブジェクトの %Save() の直後
UPDATE	BEFORE	既存のオブジェクトの %Save() の直前
UPDATE	AFTER	既存のオブジェクトの %Save() の直後
DELETE	BEFORE	既存のオブジェクトの %DeleteId() の直前
DELETE	AFTER	既存のオブジェクトの %DeleteId() の直後

この結果、SQL とオブジェクトの動作の同期を維持するためにコールバック・メソッドを実装する必要もなくなります。

Foreach トリガ・キーワードについては、"クラス定義リファレンス" を参照してください。

14.6.1 オブジェクト・アクセス時にトリガをプルしない

既定では、SQL オブジェクトは %Storage.Persistent を使用して格納されます。InterSystems IRIS では %Storage.SQL ストレージもサポートされます。

%Storage.SQL ストレージを使用するクラスでオブジェクトを保存または削除する際、文レベル (Foreach = statement)、行レベル (Foreach = row)、および行/オブジェクト・レベル (Foreach = row/object) のトリガがすべてプルされます。Foreach トリガ・キーワードが定義されていないトリガは、行レベルのトリガです。既定の動作として、すべてのトリガがプルされます。

ただし、%Storage.SQL を使用するクラスでオブジェクトを保存または削除する際、Foreach = row/object として定義されているトリガのみがプルされるように指定できます。Foreach = statement または Foreach = row として定義されているトリガはプルされません。このためには、クラス・パラメータ OBJECTSPULLTRIGGERS = 0 を指定します。既定は OBJECTSPULLTRIGGERS = 1 です。

このパラメータは、%Storage.SQL を使用して定義されているクラスにのみ適用されます。

14.7 トリガとトランザクション

トリガは、トランザクションの範囲内でトリガ・コードを実行します。トリガは、トランザクション・レベルを設定してから、トリガ・コードを実行します。トリガ・コードが正常に完了すると、トリガによってトランザクションがコミットされます。

注釈　トランザクションを使用したトリガの場合、トランザクションをコミットするコードをトリガで呼び出すと、トランザクション・レベルが既に 0 にデクリメントされているためにトリガの完了が失敗します。この状況は、プロダクションのビジネス・サービスを呼び出したときに発生することがあります。

AFTER INSERT 文レベルの ObjectScript トリガの使用では、トリガを %ok=0 にした場合、行の挿入は SQLCODE -131 エラーにより失敗します。自動トランザクションの設定によっては、トランザクション・ロールバックも発生する場合があります。SQL では、SET TRANSACTION %COMMITMODE commitmode オプションによってこの設定が制御されます。ObjectScript では、SetOption() メソッドの "AutoCommit" オプションによってこの設定が制御されます。SET status=$SYSTEM.SQL.Util.SetOption("AutoCommit",intval,.oldval) という構文が使用されます。次の表は、自動トランザクション設定がロールバックに与える影響を示します。対応する commitmode オプション (SQL に設定) と "AutoCommit" 整数値 (ObjectScript に設定) も示します。

自動トランザクション設定	%COMMITMODE オプション (SQL)	"AutoCommit" 値 (ObjectScript)	トランザクション・ロールバックの結果
トランザクション処理を自動化しない	NONE	0	トランザクションは開始されないので、INSERT はロールバックできません。
トランザクションの自動コミットをオンにする (既定)	IMPLICIT	1	INSERT に対するトランザクションがロールバックされます。
トランザクションの自動コミットをオフにする	EXPLICIT	2	INSERT に対するトランザクションのロールバックまたはコミットはアプリケーションによって決まります。

このトリガにより、トリガ内の %msg 変数にエラー・メッセージが設定されます。このメッセージは発信者宛てに返され、トリガの失敗理由についての情報が提供されます。

%ok および %msg のシステム変数は、このドキュメントの“埋め込み SQL の使用法”の章の システム変数 のセクションで説明しています。

14.8 トリガのリスト

指定したテーブルに定義されているトリガは、管理ポータルの SQL インタフェース [カタログの詳細] にリストされます。このリストでは、各トリガの基本情報が表示されます。より詳細な情報をリスト表示するには、INFORMATION.SCHEMA.TRIGGERS を使用します。

INFORMATION.SCHEMA.TRIGGERS クラスは、現在のネームスペースで定義されているトリガをリスト表示します。INFORMATION.SCHEMA.TRIGGERS は、トリガごとにさまざまなプロパティをリストします。これには、トリガの名前、関連付けられたスキーマおよびテーブル名、EVENTMANIPULATION プロパティ (INSERT、UPDATE、DELETE、INSERT/UPDATE、INSERT/UPDATE/DELETE)、ACTIONTIMING プロパティ (BEFORE、AFTER)、CREATED プロパティ (トリガ作成タイムスタンプ)、および生成された SQL トリガ・コードの ACTIONSTATEMENT プロパティが含まれます。

CREATED プロパティでは、クラス定義が最後に変更されたときのトリガ作成タイムスタンプが得られます。このため、以後このクラスを使用すると (例えば、他のトリガを定義するために)、CREATED プロパティ値が意図せずに更新されることがあります。

この INFORMATION.SCHEMA.TRIGGERS 情報には、以下の例に示すように SQL クエリからアクセスできます。

SQL

```
SELECT
TABLE_NAME,TRIGGER_NAME,CREATED,EVENT_MANIPULATION,ACTION_TIMING,ACTION_ORIENTATION,ACTION_STATEMENT
FROM INFORMATION_SCHEMA.TRIGGERS WHERE TABLE_SCHEMA='Sample'
```

15
照合

照合は、値を並べる方法と比較する方法を指定するもので、InterSystems SQL と InterSystems IRIS® データ・プラットフォーム・オブジェクトのどちらにも欠かせない要素です。基本的な照合には、数値と文字列の 2 つがあります。

- 数値照合では、完全な数値に基づいて数値を並べ替えます。この並べ替えは、NULL、負の数値（最大から最小）、ゼロ、正の数値（最小から最大）の順に行われます。これにより、-210、-185、-54、-34、-.02、0、1、2、10、17、100、120 のように順序が生成されます。
- 文字列照合は、一連の文字それぞれを照合することで文字列を並べ替えます。これにより、A、AA、AAA、AAB、AB、B のように順序が生成されます。数値の場合は、-.02、-185、-210、-34、-54、0、1、10、100、120、17、2 のように順序が生成されます。

既定の文字列照合は、SQLUPPER です。この既定値は、ネームスペースごとに設定されます。SQLUPPER 照合では、すべての文字が照合のために大文字に変換され、文字列の先頭に空白文字が追加されます。この変換は、照合のみを目的としています。InterSystems SQL の文字列は、どの照合が適用されていても、通常は大文字と小文字で表示されます。また、文字列の長さには追加された空白文字は含まれません。

タイムスタンプは文字列であるため、現在の文字列照合に従います。ただし、タイムスタンプは ODBC 形式であるため、文字列照合は時間的な順序と一致します（先頭のゼロが指定されている場合）。

- 文字列式（LEFT または SUBSTR スカラ文字列関数を使用する式など）では、結果の照合は EXACT になります。
- 2 つのリテラルの比較には、EXACT 照合を使用します。

ObjectScript の前後関係演算子を使用すると、2 つの値の相対的な照合順を判断できます。

照合は以下のように指定できます。

- ネームスペースの既定
- テーブルのフィールド/プロパティ定義
- インデックス定義
- クエリの SELECT item
- クエリの DISTINCT 節と GROUP BY 節

この章で後述する“SQL 照合と NLS 照合”も参照してください。

15.1 照合タイプ

照合は、フィールド/プロパティの定義またはインデックスの定義でキーワードとして指定できます。

照合は、クエリの節でフィールド名に照合関数を適用することで指定できます。照合関数を指定するときには、接頭語の % が必要になります。

照合は、以下の変換を使用して、ASCII または Unicode の昇順で実行されます。

- EXACT ー 文字を追加したり削除することなく、文字列データに大文字と小文字の区別を強制適用します。数値順に並べたキャノニック形式の数値を照合してから、文字ごとの ASCII 順で文字列を照合します。文字列の照合では、大文字と小文字を区別します。文字列の照合には、キャノニック形式以外の数値 (088 など) や、混合数値文字列 (22 Elm Street など) が含まれます。キャノニック形式の数値は文字列 (単一の文字やキャノニック形式以外の数値文字列を含む) より前に照合されるため、一般に、キャノニック数値形式で値を含む可能性のある文字列データ (`123`、`-.57` など) には、EXACT はお勧めしません。
- SQLSTRING ー 末尾の空白 (スペースやタブなど) を削除して、文字列の先頭に空白を 1 つ追加します。空白 (スペース、タブなど) だけを含む任意の値を SQL 空文字列として照合します。SQLSTRING は、オプションの maxlen 整数値をサポートしています。
- SQLUPPER ー すべてのアルファベット文字を大文字に変換し、末尾の空白 (スペースやタブなど) を削除して、文字列の先頭にスペース文字を 1 つ追加します。このスペース文字が先頭に追加される理由は、強制的に数値を文字列として照合するためです (スペース文字は有効な数値文字ではないため)。この変換により、SQL 空文字列 (’’) の値と空白 (スペースやタブなど) のみを含む値が単一のスペース文字として照合されるようにもなります。SQLUPPER は、オプションの maxlen 整数値をサポートしています。SQLUPPER 変換は、SQL の UPPER 関数の結果とは異なります。
- TRUNCATE ー 文字列データの大文字と小文字の区別を強制しますが、EXACT とは異なり、値を切り捨てる位置を指定できます。これは、正確なデータにインデックスを作成する際に、そのデータが添え字の最大長を超えている場合に役立ちます。`%TRUNCATE(`string,n`)` の形式で引数に正の整数を指定すると、文字列は最初の n 文字で切り捨てられます。これにより、長い文字列のインデックスの作成と並べ替えの効率が向上します。TRUNCATE に長さを指定しない場合、EXACT と同じように動作します。この動作はサポートされていますが、TRUNCATE は長さを定義している場合にのみ使用し、長さを定義していない場合は EXACT を使用することによって、定義やコードの保守がより簡単になります。
- PLUS ー 値を数値にします。非数値文字列値は 0 として返されます。
- MINUS ー 値を数値にし、その符号を変更します。非数値文字列値は 0 として返されます。

注釈　　また、さまざまな従来の照合タイプもありますが、それらは使用しないことをお勧めします。

SQL クエリでは、括弧なしの `%SQLUPPER Name` または括弧付きの `%SQLUPPER(Name)` で照合関数を指定できます。照合関数で切り捨てを指定する場合には括弧が必要です (`%SQLUPPER(Name,10)`)。

3 つの照合タイプ SQLSTRING、SQLUPPER、および TRUNCATE は、オプションの maxlen 整数値をサポートしています。maxlen を指定すると、文字列の解析は最初の n までになります。これは、長い文字列のインデックス付けや並べ替えの際に、パフォーマンスを向上させるために使用できます。maxlen は、切り詰めた文字列値を並べ替え、グループ化、または返すクエリで使用できます。

%SYSTEM.Util.Collation() メソッドを使用して照合タイプの変換を実行することもできます。

15.2 ネームスペース全体の既定の照合

各ネームスペースには、現行の文字列照合が設定されています。この文字列照合は、%Library.String のデータ型に対して定義されています。既定値は SQLUPPER です。この既定値は変更可能です。

ネームスペースごとに照合の既定を定義できます。既定では、ネームスペースには照合が割り当てられません。これは SQLUPPER 照合を使用することを意味します。既定の照合をネームスペースに個別に割り当てられます。このネームスペースの既定の照合は、すべてのプロセスに適用され、明示的に再設定するまで InterSystems IRIS を再起動しても維持されます。

ObjectScript

```
   SET stat=$$GetEnvironment^%apiOBJ("collation","%Library.String",.collval)
   WRITE "initial collation for ",$NAMESPACE,!
     ZWRITE collval
SetNamespaceCollation
  DO SetEnvironment^%apiOBJ("collation","%Library.String","SQLstring")
  SET stat=$$GetEnvironment^%apiOBJ("collation","%Library.String",.collnew)
    WRITE "user-assigned collation for ",$NAMESPACE,!
     ZWRITE collnew
ResetCollationDefault
  DO SetEnvironment^%apiOBJ("collation","%Library.String",.collval)
  SET stat=$$GetEnvironment^%apiOBJ("collation","%Library.String",.collreset)
     WRITE "restored collation default for ",$NAMESPACE,!
     ZWRITE collreset
```

ネームスペースの照合の既定を設定しないと、$$GetEnvironment は、この例の .collval のように未定義の照合変数を返します。この未定義の照合は、既定で SQLUPPER に設定されます。

注釈　データにドイツ語テキストが含まれている場合は、大文字の照合が、希望する既定どおりにならない可能性があります。これは、ドイツ語の eszett 文字 ($CHAR(223)) に小文字形式しかないためです。同意義の大文字は、“SS” という 2 文字です。大文字に変換する SQL 照合では、eszett は変換されず、変更されないまま単一の小文字として残ります。

15.3 テーブルのフィールド/プロパティ定義の照合

SQL 内に、フィールド/プロパティ定義の一部として照合を割り当てることができます。フィールドに使用するデータ型により、そのフィールドの既定の照合が決まります。行ストレージを使用するテーブルでは、文字列データ型に対する既定の照合は SQLUPPER ですが、列指向ストレージを使用するテーブルでは EXACT が既定の照合です。文字列以外のデータ型は、照合の割り当てをサポートしていません。

フィールドの照合は、CREATE TABLE および ALTER TABLE で指定できます。

SQL

```
CREATE TABLE Sample.MyNames (
    LastName CHAR(30),
    FirstName CHAR(30) COLLATE SQLstring)
```

注釈　CREATE TABLE および ALTER TABLE を使用してフィールドの照合を指定するときには、接頭語の % はオプションになります (COLLATE SQLstring または COLLATE %SQLstring)。

プロパティの照合は、永続クラス定義を使用してテーブルを定義する際に指定できます。

Class Definition

```
Class Sample.MyNames Extends %Persistent [DdlAllowed]
{
Property LastName As %String;
Property FirstName As %String(COLLATION = "SQLstring");
}
```

注釈　クラス定義およびクラス・メソッドに照合を指定するときには、照合タイプ名に接頭語の % を使用しないでください。

ここに示す各例では、LastName フィールドは既定の照合（大文字と小文字を区別しない SQLUPPER）を採用し、FirstName フィールドは大文字と小文字を区別する SQLSTRING で定義されています。

クラスのプロパティに対する照合を変更する場合に、そのクラスのデータが既に保存されているときは、プロパティのインデックスが無効になります。このプロパティに基づいてすべてのインデックスを再構築する必要があります。

15.4 インデックス定義の照合

CREATE INDEX コマンドでインデックスの照合を指定することはできません。インデックスには、インデックス作成対象のフィールドと同じ照合が使用されます。

クラス定義の一部として定義されたインデックスには、照合タイプを指定できます。既定では、特定のプロパティに対するインデックスは、プロパティ・データの照合タイプを使用します。例えば、%String タイプの Name プロパティを定義したとします。

Class Definition

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
{
Property Name As %String;
Index NameIDX On Name;
}
```

Name の照合は、SQLUPPERです (%String の既定)。Person テーブルに、以下のデータが含まれているとします。

ID	Name
1	Jones
2	JOHNSON
3	Smith
4	jones
5	SMITH

Name のインデックスは、以下のエントリを持つことになります。

Name	ID(s)
JOHNSON	2
JONES	1、4
SMITH	3、5

SQL エンジンは、このインデックスを直接使用して、Name フィールドを使用した ORDER BY や比較演算を実行します。

インデックスに使用される既定の照合は、インデックス定義に As 節を使用することでオーバーライドできます。

Class Definition

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
{
Property Name As %String;
Index NameIDX On Name As SQLstring;
}
```

この場合、NameIDX インデックスは、SQLSTRING（大文字と小文字を区別する）形式で値を保存することになります。上記の例のデータを使用すると、以下のようになります。

Name	ID(s)
JOHNSON	2
Jones	1
jones	4
SMITH	5
Smith	3

この場合、SQL エンジンは、大文字と小文字を区別する照合が必要になるあらゆるクエリに、このインデックスを利用できます。

通常、インデックスの照合は変更する必要はありません。別の照合を使用する場合は、その照合をプロパティ・レベルで定義して、そのプロパティに対するインデックスが適切な照合を採用できるようにします。

インデックスが作成されているプロパティを使用してプロパティの比較を実行する場合、その比較に指定するプロパティは、対応するインデックスと同じ照合タイプにする必要があります。例えば、SELECT の WHERE 節または JOIN の ON 節での Name プロパティは、その Name プロパティに定義されたインデックスと照合を同じにする必要があります。プロパティの照合とインデックスの照合に不一致があると、インデックスの効果が低下するか、インデックスがまったく使用されなくなります。詳細は、"SQL 最適化ガイド" の “インデックスの定義と構築” の章にある "インデックス照合" を参照してください。

使用しているインデックスが複数のプロパティを使用するために定義されている場合は、個別に照合を指定できます。

Class Member

```
Index MyIDX On (Name As SQLstring, Code As Exact);
```

15.5 クエリの照合

InterSystems SQL には、フィールドの照合または表示の変更に使用できる照合関数があります。

15.5.1 select-item 照合

クエリの select-item に照合関数を適用することで、その項目の表示を変更します。

- 大文字と小文字：既定では、クエリは大文字と小文字で文字列を表示します。ただし、照合タイプが SQLUPPER のフィールドに対する DISTINCT 演算または GROUP BY 演算は例外です。これらの演算子は、そのフィールドをすべて大文字で表示します。%EXACT 照合関数を使用すると、この大文字と小文字の変換を逆にして、フィールドを大文字と小文字で表示できます。すべて大文字でフィールドを表示するために、select-item リストで %SQLUPPER

照合関数を使用しないでください。これは、%SQLUPPER は文字列の長さに空白文字を追加するためです。その代わりに、UPPER 関数を使用してください。

SQL

```
SELECT TOP 5 Name,$LENGTH(Name) AS NLen,
             %SQLUPPER(Name) AS UpCollN,$LENGTH(%SQLUPPER(Name)) AS UpCollLen,
             UPPER(Name) AS UpN,$LENGTH(UPPER(Name)) AS UpLen
FROM Sample.Person
```

- 文字列の切り捨て：%TRUNCATE 照合関数を使用すると、表示する文字列データの長さを制限できます。%TRUNCATE は、文字列の長さに空白文字を追加する %SQLUPPER よりもお勧めです。

SQL

```
SELECT TOP 5 Name,$LENGTH(Name) AS NLen,
             %TRUNCATE(Name,8) AS TruncN,$LENGTH(%TRUNCATE(Name,8)) AS TruncLen
FROM Sample.Person
```

照合関数や大文字小文字の変換関数は、入れ子にできない点に注意してください。

- WHERE 節の比較：ほとんどの WHERE 節の述語条件の比較には、フィールド/プロパティの照合タイプが使用されます。文字列フィールドは既定で SQLUPPER に設定されているため、一般に、こうした比較では大文字と小文字が区別されません。大文字と小文字を区別するには、%EXACT 照合関数を使用します。

  以下の例では、大文字と小文字に関係なく文字列 Home_City との一致が返されます。

  SQL

  ```
  SELECT Home_City FROM Sample.Person WHERE Home_City = 'albany'
  ```

  以下の例では、大文字と小文字を区別して文字列 Home_City との一致が返されます。

  SQL

  ```
  SELECT Home_City FROM Sample.Person WHERE %EXACT(Home_City) = 'albany'
  ```

  SQL 後続関係演算子 ( ]) では、フィールド/プロパティの照合タイプが使用されます。

  ただし、SQL 包含関係演算子 ([ ) では、フィールド/プロパティの照合タイプと関係なく、EXACT 照合が使用されます。

  SQL

  ```
  SELECT Home_City FROM Sample.Person WHERE Home_City [ 'c'
  ORDER BY Home_City
  ```

  %MATCHES 述語条件と %PATTERN 述語条件では、フィールド/プロパティの照合タイプと関係なく、EXACT 照合が使用されます。%PATTERN 述語には、大文字と小文字を区別するワイルドカードと、大文字と小文字を区別しないワイルドカード ('A') の両方が用意されています。

- ORDER BY 節：ORDER BY 節は、文字列値の順序付けにネームスペースの既定の照合を使用します。そのため、ORDER BY では、大文字小文字に基づいた順序付けが行われません。%EXACT 照合を使用すると、大文字小文字に基づいて文字列を並べ替えできます。

15.5.2 DISTINCT の照合と GROUP BY の照合

既定では、これらの演算子はネームスペースの現在の演算子を使用します。ネームスペースの既定の照合は、SQLUPPER です。

- DISTINCT：DISTINCT キーワードは、重複値の削除にネームスペースの既定の照合を使用します。そのため、DISTINCT Name では、すべて大文字の値が返されます。EXACT 照合を使用すると、大文字と小文字が混ざった値を返すことができます。DISTINCT は、大文字と小文字のみが異なる重複を削除します。大文字と小文字が異なる重複を保持しながら完全な重複を削除するには、EXACT 照合を使用します。以下の例では、完全な重複（大文字と小文字が異ならないもの）を削除して、大文字と小文字が混ざった値をすべて返します。

  SQL

  ```
  SELECT DISTINCT %EXACT(Name) FROM Sample.Person
  ```

  UNION は、暗黙的に DISTINCT 演算を伴います。

- GROUP BY：GROUP BY 節は、重複値の削除にネームスペースの既定の照合を使用します。そのため、GROUP BY Name では、すべて大文字の値が返されます。EXACT 照合を使用すると、大文字と小文字が混ざった値を返すことができます。GROUP BY は、大文字と小文字のみが異なる重複を削除します。大文字と小文字が異なる重複を維持しながら完全な重複を削除するには、select-item ではなく、GROUP BY 節で %EXACT 照合関数を指定する必要があります。

  以下の例では、大文字と小文字が混ざった値が返されます。GROUP BY は大文字小文字が異なるものも含めて重複を削除します。

  SQL

  ```
  SELECT %EXACT(Name) FROM Sample.Person GROUP BY Name
  ```

  以下の例では、大文字と小文字が混ざった値が返されます。GROUP BY は完全な重複（大文字小文字が異ならないもの）を削除します。

  SQL

  ```
  SELECT Name FROM Sample.Person GROUP BY %EXACT(Name)
  ```

15.6 従来の照合タイプ

InterSystems SQL は、従来の照合タイプを複数サポートしています。これらは、従来のシステムを継続してサポートするために用意されているにすぎないため、非推奨であり、新しいコードで使用することはお勧めしません。従来の照合タイプは、以下のとおりです。

- %ALPHAUP ー 疑問符（“?”）およびコンマ（“,”）以外の句読点文字をすべて削除し、小文字をすべて大文字に変換します。主に、従来のグローバルのマッピングに使用されます。SQLUPPER によって置き換えられました。
- %STRING ー 論理値を大文字に変換し、すべての句読点（コンマを除く）および空白を削除し、文字列の先頭に空白を 1 つ追加します。空白（スペース、タブなど）だけを含む任意の値を SQL 空文字列として照合します。SQLUPPER によって置き換えられました。
- %UPPERー小文字をすべて大文字に変換します。主に、従来のグローバルのマッピングに使用されます。SQLUPPER によって置き換えられました。
- SPACE ー SPACE 照合は値の先頭にスペースを 1 つ追加します。それにより、値を文字列として解釈させます。SPACE 照合を指定するために、CREATE TABLE には SPACE 照合キーワードが用意されています。また、ObjectScript では %SYSTEM.Util クラスの Collation() メソッドに SPACE オプションが用意されています。これに対応する SQL 照合関数はありません。

注釈　文字列データ型のフィールドが EXACT 照合、UPPER 照合、または ALPHAUP 照合で定義されているときに、このフィールドにクエリで %STARTSWITH 条件を適用すると、動作結果が一貫しないことがあります。%STARTSWITH に指定した substring がキャノニック形式の数値の場合 (特に負の数値や小数値の場合)、%STARTSWITH はフィールドのインデックスが作成されているかどうかによって異なる結果を生成することがあります。%STARTSWITH は、列にインデックスが作成されていない場合に、予期したとおりに動作します。列にインデックスが作成されていると、予期しない結果を生じる可能性があります。

これらの従来の照合タイプを使用するクエリに対しては、自動並列処理は無効化されています。

15.7 SQL 照合と NLS 照合

前述の SQL 照合と InterSystems IRIS NLS 照合機能とを混同しないでください。この NLS 照合機能は、特定の言語の照合要件に従った添え字レベルのエンコードを提供するものです。照合を提供する別個のシステムが 2 つあり、これらは製品の異なるレベルで動作します。

InterSystems IRIS NLS 照合の場合、現在のプロセスにプロセスレベルの照合を 1 つ、また特定のグローバルに複数の照合を指定できます。

InterSystems SQL の使用時に適切に機能させるには、プロセスレベルの NLS 照合が、関連するすべてのグローバル (テーブルで使用されるグローバル、プロセス・プライベート・グローバルなど一時ファイルで使用されるグローバル、IRIS-TEMP グローバルで使用されるグローバルを含む) の NLS 照合と正確に一致することが必要です。そうでないと、クエリ・プロセッサで作成された異なる処理プランによって、結果が異なってしまう可能性があります。ORDER BY 節もしくは値域条件など、並べ替えの発生する状況では、クエリ・プロセッサが最も効率的な並べ替え方法を選択します。そして、インデックスの使用、プロセス・プライベート・グローバルでの一時ファイルの使用、ローカル配列内の並べ替え、または "]]" (前後関係) 比較の使用が行われます。これらはすべて、有効な InterSystems IRIS NLS 照合に従った添え字タイプの比較であるため、これらすべてのグローバル・タイプでまったく同じ NLS 照合を使用することが必要になります。

グローバルは、データベースの既定の照合で作成されます。%Library.GlobalEdit クラスの Create() メソッドの使用により、異なる照合でグローバルを作成することができます。この唯一の要件は、指定の照合が組み込み (InterSystems IRIS 標準のものなど) であるか、現在のロケールで使用可能な国固有照合のいずれかであることです。"専用のシステム/ツールおよびユーティリティ" の "%Library.GlobalEdit の使用によるグローバルの照合の設定" を参照してください。

16
データベースの変更

既存のテーブルに対して SQL 文を使用するか、対応する永続クラスに対して ObjectScript 操作を使用して、InterSystems IRIS® データ・プラットフォーム・データベースの内容を変更することができます。READONLY として定義されている永続クラス (テーブル) を変更することはできません。

SQL コマンドを使用すると、データの整合性が自動的に維持されるようになります。SQL コマンドはアトミック (全か無か) 処理です。テーブルにインデックスが定義されている場合は、SQL は変更を自動的に更新します。データや参照整合の制約が定義されている場合は、SQL は自動的にそれらを実行します。定義されたトリガがある場合、これらのアクションを実行すると、対応する挿入、更新、または削除トリガがプルされます。

16.1 データの挿入

テーブルにデータを挿入するには、SQL 文を使用するか、永続クラスのプロパティを設定して保存します。

16.1.1 SQL を使用したデータの挿入

INSERT 文は、SQL テーブルに新規のレコードを挿入します。単一のレコードまたは複数のレコードを挿入できます。

以下の例では、単一のレコードを挿入します。これは、単一レコードを挿入するために使用可能ないくつかの構文形式の 1 つです。

SQL

```
INSERT INTO MyApp.Person
   (Name,HairColor)
   VALUES ('Fred Rogers','Black')
```

以下の例では、既存のテーブルからデータのクエリを実行して、複数のレコードを挿入します。

SQL

```
INSERT INTO MyApp.Person
   (Name,HairColor)
    SELECT Name,Haircolor FROM Sample.Person WHERE Haircolor IS NOT NULL
```

INSERT コマンドを使用してさまざまなタイプのデータを挿入する方法の詳細は、"INSERT" のコマンド・リファレンス・ページを参照してください。

INSERT OR UPDATE 文を発行することもできます。この文は、そのレコードがまだ存在していない場合に、SQL テーブルに新規のレコードを挿入します。そのレコードが存在する場合は、この文は、提供されたフィールド値でそのレコード・データを更新します。

16.1.2 オブジェクト・プロパティを使用したデータの挿入

ObjectScript を使用して、1 つまたは複数のデータ・レコードを挿入できます。既存の永続クラスのインスタンスを作成し、1 つまたは複数のプロパティ値を設定してから、%Save() を使用してデータ・レコードを挿入します。

以下の例では、単一のレコードを挿入します。

ObjectScript

```
SET oref=##class(MyApp.Person).%New()
SET oref.Name="Fred Rogers"
SET oref.HairColor="Black"
DO oref.%Save()
```

以下の例では、複数のレコードを挿入します。

ObjectScript

```
SET nom=$LISTBUILD("Fred Rogers","Fred Astare","Fred Flintstone")
SET hair=$LISTBUILD("Black","Light Brown","Dark Brown")
FOR i=1:1:$LISTLENGTH(nom) {
    SET oref=##class(MyApp.Person).%New()
    SET oref.Name=$LIST(nom,i)
    SET oref.HairColor=$LIST(hair,i)
    SET status = oref.%Save() }
```

16.2 UPDATE 文

UPDATE 文は、SQL テーブルの既存の 1 つ以上のレコードの値を変更します。

SQL

```
UPDATE MyApp.Person
    SET HairColor = 'Red'
    WHERE Name %STARTSWITH 'Fred'
```

16.3 INSERT または UPDATE 時の計算フィールドの値

計算フィールドを定義するとき、そのフィールドのデータ値を計算するコードを指定できます。このデータ値は、行の挿入、更新、またはその両方が行われるとき、またはクエリが実行されるときに計算されます。以下のテーブルに、各タイプの計算処理に必要なキーワード、およびフィールド/プロパティ定義の例を示します。

計算タイプ	指定する DDL SQL キーワード	指定する永続クラスのキーワード
INSERT のみ	COMPUTECODE	SqlComputeCode および SqlComputed
	`Birthday VARCHAR(50) COMPUTECODE {SET {Birthday}=$PIECE($ZDATE({DOB},9),",")_" changed: "_$ZTIMESTAMP }`	`Property Birthday As %String(MAXLEN = 50) [ SqlComputeCode = {SET {Birthday}=$PIECE($ZDATE({DOB},9),",")_" changed: "_$ZTIMESTAMP}, SqlComputed ];`
UPDATE のみ	DEFAULT、COMPUTECODE、および COMPUTEONCHANGE	N/A
	`Birthday VARCHAR(50) DEFAULT ' ' COMPUTECODE {SET {Birthday}=$PIECE($ZDATE({DOB},9),",")_" changed: "_$ZTIMESTAMP } COMPUTEONCHANGE (DOB)`	
INSERT と UPDATE の両方	COMPUTECODE および COMPUTEONCHANGE	SqlComputeCode、SqlComputed、および SqlComputeOnChange
	`Birthday VARCHAR(50) COMPUTECODE {SET {Birthday}=$PIECE($ZDATE({DOB},9),",")_" changed: "_$ZTIMESTAMP } COMPUTEONCHANGE (DOB)`	`Property Birthday As %String(MAXLEN = 50) [ SqlComputeCode = {SET {Birthday}=$PIECE($ZDATE({DOB},9),",")_" changed: "_$ZTIMESTAMP}, SqlComputed, SqlComputeOnChange = DOB ];`
クエリ	COMPUTECODE および CALCULATED または TRANSIENT	SqlComputeCode、SqlComputed、および Calculated または Transient
	`Birthday VARCHAR(50) COMPUTECODE {SET {Birthday}=$PIECE($ZDATE({DOB},9),",")_" changed: "_$ZTIMESTAMP } CALCULATED`	`Property Birthday As %String(MAXLEN = 50) [ SqlComputeCode = {SET {Birthday}=$PIECE($ZDATE({DOB},9),",")_" changed: "_$ZTIMESTAMP}, SqlComputed, Calculated];`

DDL DEFAULT キーワードは、挿入時のデータ値の計算より優先されます。DEFAULT は空の文字列などのデータ値を取る必要があり、NULL にはできません。永続クラス定義では、挿入時に InitialExpression プロパティ・キーワードは SqlComputed データ値をオーバーライドしません。

DDL COMPUTEONCHANGE キーワードは、1 つのフィールド名、またはコンマで区切られたフィールド名のリストを取ることができます。これらのフィールド名は、更新時にこのフィールドの計算をトリガするフィールドを指定します。リストされたフィールド名はテーブルに存在する必要がありますが、計算コードに出現しなくてもかまいません。実際のフィールド名を指定する必要があります。アスタリスク構文は指定できません。

COMPUTECODE および COMPUTEONCHANGE を使用する代わりに、ON UPDATE キーワード句を使用して、レコードが変更されるときにフィールドをリテラルまたはシステム変数（現在のタイムスタンプなど）に設定できます。ON UPDATE

句は INSERT 時と UPDATE 時に変更します。UPDATE 時にのみ変更するには、DEFAULT 句と ON UPDATE 句を使用します。

DDL CALCULATED または TRANSIENT キーワードは、フィールドがクエリによってアクセスされるたびにデータ値を計算します。フィールドを選択リストで指定する必要はありません。例えば、`SELECT Name FROM MyTable WHERE LENGTH(Birthday)=36` は条件式を評価する前に、Birthday フィールドを計算します。管理ポータルの [テーブルを開く] オプションでクエリが実行され、CALCULATED および TRANSIENT データ値が計算されます。

計算フィールドの制限事項：

- UPDATE では更新しない：UPDATE ではレコード内のフィールドに前の値と同じ値を提供し、実際にはレコードを更新しません。レコードに対して実際の更新が行われない場合、COMPUTEONCHANGE は呼び出されません。レコードに対して実際の更新が行われない場合でも、ON UPDATE は更新処理時に呼び出されます。レコードが実際に更新されたかどうかに関係なく、更新時に常に計算フィールドを再計算する場合は、更新トリガを使用します。
- 計算フィールドに対するユーザ指定の明示的な値：
  - INSERT：INSERT 時には、COMPUTECODE、DEFAULT、または ON UPDATE フィールドに常に明示的な値を指定できます。InterSystems SQL は常に、生成された値ではなく明示的な値を取ります。
  - UPDATE COMPUTEONCHANGE：UPDATE 処理では COMPUTEONCHANGE フィールドに明示的な値を指定できます。InterSystems SQL は常に、計算された値ではなく明示的な値を取ります。
  - UPDATE ON UPDATE：UPDATE 処理では ON UPDATE フィールドに明示的な値を指定できません。InterSystems SQL はユーザが指定した値を無視し、ON UPDATE で生成された値を取ります。ただし、InterSystems SQL は明示的な値に対してフィールド検証を実行し、例えば、指定された値が最大データ・サイズより大きい場合は、SQLCODE -104 エラーを生成できます。
  - CALCULATED または TRANSIENT：INSERT または UPDATE 処理では CALCULATED または TRANSIENT フィールドに明示的な値を指定できません。CALCULATED または TRANSIENT フィールドはデータを格納しないためです。ただし、InterSystems SQL は明示的な値に対してフィールド検証を実行し、例えば、指定された値が最大データ・サイズより大きい場合は、SQLCODE -104 エラーを生成できます。

16.4 データの検証

この章で説明する挿入操作と更新操作では、データ検証が自動的に実行されます。これにより、テーブルに無効なデータが格納されるのを防ぐことができます。その他の方法でテーブルに格納されたデータは、検証されない可能性があります。$SYSTEM.SQL.Schema.ValidateTable() メソッドを使用して、テーブルのデータを検証できます。テーブル名は修飾 (`"schema.table"`)、未修飾 (`"table"`) のどちらでもかまいません。未修飾のテーブル名は既定のスキーマ名を取ります。スキーマ検索パスの値は使用されません。

ValidateTable() は、テーブルのデータで見つかった各検証の問題に対する行を含む結果セットを返します。このメソッドは、以下のデータ検証を実行します。

- データ型 IsValid() メソッドを使用して、各データ値をフィールドのデータ型に照らして検証します。
- 必須制約があるフィールドに Null 値がないことを検証します。
- 一意制約があるフィールドに重複する値がないことを検証します。
- 外部キー・フィールドが、参照されるテーブルの有効な行を参照していることを検証します。

検証結果セットは、%sqlcontext オブジェクトに保持されます。DO %sqlcontext.%Display() は、次のターミナルの例に示すように、データ検証結果を現在のデバイスに表示します。

```
USER>DO $SYSTEM.SQL.Schema.ValidateTable("Sample.MyTable")
USER>DO %sqlcontext.%Display()

Dumping result #1
Row(s) With an Issue    Field Or Constraint Name        Error
%ID 3   Home_City       Cannot be null
%ID 14  Home_City       Cannot be null
%ID 10  Home_City       Cannot be null
%ID 13  Home_City       Cannot be null
%ID 6   Home_PostalCode Value is invalid: BadZip
%ID 8   Home_PostalCode Value is invalid: WhoKnows
%ID 9   Home_PostalCode Value is invalid: BadZip
%ID 10  Home_PostalCode Value is invalid: WhoKnows
%ID 11  Home_PostalCode Value is invalid: BadZip
%ID 9   Home_State      Cannot be null
%ID 3   Home_State      Cannot be null
%ID 10  Home_State      Cannot be null

12 Rows(s) Affected
USER>
```

フィールドはアルファベット順で示されます。この例で、Home_City および Home_State フィールドは必須の値の検証に失敗しました。Home_PostalCode (%Integer データ型) フィールドはデータ型の検証に失敗しました。

このデータ検証処理を SQL から呼び出すこともできます。それには、次の例に示すように、ValidateTable ストアド・プロシージャを呼び出します。

SQL

```
CALL %SYSTEM.ValidateTable('Sample.MyTable')
```

シャード・テーブルの場合、ValidateTable() をシャード・マスタ・テーブルで呼び出す必要があります。

ValidateTable() はロックを実行しません。このため、同時処理を行うライブ・システム上のテーブルに対して実行した場合、誤検出エラー・レポートを受け取る可能性があります。

16.5 DELETE 文

DELETE 文は、SQL テーブルから既存の 1 つ以上のレコードを削除します。

SQL

```
 DELETE FROM MyApp.Person
     WHERE HairColor = 'Aqua'
```

TRUNCATE TABLE コマンドを発行して、テーブルからすべてのレコードを削除することもできます。DELETE を使用して、テーブルからすべてのレコードを削除することもできます。DELETE (既定) は削除トリガをプルします。一方、TRUNCATE TABLE は削除トリガをプルしません。DELETE を使用してすべてのレコードを削除しても、テーブル・カウンタはリセットされません。一方、TRUNCATE TABLE ではカウンタはリセットされます。

16.6 トランザクション処理

トランザクションは、1 つの作業単位を構成する一連のデータ変更文 (INSERT、UPDATE、DELETE、INSERT OR UPDATE、および TRUNCATE TABLE) です。

SET TRANSACTION コマンドを使用して、現在のプロセスのトランザクション・パラメータを設定することができます。START TRANSACTION コマンドを使用しても、同じパラメータを設定することができます。これらのトランザクション・パラメータは、明示的に変更されるまで複数のトランザクションにわたり有効です。

START TRANSACTION コマンドで、明示的にトランザクションを開始します。一般的に、このコマンドはオプションです。トランザクションの %COMMITMODE が IMPLICIT または EXPLICIT の場合は、トランザクションは最初のデータベース変更処理で自動的に開始されます。トランザクションの %COMMITMODE が NONE の場合は、START TRANSACTION を明示的に指定してトランザクション・プロセスを開始する必要があります。

トランザクションが成功したら、変更を暗黙的 (自動的) または明示的にコミットできます。COMMIT 文を使用してそのデータ変更をデータベースに永久的に追加して、リソースを解放する必要があるかどうかは、%COMMITMODE の値で決定します。

トランザクションが失敗した場合、そのデータ修正を元に戻す ROLLBACK 文を使用して、データベースに反映されないようにします。

注釈　SQL が管理ポータルの **[SQLクエリ実行]** インタフェースを使用して実行されている場合、SQL のトランザクション文はサポートされません。このインタフェースは SQL コードを開発するときのテスト環境として使用することを意図したもので、実際のデータを変更するためのものではありません。

16.6.1 トランザクションとセーブポイント

InterSystems SQL では、完全トランザクション処理とセーブポイントを使用したトランザクション処理の 2 種類のトランザクション処理を実行できます。完全トランザクション処理の場合、トランザクションは START TRANSACTION 文で (明示的または暗黙的に) 開始され、COMMIT 文でトランザクションを (明示的または暗黙的に) 完了して実行されたすべての処理をコミットするか、ROLLBACK 文でトランザクション中に実行されたすべての処理を戻すまで、続きます。

セーブポイントを使用する場合、InterSystems SQL はトランザクション内でレベルをサポートします。START TRANSACTION 文で、(明示的にまたは暗黙的に) トランザクションを開始します。トランザクション内で SAVEPOINT を指定して、プログラム内の 1 つ以上の名前付きセーブポイントを指定します。1 つのトランザクションに最大 255 個までの名前付きセーブポイントを指定できます。セーブポイントを追加すると、$TLEVEL トランザクション・レベル・カウンタがインクリメントされます。

- COMMIT は、トランザクションの間に実行されたすべての作業をコミットします。セーブポイントは無視されます。
- ROLLBACK は、トランザクションの間に実行されたすべての作業をロールバックします。セーブポイントは無視されます。
- ROLLBACK TO SAVEPOINT pointname により、pointname で指定された SAVEPOINT から実行されたすべての処理をロールバックし、内部トランザクション・レベル・カウンタを当該のセーブポイント・レベル数だけデクリメントします。例えば、svpt1 と svpt2 という 2 つのセーブポイントを設定して、svpt1 にロールバックする場合、ROLLBACK TO SAVEPOINT svpt1 により svpt1 から実行された処理を戻し、この場合は、トランザクション・レベル・カウンタを 2 つデクリメントします。

16.6.2 非トランザクション操作

トランザクションが有効な間でも、以下の操作はトランザクションに含まれないのでロールバックできません。

- IDKey カウンタのインクリメントは、トランザクション操作ではありません。IDKey は $INCREMENT (または $SEQUENCE) によって自動的に生成されます。そこでは SQL トランザクションとは別にカウントを保持しています。例えば、IDKeys が 17、18、および 19 のレコードを挿入し、この挿入をロールバックすると、挿入される次のレコードの IdKey は 20 になります。
- クエリ・キャッシュの作成、変更、削除は、トランザクションの操作ではありません。したがって、クエリ・キャッシュがトランザクション中に削除された場合、そのトランザクションはロールバックし、クエリ・キャッシュは削除されたままの状態で (リストアされずに) ロールバック・オペレーションになります。

- トランザクション中に発生する DDL 操作やテーブル・チューニング操作は、一時ルーチンを作成したり実行する場合があります。この一時ルーチンは、クエリ・キャッシュと同様に処理されます。つまり、一時ルーチンの作成、コンパイル、および削除はトランザクションの一部として処理されません。一時ルーチンの実行は、トランザクションの一部であると見なされます。

ロールバックされるまたはロールバックされない非 SQL 項目については、ObjectScript の "TROLLBACK" コマンドを参照してください。

16.6.3 トランザクションでのロック

トランザクションは、ロックを使用して一意のデータ値を保護します。例えば、処理が一意のデータ値を削除した場合、この値はこのトランザクションの有効期間内はロックされます。したがって、最初のトランザクションが終了するまで、別の処理でこの同じ一意のデータ値を使用してレコードを挿入することはできません。これは、一意性制約を持つフィールドに結果的に重複値を持たせるロールバックを防ぎます。これらのロックは、INSERT 文、UPDATE 文、INSERT OR UPDATE 文、および DELETE 文に %NOLOCK 制限引数が含まれている場合を除き、これらの文によって自動的に適用されます。

16.6.4 トランザクション・サイズの制限

ジャーナル・ファイルに利用できるスペースのほかには、トランザクションで指定できる処理数に制限はありません。InterSystems IRIS には自動ロック・エスカレーションがあるため、通常、ロック・テーブルのサイズが制限を課すことはありません。

既定では、テーブルごとに 1000 ロックのロックしきい値があります。1 つのテーブルは、現在のトランザクションに対して 1,000 個の一意データ値ロックを持つことが可能です。1,001 番目のロック処理は、トランザクションの有効期間内はロック・テーブルに対してそのテーブル用のロックをエスカレートします。

このロックしきい値は、以下のいずれかの方法で変更できます。

- $SYSTEM.SQL.Util.SetOption("LockThreshold") メソッドを呼び出します。このメソッドは、現在のシステム全体の値と構成ファイルの設定の両方を変更します。現在のロック制御のしきい値を確認するには、$SYSTEM.SQL.Util.GetOption("LockThreshold") メソッドを使用します。
- 管理ポータルに移動します。**[システム管理]** から、**[構成]**、**[SQL およびオブジェクトの設定]**、**[SQL]** の順に選択します。この画面で、**[ロックしきい値]** の現在の設定を表示および編集できます。

強制終了できるサブノード (子テーブル) の数に制限はありません。すべてのサブノードの強制終了はジャーナルに記録されるので、ロールバックできます。

16.6.5 コミットされていないデータの表示

クエリ発行のプロセスに対して SET TRANSACTION または START TRANSACTION を設定することで、読み取り分離レベルを指定できます。

- ISOLATION LEVEL READ UNCOMMITTED: データに対するコミットされていない挿入、更新、および削除は、別のユーザによるクエリ (読み取りのみ) アクセスで表示できます。トランザクションが指定されていない場合は、これが既定になります。
- ISOLATION LEVEL READ VERIFIED: データに対するコミットされていない挿入、更新、および削除は、別のユーザによるクエリ (読み取りのみ) アクセスで表示できます。クエリ条件で使用されるデータとクエリで表示されるデータを再チェックできます。
- ISOLATION LEVEL READ COMMITTED: コミットされていない挿入および更新によるデータへの変更は、クエリ結果セットには表示されません。クエリ結果セットには、コミット済みの挿入および更新のみが含まれます。ただし、コミットされていない削除によるデータへの変更は、クエリ結果セットに表示されます。

SELECT コマンド節である、集約関数、DISTINCT 節、GROUP BY 節、または %NOLOCK キーワードを指定した SELECT は、現在の分離レベルに関係なく、コミットされていないデータを必ず返します。詳細は、"分離レベル" を参照してください。

16.6.6 ObjectScript トランザクション・コマンド

ObjectScript と SQL のトランザクション・コマンドは完全に互換性があり、置き換え可能ですが、以下の例外があります。

ObjectScript TSTART と SQL START TRANSACTION はどちらも、トランザクションが進行中でない場合にトランザクションを開始します。ただし、START TRANSACTION では、入れ子になったトランザクションはサポートされません。そのため、入れ子になったトランザクションが必要な場合（または必要になる可能性がある場合）には、トランザクションを TSTART で始めることをお勧めします。SQL 標準との互換性が必要な場合は、START TRANSACTION を使用してください。

ObjectScript トランザクション処理は、入れ子になったトランザクションを限定的にサポートします。SQL トランザクション処理はトランザクション内のセーブポイントをサポートします。

17
データベースの問い合わせ

この章では、InterSystems IRIS® データ・プラットフォームでデータのクエリを実行する方法について説明します。

17.1 クエリのタイプ

クエリは、データの取得および結果セットの生成を実行する文です。クエリは、以下のいずれかで構成されます。

- 指定したテーブルまたはビューのデータにアクセスする簡単な SELECT 文。
- 複数のテーブルまたはビューのデータにアクセスする、JOIN 構文を使用した SELECT 文。
- 複数の SELECT 文の結果を組み合わせる UNION 文。
- 囲んでいる SELECT クエリに単一のデータ項目を提供するために SELECT 文を使用するサブクエリ。
- 埋め込み SQL で、FETCH 文を使用して複数行のデータにアクセスするために SQL カーソルを使用する SELECT 文。

17.2 SELECT 文の使用法

SELECT 文は、1 つ以上のテーブルまたはビューから 1 つ以上のデータの行を選択します。以下の例では、簡単な SELECT 文を示します。

SQL

```
SELECT Name,DOB FROM Sample.Person WHERE Name %STARTSWITH 'A' ORDER BY DOB
```

この例では、Name および DOB は Sample.Person テーブル内の列 (データ・フィールド) です。

SELECT 文での節の指定順序は、SELECT DISTINCT TOP ... selectItems INTO ...FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY です。コマンド構文の順序です。SELECT selectItems を除き、これらすべての節はオプションとなります (オプションの FROM 節は、格納されているデータに対する操作の実行に必要なため、クエリではほぼ必ず必要となります)。SELECT 節を指定する際に必要な順序の詳細は、SELECT 文の構文を参照してください。

17.2.1 SELECT 節の実行順序

SELECT 文の操作は、そのセマンティック処理順序に注目すると理解できます (SELECT 構文の順序と同じではありません)。SELECT の節は以下の順序で処理されます。

1. FROM 節 – JOIN 構文またはサブクエリを使用して、1 つまたは複数のテーブルやビューを指定します。
2. WHERE 節 – さまざまな条件を使用して、選択するデータを制限します。
3. GROUP BY 節 – 選択したデータを一致する値によるサブセットに編成します。値ごとに 1 つのレコードのみが返されます。
4. HAVING 節 – さまざまな条件を使用して、グループから選択するデータを制限します。
5. selectItem – 指定したテーブルまたはビューからデータ・フィールドを選択します。selectItem は、特定のデータ・フィールドを参照するまたは参照しない式の場合もあります。
6. DISTINCT 節 – SELECT 結果セットに適用され、返される行を、重複しない値を含む行のみに限定します。
7. ORDER BY 節 – SELECT 結果セットに適用され、返される行を、指定フィールドによる照合順でソートします。

このセマンティック順序は、テーブルのエイリアス (FROM 節で定義される) はすべての節で認識できるが、列のエイリアス (SELECT selectItems で定義される) は ORDER BY 節でしか認識できないことを示しています。

他の SELECT 節で列のエイリアスを使用するには、以下の例に示すように、サブクエリを使用します。

SQL

```
SELECT Interns FROM
      (SELECT Name AS Interns FROM Sample.Employee WHERE Age<21)
WHERE Interns %STARTSWITH 'A'
```

この例では、Name および Age は Sample.Person テーブル内の列 (データ・フィールド) であり、Interns は Name の列エイリアスです。

17.2.2 フィールドの選択

SELECT を発行すると、InterSystems SQL は、指定した各 selectItem フィールド名を、指定したテーブルに対応するクラスに定義されているプロパティと突き合わせようとします。それぞれのクラス・プロパティには、プロパティ名と SqlFieldName の両方があります。SQL を使用してテーブルを定義した場合、CREATE TABLE コマンドで指定されるフィールド名は SqlFieldName ですが、InterSystems IRIS は SqlFieldName からプロパティ名を生成しています。

フィールド名、クラス・プロパティ名、および SqlFieldName 名には、様々な名前付け規約があります。

- SELECT 文のフィールド名は、大文字/小文字を区別しません。SqlFieldName 名とプロパティ名は、大文字/小文字を区別します。
- SELECT 文内のフィールド名と SqlFieldName 名には、識別子の名前付け規約に従う特定の非英数字を含めることができます。プロパティ名には、英数字のみを含めることができます。プロパティ名の生成時に、非英数字は削除されます。InterSystems IRIS は、一意のプロパティ名を作成するために、文字を追加することが必要になる場合があります。

フィールドのこれら 3 つの名前間での変換により、クエリ動作のいくつかの面が決定されます。selectItem フィールド名は、大文字/小文字の任意の組み合わせで指定でき、InterSystems SQL は対応する該当のプロパティを識別します。結果セット表示内のデータ列ヘッダ名は SqlFieldName であり、selectItem で指定したフィールド名ではありません。この理由から、データ列ヘッダの大文字/小文字は、selectItem フィールド名とは異なる場合があります。

selectItem フィールドには、列エイリアスを指定できます。列エイリアスは、識別子の名前付け規約に従い、大文字/小文字の任意の組み合わせで指定でき、非英数字を含めることができます。列エイリアスは、大文字/小文字の任意の組み

合わせで参照でき（例えば、ORDER BY 節で）、InterSystems SQL は selectItem フィールドに指定されている大文字/小文字に解決します。InterSystems IRIS は常に、定義されたフィールドに対応するプロパティのリストへの突き合わせを試行する前に、列エイリアスのリストへの突き合わせを試行します。列エイリアスを定義した場合、結果セット表示内のデータ列ヘッダ名は、SqlFieldName ではなく、指定した大文字/小文字の列エイリアスとなります。

SELECT クエリが正常に完了すると、InterSystems SQL はそのクエリの結果セット・クラスを生成します。結果セット・クラスには、選択した各フィールドに対応するプロパティが含まれます。SELECT クエリに重複フィールド名が含まれている場合、文字を追加することによって、クエリ内のフィールドの各インスタンスに対して一意のプロパティ名が生成されます。この理由から、1 個のクエリ内では 36 個のインスタンスを超える同じフィールドを含めることはできません。

クエリの生成された結果セット・クラスには、列エイリアスのプロパティも含まれます。大文字/小文字の解決のためのパフォーマンス・コスト発生を避けるには、列エイリアスを参照するときに、SELECT 文での列エイリアスの指定時に使用したものと同じ大文字/小文字を使用する必要があります。

ユーザ指定の列エイリアスに加え、InterSystems SQL はさらに、各フィールド名に最大で 3 つのエイリアス（フィールド名の一般的な大文字/小文字バリアントに対応するエイリアス）を自動的に生成します。生成されるエイリアスは、ユーザには表示されません。エイリアスによるプロパティへのアクセスは、文字変換による大文字/小文字の解決よりも高速であるという、パフォーマンス上の理由からこれは提供されます。例えば、SELECT が FAMILYNAME を指定しており、対応するプロパティが familyname である場合、InterSystems SQL は、生成されたエイリアス（FAMILYNAME AS familyname）を使用して大文字/小文字を解決します。ただし、SELECT が fAmILyNaMe を指定しており、対応するプロパティが familyname である場合、InterSystems SQL は、低速な文字変換プロセスを使用して大文字/小文字を解決する必要があります。

selectItem 項目は、式、集約関数、サブクエリ、ユーザ定義関数、アスタリスク、または他の何らかの値である場合もあります。フィールド名以外の selectItem 項目の詳細は、SELECT コマンドのリファレンス・ページの、″selectItem″ 引数を参照してください。

17.2.3 JOIN 演算

JOIN は、テーブルのデータを別のテーブルのデータと結合する手段を提供し、レポートやクエリの定義に頻繁に使用されます。SQL では、JOIN は、2 つのテーブルからのデータを組み合わせて、制限条件に当てはまる 3 つ目のテーブルを作成する演算です。作成されたテーブルのすべての行は、制限条件を満たす必要があります。

InterSystems SQL は、CROSS JOIN、INNER JOIN、LEFT OUTER JOIN、RIGHT OUTER JOIN、および FULL OUTER JOIN の 5 種類の結合（一部は複数の構文形式）をサポートしています。外部結合は、条件式の全範囲の述語と論理演算子を使用する ON 節をサポートします。NATURAL 外部結合および USING 節を使用した外部結合は、部分的にサポートしています。これらの結合タイプの定義および詳細は、″InterSystems SQL リファレンス″ の ″JOIN″ のページを参照してください。

クエリに結合が含まれている場合、そのクエリ内のすべてのフィールド参照には、追加されたテーブル・エイリアスが必要です。InterSystems IRIS はデータ列ヘッダ名にテーブル・エイリアスを組み込まないので、データのソースであるテーブルを明確にするために、selectItem フィールドに列エイリアスを指定することもできます。

次の例では、結合操作を使用して、Sample.Person 内の ″架空の″（ランダムに割り当てられた）郵便番号を Sample.USZipCode 内の実際の郵便番号および都市名と照合しています。WHERE 節が指定されている理由は、USZipCode にはすべてのあり得る 5 桁の郵便番号が含まれているわけではないからです。

SQL

```
SELECT P.Home_City,P.Home_Zip AS FakeZip,Z.ZipCode,Z.City AS ZipCity,Z.State
FROM Sample.Person AS P LEFT OUTER JOIN Sample.USZipCode AS Z
ON P.Home_Zip=Z.ZipCode
WHERE Z.ZipCode IS NOT NULL
ORDER BY P.Home_City
```

17.2.4 多数のフィールドを選択するクエリ

クエリで 1,000 個を超える selectItem フィールドを選択することはできません。

150 個を超える selectItem フィールドを選択するクエリには、次のパフォーマンス上の考慮事項が存在する場合があります。InterSystems IRIS は、結果セット列エイリアスを自動的に生成します。それらの生成されたエイリアスは、大文字/小文字のバリエーションを迅速に解決するためのユーザ定義エイリアスを持たないフィールド名に提供されます。エイリアスを使用する大文字/小文字解決は、文字変換による大文字/小文字解決よりもはるかに高速です。ただし、生成される結果セット列エイリアスの数は、500 までに制限されています。一般に InterSystems IRIS では、各フィールドに対して (大文字/小文字の最も一般的な 3 つのバリエーションに合わせて) これらのエイリアスが 3 つ生成されるので、クエリの先頭にある 150 個ほどの指定フィールドに対してエイリアスが生成されることになります。したがって、150 個より少ないフィールドを参照するクエリは、それよりもかなり多くのフィールドを参照するクエリと比較すると、結果セットのパフォーマンスは一般に良好ということになります。非常に大規模なクエリでは、selectItem の各フィールドに対してまったく同じ列エイリアスを指定して (`SELECT FamilyName AS FamilyName` など)、列エイリアスで結果セット項目を参照するときに必ず同じ大文字/小文字が使用されるようにすることで、このパフォーマンスの問題を回避できます。

17.3 名前付きクエリの定義と実行

名前付きクエリは、以下に示すように定義および実行できます。

- CREATE QUERY を使用してクエリを定義します。このクエリはストアド・プロシージャとして定義され、CALL を使用して実行できます。
- クラス・クエリ (クラス定義で定義されるクエリ) を定義します。クラス・クエリは、ストアド・プロシージャとして投影されます。CALL を使用して実行できます。%SQL.Statement %PrepareClassQuery() メソッドを使用してクラス・クエリを準備した後、%Execute() メソッドを使用して実行することもできます。“ダイナミック SQL の使用法” を参照してください。

17.3.1 CREATE QUERY と CALL

CREATE QUERY を使用してクエリを定義し、CALL を使用して、名前でそれを実行できます。以下の例では、1 つ目はクエリ AgeQuery を定義する SQL プログラムであり、2 つ目はそのクエリを実行するダイナミック SQL です。

SQL

```
CREATE QUERY Sample.AgeQuery(IN topnum INT DEFAULT 10,IN minage INT 20)
   PROCEDURE
   BEGIN
   SELECT TOP :topnum Name,Age FROM Sample.Person
   WHERE Age > :minage
   ORDER BY Age ;
   END
```

ObjectScript

```
  SET mycall = "CALL Sample.AgeQuery(11,65)"
  SET tStatement = ##class(%SQL.Statement).%New()
  SET qStatus = tStatement.%Prepare(mycall)
    IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
  SET rset = tStatement.%Execute()
  DO rset.%Display()
```

SQL

```
DROP QUERY Sample.AgeQuery
```

17.3.2 クラス・クエリ

クエリはクラス内で定義できます。多くの場合、そのクラスは %Persistent ですが、その他のクラスでもかまいません。このクラス・クエリは、同じクラスで定義されたデータ、または同じネームスペースの別のクラスで定義されたデータを参照できます。クラス・クエリで参照されるテーブル、フィールド、およびその他のデータ・エンティティは、クエリを含むクラスのコンパイル時に存在している必要があります。

クラス・クエリは、それを含むクラスのコンパイル時にはコンパイルされません。クラス・クエリのコンパイルは、SQL コードの最初の実行時 (ランタイム) に行われます。これは、%PrepareClassQuery() メソッドを使用してダイナミック SQL でクエリが準備される際に行われます。最初の実行によって、実行可能なクエリ・キャッシュが定義されます。

以下に示すクラス定義の例では、クラス・クエリを定義しています。

```
Class Sample.QClass Extends %Persistent [DdlAllowed]
  {
  Query MyQ(Myval As %String) As %SQLQuery (CONTAINID=1,ROWSPEC="Name,Home_State") [SqlProc]
     {
     SELECT Name,Home_State FROM Sample.Person
     WHERE Home_State = :Myval  ORDER BY Name
     }

  }
```

以下に示す例では、前の例で Sample.QClass クラス内に定義した MyQ クエリを実行します。

ObjectScript

```
  SET Myval="NY"
  SET stmt=##class(%SQL.Statement).%New()
  SET status = stmt.%PrepareClassQuery("Sample.QClass","MyQ")
    IF status'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(status) QUIT}
  SET rset = stmt.%Execute(Myval)
  DO rset.%Display()
  WRITE !,"End of data"
```

以下のダイナミック SQL の例では、%SQL.Statement を使用して Sample.Person クラス内で定義されている ByName クエリを実行しています。このとき、文字列を渡すことで、その文字列値で始まる名前を返すように制限しています。

ObjectScript

```
  SET statemt=##class(%SQL.Statement).%New()
  SET cqStatus=statemt.%PrepareClassQuery("Sample.Person","ByName")
    IF cqStatus'=1 {WRITE "%PrepareClassQuery failed:" DO $System.Status.DisplayError(cqStatus) QUIT}
  SET rs=statemt.%Execute("L")
  DO rs.%Display()
```

詳細は、"クラスの定義と使用" の "クラス・クエリの定義と使用" を参照してください。

17.4 ユーザ定義関数を呼び出すクエリ

InterSystems SQL では、SQL クエリ内でクラス・メソッドを呼び出すことができます。これにより、SQL の構文を拡張するための便利な機能が実現しました。

ユーザ定義の関数を作成するには、InterSystems IRIS 永続クラス内にクラス・メソッドを定義します。メソッドにはリテラル (非オブジェクト) の返り値が必要です。これは、クラス・メソッドである必要があります。SQL クエリでは、インスタンス・メソッドを呼び出すためのオブジェクト・インスタンスが作成されないためです。また、これは、SQL ストアド・プロシージャとして定義される必要があります。

例えば、MyApp.Person クラスで Cube() メソッドを定義できます。

Class Definition

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
{
/// Find the Cube of a number
ClassMethod Cube(val As %Integer) As %Integer [SqlProc]
  {
    RETURN val * val * val
  }
}
```

CREATE FUNCTION 文、CREATE METHOD 文、または CREATE PROCEDURE 文を使用して、SQL 関数を作成できます。

SQL 関数を呼び出すには、SQL プロシージャの名前を指定します。SQL 関数は、SQL コード内でスカラ式を指定できる任意の場所から呼び出すことができます。関数名はそのスキーマ名で修飾できますが、未修飾でもかまいません。未修飾の関数名では、ユーザ指定のスキーマ検索パスまたは既定のスキーマ名が使用されます。関数名は、区切り文字付き識別子とすることができます。

SQL 関数には、括弧で囲んだパラメータ・リストが必要です。パラメータ・リストは空でもかまいませんが、括弧は必ず記述します。指定したパラメータはすべて入力パラメータとして機能します。出力パラメータはありません。

SQL 関数は値を返す必要があります。

例えば、以下の SQL クエリは、ユーザ定義の SQL 関数を組み込み SQL 関数であるかのようにメソッドとして呼び出します。

SQL

```
SELECT %ID, Age, MyApp.Person_Cube(Age) FROM MyApp.Person
```

このクエリは **Age** のそれぞれの値に対し Cube() メソッドを呼び出し、返り値を結果に表示します。

SQL 関数は入れ子にして使用できます。

指定された関数が見つからない場合、InterSystems IRIS は SQLCODE -359 エラーを発行します。指定された関数名があいまいな場合、InterSystems IRIS は SQLCODE -358 エラーを発行します。

17.5 シリアル・オブジェクト・プロパティのクエリ

既定ストレージ (%Storage.Persistent) を使用してクラスから SQL に子テーブルとして投影されるシリアル・オブジェクト・プロパティは、そのクラスによって投影されたテーブル内の 1 つの列としても投影されます。この列の値は、シリアル・オブジェクト・プロパティのシリアル化された値となります。この 1 つの列プロパティは、SQL の%List フィールドとして投影されます。

例えば、Sample.Person の Home 列は `Property Home As Sample.Address;` として定義され、`Class Sample.Address Extends (%SerialObject)` に投影されます。これには、プロパティ Street、City、State、PostalCode が含まれます。シリアル・オブジェクトの定義の詳細は、“テーブルの定義” の章の ″埋め込みオブジェクト (%SerialObject)″ を参照してください。

以下の例では、個々のシリアル・オブジェクトの列から値が返されます。

SQL

```
SELECT TOP 4 Name,Home_Street,Home_City,Home_State,Home_PostalCode
FROM Sample.Person
```

以下の例では、すべてのシリアル・オブジェクトの列の値が単一の %List 形式文字列として (各列の値が順に %List の要素として) 返されます。

SQL

```
SELECT TOP 4 Name,$LISTTOSTRING(Home,'^')
FROM Sample.Person
```

既定では、この Home 列 は非表示で、Sample.Person の列としては投影されません。

17.6 コレクションのクエリ

コレクションは、次のように SQL の WHERE 節から参照できます。

```
 WHERE FOR SOME %ELEMENT(collectionRef) [AS label] (predicate)
```

FOR SOME %ELEMENT 節は、STORAGEDEFAULT="list" を指定する配列およびリスト・コレクションに使用できます。predicate は、擬似列 %KEY または %VALUE、あるいはその両方への参照を 1 つ指定できます。以下に、FOR SOME %ELEMENT 節の使用方法の例をいくつか示します。次の例は、FavoriteColors に 'Red' がある人の名前と、それぞれの FavoriteColors のリストを返します。

SQL

```
SELECT Name,FavoriteColors FROM Sample.Person
    WHERE FOR SOME %ELEMENT(FavoriteColors) (%Value = 'Red')
```

どの SQL 述語も %Value (または %Key) の後に記述できます。例えば、次の例も有効な構文となります。

SQL

```
SELECT Name,FavoriteColors FROM Sample.Person
    WHERE FOR SOME %ELEMENT(Sample.Person.FavoriteColors)
        (%Value IN ('Red', 'Blue', 'Green'))
```

リスト・コレクションは、1 から始まる連続数値のキーを持つ配列コレクションの特別な場合と見なされます。配列コレクションには、任意の NULL でないキーを指定できます。

```
 FOR SOME (children) (%Key = 'betty' AND %Value > 5)
```

組み込みリストと配列コレクション・タイプのほか、任意のプロパティに BuildValueArray() クラス・メソッドを指定することで、汎用コレクションを作成することもできます。BuildValueArray() クラス・メソッドは、プロパティ値をローカル配列に変換します。ここで、配列の各添え字は %KEY、配列の値は対応する %VALUE になります。

%KEY または %VALUE の単純な選択のほか、以下の例のように 2 つのコレクションを論理的に結合することもできます。

```
   FOR SOME %ELEMENT(flavors) AS f
      (f.%VALUE IN ('Chocolate', 'Vanilla') AND
       FOR SOME %ELEMENT(toppings) AS t
           (t.%VALUE = 'Butterscotch' AND
            f.%KEY = t.%KEY))
```

この例には、キーによって位置的に関連付けられた flavors と toppings という 2 つのコレクションがあります。このクエリでは、flavors の要素に指定した chocolate と vanilla の行が選択され、対応する toppings としてリストされた butterscotch によって行が絞り込まれ、%KEY によって対応関係が示された結果が作成されます。

$SYSTEM.SQL.Util.SetOption() メソッドの CollectionProjection オプションを使用して、この既定値をシステム全体で変更できます。コレクションが子テーブルとして投影されている場合、SET `status=$SYSTEM.SQL.Util.SetOption("CollectionProjection",1,.oldval)` を使用して、コレクションを列として投影します。既定値は 0 です。このシステム全体に対する設定の変更は、クラスをコンパイルまたはリコンパイルしたときにクラスごとに反映されます。$SYSTEM.SQL.Util.GetOption("CollectionProjection") を使用して、現在の設定を返すことができます。

コレクションのインデックス作成の詳細は、"InterSystems SQL 最適化ガイド" の “インデックスの定義と構築” の章にある "コレクションのインデックス作成" を参照してください。

17.6.1 使用上の注意と制限事項

- FOR SOME %ELEMENT は、WHERE 節でのみ指定できます。
- %KEY または %VALUE、あるいはその両方は、FOR 述語でのみ指定できます。
- 特定の %KEY または %VALUE は、一度のみ参照できます。
- %KEY および %VALUE は、外部結合には指定できません。
- %KEY および %VALUE は、値式には指定できません (述語のみに指定できます)。

17.7 フリー・テキスト検索を呼び出すクエリ

InterSystems IRIS では、以下のサポートを含む “フリー・テキスト検索” をサポートしています。

- ワイルドカード
- 語幹解析
- 複数語による検索 (N-gram)
- 自動分類
- ディクショナリの管理

この機能によって、フル・テキストによるインデックス機能を SQL でサポートできるほか、コレクションを子テーブルとして投影せずに、SQL を使用してコレクションの個々の要素のインデックスを作成して参照できます。コレクション・インデックス機能をサポートする基本メカニズムとフル・テキスト・インデックス機能をサポートする基本メカニズムは密接に関連していますが、テキスト取得には特別なプロパティが多数あるため、特別なクラスと SQL 機能が用意されています。

詳細は、"InterSystems SQL Search の使用法" を参照してください。

17.8 疑似フィールド変数

InterSystems SQL クエリは、以下の疑似フィールド値をサポートしています。

- %ID ー 実際の RowID フィールドの名前に関係なく、RowID フィールドの値を返します。
- %TABLENAME ー FROM 節で指定された既存テーブルの修飾名を返します。この修飾テーブル名は、FROM 節で指定された大文字と小文字の組み合わせではなく、そのテーブルの定義時に使用された大文字と小文字の組み合わせに従って返されます。FROM 節で未修飾のテーブル名が指定された場合、%TABLENAME は修飾テーブル名 (schema.table) を返し、このスキーマ名はユーザ指定のスキーマ検索パスまたはシステム全体の既定のスキーマ名から取得されます。例えば、FROM 節で `mytable` と指定されている場合は、%TABLENAME 変数は `SQLUser.MyTable` を返す可能性があります。
- %CLASSNAME ー FROM 節で指定された既存テーブルに対応する修飾クラス名 (<パッケージ名>.<クラス名>) を返します。例えば、FROM 節で `SQLUser.mytable` と指定されている場合は、%CLASSNAME 変数は `User.MyTable` を返す可能性があります。

注釈　%CLASSNAME 疑似フィールド値を %ClassName() インスタンス・メソッドと混同しないでください。これらは異なる値を返します。

疑似フィールド変数を返すことができるのは、データが含まれたテーブルについてのみです。

FROM 節で複数のテーブルが指定されている場合は、以下の埋め込み SQL の例に示すように、テーブルのエイリアスを使用する必要があります。

ObjectScript

```
 &sql(SELECT P.Name,P.%ID,P.%TABLENAME,E.%TABLENAME
      INTO :name,:rid,:ptname,:etname
    FROM Sample.Person AS P,Sample.Employee AS E)
      IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
      ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
    WRITE ptname,"Person table Name is: ",name,!
    WRITE ptname,"Person table RowId is: ",rid,!
    WRITE "P alias TableName is: ",ptname,!
    WRITE "E alias TableName is: ",etname,!
```

%TABLENAME 列と %CLASSNAME 列には、Literal_n という既定の列名が割り当てられます。ここで n は、SELECT 文内の疑似フィールド変数の selectItem 位置です。

17.9 クエリ・メタデータ

ダイナミック SQL を使用すると、クエリ内で指定した列の数、クエリ内で指定した列の名前 (またはエイリアス)、クエリ内で指定した列のデータ型など、クエリに関するメタデータが返されます。

以下の ObjectScript のダイナミック SQL の例は、Sample.Person 内の全列の列名および列の ODBC データ型の整数コードを返します。

ObjectScript

```
  SET myquery="SELECT * FROM Sample.Person"
  SET rset = ##class(%SQL.Statement).%New()
  SET qStatus = rset.%Prepare(myquery)
    IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
  SET x=rset.%Metadata.columns.Count()
  WHILE x>0 {
    SET column=rset.%Metadata.columns.GetAt(x)
  WRITE !,x," ",column.colName," ",column.ODBCType
  SET x=x-1 }
  WRITE !,"end of columns"
```

この例では、列は、列の順番とは逆にリストされます。ODBC では InterSystems IRIS のリスト・データ型値をコンマで区切られた値の文字列で表すので、リスト構造化されたデータが含まれている FavoriteColors 列は、データ型 12 (VARCHAR) を返します。

詳細は、このドキュメントの "ダイナミック SQL" の章、および "インターシステムズ・クラス・リファレンス" の %SQL.Statement クラスを参照してください。

17.10 クエリと ECP (エンタープライズ・キャッシュ・プロトコル)

分散キャッシュ・クラスタなど、エンタープライズ・キャッシュ・プロトコル (ECP) を使用する InterSystems IRIS の実装では、クエリ結果を同期化することができます。ECP は分散データ・キャッシュ・アーキテクチャであり、異種のサーバ・システムで構成されるネットワークに分散されるデータとロックを管理します。

ECP の同期化がアクティブな場合、SELECT 文が実行されるたびに、InterSystems IRIS ではすべての保留中 ECP 要求が強制的にデータ・サーバに送信されます。これが完了すると、クライアント・キャッシュの同期が保証されます。この同期化はクエリの Open ロジックで行われます。これがカーソル・クエリである場合は、OPEN カーソル実行となります。

システム全体で ECP の同期を有効にするには、次のように $SYSTEM.SQL.Util.SetOption() メソッドを `SET status=$SYSTEM.SQL.Util.SetOption("ECPSync",1,.oldval)` のように使用します。既定値は 0 です。現在の設定を確認するには、$SYSTEM.SQL.CurrentSettings() を呼び出します。

詳細は、"スケーラビリティ・ガイド" の "InterSystems 分散キャッシュによるユーザ数に応じたシステムの水平方向の拡張" の章を参照してください。

18
ストアド・プロシージャの定義と使用

この章では、InterSystems IRIS® データ・プラットフォーム上の InterSystems SQL でストアド・プロシージャを定義および使用する方法について説明します。

18.1 概要

SQL ルーチンは、SQL クエリ・プロセッサが呼び出すことのできるコードの実行可能単位です。SQL ルーチンには、関数とストアド・プロシージャの 2 つのタイプがあります。関数は、`functionname()` 構文をサポートする SQL 文から呼び出されます。ストアド・プロシージャは、CALL 文からのみ呼び出すことができます。関数は、いくつかの入力が誘導される引数を受け入れ、単一の結果値を返します。ストアド・プロシージャはいくつかの入力、入出力、および出力引数を受け付けます。ストアド・プロシージャは、単一値を返すユーザ定義関数の場合もあります。関数は、CALL 文から呼び出すこともできます。

ほとんどのリレーショナル・データベース・システムと同様に、InterSystems IRIS では、SQL ストアド・プロシージャを作成できます。ストアド・プロシージャ (SP) は、データベースに保存され、SQL コンテキスト内で (CALL 文の使用、または ODBC や JDBC から) 呼び出し可能なルーチンを提供します。

InterSystems IRIS は、ストアド・プロシージャをクラスのメソッドとして定義できるという点で、従来のリレーショナル・データベースより優れています。実際、ストアド・プロシージャは、SQL でも使用できるようにしたクラス・メソッドに過ぎません。ストアド・プロシージャでは、InterSystems IRIS のオブジェクト・ベースのさまざまな機能を利用できます。

- ストアド・プロシージャは、データベースを問い合わせてデータの単一の結果セットを返すクエリとして定義できます。
- ストアド・プロシージャは、単一の値を返すユーザ定義関数として機能する関数プロシージャとして定義できます。
- ストアド・プロシージャは、データベースのデータに変更を加えて 1 つの値または 1 つ以上の結果セットを返すメソッドとして定義できます。

$SYSTEM.SQL.Schema.ProcedureExists() メソッドを使用して、プロシージャが既に存在するかどうかを確認できます。このメソッドは、プロシージャ・タイプ (“function” または “query”) も返します。

18.2 ストアド・プロシージャの定義

InterSystems SQL のほとんどの機能と同様に、ストアド・プロシージャを定義するには、DDL を使用する場合とクラスを使用する場合の 2 種類の方法があります。これらは以下のセクションで説明しています。

18.2.1 DDL を使用したストアド・プロシージャの定義

InterSystems SQL がサポートする、クエリを作成するためのコマンドは以下のとおりです。

- CREATE PROCEDURE では、常にストアド・プロシージャとして投影されるクエリを作成できます。クエリは、単一の結果セットを返すことができます。
- CREATE QUERY では、必要に応じてストアド・プロシージャとして投影できるクエリを作成します。クエリは、単一の結果セットを返すことができます。

InterSystems SQL がサポートする、メソッドまたは関数を作成するためのコマンドは以下のとおりです。

- CREATE PROCEDURE では、常にストアド・プロシージャとして投影されるメソッドを作成できます。メソッドは、単一の値、または 1 つ以上の結果セットを返すことができます。
- CREATE METHOD では、必要に応じてストアド・プロシージャとして投影できるメソッドを作成できます。メソッドは、単一の値、または 1 つ以上の結果セットを返すことができます。
- CREATE FUNCTION では、必要に応じてストアド・プロシージャとして投影できる関数プロシージャを作成できます。関数は、単一の値を返すことができます。

これらのコマンド内に指定する実行可能コードのブロックは、InterSystems SQL または ObjectScript で記述できます。ObjectScript コード・ブロックには、埋め込み SQL を含めることができます。

18.2.2 SQL からクラス名への変換

ストアド・プロシージャの作成に DDL を使用すると、指定した名前はクラス名に変換されます。そのクラスが存在しない場合は、作成されます。

- 名前が修飾されておらず、FOR 節の指定がない場合は、システム全体の既定のスキーマ名がパッケージ名として使用され、その名前の後にドットが続き、その後に生成されたクラス名が続きます。このクラス名は、‘func’、‘meth’、‘proc’、または ‘query’ という文字列の後に句読点文字を削除した SQL 名を組み合わせたものになります。例えば、未修飾のプロシージャ名 `Store_Name` は、`User.procStoreName` のようなクラス名になります。このプロシージャ・クラスには、StoreName() メソッドが含まれます。
- 名前が修飾されていて、FOR 節の指定がない場合は、そのスキーマの名前がパッケージ名に変換され、その名前の後にドットが続き、その後に ‘func’、‘meth’、‘proc’、または ‘query’ という文字列が続き、その後に句読点文字を削除した SQL 名が続きます。指定したパッケージ名は、必要に応じて有効なパッケージ名に変換されます。

  名前が修飾され、FOR 節が指定されている場合は、その FOR 節で指定した修飾されたクラス名により、関数名、メソッド名、プロシージャ名、またはクエリ名で指定したスキーマ名がオーバーライドされます。
- SQL ストアド・プロシージャ名は、識別子の名前付け規約に従います。InterSystems IRIS では、SQL 名から句読点文字を削除して、プロシージャ・クラスとそのクラス・メソッドの一意のクラス・エンティティ名を生成します。

有効なパッケージ名へのスキーマ名の変換は、以下のルールで制御されます。

- スキーマ名にアンダースコアが含まれている場合、この文字はサブパッケージを示すドットに変換されます。例えば、修飾名 `myprocs.myname` により、パッケージ `myprocs` が作成されます。修飾名 `my_procs.myname` により、サブパッケージ `procs` を含むパッケージ `my` が作成されます。

以下の例では、クラス名とその SQL 呼び出しにおける句読点の違いを示します。ここでは、クラス名に 2 つのドットを含んでいるメソッドを定義します。この例では、SQL から呼び出したときは、1 つ目のドットがアンダースコア文字に置換されます。

Class Definition

```
Class Sample.ProcTest Extends %RegisteredObject
 {  ClassMethod myfunc(dummy As %String) As %String [ SqlProc ]
    { /* method code */
      Quit "abc" }
 }
```

SQL

```
SELECT Sample.ProcTest_myfunc(Name)
FROM Sample.Person
```

18.2.3 クラスを使用したメソッド・ストアド・プロシージャの定義

クラス・メソッドはストアド・プロシージャとして公開されます。このようなメソッドは、値を計算して、それを返さずにデータベースに保存するストアド・プロシージャのような動作に対して理想的です。ほぼすべてのクラスで、メソッドをストアド・プロシージャとして公開できます。この例外は、データ型クラス ([ClassType = datatype]) などのジェネレータ・クラスです。ジェネレータ・クラスには実行時コンテキストがありません。プロパティなど、他の一部のエンティティの実行時において、データ型コンテキストを使用するためのみに有効です。

メソッド・ストアド・プロシージャを定義するには、クラス・メソッドを定義し、その SqlProc キーワードを設定します。

Class Definition

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
{

/// This procedure finds total sales for a territory
ClassMethod FindTotal(territory As %String) As %Integer [SqlProc]
{
    // use embedded sql to find total sales
    &sql(SELECT SUM(SalesAmount) INTO :total
            FROM Sales
            WHERE Territory = :territory
    )

    Quit total
}
}
```

このクラスがコンパイルされた後、ストアド・プロシージャ MyApp.Person_FindTotal() として FindTotal() メソッドが SQL に投影されます。メソッドの SqlName キーワードを使用すると、SQL がプロシージャに使用している名前を変更できます。

このメソッドは、プロシージャ・コンテキスト・ハンドラを使用して、プロシージャとその呼び出し元 (ODBC サーバなど) 間でプロシージャ・コンテキストの受け渡しを行います。このプロシージャ・コンテキスト・ハンドラは、**%sqlcontext** オブジェクトを使用して、InterSystems IRIS によって (%qHandle:%SQLProcContext として) 自動的に生成されます。

%sqlcontext は、SQLCODE エラー・ステータス、SQL 行カウント、エラー・メッセージなどのプロパティで構成されます。それらのプロパティは、次のように、対応する SQL 変数を使用して設定されます。

```
  SET %sqlcontext.%SQLCode=SQLCODE
  SET %sqlcontext.%ROWCOUNT=%ROWCOUNT
  SET %sqlcontext.%Message=%msg
```

これらの値に関しては何も行う必要はありませんが、クライアントによって解読されます。**%sqlcontext** オブジェクトは、実行される前に毎回リセットされます。

メソッドは値を返しません。

1 つのクラスのユーザ定義メソッドの最大数は 2000 です。

例えば、CalcAvgScore() メソッドがあるとします。

```
ClassMethod CalcAvgScore(firstname As %String,lastname As %String) [sqlproc]
{
  New SQLCODE,%ROWID
  &sql(UPDATE students SET avgscore =
    (SELECT AVG(sc.score)
     FROM scores sc, students st
     WHERE sc.student_id=st.student_id
       AND st.lastname=:lastname
       AND st.firstname=:firstname)
     WHERE students.lastname=:lastname
       AND students.firstname=:firstname)

  IF ($GET(%sqlcontext)'= "") {
    SET %sqlcontext.%SQLCODE = SQLCODE
    SET %sqlcontext.%ROWCOUNT = %ROWCOUNT
  }
  QUIT
}
```

18.2.4 クラスを使用したクエリ・ストアド・プロシージャの定義

データベースからデータを返す多くのストアド・プロシージャは、標準クエリ・インタフェースで実装されます。この方法は、プロシージャが埋め込み SQL で記述されている限りうまくいきます。以下の例では、WHERE 節に値を提供する埋め込み SQL ホスト変数の使用法に注意してください。

Class Definition

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
{

    /// This procedure result set is the persons in a specified Home_State, ordered by Name
    Query ListPersons(state As %String = "") As %SQLQuery [ SqlProc ]
    {
        SELECT ID,Name,Home_State
        FROM Sample.Person
        WHERE Home_State = :state
        ORDER BY Name
    }
}
```

クエリをストアド・プロシージャとして公開するには、スタジオ・インスペクタで、公開するクエリのエントリの SQLProc フィールドの値を True に変更するか、以下の“[SqlProc]”文字列をクエリ定義に追加します。

```
Query QueryName() As %SQLQuery( ... query definition ... )
    [ SqlProc ]
```

このクラスがコンパイルされた後、ストアド・プロシージャ MyApp.Person_ListPersons として ListPersons クエリが SQL に投影されます。クエリの SqlName キーワードを使用すると、SQL がプロシージャに使用している名前を変更できます。

SQL から MyApp.Person_ListPersons が呼び出されると、クエリの SQL 文によって定義されている結果セットが自動的に返されます。

次の例は、結果セットを使用したストアド・プロシージャです。

Class Definition

```
Class apc.OpiLLS.SpCollectResults1 [ Abstract ]
{

/// This SP returns a number of rows (pNumRecs) from WebService.LLSResults, and updates a property for
 each record
Query MyQuery(pNumRecs As %Integer) As %Query(ROWSPEC = "Name:%String,DOB:%Date") [ SqlProc ]
{
}

/// You put initial code here in the Execute method
ClassMethod MyQueryExecute(ByRef qHandle As %Binary, pNumRecs As %Integer) As %Status
{
    SET mysql="SELECT TOP ? Name,DOB FROM Sample.Person"
```

```
    SET rset=##class(%SQL.Statement).%ExecDirect(,mysql,pNumRecs)
            IF rset.%SQLCODE'=0 {QUIT rset.%SQLCODE}
    SET qHandle=rset
    QUIT $$$OK
}

/// This code is called by the SQL framework for each row, until no more rows are returned
ClassMethod MyQueryFetch(ByRef qHandle As %Binary, ByRef Row As %List,
                         ByRef AtEnd As %Integer = 0) As %Status [ PlaceAfter = NewQuery1Execute ]
{
     SET rset=qHandle
     SET tSC=$$$OK

     FOR {
        ///Get next row, quit if end of result set
        IF 'rset.%Next() {
                SET Row = "", AtEnd = 1
                SET tSC=$$$OK
                QUIT
                }
        SET name=rset.Name
        SET dob=rset.DOB
        SET Row = $LISTBUILD(name,dob)
        QUIT
        }
        QUIT tSC
}

ClassMethod MyQueryClose(ByRef qHandle As %Binary) As %Status [ PlaceAfter = NewQuery1Execute ]
{
        KILL qHandle   //probably not necesary as killed by the SQL Call framework
        QUIT $$$OK
}

}
```

単純な SQL 文としてクエリを記述し、それをクエリ・ウィザードで作成することが可能な場合、クエリを実装する基本的なメソッドに関する知識は必要ありません。

内部で、各クエリに対して、クラス・コンパイラはストアド・プロシージャの名前を基にしてメソッドを生成します。以下のものが含まれます。

- stored-procedure-nameExecute()
- stored-procedure-nameFetch()
- stored-procedure-nameFetchRows()
- stored-procedure-nameGetInfo()
- stored-procedure-nameClose()

クエリのタイプが %SQLQuery の場合、クラス・コンパイラは、生成されたメソッドに自動的にいくつかの埋め込み SQL を挿入します。Execute() は、SQL のストアド・カーソルを宣言して開きます。Fetch() は、それが空の行 (SET Row="") を返すまで繰り返し呼び出されます。オプションで、Fetch() を AtEnd=1 のブーリアン・フラグを返すようにすることもできます。これは、現在の Fetch が最後の行を取り、次の Fetch で空白の行を返すようにすることを示します。ただし、空白行 (Row="") は、結果セットの終了を判断するためのテスト用として常に使用する必要があります。AtEnd=1 を設定するときには、常に Row="" を設定する必要があります。

FetchRows() は、Fetch() を繰り返し呼び出すことと論理的に同等です。GetInfo() は、ストアド・プロシージャに対するシグニチャの詳細を返すために呼び出されます。Close() は、カーソルをクローズします。

これらすべてのメソッドは、クライアントからストアド・プロシージャが呼び出されるたびに自動的に呼び出されますが、論理的には、サーバで起動している ObjectScript から直接呼び出されることもあります。

オブジェクトを Execute() から Fetch() に渡すために、また Fetch() から次の Fetch() の呼び出しに渡すために、クエリ・ハンドラをそのオブジェクトのオブジェクト参照 (oref) に設定することができます。複数のオブジェクトを渡す場合、qHandle を配列として設定できます。

ObjectScript

```
SET qHandle(1)=oref1,qHandle(2)=oref2
```

ユーザが記述したコード（SQL 文ではありません）に基づいて結果セット・ストアド・プロシージャを生成することもできます。

1 つのクラスのユーザ定義クエリの最大数は 200 です。

18.2.5 カスタマイズされたクラス・クエリ

クエリ・モデルに一致しない複雑なクエリやストアド・プロシージャに対しては、多くの場合で、そのメソッドの一部、またはすべてを置き換えることでクエリをカスタマイズする必要があります。このセクションで説明するように、%Library.Query を使用できます。

多くの場合、%SQLQuery（%Library.SQLQuery）タイプの代わりに %Query（%Library.Query）タイプを選択する方が、クエリの実装が簡単になります。これによって、同じ 5 つのメソッドが生成されますが、FetchRows() は単純に Fetch() の呼び出しを繰り返します（%SQLQuery には他の動作を引き起こすいくつかの最適化機能があります）。GetInfo() は単純にシグニチャから情報を得るため、コードを変更する必要が生じることはほとんどありません。これによって、他の 3 つのメソッドそれぞれに対し、クラス・メソッドを作成する上での問題が削減されます。クラスがコンパイルされるとき、コンパイラはこれらのメソッドの存在を検出します。上書きすることはありません。

メソッドは特定のシグニチャを必要とします。それらはすべて %Binary タイプの Qhandle（クエリ・ハンドラ）を使用します。これは、クエリの特性と現状を維持する構造に対するポインタです。Execute() と Fetch() に対する参照と、Close() に対する値によって渡されます。

```
ClassMethod SP1Close(qHandle As %Binary) As %Status
{
   // ...
}

ClassMethod SP1Execute(ByRef qHandle As %Binary,
    p1 As %String) As %Status
{
   // ...
}

ClassMethod SP1Fetch(ByRef qHandle As %Binary,
    ByRef Row As %List, ByRef AtEnd As %Integer=0) As %Status
{
   // ...
}

Query SP1(p1 As %String)
   As %Query(CONTAINID=0,ROWSPEC="lastname:%String") [sqlproc ]
{
}
```

コードは通常、宣言を含み、SQL カーソルを使用します。%SQLQuery タイプのクエリから生成されたカーソルは、自動的に Q14 などの名前を持ちます。クエリに個別の名前が与えられていることを確認してください。

クラス・コンパイラは、カーソルを使用する前に、カーソル宣言を見つける必要があります。したがって、DECLARE 文（通常は Execute 内）は、Close や Fetch と同じ MAC ルーチン内にあるべきで、Close や Fetch よりも先に来る必要があります。ソースを直接編集するには、カーソル宣言が先に来るように Close 定義と Fetch 定義の両方で PLACEAFTER メソッド・キーワードを使用します。

エラー・メッセージは、通常、桁を 1 桁余分に持っている内部カーソル名を参照します。したがって、カーソル Q140 に対するエラー・メッセージは、おそらく Q14 を参照しています。

18.3 ストアド・プロシージャの使用法

ストアド・プロシージャを使用するには、以下のように 2 種類の異なる方法があります。

- SQL CALL 文を使用してストアド・プロシージャを呼び出すことができます。詳細は、"InterSystems SQL リファレンス" の "CALL" 文を参照してください。
- ストアド関数 (単一の値を返すメソッドベースのストアド・プロシージャ) は、SQL クエリ内で組み込み関数であるかのように使用できます。

注釈　SQL 関数を引数として取るストアド・プロシージャを実行する際は、以下の例のように、CALL を使用してストアド・プロシージャを呼び出します。

SQL

```
CALL sp.MyProc(CURRENT_DATE)
```

SELECT クエリでは、SQL 関数引数が指定されたストアド・プロシージャの実行はサポートされません。SELECT では、SQL 関数引数が指定されたストアド関数の実行はサポートされます。

xDBC では、SELECT または CALL を使用した、SQL 関数引数が指定されたストアド・プロシージャの実行はサポートされません。

18.3.1 ストアド関数

ストアド関数は、単一の値を返すメソッド・ベースのストアド・プロシージャです。例えば、以下のクラスは、指定の値の 2 乗を返すストアド関数 Square を定義します。

Class Definition

```
Class MyApp.Utils Extends %Persistent [DdlAllowed]
{
ClassMethod Square(val As %Integer) As %Integer [SqlProc]
{
    Quit val * val
}
}
```

ストアド関数は単に、SqlProc キーワードが指定されたクラス・メソッドです。

注釈　ストアド関数の場合は、ReturnResultsets キーワードを未指定にするか (既定)、このキーワードの前に Not キーワードを指定する必要があります。

SQL クエリ内で、ストアド関数を組み込みの SQL 関数であるかのように使用できます。この場合の関数の名前は、ストアド関数の SQL 名 (この場合は “Square”) を、その関数が定義されているスキーマ (パッケージ) 名 (この場合は “MyApp”) で修飾したものになります。

以下のクエリは、Square 関数を使用します。

SQL

```
SELECT Cost, MyApp.Utils_Square(Cost) As SquareCost FROM Products
```

1 つのパッケージ (スキーマ) に複数のストアド関数を定義する場合は、それぞれが一意の SQL 名を持つ必要があります。

次の例では、Sample.Wages という名前のテーブルを定義しており、このテーブルでは 2 つのデータ・フィールド (プロパティ) と 2 つのストアド関数 (TimePlus と DTime) が定義されています。

Class Definition

```
Class Sample.Wages Extends %Persistent [ DdlAllowed ]
{
  Property Name As %String(MAXLEN = 50) [ Required ];
  Property Salary As %Integer;
  ClassMethod TimePlus(val As %Integer) As %Integer [ SqlProc ]
  {
   QUIT val * 1.5
  }
  ClassMethod DTime(val As %Integer) As %Integer [ SqlProc ]
  {
   QUIT val * 2
  }
}
```

次のクエリでは、これらのストアド・プロシージャを使用して、同じ Sample.Wages テーブル内の各従業員の本給、5 割増賃金、および倍額賃金を返します。

SQL

```
SELECT Name,Salary,
       Sample.Wages_TimePlus(Salary) AS Overtime,
       Sample.Wages_DTime(Salary) AS DoubleTime FROM Sample.Wages
```

次のクエリでは、これらのストアド・プロシージャを使用して、異なる Sample.Employee テーブル内の各従業員の本給、5 割増賃金、および倍額賃金を返します。

SQL

```
SELECT Name,Salary,
       Sample.Wages_TimePlus(Salary) AS Overtime,
       Sample.Wages_DTime(Salary) AS DoubleTime FROM Sample.Employee
```

18.3.2 特権

プロシージャを実行するには、ユーザはそのプロシージャに対する EXECUTE 特権が必要です。GRANT コマンドまたは $SYSTEM.SQL.Security.GrantPrivilege() メソッドを使用して、指定のプロシージャの EXECUTE 特権を指定のユーザに割り当てます。

指定ユーザが指定プロシージャの EXECUTE 特権を所有しているかどうかを判断するには、$SYSTEM.SQL.Security.CheckPrivilege() メソッドを呼び出します。

どのクラス・クエリで特権が確認されるかに関する詳細は、"SQL のユーザ、ロール、および特権" を参照してください。

ユーザが EXECUTE 特権を持っているすべてのプロシージャをリストするには、管理ポータルに移動します。**[システム管理]** から、**[セキュリティ]** を選択し、**[ユーザ]** または **[ロール]** を選択します。目的のユーザまたはロールの **[編集]** を選択し、**[SQL プロシージャ]** タブを選択します。ドロップダウン・リストから目的の **[ネームスペース]** を選択します。

18.4 プロシージャの一覧表示

INFORMATION.SCHEMA.ROUTINES 永続クラスは、現在のネームスペース内のすべてのルーチンとプロシージャに関する情報を表示します。

埋め込み SQL で指定する場合、INFORMATION.SCHEMA.ROUTINES には、#include %occInclude マクロ・プリプロセッサ指示文が必要です。この指示文は、ダイナミック SQL の場合は不要です。

次の例は、現在のネームスペース内のスキーマ “Sample” のすべてのルーチンのルーチン名、メソッドまたはクエリ名、ルーチン・タイプ (PROCEDURE または FUNCTION)、ルーチン本体 (SQL = SQL を使用するクラス・クエリ、EXTERNAL = SQL を使用するクラス・クエリでない)、戻り値のデータ型、およびルーチン定義を返します。

SQL

```
SELECT ROUTINE_NAME,METHOD_OR_QUERY_NAME,ROUTINE_TYPE,ROUTINE_BODY,SQL_DATA_ACCESS,IS_USER_DEFINED_CAST,
DATA_TYPE||' '||CHARACTER_MAXIMUM_LENGTH AS Returns,NUMERIC_PRECISION||':'||NUMERIC_SCALE AS
PrecisionScale,
ROUTINE_DEFINITION
FROM INFORMATION_SCHEMA.ROUTINES WHERE ROUTINE_SCHEMA='Sample'
```

INFORMATION.SCHEMA.PARAMETERS 永続クラスは、現在のネームスペース内のすべてのルーチンとプロシージャの入力および出力パラメータに関する情報を表示します。

次の例は、現在のネームスペース内のスキーマ “Sample” のすべてのルーチンのルーチン名、パラメータ名、入力パラメータまたは出力パラメータのいずれであるか、およびパラメータ・データ型情報を返します。

SQL

```
SELECT SPECIFIC_NAME,PARAMETER_NAME,PARAMETER_MODE,ORDINAL_POSITION,
DATA_TYPE,CHARACTER_MAXIMUM_LENGTH AS MaxLen,NUMERIC_PRECISION||':'||NUMERIC_SCALE AS PrecisionScale
FROM INFORMATION_SCHEMA.PARAMETERS WHERE SPECIFIC_SCHEMA='Sample'
```

管理ポータルの SQL インタフェースの [カタログの詳細] タブを使用して、単一プロシージャに対してほぼ同じ情報を表示できます。プロシージャの [カタログの詳細] には、プロシージャ・タイプ (クエリまたは関数)、クラス名、メソッド名またはクエリ名、説明、および入出力パラメータ数が含まれます。[カタログの詳細] の [ストアド・プロシージャ情報] の表示では、ストアド・プロシージャを実行するオプションも用意されています。

19

ストリーム・データ(BLOB と CLOB)の格納と使用

InterSystems SQL には、InterSystems IRIS® データ・プラットフォーム・データベースにストリーム・データを BLOB (Binary Large Object) または CLOB (Character Large Object) として格納する機能があります。

19.1 ストリーム・フィールドと SQL

InterSystems SQL は 2 種類のストリーム・フィールドをサポートしています。

- 文字ストリーム。大量のテキストに使用します。
- バイナリ・ストリーム。イメージ、音声、またはビデオに使用します。

19.1.1 BLOB と CLOB

InterSystems SQL には、データベースに BLOB (Binary Large Object) や CLOB (Character Large Object) をストリーム・オブジェクトとして保存する機能があります。BLOB は、イメージなどのバイナリ情報の保存に、CLOB は文字情報の保存に使用します。BLOB と CLOB は、最大 4 ギガバイトのデータを保存できます (この制限は JDBC と ODBC の仕様により決められています)。

BLOB と CLOB の処理方法はほぼ同じですが、ODBC または JDBC クライアントからアクセスされたときに、文字エンコード変換を処理する方法 (Unicode からマルチバイト文字への変換など) のみ異なります。BLOB のデータはバイナリ・データとして処理され、他の文字コードに変換されませんが、CLOB のデータは文字データとして処理され、必要に応じて変換されます。

バイナリ・ストリーム・ファイル (BLOB) に 1 つの出力不能文字 $CHAR(0) が含まれる場合、これは空のバイナリ・ストリームと見なされます。これは、空のバイナリ・ストリーム値 "" と同じです。存在はしていますが (NULL ではありませんが)、その長さは 0 です。

オブジェクトの観点から見ると、BLOB と CLOB は ストリーム・オブジェクトと見なされます。詳細は、"クラスの定義と使用" の "ストリームを使用した作業" の章を参照してください。

19.1.2 ストリーム・データ・フィールドの定義

InterSystems SQL では、ストリーム・フィールドにさまざまなデータ型名をサポートしています。これらの InterSystems のデータ型名は、以下に対応する同義語です。

- 文字ストリーム：データ型 LONGVARCHAR。%Stream.GlobalCharacter クラスと ODBC/JDBC データ型 -1 にマップされます。
- 文字ストリーム：データ型 LONGVARBINARY。%Stream.GlobalBinary クラスと ODBC/JDBC データ型 -4 にマップされます。

一部の InterSystems ストリーム・データ型では、データ精度値を指定できます。この値は空命令で、許容されたサイズのストリーム・データには影響を与えません。予期されるサイズの将来のデータをドキュメント化するために提供されています。

ストリーム・データ型のデータ型マッピングについては、"InterSystems SQL リファレンス" の "データ型" リファレンス・ページを参照してください。

テーブル (永続クラス) のフィールド (プロパティ) を定義する方法については、"永続クラスの作成によるテーブルの定義" および "DDL を使用したテーブルの定義" を参照してください。永続クラスのストリーム・プロパティを定義する際、オプションで LOCATION パラメータを指定できます。"クラスの定義と使用" の "ストリームを使用した作業" の章の "ストリーム・プロパティの宣言" を参照してください。

以下の例では、2 つのストリーム・フィールドを含むテーブルを定義します。

SQL

```
CREATE TABLE Sample.MyTable (
    Name VARCHAR(50) NOT NULL,
    Notes LONGVARCHAR,
    Photo LONGVARBINARY)
```

19.1.2.1 ストリーム・フィールドの制約

ストリーム・フィールドの定義には、以下のフィールドのデータ制約が適用されます。

ストリーム・フィールドは NOT NULL として定義できます。

ストリーム・フィールドは DEFAULT 値、ON UPDATE 値、または COMPUTECODE 値を取ることができます。

ストリーム・フィールドを UNIQUE、主キー・フィールド、または IdKey として定義することはできません。このように定義しようとすると、SQLCODE -400 の致命的なエラーが発生し、%msg は、"ERROR #5414: : Sample.MyTable::MYTABLEUNIQUE2::Notes Unique PrimaryKey IdKey > ERROR #5030: 'Sample.MyTable' " のようになります。

ストリーム・フィールドを COLLATE 値を指定して定義することはできません。このように定義しようとすると、SQLCODE -400 の致命的なエラーが発生し、%msg は、"ERROR #5480: : Sample.MyTable:Photo:COLLATION > ERROR #5030: 'Sample.MyTable' " のようになります。

19.1.3 ストリーム・データ・フィールドへのデータの挿入

ストリーム・フィールドにデータを INSERT するには、以下の 3 つの方法があります。

- %Stream.GlobalCharacter フィールド：文字ストリーム・データを直接挿入できます。以下に例を示します。

  SQL

  ```
  INSERT INTO Sample.MyTable (Name,Notes)
      VALUES ('Fred','These are extensive notes about Fred')
  ```

- %Stream.GlobalCharacter および %Stream.GlobalBinary フィールド：OREF を使用して、ストリーム・データを挿入できます。Write() メソッドを使用して文字列を文字ストリームに追加するか、WriteLine() メソッドを使用して文字列と行ターミネータを文字ストリームに追加します。既定では、行ターミネータは $CHAR(13,10) (キャリッジ・リターン/改行) です。LineTerminator プロパティを設定して、この行ターミネータを変更できます。次の例の最初の部分では、

2 つの文字列とそのターミネータで構成される文字ストリームを作成し、埋め込み SQL を使用してストリーム・フィールドに挿入します。この例の 2 つ目の部分では、文字ストリーム長を返して、ターミネータを示す文字ストリーム・データを表示します。

ObjectScript

```
CreateAndInsertCharacterStream
  SET gcoref=##class(%Stream.GlobalCharacter).%New()
  DO gcoref.WriteLine("First Line")
  DO gcoref.WriteLine("Second Line")
  &sql(INSERT INTO Sample.MyTable (Name,Notes)
    VALUES ('Fred',:gcoref))
       IF SQLCODE<0 {WRITE "SQLCODE ERROR:"_SQLCODE_" "_%msg  QUIT}
       ELSE {WRITE "Insert successful",!}
DisplayTheCharacterStream
  KILL ^CacheStream
  WRITE gcoref.%Save(),!
  ZWRITE ^CacheStream
```

- %Stream.GlobalCharacter および %Stream.GlobalBinary フィールド：ファイルから読み取ることで、ストリーム・データを挿入できます。以下に例を示します。

  ObjectScript

  ```
    SET myf="C:\InterSystems\IRIS\mgr\temp\IMG_0190.JPG"
    OPEN myf:("RF"):10
    USE myf:0
    READ x(1):10
    &sql(INSERT INTO Sample.MyTable (Name,Photo) VALUES ('George',:x(1)))
         IF SQLCODE<0 {WRITE "INSERT Failed:"_SQLCODE_" "_%msg  QUIT}
         ELSE {WRITE "INSERT successful",!}
    CLOSE myf
  ```

  詳細は、"入出力デバイス・ガイド" の "シーケンシャル・ファイルの入出力" を参照してください。

DEFAULT 値または算出値として挿入される文字列データは、ストリーム・フィールドに適した形式で格納されます。

19.1.4 ストリーム・フィールド・データのクエリ

クエリ select-item は、以下の例に示すように、ストリーム・フィールドを選択し、ストリーム・オブジェクトの整形式 OID (オブジェクト ID) 値を返します。

SQL

```
SELECT Name,Photo,Notes
FROM Sample.MyTable WHERE Photo IS NOT NULL
```

OID は、`$lb("1","%Stream.GlobalCharacter","^EW3K.Cn9X.S")` のような %List 形式のデータ・アドレスです。

- OID の最初の要素は、テーブルに挿入された各ストリーム・データ値に割り当てられる、連続する正の整数 (1 で始まる) です。例えば、行 1 にストリーム・フィールド Photo および Notes の値が挿入される場合、これらには 1 と 2 が割り当てられます。行 2 に Notes の値が挿入される場合、3 が割り当てられます。行 3 に Photo および Notes の値が挿入される場合、これらには 4 と 5 が割り当てられます。割り当て順序は、INSERT コマンドで指定されている順序ではなく、テーブル定義でフィールドがリストされている順序です。既定では、ストリーム位置のグローバル・カウンタに対応する、単一の整数シーケンスが使用されます。ただし、以下に説明するように、テーブルに複数のストリーム・カウンタが存在することがあります。

  UPDATE 操作では、初期整数値は変更されません。DELETE 操作では、整数シーケンスにギャップが生じる可能性がありますが、これらの整数値は変更されません。DELETE を使用してすべてのレコードを削除しても、この整数カウンタはリセットされません。すべてのテーブル・ストリーム・フィールドで既定の StreamLocation 値が使用されている場合、TRUNCATE TABLE を使用してすべてのレコードを削除すると、この整数カウンタはリセットされます。

TRUNCATE TABLE を使用して、埋め込みオブジェクト (%SerialObject) クラスのストリーム整数カウンタをリセットすることはできません。

- OID の 2 番目の要素はストリーム・データ型 (%Stream.GlobalCharacter または %Stream.GlobalBinary) です。
- OID の 3 番目の要素はグローバル変数です。既定では、その名前はパッケージ名およびテーブルに対応する永続クラス名から生成されます。“S” (ストリームの場合) が追加されます。
  - テーブルが SQL CREATE TABLE コマンドを使用して作成された場合、これらのパッケージ名および永続クラス名はそれぞれ 4 つの文字にハッシュ化されます (例 : `^EW3K.Cn9X.S`)。このグローバルには、最後に割り当てられたストリーム・データ挿入カウンタの値が含まれます。ストリーム・フィールド・データが挿入されていない場合、または TRUNCATE TABLE を使用してすべてのテーブル・データが削除されている場合、このグローバルは定義されません。
  - テーブルが永続クラスとして作成された場合、これらのパッケージ名および永続クラス名はハッシュ化されません (例 : `^Sample.MyTableS`)。既定では、これは StreamLocation ストレージ・キーワード `<StreamLocation>^Sample.MyTableS</StreamLocation>` の値です。

    既定のストリーム位置は、`^Sample.MyTableS` などのグローバルです。このグローバルを使用して、カスタム LOCATION のない、すべてのストリーム・プロパティ (フィールド) への挿入をカウントします。例えば、Sample.MyTable 内のすべてのストリーム・プロパティが既定のストリーム位置を使用するとします。Sample.MyTable のストリーム・プロパティに 10 個のストリーム・データ値が挿入されている場合、`^Sample.MyTableS` グローバルには値 10 が含まれます。このグローバルには、最後に割り当てられたストリーム・データ挿入カウンタの値が含まれます。ストリーム・フィールド・データが挿入されていない場合、または TRUNCATE TABLE を使用してすべてのテーブル・データが削除されている場合、このグローバルは定義されません。

    ストリーム・フィールド・プロパティを定義する際、次のようなカスタム LOCATION を定義できます : `Property Note2 As %Stream.GlobalCharacter (LOCATION="^MyCustomGlobalS");`。この状況では、`^MyCustomGlobalS` グローバルは、この LOCATION を指定するストリーム・プロパティのストリーム・データ挿入カウンタとして機能します。LOCATION を指定しないストリーム・プロパティは、既定のストリーム位置グローバル (`^Sample.MyTableS`) をストリーム・データ挿入カウンタとして使用します。各グローバルは、その位置に関連付けられているストリーム・プロパティの挿入をカウントします。ストリーム・フィールド・データが挿入されていない場合、位置グローバルは定義されません。1 つまたは複数のストリーム・プロパティが LOCATION を定義している場合、TRUNCATE TABLE はストリーム・カウンタをリセットしません。

これらのストリーム位置グローバル変数の添え字には、各ストリーム・フィールドのデータが含まれます。例えば、`^EW3K.Cn9X.S(3)` は 3 番目に挿入されたストリーム・データ項目を表します。`^EW3K.Cn9X.S(3,0)` はデータの長さです。`^EW3K.Cn9X.S(3,1)` は実際のストリーム・データ値です。

注釈　ストリーム・フィールドの OID は、RowID または参照フィールドに返された OID と同じではありません。%OID 関数は RowID または参照フィールドの OID を返します。%OID をストリーム・フィールドと一緒に使用することはできません。ストリーム・フィールドを %OID の引数として使用しようとすると、SQLCODE -37 エラーが返されます。

クエリの WHERE 節または HAVING 節でのストリーム・フィールドの使用は、厳しく制限されます。ストリーム・フィールドで等値条件またはその他の関係演算子 (=、!=、<、>)、包含関係演算子 (]) または後続関係演算子 ([) を使用することはできません。ストリーム・フィールドでこれらの演算子を使用しようとすると、SQLCODE -313 エラーが返されます。ストリーム・フィールドを使用する有効な述語については、″述語条件とストリーム″ を参照してください。

19.1.4.1 結果セットの表示

- プログラムから実行されたダイナミック SQL は、形式 `$lb("6","%Stream.GlobalCharacter","^EW3K.Cn9X.S")` で OID を返します。

- SQL シェルはダイナミック SQL として実行され、形式 `$lb("6","%Stream.GlobalCharacter","^EW3K.Cn9X.S")` で OID を返します。
- 埋め込み SQL は同じ OID を返しますが、エンコードされた %List として返します。$LISTTOSTRING 関数を使用すると、要素をコンマで区切った文字列として OID を表示できます：`6,%Stream.GlobalBinary,^EW3K.Cn9X.S`。

クエリを管理ポータルの SQL 実行インタフェースから実行すると、OID は返されません。代わりに以下のようになります。

- 文字ストリーム・フィールドが文字ストリーム・データの最初の 100 文字を返します。文字ストリーム・データが 100 文字より長い場合、100 番目の文字の後の省略記号 (...) によって、それが示されます。これは SUBSTRING(cstream-field,1,100) と同等です。
- バイナリ・ストリーム・フィールドは、文字列 `<binary>` を返します。

同じ値が、管理ポータルの SQL インタフェースのテーブル・データの [テーブルを開く] 表示に示されます。

OID 値を管理ポータルの SQL **実行**インタフェースから表示するには、`SELECT Name, ''||Photo, ''||Notes FROM Sample.MyTable` のように、空の文字列をストリーム値に連結します。

19.1.5 DISTINCT、GROUP BY、および ORDER BY

データそのものに重複が含まれる場合でも、各ストリーム・データ・フィールドの OID 値は一意です。これらの SELECT 節は、データ値ではなくストリーム OID 値に対して機能します。このため、クエリでストリーム・フィールドに適用された場合、以下のようになります。

- DISTINCT 節は、重複するストリーム・データ値には影響を与えません。DISTINCT 節は、ストリーム・フィールドが NULL のレコード数を、1 つの NULL レコードに削減します。
- GROUP BY 節は、重複するストリーム・データ値には影響を与えません。GROUP BY 節は、ストリーム・フィールドが NULL のレコード数を、1 つの NULL レコードに削減します。
- ORDER BY 節は、ストリーム・データ値をデータ値ではなく OID 値で並べます。ORDER BY 節は、ストリーム・フィールド・データ値を持つレコードをリストする前に、ストリーム・フィールドが NULL のレコードをリストします。

19.1.6 述語条件とストリーム

IS [NOT] NULL 述語は、以下の例に示すように、ストリーム・フィールドのデータ値に適用可能です。

SQL

```
SELECT Name,Notes
FROM Sample.MyTable WHERE Notes IS NOT NULL
```

BETWEEN、EXISTS、IN、%INLIST、LIKE、%MATCHES、および %PATTERN 述語は、以下の例に示すようにストリーム・オブジェクトの OID 値に適用可能です。

SQL

```
SELECT Name,Notes
FROM Sample.MyTable WHERE Notes %MATCHES '*1[0-9]*GlobalChar*'
```

ストリーム・フィールドでその他の述語条件を使用しようとすると、SQLCODE -313 エラーが返されます。

19.1.7 集約関数とストリーム

COUNT 集約関数は、以下の例に示すように、ストリーム・フィールドを取得し、フィールドに NULL でない値を含む行をカウントします。

SQL

```
SELECT COUNT(Photo) AS PicRows,COUNT(Notes) AS NoteRows
FROM Sample.MyTable
```

ただし、COUNT(DISTINCT) はストリーム・フィールドではサポートされません。

その他の集約関数は、ストリーム・フィールドではサポートされません。ストリーム・フィールドをその他の集約関数で使用しようとすると、SQLCODE -37 エラーが返されます。

19.1.8 スカラ関数とストリーム

InterSystems SQL では、関数をストリーム・フィールドに適用できません。ただし、%OBJECT、CHARACTER_LENGTH (または CHAR_LENGTH または DATALENGTH)、SUBSTRING、CONVERT、XMLCONCAT、XMLELEMENT、XMLFOREST、および %INTERNAL 関数は例外です。ストリーム・フィールドをその他の SQL 関数で引数として使用しようとすると、SQLCODE -37 エラーが返されます。

- %OBJECT 関数は、以下の例に示すように、ストリーム・オブジェクトを開き (OID を取得し)、oref (オブジェクト参照) を返します。

  SQL

  ```
  SELECT Name,Notes,%OBJECT(Notes) AS NotesOref
  FROM Sample.MyTable WHERE Notes IS NOT NULL
  ```

- CHARACTER_LENGTH 関数、CHAR_LENGTH 関数、および DATALENGTH 関数は、以下の例に示すように、ストリーム・フィールドを取得し、実際のデータ長を返します。

  SQL

  ```
  SELECT Name,DATALENGTH(Notes) AS NotesNumChars
  FROM Sample.MyTable WHERE Notes IS NOT NULL
  ```

- SUBSTRING 関数は、以下の例に示すように、ストリーム・フィールドを取得し、ストリーム・フィールドの実際のデータ値の指定された部分文字列を返します。

  SQL

  ```
  SELECT Name,SUBSTRING(Notes,1,10) AS Notes1st10Chars
  FROM Sample.MyTable WHERE Notes IS NOT NULL
  ```

  管理ポータルの SQL **実行**インタフェースから発行された場合、SUBSTRING 関数は最大 100 文字のストリーム・フィールド・データの部分文字列を返します。ストリーム・データの指定された部分文字列が 100 文字より長い場合、これは 100 番目の文字の後の省略記号 (...) で示されます。

- CONVERT 関数を使用して、以下の例に示すように、ストリーム・データ型を VARCHAR に変換できます。

  SQL

  ```
  SELECT Name,CONVERT(VARCHAR(100),Notes) AS NotesTextAsStr
  FROM Sample.MyTable WHERE Notes IS NOT NULL
  ```

  `CONVERT(datatype,expression)` 構文は、ストリーム・データの変換をサポートします。VARCHAR の精度が実際のストリーム・データの長さより小さい場合、返り値を VARCHAR の精度まで切り捨てます。VARCHAR の精度

が実際のストリーム・データの長さより大きい場合、返り値は実際のストリーム・データの長さになります。埋め込みは行われません。

`{fn CONVERT(expression,datatype)}` 構文はストリーム・データの変換をサポートしていません。SQLCODE -37 エラーが返されます。

- %INTERNAL 関数は、ストリーム・フィールドに使用できますが、演算を実行しません。

19.2 ストリーム・フィールドの同時処理ロック

InterSystems IRIS は、ストリーム・データに対するロックを取得することで、別のプロセスによる同時操作からストリーム・データの値を保護します。

InterSystems IRIS は、書き込み操作を実行する前に、排他ロックを取得します。この排他ロックは、書き込み操作の完了直後に解放されます。

InterSystems IRIS は、最初の読み取り操作が行われるときに共有ロックを取得します。共有ロックは、ストリームが実際に読み取られるときにのみ取得され、ストリーム全体がディスクから内部の一時入力バッファに読み込まれた直後に解放されます。

19.3 InterSystems IRIS メソッドでのストリーム・フィールドの使用

InterSystems IRIS メソッド内で、埋め込み SQL またはダイナミック SQL を使用して、直接的に BLOB 値または CLOB 値を使用することはできません。代わりに、SQL を使用して BLOB または CLOB のストリーム識別子を見つけてから、%AbstractStream オブジェクトのインスタンスを作成してデータにアクセスします。

19.4 ODBC からのストリーム・フィールドの使用

ODBC の規格では、BLOB と CLOB フィールドの認識や特別な処理方法などは、提供されていません。InterSystems SQL は、ODBC 内の CLOB フィールドを LONGVARCHAR (-1) タイプとして表します。BLOB フィールドは、LONGVARBINARY (-4) タイプとして表されます。ストリーム・データ型の ODBC/JDBC データ型マッピングについては、"InterSystems SQL リファレンス" の "データ型" リファレンス・ページの "データ型の整数コード" を参照してください。

ODBC ドライバおよびサーバは、特別なプロトコルを使用して、BLOB や CLOB フィールドへアクセスします。通常、CLOB や BLOB フィールドを使用するには、ODBC アプリケーションに特別なコードを書く必要があります。一般的に、標準のレポート・ツールはこれらをサポートしていません。

19.5 JDBC からのストリーム・フィールドの使用

Java プログラムで、標準 JDBC BLOB/CLOB インタフェースを使用して、BLOB や CLOB からデータの検索や設定を実行できます。以下はその例です。

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT MyCLOB,MyBLOB FROM MyTable");
rs.next();      // fetch the Blob/Clob

java.sql.Clob clob = rs.getClob(1);
java.sql.Blob blob = rs.getBlob(2);

// Length
System.out.println("Clob length = " + clob.length());
System.out.println("Blob length = " + blob.length());

// ...
```

注釈　BLOB または CLOB の使用を終了したら、free() メソッドを明示的に呼び出して Java のオブジェクトを閉じ、サーバにメッセージを送信してストリーム・リソース (オブジェクトとロック) を解放する必要があります。Java オブジェクトを範囲外にするだけでは、サーバ・リソースをクリーンアップするためのメッセージは送信されません。

20

SQL のユーザ、ロール、および特権

InterSystems IRIS® は、システム・レベルのセキュリティと、SQL 関連セキュリティ機能の追加セットの両方を備えています。インターシステムズの SQL セキュリティは、インターシステムズのデータベース・レベル保護を超えるレベルのセキュリティ機能を提供します。SQL セキュリティとシステム・レベルのセキュリティの主な違いは以下のとおりです。

- SQL 保護では、システム・レベルのセキュリティに比べ、詳細な保護が可能です。テーブル、ビュー、およびストアド・プロシージャに対する特権を定義できます。
- SQL 特権はユーザとロールに与えることができます。システム・レベルの特権はロールに対してのみ与えられます。
- SQL 特権を保持することにより、SQL アクションの実行に必要なすべての関連システム特権が暗黙的に与えられます(反対に、システム・レベルの特権はテーブル・レベルの特権を意味しません)。特権の種類については、“SQL 特権とシステム特権” のセクションで説明します。

注釈　InterSystems SQL は、InterSystems IRIS データ・プラットフォームで ODBC、JDBC、ダイナミック SQL、および SQL シェル・インタフェースの特権チェックを実行します。埋め込み SQL 文は、特権チェックを実行しません。これは、埋め込み SQL 文を使用する前に、その埋め込み SQL を使用するアプリケーションで特権をチェックしていることが前提となっているからです。同様に、%SQL.Statement オブジェクトを伴わないクラス・クエリの直接呼び出しはアプリケーション・アクセスと見なされ、SQL 特権が確認されません。

20.1 SQL 特権とシステム特権

SQL 固有のメカニズムを使用してテーブルやその他の SQL エンティティを操作するには、適切な SQL 特権が必要です。システム・レベルの特権では十分ではありません。ユーザに直接 SQL 特権を付与することも、ユーザが SQL 特権を持つロールに属することもできます。

注釈　SQL セキュリティとシステム・レベルのセキュリティで、ロールを共有できます。つまり、システム・レベル特権と SQL 特権の両方を 1 つのロールに備えることができます。

Windows マシンの InterSystems IRIS インスタンスについて、以下の例を考えてみましょう。

- USER ネームスペースに User.MyPerson という永続クラスがあります。このクラスは SQL に SQLUser.MyPerson テーブルとして投影されます。
- Test という名前のユーザがいます。このユーザはどのロールにも属しません (したがって、システム特権を何も持ちません) が、SQLUser.MyPerson テーブルに対するすべての特権を持ちます (その他の SQL 特権は持ちません)。

- Test2 という 2 人目のユーザがいます。このユーザは、**%DB_USER** ロール (このため、USER データベースに対するデータの読み書きが可能)、および **%SQL** ロール (このため、%Service_Bindings サービスを通じた SQL アクセスが可能) を持ちます。また、カスタム・ロールを通じて、コンソールと **%Development** を使用するための特権を持ちます。

ユーザ Test が SQL 固有の任意のメカニズム (ODBC を使用するものなど) を通じて SQLUser.MyPerson テーブル内のデータに対して読み取りまたは書き込みを実行すると、この操作は成功します。これは、InterSystems IRIS がユーザ Test を **%SQL** ロール (**%Service_SQL:Use** 特権を含む) および **%DB_USER** ロールのメンバにして、ユーザが接続の確立に必要な特権を持つようにするためです。これは、%System/%Login/Login イベントなど、接続により生成される監査イベントで確認できます (ユーザ Test が、ターミナル・オブジェクト・メカニズムを使用しようとすると失敗します。これは、ユーザ Test がこれらの操作に必要な特権を持たないためです)。

ユーザ Test2 が SQL 固有の任意のメカニズム (ODBC を使用するものなど) を通じて SQLUser.MyPerson テーブル内のデータに対して読み取りまたは書き込みを実行すると、この操作は失敗します。これはユーザ Test2 がこのテーブルに対して必要な特権を持たないからです(ユーザ Test2 が、オブジェクト・メカニズムを使用してターミナルで同じデータを表示しようとすると、この操作は成功します。これはユーザ Test2 が、この種類の接続に必要な特権を持っているためです)。

SQL 特権に関する詳細は、"SQL 特権" を参照してください。

InterSystems IRIS の永続クラスは、行レベル・セキュリティの特権もサポートします。

20.2 %Admin_Secure 権限

USE 権限のある %Admin_Secure 管理リソースを持つユーザは、次の操作を実行できます・

- ユーザの作成、変更、削除。
- ロールの作成、変更、削除。
- ユーザに付与された特権の参照。
- ロールに付与された特権の参照。
- 別のユーザから付与された SQL 特権の取り消し。

%Admin_Secure を使用することにより、ユーザはシステム上で完全なセキュリティ特権が付与されていなくてもこれらの操作を実行できます。

20.3 %Admin_RoleEdit 権限

USE 権限のある %Admin_RoleEdit 管理リソースを持つユーザは、次の操作を実行できます。

- ロールの作成または削除。

%Admin_RoleEdit を使用することにより、ユーザはシステム上で完全なセキュリティ特権が付与されていなくてもこれらの操作を実行できます。

20.4 %Admin_UserEdit 権限

USE 権限のある %Admin_UserEdit 管理リソースを持つユーザは、次の操作を実行できます・

- ユーザの作成、変更、削除。

%Admin_UserEdit を使用することにより、ユーザはシステム上で完全なセキュリティ特権が付与されていなくてもこれらの操作を実行できます。

20.5 ユーザ

InterSystems SQL ユーザは、InterSystems セキュリティで定義されているユーザと同じです。SQL コマンドまたは管理ポータルを使用してユーザを定義できます。

- SQL でユーザを作成するには、CREATE USER 文を使用します。これによって、ユーザ名とユーザ・パスワードのみが作成されます。新規に作成したユーザにはロールがありません。ユーザに特権とロールを割り当てるには、GRANT 文を使用する必要があります。既存のユーザ定義を変更するには、ALTER USER 文と DROP USER 文を使用します。
- 管理ポータルで、**[システム管理]**、**[セキュリティ]**、**[ユーザ]** の順に選択します。ページ上部の **[新規ユーザ作成]** ボタンをクリックします。これによって、**[ユーザ編集]** ページが表示されます。ここで、ユーザ名、ユーザ・パスワード、およびその他のパラメータを指定できます。ユーザを作成すると、その他のタブが使用可能になります。これらのタブでは、ユーザのロール、ユーザの一般的な SQL 特権、ユーザのテーブルレベルの特権、使用可能なビュー、および実行可能なストアド・プロシージャを指定できます。

ユーザに SQL テーブルの特権または一般的な SQL 特権がある場合、ユーザの **[ロール]** タブで付与または削除されるロールは、ODBC などの SQL ベースのサービスを使用したテーブルへのユーザ・アクセスには影響しません。SQL ベースのサービスでは、テーブルベースの特権がリソース・ベースの特権よりも優先されるためです。

%Library.SQLCatalogPriv クラスのクエリを使用して、以下を示すことができます。

- すべてのユーザ SQLUsers()
- 指定したユーザに付与されているすべての特権 SQLUserPrivs(“username”)
- 指定したユーザに付与されているすべてのシステム特権 SQLUserSysPrivs(“username”)
- 指定したユーザに付与されているすべてのロール SQLUserRole(“username”)

以下の例は、現在のユーザに付与されている特権を示しています。

ObjectScript

```
SET statemt=##class(%SQL.Statement).%New()
SET cqStatus=statemt.%PrepareClassQuery("%Library.SQLCatalogPriv","SQLUserPrivs")
  IF cqStatus'=1 {WRITE "%PrepareClassQuery failed:" DO $System.Status.DisplayError(cqStatus) QUIT}

SET rset=statemt.%Execute($USERNAME)
WRITE "Privileges for ",$USERNAME
DO rset.%Display()
```

20.5.1 スキーマ名としてのユーザ名

状況によっては、ユーザ名を暗黙的に SQL スキーマ名として使用できます。SQL 識別子では使用できない文字がユーザ名に使用されていると、この処理によって問題が発生します。例えば、複数ドメイン構成ではユーザ名に “@” 文字が使用されます。

InterSystems IRIS では、区切り識別子の構成パラメータの設定に応じて、この状況が区別して扱われます。

- 区切り識別子が使用できるようになっていると、特別な処理は発生しません。

- 区切り識別子が使用できないようになっていると、スキーマ名に使用できない文字がユーザ名から削除されてスキーマ名が形成されます。例えば、“documentation@intersystems.com”というユーザ名から“documentationintersys-temscom”というスキーマ名が生成されます。

これによって、SQL CURRENT_USER 関数から返される値が変化することはありません。必ず $USERNAME と同じ値が返されます。

20.6 ロール

ユーザまたはロールには SQL 特権が割り当てられます。ロールを使用すれば、複数のユーザに同じ特権を設定できます。SQL セキュリティとシステム・レベルのセキュリティで、ロールを共有できます。つまり、システム特権と SQL 特権の両方を 1 つのロールに備えることができます。

管理ポータルの **[システム管理]→[セキュリティ]→[ロール]** ページに、InterSystems IRIS インスタンスのロール定義のリストが含まれています。特定のロールの詳細を表示または変更するには、そのロールの **[名前]** リンクを選択します。表示される **[ロール編集]** ページには、ロール特権と、その特権を持つユーザまたはロールに関する情報が含まれています。

[一般] タブには、InterSystems セキュリティ・リソースに対するロールの特権が示されます。ロールに SQL 特権のみが含まれている場合、[一般] タブのリソース・テーブルには、ロールの特権として“定義なし”が示されます。

[SQL権限] タブには、InterSystems SQL リソースに対するロールの特権が示されます。ここでネームスペースのドロップダウン・リストを使用すると、各ネームスペースのリソースを表示できます。特権はネームスペース別に示されているため、特定のネームスペースで特権を持たないロールのリストは“なし”と表示されます。

注釈　特権の定義にはロールを使用し、それらのロールに特定のユーザを指定してください。これには、以下のような利点があります。

1. SQL エンジンにとって、各ユーザのエントリを確認するより、比較的小規模なロール・データベースを確認することにより特権のレベルを決定する方が、非常に効率的です。
2. 多数の個人ユーザ設定を持つシステムに比べ、小規模なロールの集合を使用したシステムの方が、管理が容易です。

例えば、あるアクセス特権を持つ“ACCOUNTING”というロールを定義するとします。Accounting Department（経理部）の人員が増加するにつれ、新しいユーザを定義し、ACCOUNTING ロールに追加します。ACCOUNTING へのアクセス権を変更する必要がある場合、1 回変更を加えれば、その変更が経理部のすべてのメンバに対し自動的に適用されます。

ロールには、他のロールを含めることができます。例えば、ACCOUNTING ロールには BILLINGCLERK ロールを含めることができます。ACCOUNTING ロールを付与されたユーザは、ACCOUNTING ロールと BILLINGCLERK ロールの両方の特権を持つことになります。

CREATE USER、CREATE ROLE、ALTER USER、GRANT、DROP USER、および DROP ROLE の各 SQL コマンドを使用してユーザおよびロールを定義することもできます。

%Library.SQLCatalogPriv クラスのクエリを使用して、以下を示すことができます。

- すべてのロール SQLRoles()
- 指定したロールに付与されているすべての特権 SQLRolePrivileges(“rolename”)
- 指定したロールに付与されているすべてのロールまたはユーザ SQLRoleUser(“rolename”)
- 指定したユーザに付与されているすべてのロール SQLUserRole(“username”)

20.7 SQL 特権

ユーザまたはロールには SQL 特権が割り当てられます。ロールを使用すれば、複数のユーザに同じ特権を設定できます。

InterSystems SQL は、管理とオブジェクトという 2 つのタイプの特権をサポートしています。

- 管理特権はネームスペース固有のものです。

  管理特権は、テーブルの作成に必要な %CREATE_TABLE 特権などの、オブジェクトのタイプの作成、変更、および削除を管理します。%ALTER_TABLE 特権は、テーブルを変更するためだけでなく、インデックスの作成または削除、トリガの作成または削除、および TUNE TABLE の実行にも必要です。

  管理特権には、%NOCHECK、%NOINDEX、%NOLOCK、%NOJOURN、および %NOTRIGGER も含まれます。これらにより、INSERT、UPDATE、INSERT OR UPDATE、または DELETE を実行するユーザが、対応するキーワード制約を適用できるかどうかが決まります。TRUNCATE TABLE を実行するユーザには、%NOTRIGGER 管理特権を割り当てる必要があります。

- オブジェクト特権は、テーブル、ビュー、またはストアド・プロシージャに固有です。オブジェクト特権は、(テーブル、ビュー、列、またはストアド・プロシージャという SQL 用語での) 特定の名前付き SQL オブジェクトに対するアクセスのタイプを指定します。ユーザが SQL オブジェクトの所有者 (作成者) である場合、ユーザにはそのオブジェクトに対するすべての特権が自動的に付与されます。

  テーブルレベルのオブジェクト特権では、テーブルまたはビューのすべての列のデータにアクセス (%ALTER、DELETE、SELECT、INSERT、UPDATE、EXECUTE、REFERENCES) できます。現在存在している列と今後追加される列の両方が対象です。

  列レベルのオブジェクト特権では、テーブルまたはビューの指定した列のデータのみにアクセスできます。RowID や Identity (ID) などのシステム定義の値を使用して列に列レベルの特権を割り当てる必要はありません。

  ストアド・プロシージャ・オブジェクト特権では、プロシージャの EXECUTE 特権の割り当てが指定のユーザまたはロールに対して許可されます。

詳細は、"GRANT コマンド" を参照してください。

20.7.1 SQL 特権の付与

以下のようにして、特権を与えます。

- 管理ポータルを使用します。[**システム管理**] から、[**セキュリティ**] を選択し、[**ユーザ**] または [**ロール**] を選択します。目的のユーザまたはロールを選択してから、該当するタブを選択します。管理特権の場合は [**SQL権限**]、オブジェクト特権の場合は [**SQLテーブル**]、[**SQLビュー**]、または [**SQLプロシージャ**] となります。
- SQL から GRANT コマンドを使用して、指定されたユーザまたはロール (あるいはユーザまたはロールのリスト) に、特定の管理特権またはオブジェクト特権を付与します。REVOKE コマンドを使用して、特権を削除できます。
- ObjectScript から $SYSTEM.SQL.Security.GrantPrivilege() メソッドを使用して、指定されたユーザ (またはユーザのリスト) に特定のオブジェクト特権を付与します。

20.7.2 SQL 特権のリスト

- 管理ポータルを使用します。[**システム管理**] から、[**セキュリティ**] を選択し、[**ユーザ**] または [**ロール**] を選択します。目的のユーザまたはロールを選択してから、該当するタブを選択します。管理特権の場合は [**SQL権限**]、オブジェクト特権の場合は [**SQLテーブル**]、[**SQLビュー**]、または [**SQLプロシージャ**] となります。

- SQL から、%CHECKPRIV コマンドを使用して、現在のユーザに特定の管理特権またはオブジェクト特権があるかどうかを特定できます。
- ObjectScript から、$SYSTEM.SQL.Security.CheckPrivilege() メソッドを使用して、指定されたユーザに特定のオブジェクト特権があるかどうかを特定できます。

20.7.3 特権エラーの監査

InterSystems IRIS プロセスが、ユーザが特権を持っていない SQL 文を呼び出すと、操作は失敗し、SQLCODE -99 エラーが生成されます。監査イベント %System/%SQL/PrivilegeFailure が有効になっている場合、発生した SQLCODE -99 エラーごとに、監査データベースにレコードが配置されます。監査データベース・オプションは、既定では無効になっています。

21

SQL コードのインポート

この章では、SQL コードをテキスト・ファイルから InterSystems SQL にインポートする方法を説明します。SQL コードをインポートすると、InterSystems IRIS® データ・プラットフォームは、ダイナミック SQL を使用して SQL の各行を作成して実行します。解析できないコード行が検出された場合は、SQL インポートではそのコード行がスキップされて、ファイルの末尾に達するまで後続行の作成と実行が続行されます。すべての SQL コード・インポート操作では、現在のネームスペースにインポートされます。

SQL インポートの主な用途は、CREATE TABLE などのデータ定義言語 (DDL) コマンドをインポートすること、および INSERT、UPDATE、および DELETE コマンドを使用してテーブルを操作することです。SQL インポートは SELECT クエリを作成して実行しますが、結果セットは作成しません。

SQL インポートを使用して InterSystems SQL コードをインポートできます。SQL インポートはコード移行のためにも使用できます。コード移行とは、他のベンダの SQL コードをインポートすることです (FDBMS、Informix、InterBase、MSSQLServer、MySQL、Oracle、Sybase)。他のベンダのコードは、InterSystems SQL コードに変換されて実行されます。SQL インポートは、すべての SQL コマンドを InterSystems SQL にインポートできるわけではありません。SQL インポートは、InterSystems IRIS で実装されている SQL 標準に対応しているコマンドと節をインポートします。対応していない機能は通常は解析されますが、無視されます。

SQL インポートで SQL クエリを正常に準備することはできます (必要に応じて、対応するクエリ・キャッシュを作成) が、クエリは実行しません。

SQL コードのインポートを実行するには、**%SYSTEM.SQL.Schema** クラスから適切なメソッドを呼び出します。SQL コードをインポートする際、これらのメソッドによって Errors.log と Unsupported.log という他の 2 つのファイルが作成される場合があります。Errors.log ファイルには SQL コマンドの解析に関するエラーが記録され、Unsupported.log ファイルにはそのメソッドで SQL コマンドとして認識されないリテラル・テキスト行が含まれます。

注釈　注目すべき点として、**%SYSTEM.SQL.Schema** クラスでは ExportDDL() メソッドを使用して DDL コマンドをエクスポートすることもできます。つまり、このメソッドを使用すると、このページで説明するメソッドを使用して後でインポートできる DDL スクリプト・ファイルをエクスポートできます。

この章では、各種 SQL コードのインポートについて説明します。

- InterSystems SQL のインポート
- 非 InterSystems SQL のインポート：FDBMS、Informix、InterBase、MSSQLServer、MySQL、Oracle、Sybase

21.1 InterSystems SQL のインポート

以下のいずれかの %SYSTEM.SQL.Schema メソッドを使用して、InterSystems SQL コードをテキスト・ファイルからインポートできます。

- ImportDDL() は汎用の SQL インポート・メソッドです。このメソッドは、バックグラウンド (非インタラクティブ) プロセスとして実行されます。InterSystems SQL をインポートするには、3 つ目のパラメータとして “IRIS” を指定します。
- Run() は InterSystems SQL インポート・メソッドです。このメソッドは、ターミナルからインタラクティブに実行されます。ターミナルでは、インポート・テキスト・ファイルの場所や、Errors.log ファイルと Unsupported.log ファイルを作成する場所などの情報を指定するように求められます。

注釈　この SQL DDL コードのインポートと実行を、管理ポータルの SQL インタフェースの [**文のインポート**] アクションと混同しないでください。この操作では、SQL 文を XML 形式でインポートします。

次の例では、InterSystems IRIS SQL コード・ファイル・パス名 mysqlcode.txt をインポートして、そのファイルにリストされている SQL コマンドを現在のネームスペース内で実行します。

ObjectScript

```
  DO $SYSTEM.SQL.Schema.ImportDDL("c:\InterSystems\mysqlcode.txt",,"IRIS")
```

既定では、ImportDDL() は 2 つ目のパラメータで指定されたとおりに、エラー・ログ・ファイルを作成します。この例では、2 つ目のパラメータを省略して、既定で `mysqlcode_Errors.log` という名前のファイルが SQL コード・ファイルと同じディレクトリ内に作成されます。このログ・ファイルは、ファイルに書き込む内容がない場合でも作成されます。

ターミナルから ImportDDL() を実行すると、次の例に示すように、まず入力ファイルがリストされ、次にエラー・ログ・ファイル、その次にインポートされた各 SQL コマンドがリストされます。

```
Importing SQL Statements from file: c:\InterSystems\mysqlcode.txt

Recording any errors to principal device and log file: c:\InterSystems\mysqlcode_Errors.log

 SQL statement to process (number 1):
     CREATE TABLE Sample.NewTab (Name VARCHAR(40))
      Preparing SQL statement...
      Executing SQL statement...
  DONE

  SQL statement to process (number 2):
     CREATE INDEX NameIDX ON Sample.NewTab (Name)
      Preparing SQL statement...
      Executing SQL statement...
  DONE

Elapsed time: 7.342532 seconds
```

SQL コマンドでエラーが発生した場合、ターミナルに以下の例のようなエラーが表示されます。

```
SQL statement to process (number 3):
       INSERT INTO Sample.MyStudents (StudentName,StudentDOB) SELECT Name,
       DOB FROM Sample.Person WHERE Age <= '21'
    Preparing SQL statement...
ERROR #5540: SQLCODE: -30 Message:  Table 'SAMPLE.PERSON' not found
       Pausing 5 seconds - read error message!  (Type Q to Quit)
```

5 秒以内に終了しない場合、ImportDDL() は次の SQL コマンドを実行します。エラー・ログ・ファイルにエラーがタイムスタンプ、ユーザ名、ネームスペース名と共に記録されます。

21.1.1 インポート・ファイル形式

SQL テキスト・ファイルは、.txt ファイルなどの書式なしファイルである必要があります。各 SQL コマンドはそれぞれ新しい行で始まる必要があります。1 つの SQL コマンドを複数の行に分けることができ、インデントが許可されています。各 SQL コマンドの後ろには、GO 文をそれぞれの行に記述する必要があります。

以下は、有効な InterSystems SQL インポート・ファイル・テキストの例です。

```
 CREATE TABLE Sample.MyStudents (StudentName VARCHAR(32),StudentDOB DATE)
GO
 CREATE INDEX NameIdx ON TABLE Sample.MyStudents (StudentName)
GO
 INSERT INTO Sample.MyStudents (StudentName,StudentDOB) SELECT Name,
 DOB FROM Sample.Person WHERE Age <= '21'
GO
 INSERT INTO Sample.MyStudents (StudentName,StudentDOB)
       VALUES ('Jones,Mary',60123)
GO
 UPDATE Sample.MyStudents SET StudentName='Smith-Jones,Mary' WHERE StudentName='Jones,Mary'
GO
 DELETE FROM Sample.MyStudents WHERE StudentName %STARTSWITH 'A'
GO
```

21.1.2 サポートされている SQL コマンド

すべての有効な InterSystems SQL コマンドをインポートできるわけではありません。以下は、サポートされている InterSystems SQL コマンドのリストです。

- CREATE TABLE、ALTER TABLE、DROP TABLE
- CREATE VIEW、ALTER VIEW、DROP VIEW
- CREATE INDEX (ビットスライスを除くすべてのインデックス・タイプ)
- CREATE USER、DROP USER
- CREATE ROLE
- GRANT、REVOKE
- INSERT、UPDATE、INSERT OR UPDATE、DELETE
- SET OPTION
- SELECT (オプティマイザ・プラン・モード専用)

21.2 コード移行 : 非 InterSystems SQL のインポート

他のベンダで使用されている SQL フォーマットに従った SQL コードをインポートできます。他のベンダのコードは、InterSystems SQL コードに変換されて実行されます。以下のメソッドが用意されています。

- ImportDDL() は汎用の SQL インポート・メソッドです。このメソッドは、バックグラウンド (非インタラクティブ) プロセスとして実行されます。このメソッドの使用法に関する一般的な情報は、"InterSystems SQL のインポート" を参照してください。

  特定フォーマットの SQL をインポートするには、そのフォーマットの名前を 1 つ目のパラメータとして指定します (FDBMS、Informix、InterBase、MSSQLServer (または MSSQL)、MySQL、Oracle、または Sybase)。

  次の例では、MSSQL コード・ファイル mssqlcode.txt をインポートして、そのファイルにリストされている SQL コマンドを現在のネームスペース内で実行します。

ObjectScript

```
DO $SYSTEM.SQL.Schema.ImportDDL($lb("C:\temp\somesql.sql","UTF8"),,"MSSQL")
```

3 つ目のパラメータが MSSQL、Sybase、Informix、または MySQL の場合、最初のパラメータは SQL コード・ファイル・パス名または 2 要素の %List (最初の要素が SQL コード・ファイル・パス名で 2 つ目の要素が使用する入出力変換テーブル) です。

- 次のタイプの SQL をインポートするための個別のインタラクティブ・メソッドが %SYSTEM.SQL.Schema で用意されています：LoadFDBMS()、LoadInformix()、LoadInterBase()、LoadMSSQLServer()、LoadOracle()、および LoadSybase()。これらのメソッドは、ターミナルからインタラクティブに実行されます。ターミナルでは、インポート・テキスト・ファイルの場所や、Errors.log ファイルと Unsupported.log ファイルを作成する場所などの情報を指定するように求められます。
- ImportDDLDir() を使用すると、同じディレクトリ内の複数のファイルから SQL コードをインポートできます。このメソッドは、バックグラウンド (非インタラクティブ) プロセスとして実行されます。このメソッドは、Informix、MSSQLServer、および Sybase をサポートしています。すべてのインポート対象ファイルには .sql という拡張子接尾語が付加されている必要があります。
- ImportDir() を使用すると、同じディレクトリ内の複数のファイルから SQL コードをインポートできます。このメソッドでは、ImportDDLDir() より多くのオプションが用意されています。このメソッドは、バックグラウンド (非インタラクティブ) プロセスとして実行されます。このメソッドは、MSSQLServer と Sybase をサポートしています。許可されるファイル拡張子接尾語のリストを指定できます。

A

SQL データのインポートとエクスポート

InterSystems IRIS® データ・プラットフォームの管理ポータルには、データをインポートおよびエクスポートするためのツールがあります。インポートまたはエクスポートできる行の最大サイズは、3,641,144 文字です。

%SQL.Import.Mgr クラスを使用してデータをインポートし、%SQL.Export.Mgr クラスを使用してデータをエクスポートすることもできます。

A.1 LOAD DATA を使用したデータのインポート

LOAD DATA SQL コマンドを使用して、InterSystems IRIS にデータをインポートできます。このユーティリティを使用して、ファイルのデータまたは JDBC でアクセスしたテーブルのデータをインポートできます。LOAD DATA を呼び出す前に CREATE TABLE を使用してテーブルとその列を定義する必要があります。

データをロードするときにテーブルが空の場合は、LOAD DATA によってデータ・ソースの行がテーブルに取り込まれます。テーブルに既にデータが格納されている場合は、LOAD DATA によってテーブルに行が挿入され、テーブルにあるデータは上書きされません。

以下の例では、テーブルを作成し、ローカル システムに格納されているファイル people.csv からそのテーブルにデータをロードします。

```
>>> CREATE TABLE Sample.Person (
        Name VARCHAR(25),
        Age INT,
        DOB DATE)

>>> LOAD DATA FROM FILE 'C://sampledata/people.csv' INTO Sample.Person
```

LOAD DATA には、BULK キーワードを使用することでロード処理を高速にするオプションも用意されています。この動作の詳細な説明は、LOAD DATA のリファレンスを参照してください。

A.2 テキスト・ファイルへのデータのエクスポート

所定のクラスのデータをテキスト・ファイルにエクスポートできます。そのためには、以下の操作を実行します。

1. 管理ポータルで、[**システム・エクスプローラ**]、[SQL] の順に選択します。ページ上部の [**切り替え**] オプションを使ってネームスペースを選択します。利用可能なネームスペースのリストが表示されます。
2. ページ上部の [ ] ドロップダウン・リストをクリックし、[**データ・インポート**] を選択します。

3. ウィザードの最初のページで、以下の操作を行います。
   - エクスポートしたデータを保持するために作成するファイルの完全なパスとファイル名を入力します。
   - ドロップダウン・リストから、ネームスペース、スキーマ名、およびデータのエクスポート元のテーブル名を選択します。
   - オプションで、**[文字セット]**ドロップダウン・リストから文字セットを選択します。既定値は［デバイスデフォルト］です。

   次に **[次へ]** をクリックします。
4. ウィザードの 2 ページ目で、エクスポートする列を選択します。次に **[次へ]** をクリックします。
5. ウィザードの 3 ページ目では、外部ファイルの形式を指定します。
   - **[列を区切る区切り文字]** で、このファイルの区切り文字に対応するオプションをクリックします。
   - 列ヘッダをファイルの最初の行としてエクスポートする場合は、**[列ヘッダのエクスポート]** チェック・ボックスにチェックを付けます。
   - **[文字列の引用]** で、このファイルで文字列データを開始および終了する方法を示すオプションをクリックします。
   - **[日付形式]** で、このファイルで使用する日付形式を示すオプションをクリックします。
   - **[時刻形式]** で、このファイルで使用する時刻形式を示すオプションをクリックします。
   - オプションで、**[データのプレビュー]** をクリックすると、結果がどのように表示されるか確認できます。

   次に **[次へ]** をクリックします。
6. 入力内容を確認して、**[完了]** をクリックします。ウィザードに、**[データのエクスポート結果]** ダイアログ・ボックスが表示されます。
7. **[閉じる]** をクリックします。または、バックグラウンド・タスクのページを表示する特定のリンクをクリックします。

   いずれの場合も、ウィザードによって、処理を実行するバックグラウンド・タスクが開始されます。