
# Transact-SQL (TSQL) 移行ガイド

Version 2023.1
2024-01-02

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

Transact-SQL (TSQL) 移行ガイド
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

# 1

# TSQL の移行の計画と実行

InterSystems TSQL では、Sybase と Microsoft の両方で実装されている多彩な機能をサポートするプロシージャ型言語 Transact-SQL を実装しています。Transact-SQL は Sybase Adaptive Server、Microsoft SQL Server (MSSQL)、および他のプラットフォームと共に使用されます。

注釈　この章では主に Sybase Adaptive Server (ASE) の実装について説明しますが、この情報のほとんどは Transact-SQL のすべての実装に関係があります。

また、InterSystems TSQL には、Microsoft SQL Server や Sybase Adaptive Server には含まれない専用の拡張機能がいくつかあります。これらの拡張機能については、"TSQL コマンド" および "TSQL システム・ストアド・プロシージャ" の章で説明します。

このドキュメントでは、Transact-SQL データベース・アプリケーションからスキーマ、ストアド・プロシージャ、およびデータを効果的に移行する方法、および InterSystems IRIS® データ・プラットフォームに TSQL (Transact-SQL) を実装し、最初の移行後に TSQL を管理および強化する方法を説明します。

## 1.1 InterSystems IRIS に移行すべき理由

InterSystems IRIS は効率的な最新の SQL 実装を備えており、さまざまな業界でミッション・クリティカルなアプリケーションにスピード、拡張性、およびセキュリティを提供します。こうしたパフォーマンス上のメリット、垂直方向と水平方向の拡張性、およびエンタープライズ・グレードのセキュリティは、InterSystems IRIS に移行した TSQL アプリケーションにも適用されます。

InterSystems IRIS データベースのすべてのデータは、グローバルと呼ばれる、ツリー・ベースの効率的な多次元スパース配列に保存されます。グローバルは直接アクセスされ、ファイル・システム層が必要ないため、InterSystems IRIS グローバルは非常に高速で柔軟な格納と取得を実現します。グローバルは InterSystems IRIS オブジェクトと SQL インタフェースの基盤になっており、キー/値と NoSQL のパラダイムをサポートしています。また、グローバルによって、InterSystems IRIS は XML や JSON などのダイナミック・データ型を容易に扱うことができます。

InterSystems SQL は、TSQL 構文を完全にサポートするように強化された、高度に最適化された ODBC および JDBC ドライバによって高性能なアクセスを提供します。また、SQL ゲートウェイ機能も備えているため、他のリレーショナル・データベースのデータに簡単にアクセスしたり、データをインポートしたりできます。

### 1.1.1 InterSystems IRIS での TSQL アプリケーションの実行

InterSystems IRIS プラットフォームで TSQL コードを実行する際に、コードは対応する InterSystems SQL と ObjectScript コードにコンパイルされます。ObjectScript はインターシステムズのオブジェクト指向プログラミング言語です。コンパイルされたコードは InterSystems IRIS 上で実行され、必要に応じてデバッグのために利用できます。このコンパイル手順は

完全に自動化されており、1 回限りの移行タスクではありません。移行作業は、旧環境から InterSystems IRIS にスキーマ、データ、および TSQL コードをロードすることで構成されます。その後、TSQL アプリケーション・コードの使用や変更に進むことできます。コードを変更した場合は、変更後にコードをリコンパイルするだけです。

InterSystems IRIS では、TSQL コードを実行するためのインタフェースが多数提供されています。これらのインタフェースには、Sybase または MSSQL を指定する言語オプションが用意されています。

どのインタフェースや言語を使用するかにかかわらず、InterSystems IRIS では、対応する InterSystems SQL と ObjectScript コードのみが実行されます。TSQL がネイティブに実行されることはありません。

### 1.1.2 Sybase 製品からの移行

InterSystems TSQL の初期の実装は、Sybase ASE のコードの移行をサポートすることを目的に設計されました。この実装は、ネイティブの ASE TSQL コマンドとシステム・ストアド・プロシージャの大半をサポートしています。

また、日付/時刻形式を含むほとんどの ASE スキーマ・オプションを完全にサポートするか受け入れます。さらに、ASE TSQL ストアド・プロシージャ・コードから InterSystems SQL および ObjectScript へのコンパイルもサポートしています。

同じハードウェア上で実行した場合、InterSystems IRIS は Sybase ASE よりも大幅に高いパフォーマンスを発揮します。インターシステムズのミラーリング機能は、Sybase ASE のデータベース・レプリケーションよりも堅牢な復元可能性を実現し、インターシステムズのエンタープライズ・キャッシュ・プロトコル (ECP) とシャーディングは、より柔軟なスケール・アウト・オプションを提供します。

他の Sybase 製品 (Sybase ASA および Sybase IQ) では、TSQL のサポートと言語がわずかに異なります。Sybase ASA から正常に移行できたお客様もいますが、このような移行を計画する際には InterSystems サポートと協力することをお勧めします。

Sybase IQ はその高いパフォーマンスで有名ですが、Sybase IQ から InterSystems IRIS にワークロードを移行したお客様は、以前のプラットフォームよりもさらに高いパフォーマンス・レベルを実感しています。

### 1.1.3 Microsoft 製品からの移行

Microsoft SQL Server (MSSQL) と Azure データベースでは、Sybase TSQL から分離した後に大きく進化した別の TSQL 言語が使用されています。InterSystems IRIS は、MSSQL 言語のサブセットをサポートしています。ただし、InterSystems IRIS TSQL は主に Sybase 言語のコードの移行をサポートすることを目的に設計されているため、Microsoft TSQL との互換性および移行のサポートは限定的です。

## 1.2 移行の計画

TSQL アプリケーションを移行するということは、結果的に同じスキーマ、データ、およびアプリケーション・コードを (ただし異なるプラットフォームで) 使用することを意味しますが、それでもこのプロジェクトは慎重に計画する価値があります。

### 1.2.1 インフラストラクチャの計画

InterSystems IRIS では一般的に、同じ TSQL ワークロードを実行するために必要なハードウェア・リソースは他のデータベース・プラットフォームよりも少なくなります。つまり、InterSystems IRIS への移行は、TSQL アプリケーションを実行するインフラストラクチャを検討する格好のタイミングです。

- InterSystems IRIS の効率性と垂直方向の拡張性を利用して、TSQL アプリケーションの最小限のハードウェアを判断します。
- InterSystems IRIS はクラウドでも動作します。クラウドでは、さまざまなクラウド・サービス・プロバイダから環境のサイズを適正化するための幅広いオプションが提供されています。

- ミラーリングは、高可用性と災害復旧の要件を実装するための堅牢で実証済みのメカニズムを提供します。
- InterSystems IRIS では、非同期ミラー・レポート・サーバを使用してクエリをデータベース・サーバからオフロードし、クエリの使用量が多いシステムのパフォーマンスを大幅に向上させることができます。必要に応じて、データベース・サーバのデータのサブセットをレポート・サーバで提供できます。
- 最も負荷の高いワークロードの場合、InterSystems IRIS では、クラスタに複数のマシンを導入してユーザとデータ・ボリュームを別々にスケール・アウトすることもできます。

このプロセスの支援については、インターシステムズのテクノロジ・アーキテクトにご相談ください。

## 1.2.2 アプリケーションのスキーマとコードの確認

InterSystems IRIS TSQL は、TSQL の大部分の概念と言語要素をサポートしており、ほとんどのスキーマ・オプションを使用できます。それでも、TSQL アプリケーションを確認して、現在はサポートされていないいくつかの構成要素に依存していないかどうかを判断することをお勧めします。最も簡単な検証方法は、単純に TSQL スキーマとコードを IRIS にインポートしてコンパイルすることです。InterSystems IRIS TSQL コンパイラは、インポートしたコードで見つかった問題にフラグを設定します。これにより、元の TSQL コードをサポートされる別のコードに変更するか、特定のプロシージャを ObjectScript で再実装することによって問題を解決できます。

### 1.2.2.1 InterSystems IRIS でサポートされない TSQL の機能

InterSystems IRIS が対応する TSQL の機能は、コードが元々どの言語を対象に開発されたかによって異なります。スキーマとエイリアスの処理方法に違いがあるため、SQL エイリアスの特定の処理とスキーマの名前付けに制限がいくつかあります。これらの制限を回避するには、DDL でスキーマ名を適切に一致させます。また、SQL クエリでテーブル名とエイリアスを混在させる場合は注意して使用します。

多くのテーブル・プロパティと他のスキーマ・オプションはプラットフォーム固有です。これらが使われていても、ユーザやアプリケーションの観点からはテーブルの動作に違いがない場合があります。ストレージ関連のさまざまなプロパティ（分割や圧縮など）だけでなく、直交機能（暗号化など）もこれに該当します。InterSystems IRIS TSQL は、動作に影響のないプロパティの多くを通知なしで受け入れて無視しますが、動作に影響がある場合はコンパイル・エラーを報告します。

### 1.2.2.2 InterSystems IRIS での TSQL コードへのクライアント・アクセス

InterSystems IRIS TSQL はコマンド行 SQL シェルをサポートしており、ユーザはこれを使用して TSQL コマンドとクエリを直接発行できます。最初に現在の SQL シェル・セッションの言語を好みの値に設定する必要があります。例えば、`SET DIALECT = Sybase` のように設定します。

言語を適切に設定することにより、Java ベースのアプリケーションから JDBC または ODBC ドライバを通して TSQL コマンドを発行することもできます。

InterSystems SQL と ObjectScript を使用すると、InterSystems SQL で提供される一連のアクセス性オプションがさらに広がります。これには、基本となるストレージ・パラダイムへの超高速なネイティブ・アクセスなどがあります。詳細は、InterSystems IRIS のドキュメントを参照してください。

## 1.2.3 データの確認

InterSystems IRIS は完全なマルチモデル・データベースで、多種多様なデータ型をサポートします。TSQL で使用可能なほとんどのデータ型は InterSystems IRIS でも使用可能です。これらの型は、テーブルを作成するためのデータ定義言語（DDL）文によって自動的に適切にマップされます。

Sybase ASE、Sybase ASA、および Microsoft SQL Server は、InterSystems IRIS SQL と同様に、リレーショナル・データについては主に行ベースのストレージ・モデルに従います。Sybase IQ は、InterSystems IRIS では現在サポートされていない列指向のストレージ・パラダイムを使用します。これは内部的なストレージ・モデルであって、SQL 経由でテーブ

ルにアクセスする場合はテーブルそのものの動作には影響しませんが、同じデータを別のモデルで保存した場合、ストレージの使用量に影響する可能性があります。

### 1.2.4 プロジェクトの計画

以下に、InterSystems IRIS への移行範囲を判断する際に重要な考慮事項を示します。

- TSQL の専門知識があり、コードを全般的に理解できること。
- TSQL アプリケーションの元の開発者に機能について質問できること。
- 移行を検証するためのユニット・テストまたは他のテスト・スクリプトがあること。
- スキーマとコードの初期評価。複雑さ、およびサポートされない機能の有無の判断。
- データ・セットのサイズの評価。
- 切り替え (一括変換または段階的変換) の要件に関する合意。

移行プロセスでは、以下のアプローチを組み合わせて使用することができます。

- すべての TSQL コードを移行してから、互換性のないコードを修正する。
- 互換性のある TSQL コードを移行してから、InterSystems SQL と ObjectScript で新規に開発して拡張する。
- TSQL システムと IRIS SQL システムを並行して実行できるようにし、それぞれの利点を活かしてデータの同期を保つ。インターシステムズは、合意したパフォーマンスおよびサービス・レベルに基づいて何を移行すべきかを判断する際にお客様を支援できます。

### 1.2.5 移行のテスト

ユニット・テストはあらゆるアプリケーション開発プロジェクトにおいて重要な要素であり、アプリケーション・コードの変更や、基盤となるインフラストラクチャのアップグレードをテストする場合には非常に重要です。したがって、アプリケーションを InterSystems IRIS などの新しいプラットフォームに移行する場合にも大いに役立ちます。TSQL アプリケーションの場合は、TSQL で記述されたユニット・テストを TSQL アプリケーション・コードと共に移行できます。Jenkins のような Java ベースのアプリケーション・コードやフレームワークなど、外部の機能から実行するユニット・テストも、そのまま再利用できます。

TSQL コードとデータ・セットの移行をテストする場合、以下も重要な考慮事項です。

- パフォーマンス・テスト。アプリケーションの応答時間は前のプラットフォームと同等以上か。
- スケーラビリティ・テスト。オプションでハードウェア・リソースを追加して、前のプラットフォームと比較してより多くのユーザやデータ・ボリュームをサポートできるか。
- 正確性テスト。InterSystems TSQL と前のプラットフォームとの間でセマンティクスが一部異なる場合があります (TSQL の異なる実装で異なるのと同様)。例えば、Sybase ASE で提供されている SQL クエリの外部結合の解決などです。正確性テストは特に、移行後にユーザが同じ動作と結果を利用できることを保障するために重要です。

## 1.3 計画の実行

既存の TSQL アプリケーションを InterSystems TSQL に移行するには、4 つの操作 (TSQL 用の InterSystems IRIS の構成、TSQL ソース・コードの移行、メタデータ (DDL) の移行、およびデータの移行) を実行する必要があります。インターシステムズとその実装パートナーがこれらの各タスクを支援します。

## 1.3.1 システムの設定

InterSystems IRIS の既定のシステム設定の中には、Sybase ASE や他の TSQL プラットフォームの既定の構成と異なるものがあります。インターシステムズおよび TSQL の移行を手がけるプロフェッショナル・サービス・パートナーは、新規インストールした InterSystems IRIS インスタンスを TSQL アプリケーションで使用できるよう準備するためのスクリプトを提供できます。

InterSystems IRIS 構成ユーティリティを使用してシステムを手動で TSQL 用に構成するには、以下の手順を実行します。

- InterSystems IRIS 管理ポータルに移動します。**[システム管理]**、**[構成]**、**[SQL およびオブジェクトの設定]**、**[TSQL 互換性]** の順に選択します。ここでは、言語 (Sybase または MSSQL) を指定したり、ANSI_NULLS、CASEINSCOMPARE、および QUOTED_IDENTIFIER をオンまたはオフに設定したりできます。この 3 つの既定値はすべて "オフ" で、これは Sybase ASE に適した設定です。
- 管理ポータルで、**[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。ここから、**[既定のスキーマ]** を設定できます。これは、未修飾の DDL エンティティ (テーブル名やプロシージャ名など) すべてに使用される既定のスキーマ名です (パッケージにマッピングされます)。
- 等値リテラル置換をオフに設定します。これは Sybase ASE に適した設定です。
- 既定の照合順を %SQLSTRING に設定します。照合オプションは、インデックス照合の目的でのみ文字列の変換を実行します。保存済みのデータは変更されません。%SQLSTRING オプションは、Sybase ASE の既定のバイナリ照合に対応します。この設定が Sybase のソート順に一致していることを確認することが重要です。このステップはデータをロードする前に実行する必要があります。
- 切り捨て設定は、データベース内で文字列の最後に後続スペースが存在する Sybase 実装で必要になる場合があります。これらの文字列を切り捨てることで、文字列を適切に照合できるようになります。"データの照合と文字列の切り捨て" を参照してください。
- 既定のタイムスタンプの代わりに Posix 時間を使用してパフォーマンスを向上させることを検討します。"タイムスタンプと時刻精度" を参照してください。
- 一時テーブルは完全にサポートされます。一時テーブルを頻繁に使用する場合は、その設定を最適化して速度を向上させる必要があります。"一時データベースの設定" を参照してください。
- 管理ポータルで、**[システム管理]**、**[構成]**、**[SQL およびオブジェクトの設定]**、**[ユーザ DDL マッピング]** の順に選択します。このオプションを使用して、必要なユーザ定義のデータ型をマッピングできます。

## 1.3.2 コードの移行

最初のアプリケーション移行は簡単です。

1. DDL のインポート : %SYSTEM.SQL.Schema.ImportDDL() メソッド (単一ファイルの場合) または $SYSTEM.SQL.Schema.ImportDDLDir() メソッド (1 つのディレクトリに複数のファイルがある場合) のいずれかを使用して、テーブルとビューの定義をインポートします。これらのメソッド内で、DDLMode パラメータを `"MSSQLServer"` または `"Sybase"` のいずれかに設定します。これらのメソッドは、DDL 文に加えて、INSERT などの DML 文をインポートし、それらを対応する InterSystems IRIS SQL に変換して実行します。詳細は、"SQL コードのインポート" を参照してください。

   また、$SYSTEM.SQL.Schema.LoadSybase() メソッドまたは $SYSTEM.SQL.Schema.LoadMSSQLServer() メソッドを呼び出してスキーマをインポートすることもできます。詳細は、"SQL コードのインポート" を参照してください。

   TSQL ソースに CREATE PROC 文が含まれる場合は、CREATE PROC ソースを含むクラス・メソッドが作成されます。InterSystems IRIS では、このクラス・メソッドが既存のクラスに配置されるか、スキーマおよびプロシージャ名に基づく新しいクラスに配置されます。プロシージャが既に存在する場合は、既存のバージョンが新しいバージョンに置き換えられます。スキーマおよびプロシージャから生成されたクラス名に一致するクラスが既に存在する場合は、既

存のクラス名が使用されます (以前に TSQL ユーティリティによって生成されている場合)。一致するクラスが存在しない場合は、スキーマおよびプロシージャ名に基づく一意のクラス名が生成されます。プロシージャが正常に作成されると、結果のクラスがコンパイルされます。ロギングが要求される場合は、ソース文が、それを含むクラス名、クラス・メソッド、および生成された仮引数と共に記録されます。プロセスで発生したエラーもログに記録されます。CREATE PROC の処理中にエラーが検出された場合、InterSystems IRIS では、そのプロシージャに対して生成された新しいクラスが削除されます。

2. エラーのログ・ファイルの検査：エラー番号で検索します。エラーと正しく実行されたインポートの合計数がログの最後に表示されます。ほとんどの場合、エラーは、このドキュメントの情報を使用して回避または対処できます。
3. コンパイル：DDL をインポートすると、テーブルおよびビュー定義のコンパイルが自動的に実行されます。その他の TSQL ソース・コードをコンパイルするには、以下のようにコマンドを使用します。

   ObjectScript

   ```
   DO $SYSTEM.OBJ.CompileAll("-l")
   ```

   小文字の “L” 修飾子フラグによって、コンパイルの際にロックを適用しないように設定します。フラグ修飾子の完全なリストを確認するには、`DO $SYSTEM.OBJ.ShowFlags()` を呼び出します。

## 1.3.3 データの移行

データを移行するためのオプションは以下のとおりです。

- JDBC または ODBC を使用して SQL ゲートウェイ接続を設定し、データをエクスポート/インポートできるようにします。管理ポータルで **[システム管理]**、**[構成]**、**[接続性]**、**[SQLゲートウェイ接続]** の順に選択し、**[新規接続作成]** ボタンを選択して **SQL ゲートウェイ接続**を定義します。
- データ移行ウィザードの使用管理ポータルで **[システムエクスプローラ]**、[SQL] の順に選択し、**[ウィザード]** ドロップダウン・リストから **[データ移行]** を選択して、データ移行ウィザードを構成します。ドロップダウン・リストから既存の SQL ゲートウェイ接続を選択します。これにより、外部ソースからデータを移行するウィザードが実行され、そのデータを格納する InterSystems IRIS クラス定義が作成されます。
- バルク・ローダを使用します。インターシステムズは、Sybase のさまざまな日付/時刻フィールド保存形式を認識して InterSystems IRIS の標準の内部形式に正規化できるユーティリティを提供しています。この変換を行わないと、Sybase での日付/時刻フィールドの保存方法によっては、データ・ロード時に日付/時刻フィールドで問題が発生する可能性があります。

  データが大量にある場合、一括コピー・プログラム (BCP) ファイルの読み取りをサポートするバルク・ローダを使用するか、好みの抽出、変換、およびロード (ETL) ユーティリティを使用します。InterSystems サポートでデータの一括取り込み用のツールを提供できます。詳細は InterSystems サポートにお問い合わせください。

## 1.3.4 トラブルシューティング

TSQL Trace 機能をオンにして、コンパイル・ログを検査します。"TRACE" を参照してください。これにより、各操作のタイムスタンプ、各操作の経過時間、グローバル参照の数、および %ROWCOUNT を記録するログが生成されます。このログは、処理されたストアド・プロシージャの詳細情報を TSQL 文ごとに提供します。

さらに、クエリ・キャッシュのソース・コードと生成されたクエリ・キャッシュを保持して、コンパイルされたコードの詳細な情報を提供することも検討してください。"クエリキャッシュのソース" を参照してください。

# 1.4 InterSystems IRIS での TSQL の記述と実行

InterSystems スタジオ統合開発環境または InterSystems SQL を使用して、TSQL を作成および実行できます。

## 1.4.1 IDE を使用した TSQL の操作

InterSystems IRIS には、サーバ側アプリケーション・コードを構築および管理するための統合開発環境 (IDE)、InterSystems スタジオが付属します。スタジオは、TSQL メソッド本体を ObjectScript クラスに組み込んで SQL 経由でアクセス可能なストアド・プロシージャとして投影する機能により、TSQL の開発を完全にサポートします。

スタジオで TSQL ストアド・プロシージャ (SP) を記述および管理することができます。TSQL SP はクラス・メソッドまたはクエリです。クラス・メソッドは、パラメータを受け取って単一のスカラ結果を返します。クエリはパラメータを受け取って行を返します。単純な SELECT 文をクラス・メソッドに埋め込むと、それらの文は実行されますが、行は返されません。

InterSystems スタジオで TSQL ストアド・プロシージャを作成するには、SqlProc キーワードを使用して、ストアド・プロシージャとしてマークされたクラス・メソッドを作成し、言語を tsql として入力します。以下のテンプレートをベースとして使用できます。

```
ClassMethod MyTestMethod() As %Integer
   [ Language = tsql, ReturnResultSets, SqlName=name, SqlProc ]
{
}
```

"クラス定義リファレンス" のメソッド定義の "Language"、"SqlProc"、および "SqlName" の各キーワードを参照してください。

TSQL でトリガを作成および管理できます。トリガは、TSQL コード内の命令のセットであり、指定した SQL イベントに応答して実行されます。Language=tsql クラス定義キーワードを使用して、トリガを TSQL で記述することを指定できます。UpdateColumnList クラス定義キーワードは、TSQL でのみサポートされています。TSQL では、行レベル・トリガはサポートされていません。"InterSystems SQL の使用法" のドキュメントの "トリガの使用法" を参照してください。

## 1.4.2 SQL を使用した TSQL の操作

スタジオを使用して TSQL コードを開発する代わりに、SQL 文を発行できるインタフェースを使用して TSQL コードを呼び出すことができます。同じインタフェースのセットを使用し、データ定義言語 (DDL) を介して TSQL ストアド・プロシージャを作成、置換、および削除することもできます。

- TSQL シェルの使用法

  InterSystems TSQL シェルを使用すると、Transact-SQL コードを InterSystems IRIS 上で直接実行できます。TSQL シェルを使用するには、`DO $SYSTEM.SQL.TSQLShell()` のように、ターミナルから TSQLShell() (または $SYSTEM.SQL.Schema.LoadTSQL()) メソッドを呼び出します。これにより、InterSystems SQL シェルが呼び出され、その DIALECT 構成パラメータが現在構成されている TSQL 言語 (MSSQL または Sybase) に設定されます。初期構成の既定値は MSSQL です。

  SQL コードをインタラクティブに入力する際、TSQL シェルでは、各 SQL 文の末尾に文の区切り文字としてセミコロン (;) を配置することがサポートされますが、必須ではありません。

  シェルの RUN コマンドを使用して、TSQL スクリプト・ファイルを実行できます。RUN コマンドでは、`"Please enter the end-of-statement delimiter (Default is 'GO'): GO=>"` など、一連のプロンプトが表示されます。これにより、InterSystems IRIS の既定の GO 文の代わりに、TSQL のセミコロン (;) をスクリプト・ファイルの文の区切り文字として指定できます。"InterSystems SQL の使用法" のドキュメントの "SQL シェル・インタフェースの使用法" を参照してください。

- InterSystems SQL シェルの使用法

InterSystems SQL シェルを使用すると、SET DIALECT コマンドを使用してシェルの言語を Sybase または MSSQL に設定することによって、TSQL コードの行を実行できます。

シェルの言語を Sybase または MSSQL に設定した場合、SQL シェルでは、各 SQL 文の末尾に文の区切り文字としてセミコロン (;) を配置することがサポートされますが、必須ではありません。シェルの言語を IRIS に設定した場合、文の区切り文字としてセミコロン (;) を使用すると SQLCODE -25 エラーが発生します。

シェルの RUN コマンドを使用して、TSQL スクリプト・ファイルを実行できます。RUN コマンドでは、"`Please enter the end-of-statement delimiter (Default is 'GO'): GO=>`" など、一連のプロンプトが表示されます。これにより、InterSystems IRIS の既定の GO 文の代わりに、TSQL のセミコロン (;) をスクリプト・ファイルの文の区切り文字として指定できます。"InterSystems SQL の使用法" のドキュメントの "SQL シェル・インタフェースの使用法" を参照してください。

- 管理ポータルの SQL インタフェースの使用法

  管理ポータルの SQL インタフェースの [Dialect] オプションでは、SQL 言語を IRIS、Sybase、または MSSQL に設定できます。既定値は IRIS です。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。"InterSystems SQL の使用法" のドキュメントの "管理ポータルの SQL インタフェースの使用法" を参照してください。

- ダイナミック SQL の使用法

  ObjectScript の機能である InterSystems IRIS ダイナミック SQL を使用すると、TSQL コード・クエリ、および他の DML 文と DDL 文の限定されたサブセットを ObjectScript コードから実行できます。

  - ダイナミック SQL 文クラスのインスタンスを作成し、%Dialect プロパティを Sybase または MSSQL に設定することができます。その後、そのオブジェクト・インスタンス内で TSQL コマンドを準備して実行します。
  - SQL コマンドを準備する %SYSTEM.SQL.Prepare() メソッド、または SQL コマンドの準備と実行の両方を行う %SYSTEM.SQL.Execute() メソッドを呼び出すと、文クラスのインスタンスを作成せずにダイナミック SQL を実行することができます。これらのどちらのメソッドでも Dialect パラメータを指定します。

  "InterSystems SQL の使用法" のドキュメントの "ダイナミック SQL の使用法" を参照してください。

# 2

# InterSystems TSQL 構文

## 2.1 テーブル参照

InterSystems TSQL では、InterSystems IRIS® データ・プラットフォームの SQL 形式を使用したテーブル参照がサポートされています。

```
schema.table
```

必須のテーブル参照コンポーネントは `table` のみです。スキーマを省略すると、TSQL では既定のスキーマ名が使用されます。

その他の形式の Transact-SQL テーブル参照では、ドットで区切られた最大 4 つのコンポーネント (`server.database.owner.table`) を使用できます。Transact-SQL でのテーブル参照の処理方法を以下に示します。

- `server.` コンポーネントは無視されます (存在する場合)。
- `database.` コンポーネントが存在し、`owner.` コンポーネントが省略されている場合、`database` は schema 名にマップされます。したがって、`database..table` は `schema.table` にマップされます。データベース名が 'master' である場合、この変換は実行されません。
- `owner.` コンポーネントが存在する場合、schema 名にマップされます。

名前変換のため、トランザクションの実行中はフィールド名からフィールド接尾語がいったん削除され、後で元に戻されます。

## 2.2 一時テーブル

InterSystems TSQL は `#tablename` 一時テーブルをサポートしています。`#tablename` 一時テーブルは現在のプロセスの現在のプロシージャから参照できます。また、現在のプロシージャから呼び出された任意のプロシージャからも参照できます。`#tablename` 構文は、TSQL プロシージャ (TSQL 言語でプロシージャとして投影されるクラス・メソッド) でのみサポートされます。

一時テーブルは、CREATE TABLE で、"#" で始まるテーブル名を使用して定義します。一時テーブルは実行時に作成されます。プロシージャを終了すると、`#tablename` テーブル定義は範囲外になります。接続が切断されると、すべての一時テーブルの定義は範囲外になります。DROP TABLE を使用して一時テーブルを明示的に削除することもできます。

ただし、一時テーブルがアクティブな結果セットで参照される場合、その一時テーブルはプロセスから参照できなくなる可能性がありますが、データおよび定義は結果セットが範囲外になるまで保持されます。

#tablename 一時テーブルは、作成プロシージャと、そのプロシージャが呼び出す任意のプロシージャの両方から参照できます。一時テーブルは、入れ子にされたプロシージャの呼び出しで参照できます。呼び出し先のプロシージャで一時テーブルを宣言する必要はありません。呼び出し先のプロシージャでも同じ名前の一時テーブルを作成する場合は、最後に作成されたテーブル定義が使用されます。一時テーブルは ObjectScript ローカル変数を使用して定義されるので、そのテーブルの作成、変更、および削除はジャーナルされるトランザクション・イベントではありません。トランザクションをロールバックしても、これらの操作に影響しません。

# 2.3 システム・テーブル

システム・テーブルは、InterSystems IRIS ネームスペース単位で存在します。

### Systypes

部分的にサポートしています。

# 2.4 トランザクション

BEGIN TRAN、COMMIT、および ROLLBACK のコマンドに対して生成されたコードは明示的なトランザクション・モードを使用しますが、次のトランザクション TSQL では通常、BEGIN TRAN 文の前にアクティブであったモードをリストアします。TSQL では、プロシージャが終了したとき、または COMMIT コマンドまたは ROLLBACK コマンドが発行されたときのモードをリストアします。

# 2.5 カーソル名の管理

実行時に 1 つのバージョンのカーソルのみが開いていれば、同じカーソルを複数回宣言することができます。あるストアド・プロシージャで同じカーソルが複数宣言された場合、最初の宣言以外は、名前変更されたカーソルと関連付けられます。OPEN、FETCH、CLOSE、および DEALLOCATE の文は、該当するカーソルの最後の DECLARE を参照することを前提としています。ストアド・プロシージャに含まれる文の字句の位置は、カーソル名と DECLARE とを対応付けるために使用されます。つまり、そのコードのランタイム・パスは考慮されないので注意してください。

クエリ内のカーソルは、InterSystems SQL クエリで使用するスキームの拡張子を使用して名前が付けられます。例えば、以下のようになります。

TSQL

```
DECLARE C CURSOR FOR SELECT A FROM B
--
OPEN C
FETCH C
CLOSE C
DEALLOCATE C
--
DECLARE C CURSOR FOR SELECT D FROM E
--
OPEN C
FETCH C
CLOSE C
DEALLOCATE C
```

これは次のように変換されます。

TSQL

```
DECLARE C CURSOR FOR SELECT A FROM B
--
OPEN C
FETCH C
CLOSE C
DEALLOCATE C
--
DECLARE Cv2 CURSOR FOR SELECT D FROM E
--
OPEN Cv2
FETCH Cv2
CLOSE Cv2
DEALLOCATE Cv2
```

# 2.6 SYSOBJECTS 参照

アプリケーションには一般的に、テーブル、ビュー、そのアプリケーション環境で使用するメタデータを作成する設定プロシージャがあります。これらのプロシージャには次のような式があります。

TSQL

```
IF EXISTS (SELECT * FROM SYSOBJECTS
WHERE ID = OBJECT_ID('People'))
```

この例では、これによってテーブルが存在するかどうかが判断されます。通常は DROP および CREATE 文が続き、テーブルのメタデータが再構築されます。

TSQL プロシージャとトリガは、SYSOBJECTS システム・テーブルを参照できます。InterSystems TSQL は、SYSOBJECTS テーブルの以下の列 (%TSQL.sys.objects クラス・プロパティ) をサポートします。

| 列 | 説明 |
|---|---|
| name | オブジェクト名。 |
| id | オブジェクト ID。 |
| type | オブジェクト型。次の値のいずれかです。K= PRIMARY KEY または UNIQUE 制約、P= ストアド・プロシージャ、RI=FOREIGN KEY 制約、S= システム・テーブル、TR= トリガ、U= ユーザ・テーブル、V= ビュー。 |
| deltrig | エントリがテーブルの場合は、削除トリガのオブジェクト ID。エントリがトリガの場合は、テーブルのテーブル ID。 |
| instrig | エントリがテーブルの場合は、挿入トリガのオブジェクト ID。 |
| updtrig | エントリがテーブルの場合は、更新トリガのオブジェクト ID。 |
| parent_obj | 親オブジェクトのオブジェクト ID 番号。例えば、トリガまたは制約の場合はテーブル ID。 |
| schema | オブジェクトが存在するスキーマの名前。 |
| parent_obj_name | parent_obj のオブジェクト名。parent_obj=0 の場合は、parent_obj_name は name と同じです。 |

SYSOBJECTS テーブルは読み取り専用です。SYSOBJECTS テーブルは、TSQL プロシージャまたはトリガ外部から名前 %TSQL_sys.objects で参照できます。SYSOBJECTS は、複数のネームスペースにマップされるテーブルではサポートされていません。

注釈　InterSystems IRIS には、SYSOBJECTS 参照と同じ操作を実行できる、クラス・オブジェクトの %Dictionary パッケージが用意されています。詳細は、"インターシステムズ・クラス・リファレンス" の %Dictionary パッケージを参照してください。

# 3

# InterSystems TSQL 言語要素

この章では、InterSystems IRIS® データ・プラットフォームの以下の TSQL 言語要素について説明します。

- リテラル、予約語、コメント
- 変数
- 識別子
- データ型
- 算術演算子、比較演算子、文字列演算子、論理演算子、ビット演算子

## 3.1 リテラル

### 3.1.1 文字列リテラル

文字列リテラルは引用符で区切る必要があります。優先的に使用される区切り文字は一重引用符です。SET DELIMITED_IDENTIFER OFF を指定すれば二重引用符も使用できます。それ以外の場合、二重引用符は識別子の区切り文字として解析されます。

一重引用符で文字列リテラルを区切ると、文字列内にリテラル二重引用符を含めることができます。文字列内にリテラル一重引用符を含めるには、一重引用符を 2 つ入力して引用符を二重にします。

リテラルの一重引用符が含まれる文字列 (`'this is an ''embedded'' string'` など) を InterSystems IRIS でコンパイルすると、二重引用符の中に一重引用符が含まれた文字列 (`"this is an 'embedded' string"`) に変換されます。

## 3.1.2 空の文字列

Transact-SQL コードを InterSystems TSQL に移行する場合は、空文字列の再定義が必要となる場合があります。この再定義を行うには、以下の InterSystems IRIS システム・グローバルを設定します。

```
^%SYS("sql","sys","namespace",nspace,"empty string")
```

引用符付き文字列として指定されたネームスペース名 nspace を除き、ここで指定される値はすべてキーワードのリテラルです。

注意　空文字列定義の変更には特に注意が必要です。この変更により、空文字列に対して異なる表現を含むデータが作成される可能性があります。また、このネームスペースで既存のプログラムを実行すると失敗する場合もあります。空文字列を定義したら、すべてのクエリ・キャッシュを削除して、変更前の空文字列定義を使用するネームスペースのすべてのクラスとルーチンをリコンパイルする必要があります。

以下の ObjectScript の例では、SAMPLES ネームスペースの空文字列定義を変更します。最初に、空文字列の値を 1 つの空白に設定します。次に、空文字列の値を ASCII コード 0 で表される出力不能文字に設定します (この例では、その直後に空文字列を InterSystems IRIS の既定値にリセットします)。

ObjectScript

```
SET ^%SYS("sql","sys","namespace","SAMPLES","empty string")=" "
WRITE !,"Empty string set to:"
ZZDUMP ^%SYS("sql","sys","namespace","SAMPLES","empty string")
SET ^%SYS("sql","sys","namespace","SAMPLES","empty string")=$CHAR(0)
WRITE !,"Empty string set to:"
ZZDUMP ^%SYS("sql","sys","namespace","SAMPLES","empty string")
SET ^%SYS("sql","sys","namespace","SAMPLES","empty string")=""
WRITE !,"Empty string reset to:"
ZZDUMP ^%SYS("sql","sys","namespace","SAMPLES","empty string")
WRITE !,!,"End of sample program"
```

## 3.1.3 NULL

TSQL では、ブーリアン演算に指定された NULL は、以下の例に示すように、False として返されます。

```
DECLARE @var BINARY(1)
SELECT @var=NULL
IF @var PRINT "true" ELSE PRINT "false"
```

Sybase 言語では、NULL は NULL と等しくなります。NULL=NULL 比較は True を返し、NULL != NULL 比較は False を返します。

MSSQL 言語では、任意の値と NULL の比較は False を返します。したがって、NULL=NULL 比較と NULL != NULL 比較の両方が False を返します。

```
DECLARE @var BINARY(1)
SELECT @var=NULL
IF @var=NULL PRINT "true" ELSE PRINT "false"
```

Sybase 言語では、NULL はいずれの値とも等しくなりません。したがって、NULL およびブーリアン、数値、または文字列値 (空の文字列 (″″) も含まれます) を含む Not Equals (!=) 比較は True を返します。Equals (=)、Greater Than (>)、Less Than (<) 比較はすべて、False を返します。

MSSQL 言語では、NULL を値と比較することはできません。したがって、Equals (=)、Not Equals (!=)、Greater Than (>)、Less Than (<) 比較はすべて、False を返します。

TSQL の文字列連結演算では、NULL は空の文字列に相当します。TSQL の数値演算では、NULL は 0 に相当します。

## 3.1.4 16 進数

InterSystems TSQL は、TSQL ソース・コード内の 16 進数の数値リテラルを対応する 10 進数 (10 進法) の数値リテラルに自動的に変換します。

## 3.1.5 予約語

InterSystems TSQL では、識別子に SQL Server の予約語を使用できません。SQL 構成設定の [QUOTED_IDENTIFIER] が [はい] に設定されている場合は、InterSystems TSQL で InterSystems SQL 予約語 (SQL Server の予約語ではない) を使用できます。

## 3.1.6 コメント、空白行、セミコロン

InterSystems TSQL では、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。

- その行の最後までが 1 行コメントです。TSQL シェルでコメントを使用すると、/x や /c などの文末修飾子はコメントに含まれません。InterSystems TSQL では、1 行コメントの区切り文字として -- と // の両方をサポートします。
- 複数行コメントは、/* で開始し、*/ で終了します。コメントでは、/* ...*/ コメントを入れ子にすることもできます。

TSQL

```
PRINT 'these are comments'
-- this is a single-line comment
// this is a single-line comment
/* This is a multi-line comment
The command
PRINT 'do not print'
is part of the comment and is not executed */
```

### 3.1.6.1 TSQL 専用の文

InterSystems TSQL では、InterSystems IRIS TSQL コード内で実行可能文を記述する手段になり、Transact-SQL では実行されないコメントとして解析されます。2 つのハイフンと垂直バーから始まる文は、InterSystems IRIS では実行可能な文として解析されます。Sybase Adaptive Server と Microsoft SQL Server では、Transact-SQL コメントと見なされます。

TSQL

```
PRINT 'any context'
-- PRINT 'commented out'
--| PRINT 'InterSystems only'
```

### 3.1.6.2 セミコロン

2 つのハイフンまたはセミコロンのいずれかを使用することにより、空白行を指定できます。

TSQL 文の前後のセミコロンは無視されます。これは、セミコロンで文が終了する Transact-SQL コード (ストアド・プロシージャなど) との互換性を保つためにサポートされています。

TSQL

```
PRINT 'no semicolon'
--
PRINT 'trailing semicolon';
;
;PRINT 'leading semicolon'
```

# 3.2 変数

TSQL では、ローカル変数のデータ型を宣言するために DECLARE を使用します。TSQL では、ローカル変数の値を設定するために SET を使用します。

ローカル変数名は、有効な識別子である必要があります。識別子のアットマーク (@) の接頭語は、それがローカル変数の名前であることを示します。

大文字/小文字の区別の有無は、TSQL 言語によって異なります。

- Sybase：ローカル変数名では、大文字と小文字が区別されます。
- MSSQL：ローカル変数名では、大文字と小文字が区別されません。

InterSystems IRIS のローカル変数名では、大文字と小文字が区別されます。

# 3.3 識別子

識別子とは、テーブル、列、ビュー、キー、インデックス、トリガ、ストアド・プロシージャなどのTSQL オブジェクトの名前です。識別子の命名規約は、以下のとおりです。

- 識別子の最初の文字は、英字、アンダースコア (_)、またはパーセント (%) 記号にする必要があります。
- 識別子の 2 文字目以降は、英字、数字、アンダースコア (_)、ドル記号 ($)、またはシャープ記号 (#) を使用できます。
- 識別子の長さに制限はなく、最初の 30 文字は一意である必要があります。
- 識別子では、大文字と小文字は区別されません。
- 識別子には SQL 予約語を使用できません。
- 識別子のシャープ記号 (#) の接頭語は、それが一時テーブルの名前であることを示します。
- 識別子のアットマーク (@) の接頭語は、それが変数名であることを示します。

一部の識別子はスキーマ名で修飾されています。例えば、schema.tablename や schema.storedprocedure のように修飾されています。スキーマ名が省略されている場合は、その識別子は未修飾です。TSQL は、システム全体の既定のスキーマ (DDL の場合) または schemaPath プロパティ (DML の場合) を使用して未修飾の識別子を解決します。schemaPath プロパティでは、指定されたテーブル名やストアド・プロシージャ名を検索する場所となるスキーマ検索パスが指定されます。

## 3.3.1 区切り識別子および引用符で囲まれた識別子

区切り識別子は、通常の識別子の名前付け規約による制限を受けません。例えば、区切り識別子は SQL 予約語と同じ語にできます。区切り識別子にはスペース文字を含めることができます。

既定では、角括弧および二重引用符の両方が、識別子を区切るために使用できます。これらの区切り文字は置き換えできます。区切り識別子は角括弧で囲んで定義します。また、二重引用符で囲んで指定すると、同じ区切り識別子を呼び出すことができます。

SQL 構成設定の [QUOTED_IDENTIFIER] が [はい] に設定されている場合は、引用符で囲まれた識別子を指定できます。引用符で囲まれた識別子を指定するには、その識別子を二重引用符で囲みます。QUOTED_IDENTIFIER がオンの場合、二重引用符が識別子の区切り文字として解析されます。QUOTED_IDENTIFIER がオフの場合、二重引用符が

文字列リテラルの代替の区切り文字として解析されます。文字列リテラルの望ましい区切り文字は一重引用符です。引用符で囲まれた識別子には、空白を含め、あらゆる文字を使用できます。

# 3.4 データ型

ローカル変数とテーブルの列では、次のデータ型がサポートされています。これらのデータ型は、有効なデータ型として解析される場合はサポートされます。範囲や値は検証されません。

BINARY(n) および VARBINARY(n)。(n) のサイズ指定は必須です。

BIT

BOOLEAN

CHAR と VARCHAR

CHAR(n)、NCHAR(n)、VARCHAR(n)、および NVARCHAR(n)

VARCHAR(MAX) および NVARCHAR(MAX)。既定では、これらは **%Stream.GlobalCharacter** にマップされます。

DATETIME および SMALLDATETIME

DECIMAL、DECIMAL(p)、および DECIMAL(p,s)。ここで p と s は、精度 (合計桁) とスケール (小数桁) を指定する整数です。

DOUBLE および DOUBLE PRECISION

FLOAT および FLOAT(n)

INT、BIGINT、SMALLINT、および TINYINT

MONEY および SMALLMONEY

NATIONAL

NUMERIC、NUMERIC(p)、および NUMERIC(p,s)。ここで p と s は、精度 (合計桁) とスケール (小数桁) を指定する整数です。

REAL

TIMESTAMP

注釈　Microsoft SQL Server の TIMESTAMP データ型は、日付や時刻の情報を示す目的では使用されません。これは、テーブルでレコードの挿入または更新が行われた回数を示す整数カウンタです。YYYY-MM-DD HH:MM:SS.nnnnnnnnn 形式で日付や時刻を示す InterSystems SQL および ODBC の TIMESTAMP データ型と混同しないでください。TSQL では、日付や時刻の値を示すのに DATETIME や SMALLDATETIME を使用します。

ROWVERSION

SQL_VARIANT

次の SQL Server データ型は、特定のコンテキストにおいてサポートされます。

CURSOR

NTEXT、TEXT。既定では、これらは **%Stream.GlobalCharacter** にマップされます。

IMAGE

TABLE

以下は実装されていません。

- 16 バイトのバイナリ文字列として格納される UNIQUEIDENTIFIER。代わりに VARCHAR(32) をグローバルに一意な ID のデータ型として使用します。
- SQL92 オプションおよび TSQL オプション
- UPDATE OF

# 3.5 演算子

## 3.5.1 算術演算子および等値演算子

InterSystems TSQL では、+ (加算)、- (減算)、* (乗算)、/ (除算)、および % (モジュロ算術) の各演算子がサポートされます。

InterSystems TSQL では、以下の等値および比較演算子がサポートされます。

- = (等しい)
- <> (等しくない) および != (等しくない)
- < (より小さい)、!< (より小さくない)、<= (以下)
- > (より大きい)、!> (より大きくない)、>= (以上)

異なるデータ型の日付値で等値比較 (= または <>) を行う場合、すべての日付値および時間値は TIMESTAMP データ型を使用して比較されます。したがって、異なる形式の 2 つの日付を意味のある方法で比較できます。例えば、STRING データ型として宣言されている日付値を、DATETIME データ型として宣言されている日付値と比較できます。

## 3.5.2 連結演算子

InterSystems TSQL では、連結演算子および加算演算子の両方で + (プラス記号) がサポートされます。文字列と共に使用すると、プラス記号は連結演算子として機能します。この演算子を使用すれば複数の文字列を連結できます。すべての項目が文字列であれば連結演算が実行されますが、項目に 1 つでも数値が含まれている場合、TSQL は非数値文字列を 0 として加算演算を実行します。

'world'+'wide'+'web' は 'worldwideweb' に連結されます。

'world'+'33'+'web' は 'world33web' に連結されます。

'world'+33+'web' では加算演算が実行されます (0+33+0=33)。

TSQL の文字列連結演算では、NULL は空の文字列に相当します。TSQL の数値演算では、NULL は 0 に相当します。プラス記号 (+) は連結と加算の両方に使用されるため、NULL 変数のデータ型宣言は重要です。以下の例はすべて、"bigdeal" を返します。

```
DECLARE @var1 BINARY(1)
DECLARE @var2 VARCHAR(10)
SELECT @var1=NULL,@var2=NULL
PRINT "big"+NULL+"deal"
PRINT "big"+@var1+"deal"
PRINT "big"+@var2+"deal"
```

以下の例は 0 を返します。この例では、+ を数値演算子として扱い、引数を 0 + 0 + 0 = 0 と解釈します。

```
DECLARE @var1 INT
SELECT @var1=NULL
PRINT "big"+@var1+"deal"
```

InterSystems TSQL では、連結演算子として || もサポートされます。

## 3.5.3 比較演算子

### 3.5.3.1 BETWEEN

InterSystems TSQL では、`BETWEEN num1 AND num2` という形式で、範囲をチェックする BETWEEN 演算子がサポートされます。BETWEEN には指定された範囲の上限値と下限値が含まれます。

### 3.5.3.2 IS NULL

InterSystems TSQL では、IS NULL マッチ演算子がサポートされます。宣言されても値が割り当てられない変数および明示的に NULL と指定されている変数は、NULL として処理されます。空の文字列は NULL ではありません。

### 3.5.3.3 LIKE

InterSystems TSQL では、LIKE パターン・マッチ演算子がサポートされます。LIKE では、文字のマッチングで大文字と小文字が区別されません。InterSystems TSQL では、NOT LIKE もサポートされます。

## 3.5.4 NOT 論理演算子

NOT 論理演算子は、後に続く文の真偽値を反転します。例えば、`IF NOT EXISTS(...)` のように指定します。NOT では大文字と小文字が区別されません。

## 3.5.5 ビット単位論理演算子

InterSystems TSQL では、INTEGER データ型に対して AND (&)、OR (|)、XOR (^)、および NOT (~) ビット演算子がサポートされます。10 進数の整数値がバイナリ値に変換されて論理演算が実行され、その結果として得られるバイナリ値が 10 進数の整数値に変換されます。NOT (~) 演算子はビットを反転する単項演算子です。

# 4

# TSQL コマンド

この章では、InterSystems IRIS® データ・プラットフォームでサポートされている TSQL コマンドを以下のグループに分けて説明します。

- データ定義言語 (DDL) 文：

  ALTER TABLE、CREATE TABLE、DROP TABLE

  CREATE INDEX、DROP INDEX

  CREATE TRIGGER、DROP TRIGGER

  CREATE VIEW、DROP VIEW

  CREATE DATABASE、DROP DATABASE (解析されるが無視される)
- データ管理言語 (DML) 文：

  INSERT、UPDATE、DELETE、TRUNCATE TABLE

  READTEXT、WRITETEXT、UPDATETEXT
- クエリ文：

  SELECT、JOIN、UNION、FETCH カーソル
- 制御文のフロー：

  IF、WHILE、CASE、GOTO、WAITFOR
- 代入文：

  DECLARE、SET
- トランザクション文：

  SET TRANSACTION ISOLATION LEVEL、BEGIN TRANSACTION、COMMIT、ROLLBACK、LOCK TABLE

  解析されるが無視される：SAVE TRANSACTION
- プロシージャ文

  CREATE PROCEDURE、DROP PROCEDURE

  CREATE FUNCTION、ALTER FUNCTION、DROP FUNCTION

  RETURN、EXECUTE、EXECUTE IMMEDIATE、CALL
- その他の文

  CREATE USER、GRANT、REVOKE、PRINT、RAISERROR、UPDATE STATISTICS

- InterSystems IRIS 拡張機能

  OBJECTSCRIPT、IMPORTASQUERY

InterSystems IRIS における TSQL 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。TSQL 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。

# 4.1 データ定義言語 (DDL) 文

以下の DDL 文がサポートされます。

## 4.1.1 CREATE TABLE

テーブル、そのフィールド、データ型、および制約を定義します。

```
CREATE TABLE [schema. | #]tablename (fieldname datatype constraint [,...])
```

"テーブル参照" の説明に従って tablename を指定します。

CREATE TABLE では、テーブル名の先頭に # 文字を付けることにより、一時テーブルを作成できます。一時テーブルは、ストアド・プロシージャからのみ定義できます。ストアド・プロシージャの外部にあるダイナミック SQL から一時テーブルを定義することはできません。完全修飾の一時テーブル名を作成するには、"SQLUser"."#mytemp" のように名前の各要素を引用符で囲みます。

有効なテーブル名は、英字、アンダースコア (_)、または # 記号 (ローカルの一時テーブル用) で始まる必要があります。2 文字目以降は、英字、数字、または #、$、_ 記号を使用できます。テーブル名では、大文字と小文字は区別されません。

フィールド名は、有効な TSQL 識別子である必要があります。フィールド名は角括弧を使用して区切ることができます。これは特に、予約語と同じ名前を持つフィールドを定義する際に便利です。以下の例では、Check および Result という名前の 2 つフィールドを定義しています。

### TSQL

```
CREATE TABLE mytest ([Check] VARCHAR(50),[Result] VARCHAR(5))
```

オプションの CONSTRAINT キーワードを使用して、フィールド制約またはテーブル制約に使用するユーザ定義の制約名を指定できます。1 つのフィールドに複数の CONSTRAINT name type 文を指定できます。

InterSystems SQL では制約名を保持しません。そのため、これらの名前は後の ALTER TABLE 文で使用することはできません。

テーブル・フィールド制約 DEFAULT、IDENTITY、NULL、NOT NULL、PRIMARY KEY、[FOREIGN KEY] REFERENCES (キーワード FOREIGN KEY はオプション)、UNIQUE、CLUSTERED、および NONCLUSTERED がサポートされます。テーブル制約 FOREIGN KEY REFERENCES がサポートされます。

フィールド定義の DEFAULT 値には、CURRENT_TIMESTAMP、CURRENT_USER、GETDATE、HOST_NAME、ISNULL、NULLIF、および USER の各 TSQL 関数を指定できます。

フィールド定義の IDENTITY 制約がサポートされ、システムで生成される連続番号が割り当てられます。IDENTITY 引数 seed と increment は解析されますが、無視されます。

TSQL の CREATE TABLE コマンドでは、シャード・テーブルを作成できます。SHARD 節の構文は、InterSystems SQL の CREATE TABLE 文の場合と同じです。

```
SHARD [ KEY fieldname { , fieldname2 } ] [ COSHARD [ WITH ] [(] tablename [)] ]
```

CHECK フィールド制約はサポートされません。CHECK 制約が TSQL ソースのコンパイル中に見つかった場合、InterSystems IRIS はエラー・メッセージを生成して CHECK 制約をサポートしていないことを示します。エラーはコンパイル・ログに記録され (アクティブの場合)、ソースは未サポート・ログに置かれます (アクティブの場合)。

既にテーブルが存在している場合は、SQLCODE -201 エラーが発行されます。

以下のダイナミック SQL の例では、4 つのフィールドがある #mytest という名前の一時テーブルを作成し、そのテーブルにデータを移入してから、結果を表示します。LastName フィールドには、複数の制約があります。FirstName フィールドは既定値を取ります。DateStamp フィールドは、システム定義の既定値を取ります。

ObjectScript

```
SET sql=9
SET sql(1)="CREATE TABLE #mytest (MyId INT PRIMARY KEY,"
SET sql(2)="LastName VARCHAR(20) CONSTRAINT unq_lname UNIQUE "
SET sql(3)="  CONSTRAINT nonull_lname NOT NULL,"
SET sql(4)="FirstName VARCHAR(20) DEFAULT '***TBD***',"
SET sql(5)="DateStamp DATETIME DEFAULT CURRENT_TIMESTAMP)"
SET sql(6)="INSERT INTO #mytest(MyId,LastName,FirstName) VALUES (1224,'Smith','John')"
SET sql(7)="INSERT INTO #mytest(MyId,LastName) VALUES (1225,'Jones')"
SET sql(8)="SELECT MyId,FirstName,LastName,DateStamp FROM #mytest"
SET sql(9)="DROP TABLE #mytest"
SET statement=##class(%SQL.Statement).%New()
SET statement.%Dialect="MSSQL"
SET status=statement.%Prepare(.sql)
WRITE status,!
SET result=statement.%Execute()
DO result.%Display()
```

#### 4.1.1.1 解析されるが無視される

テーブル制約節 WITH、ON、および TEXTIMAGE ON は互換性を保つために解析されますが無視されます。UNIQUE 制約または PRIMARY KEY 制約の index_options 節は互換性を保つために解析されますが、無視されます。

テーブル制約の以下の SQL サーバの括弧で囲んだ WITH オプションは解析されますが、無視されます。ALLOW_PAGE_LOCKS、ALLOW_ROW_LOCKS、DATA_COMPRESSION、FILLFACTOR、IGNORE_DUP_KEY、PAD_INDEX および STATISTICS_NORECOMPUTE。

フィールド制約 CLUSTERED および NONCLUSTERED は互換性を保つために解析されますが、無視されます。

### 4.1.2 ALTER TABLE

テーブル、そのフィールド、データ型、および制約を変更します。

以下の構文形式がサポートされます。

```
ALTER TABLE tablename ADD fieldname datatype [DEFAULT value]
     [{UNIQUE | NOT NULL} | CONSTRAINT constraintname {UNIQUE | NOT NULL} ]
ALTER TABLE tablename ALTER COLUMN fieldname newdatatype
ALTER TABLE tablename DROP COLUMN fieldname [,fieldname2]
ALTER TABLE tablename ADD tableconstraint FOR fieldname
ALTER TABLE tablename DROP tableconstraint
ALTER TABLE tablename DROP FOREIGN KEY role
ALTER TABLE tablename ADD CONSTRAINT constraint DEFAULT defaultvalue FOR fieldname
ALTER TABLE tablename ADD CONSTRAINT constraint FOREIGN KEY
ALTER TABLE tablename DROP CONSTRAINT constraint
```

"テーブル参照" の説明に従って tablename を指定します。

- ALTER TABLE...ADD fieldname では、1 つのフィールド定義またはフィールド定義のコンマ区切りリストを追加できます。
  - DEFAULT がサポートされます。
  - テーブルにデータが含まれていない場合、NOT NULL がサポートされます。テーブルにデータが含まれている場合、NOT NULL を指定できるのは、フィールドで DEFAULT 値も指定している場合のみです。
  - UNIQUE は解析されますが、無視されます。一意制約を設定するには、UNIQUE キーワードを指定して CREATE INDEX コマンドを使用します。

  ALTER TABLE...ADD fieldname でサポートされる完全な構文は、次のとおりです。

```
ALTER TABLE tablename
   [ WITH CHECK | WITH NOCHECK ]
  ADD fieldname datatype [DEFAULT value]
     [{UNIQUE | NOT NULL} | CONSTRAINT constraintname {UNIQUE | NOT NULL} ]
     [ FOREIGN KEY (field1[,field2[,...]])
     REFERENCES tablename(field1[,field2[,...]]) ]
```

  WITH CHECK | WITH NOCHECK は InterSystems IRIS により解析はされますが、無視されます。Transact-SQL では、WITH CHECK | WITH NOCHECK により新規の制約もしくは新たに有効となった制約に対して、既存データの実行時間チェックが用意されています。InterSystems TSQL では特にそのサポートはありませんが、InterSystems SQL では新規の制約に対する既存データのチェックが行われます。

  Sybase の PARTITION BY 節はサポートされません。

- ALTER TABLE...ALTER COLUMN fieldname datatype では、既存のフィールドのデータ型を変更できます。指定された datatype がフィールドの既存のデータ型と同じである場合も、このコマンドはエラーなしで完了します。
- ALTER TABLE...DROP [COLUMN] fieldname では、1 つの定義済みフィールドまたは定義済みフィールドのコンマ区切りリストを削除できます。キーワード DELETE はキーワード DROP の同義語です。
  - Sybase : COLUMN キーワードは許可されません。CONSTRAINT キーワードは必須です : ALTER TABLE...DROP fieldname, CONSTRAINT constraint
  - MSSQL : COLUMN キーワードは必須です。CONSTRAINT キーワードはオプションです : ALTER TABLE...DROP COLUMN fieldname, constraint
- ALTER TABLE...DROP [CONSTRAINT] constraintname では、フィールドから制約を削除できます。キーワード DELETE はキーワード DROP の同義語です。
  - Sybase : CONSTRAINT キーワードは必須です。
  - MSSQL : CONSTRAINT キーワードはオプションです。
- ALTER TABLE...ADD CONSTRAINT...DEFAULT 構文では、フィールド制約は作成されません。代わりに、この構文は ALTER TABLE...ALTER COLUMN...DEFAULT 文と同等に実行されます。これは、InterSystems IRIS では、指定したフィールドの既定値がフィールド・プロパティの初期値式として設定されることを意味します。フィールド制約は定義されないので、この “制約” を後で削除したり、変更したりすることはできません。

  CHECK | NOCHECK CONSTRAINT は、InterSystems IRIS TSQL ではサポートされていません。この CHECK または NOCHECK キーワードの指定により、エラー・メッセージが生成されます。

## 4.1.3 DROP TABLE

テーブル定義を削除します。

```
DROP TABLE [IF EXISTS] tablename
```

テーブル定義を削除します。通常のテーブルと一時テーブルの両方を削除できます (一時テーブル名は '#' 文字で開始します)。存在しない一時テーブル名を指定しても DROP TABLE では無視され、エラーも生成せずに終了します。

"テーブル参照" の説明に従って tablename を指定します。

tablename に関連付けられているビューがある場合、テーブルを削除する前に、ビューを削除する必要があります。

IF EXISTS 節は解析されますが、無視されます。

## 4.1.4 CREATE INDEX

指定したテーブルまたはビューにインデックスを作成します。

```
CREATE [UNIQUE] INDEX indexname ON tablename (fieldname [,fieldname2])
```

1 つのフィールドまたはフィールドのコンマ区切りリストに対してインデックスを作成できます。

インデックスを IDKEY (クラスタ化インデックスとして扱われる)、IDENTITY フィールド (%%ID フィールドにインデックスを作成する)、主キー、またはその他のフィールドに作成できます。

"テーブル参照" の説明に従って tablename を指定します。

UNIQUE キーワードは、指定されたフィールドに対して一意の値制約インデックスを作成します。

次の Transact-SQL 機能は解析されますが無視されます。

- CLUSTERED/NONCLUSTERED キーワード暗黙的にクラスタ化インデックスとして扱われる IDKEY 以外に、InterSystems TSQL ではクラスタ化インデックスはサポートされません。
- ON dbspace 節。
- ASC/DESC キーワード
- INCLUDE 節
- WITH FILLFACTOR=n や WITH DROP_EXISTING=ON などの WITH 節オプション。コンマで区切られた WITH 節のオプションのリストは、必要に応じて括弧で囲むことができます。
- ON ファイルグループまたは IN DB スペース名の節

現在、以下の Transact-SQL 機能はサポートされていません。

- Sybase のインデックス・タイプ。
- IN dbspace 節。
- NOTIFY integer 節。
- LIMIT integer 節。
- フィールド名の代わりとしての関数名の使用。

ALTER INDEX 文はサポートされません。

## 4.1.5 DROP INDEX

インデックス定義を削除します。以下の構文形式のいずれかを使用して、単一のインデックス、またはコンマ区切りのインデックスのリストを削除できます。

```
DROP INDEX tablename.indexname [,tablename.indexname]

DROP INDEX indexname ON tablename [WITH (...)] [,indexname ON tablename [WITH (...)]
 ]
```

tablename は、インデックスが付いたフィールドを含むテーブルの名前です。"テーブル参照" の説明に従って tablename を指定します。#temptable を指定できるのは、現在のネームスペースがシャード・クラスタの一部である場合のみです。

indexname はインデックスの名前です。標準的な識別子または引用符で囲まれた識別子を指定できます。

括弧内に値が記述された WITH (...) 節は、互換性の構文チェックは通過しますが、検証されず、操作は何も実行されません。

IF EXISTS 節はサポートされません。

## 4.1.6 CREATE TRIGGER

文レベルのトリガを作成します。

```
CREATE TRIGGER triggername ON tablename
[WITH ENCRYPTION]
{FOR | AFTER | INSTEAD OF} {INSERT | DELETE | UPDATE}
[WITH APPEND]
[NOT FOR REPLICATION]
AS tsql_trigger_code
```

トリガは、1 つのイベント (INSERT) について作成することも、イベントのコンマ区切りリスト (INSERT,UPDATE) について作成することもできます。

"テーブル参照" の説明に従って tablename を指定します。

FOR、AFTER、および INSTEAD OF キーワードは同義語です。イベント操作が実行されると、常にトリガがプルされます。

同じイベントまたはイベントのコンマ区切りリストに対して複数のトリガがある場合は、トリガが作成された順に実行されます。

WITH ENCRYPTION、WITH APPEND、NOT FOR REPLICATION の各節は解析されますが、無視されます。

InterSystems TSQL では、行レベル・トリガはサポートされていません。

CREATE PROCEDURE コードに CREATE TRIGGER 文を含めることはできません。

## 4.1.7 DROP TRIGGER

トリガ定義を削除します。

```
DROP TRIGGER [owner.]triggername
```

## 4.1.8 CREATE VIEW

ビュー定義を作成します。

```
CREATE VIEW [owner.]viewname
    [WITH {ENCRYPTION | SCHEMABINDING | VIEW_METADATA}]
    AS select_statement
   [WITH CHECK OPTION]
```

viewname は、一意の TSQL 識別子である必要があります。"テーブル参照" の説明に従って viewname を指定します。既にビューが存在している場合は、SQLCODE -201 エラーが発行されます。viewname は、区切り識別子にできます。例えば、`CREATE VIEW Sample.[Name/Age View]` のようにできます。

既定では、ビューのフィールドの名前は SELECT テーブルのフィールドと同じです。ビューのフィールドに異なる名前を指定するには、SELECT 文でフィールドのエイリアスを指定します。これらのエイリアスがビューのフィールド名として使用されます。

TSQL

```
CREATE VIEW NameAgeV
AS SELECT Name AS FullName,Age AS Years FROM Sample.Person
```

単一のキーワードまたはキーワードのコンマ区切りのリストを含む WITH 節を指定できます。例えば、`WITH SCHEMABINDING, ENCRYPTION, VIEW_METADATA` のように指定できます。ENCRYPTION、SCHEMABINDING、VIEW_METADATA の各キーワードは解析されますが、無視されます。

select_statement 節は、TOP 節と組み合わせた場合にのみ、ORDER BY 節を含めることができます。ビュー内のすべての行を含める場合は、ORDER BY 節と TOP ALL 節を組み合わせることができます。TOP 節は ORDER BY 節を指定しなくても含めることができます。ただし、TOP 節を使用せずに ORDER BY 節を含めると、SQLCODE -143 エラーが生成されます。

select_statement には、UNION または UNION ALL を含めることができます。

オプションの WITH CHECK OPTION 節を使用すると、レコードからビューへのアクセスをできなくするビューを通じて更新することができなくなります。これには、SELECT 文の WITH 節をチェックします。WITH CHECK OPTION は、CASCADE の既定値を使用して InterSystems SQL にバインドします。

ALTER VIEW 文はサポートされません。

## 4.1.9 DROP VIEW

ビュー定義を削除します。

```
DROP VIEW viewname [,viewname2 [,...] ]
```

単一のビュー、またはコンマ区切りのビューのリストを削除できます。"テーブル参照" の説明に従って viewname を指定します。

DROP VIEW は、全か無かの操作ではありません。存在しないビューをビューのリスト内で検出するまで、リスト内の既存のビューを削除します。存在しないビューを検出した時点で、削除操作は停止し、SQLCODE -30 エラーを返します。

IF EXISTS 節はサポートされません。

### 4.1.10 CREATE DATABASE

CREATE DATABASE 構文は、互換性を提供するために解析されます。機能は何もありません。

```
CREATE DATABASE dbname
```

この基本的な CREATE DATABASE 構文のみが解析されます。

Sybase における追加の CREATE DATABASE 節はサポートされません。

MSSQL におけるデータベースのアタッチとデータベース・スナップショットの作成の構文オプションはサポートされません。

ALTER DATABASE 文はサポートされません。

### 4.1.11 DROP DATABASE

DROP DATABASE 構文は、互換性を提供するために解析されます。機能は何もありません。

```
DROP DATABASE dbname
```

## 4.2 データ管理言語 (DML) 文

- TSQL は、ダイナミック SQL 内の単一の DML 文の場合、スキーマ検索パスを使用して未修飾テーブル名を解決できます。
- TSQL は、ダイナミック SQL 内の複数の DML 文の場合には、スキーマ検索パスを使用して未修飾テーブル名を解決することはできません。これには、明示的な BEGIN TRANSACTION の後に単一の DML 文が続く場合などの複数文が含まれます。

### 4.2.1 DELETE

テーブルからデータ行を削除します。DELETE と DELETE ... FROM の両方がサポートされます。

```
DELETE FROM tablename  WHERE condition

DELETE FROM tablename FROM matchtablename WHERE tablename.fieldname =
matchtablename.fieldname
```

非常に単純なシータ結合のみがサポートされます (`FROM table` 節は入れ子のサブクエリに変換されます)。

1 つ以上の実行オプションをコンマ区切りリストとして指定することによって、DELETE の実行方法を指定できます。これらのオプションは、コメント内で以下の固有の構文を使用して指定します。

```
/* IRIS_DELETE_HINT: option,option2 */
```

option には、%NOCHECK、%NOFPLAN、%NOINDEX、%NOLOCK、%NOTRIGGER、%PROFILE、%PROFILE_ALL を指定できます。詳細は、InterSystems SQL の "DELETE" コマンドを参照してください。

最適化ヒントをコンマ区切りリストとして DELETE FROM 節に指定できます。これらのヒントは、コメント内で以下の固有の構文を使用して指定します。

```
/* IRIS_DELETEFROM_HINT: hint,hint2 */
```

hint には、%ALLINDEX、%FIRSTTABLE tablename、%FULL、%INORDER、%IGNOREINDICES、%NOFLATTEN、%NOMERGE、%NOSVSO、%NOTOPOPT、%NOUNIONOROPT、および %STARTTABLE を指定できます。詳細は、InterSystems SQL の "FROM" 節を参照してください。

FASTFIRSTROW、HOLDINDEX、INDEX(name)、NOLOCK、PAGLOCK、READCOMMITTED、READPAST、READUNCOMMITTED、REPEATABLEREAD、ROWLOCK、SERIALIZABLE、SHARED、TABLOCK、TABLOCKX、UPDLOCK、XLOCK という table_hints は解析されますが、無視されます。テーブル・ヒントの前に WITH キーワードを付けることもできます。WITH を指定した場合、括弧で囲むかどうかはオプションです。テーブル・ヒントのリストはコンマか空白のスペースで区切られます。

DELETE では、@@ROWCOUNT システム変数に削除する行数を設定し、@@IDENTITY システム変数に最後に削除した行の IDENTITY 値を設定します。

@@ROWCOUNT が 100,000 を超える DELETE では、自動的に UPDATE STATISTICS が呼び出されて、今後のクエリのためにテーブルが最適化されます。

DELETE または TRUNCATE TABLE のいずれかを使用して、テーブルからすべての行を削除することができます。DELETE は @@ROWCOUNT を削除される行の数に設定します。TRUNCATE TABLE の方が効率的ですが、削除される行の数は保持されず、@@ROWCOUNT は -1 に設定されます。DELETE では、RowID カウンタや他の行カウンタはリセットされません。TRUNCATE TABLE では、これらのカウンタはリセットされます。

以下のオプションはサポートされません。

- MSSQL の行セット関数。
- MSSQL の `OPTION` 節。

## 4.2.2 INSERT

テーブルにデータ行を挿入します。以下の構文形式がサポートされます。

```
INSERT [INTO] tablename (fieldname[,fieldname2[,...]]) VALUES (list_of_values)
INSERT [INTO] tablename (fieldname[,fieldname2[,...]]) SELECT select_list
```

INTO キーワードはオプションです。"テーブル参照" の説明に従って tablename を指定します。

VALUES 構文については、VALUES キーワードは MSSQL と Sybase の両方で必須です。list_of_values に、テーブルで定義されている順にユーザ指定のすべてのフィールドがリストされている場合、(fieldname) リストはオプションです。フィールド名が指定されている場合、list_of_values は、数およびデータ型がフィールド名のリストと一致する値のコンマ区切りのリストです。

1 つ以上の実行オプションをコンマ区切りリストとして指定することによって、INSERT の実行方法を指定できます。これらのオプションは、コメント内で以下の固有の構文を使用して指定します。

```
/* IRIS_INSERT_HINT: option,option2 */
```

option には、%NOCHECK、%NOFPLAN、%NOINDEX、%NOLOCK、%NOTRIGGER、%PROFILE、%PROFILE_ALL を指定できます。詳細は、InterSystems SQL の "INSERT" コマンドを参照してください。

FASTFIRSTROW、HOLDINDEX、INDEX(name)、NOLOCK、PAGLOCK、READCOMMITTED、READPAST、READUNCOMMITTED、REPEATABLEREAD、ROWLOCK、SERIALIZABLE、SHARED、TABLOCK、TABLOCKX、UPDLOCK、XLOCK という table_hints は解析されますが、無視されます。テーブル・ヒントの前に WITH キーワードを付けることもで

きます。WITH を指定した場合、括弧で囲むかどうかはオプションです。テーブル・ヒントのリストはコンマか空白のスペースで区切られます。

INSERT では、@@ROWCOUNT システム変数に挿入する行数を設定し、@@IDENTITY システム変数に最後に挿入した行の IDENTITY 値を設定します。

@@ROWCOUNT が 100,000 を超える INSERT では、自動的に UPDATE STATISTICS が呼び出されて、今後のクエリのためにテーブルが最適化されます。

以下のオプションはサポートされません。

- `(fieldname) DEFAULT VALUES` または `(fieldname) VALUES (DEFAULT)`。フィールドの既定値は、フィールドが INSERT 文で指定されていない場合に使用されます。
- `(fieldname) EXECUTE procname`。
- Sybase の insert load option 節：LIMIT、NOTIFY、SKIP、または START ROW ID。
- Sybase の insert select load option 節：WORD SKIP、IGNORE CONSTRAINT、MESSAGE LOG、または LOG DELIMITED BY。
- Sybase の LOCATION 節。
- MSSQL の `INSERT TOP` 節。
- MSSQL の行セット関数。

## 4.2.3 UPDATE

テーブルの既存のデータ行の値を更新します。

```
UPDATE tablename SET fieldname=value [,fieldname2=value2[,...]]
   [FROM tablename [,tablename2]] WHERE fieldname=value

UPDATE tablename SET fieldname=value[,fieldname2=value2[,...]]
   WHERE [tablename.]fieldname=value
```

次の構文形式はベンダ固有です。

- Sybase：オプションの FROM キーワード構文を使用して、条件で使用するオプションのテーブル（または結合されるテーブル）を指定します。非常に単純なシータ結合のみがサポートされます（`FROM table` 節は入れ子のサブクエリに変換されます）。
- MSSQL：tablename.fieldname 構文を使用して、条件で使用するオプションのテーブルを指定します。

value のデータ型および長さが、fieldname で定義されたデータ型および長さと一致している必要があります。value には、リテラル値に解決される式を指定することも、NULL キーワードを指定することもできます。DEFAULT キーワードを指定することはできません。

"テーブル参照" の説明に従って tablename を指定します。

UPDATE では、SET 節の左側でローカル変数を使用することをサポートしています。このローカル変数は、フィールド名の代わりに使用することも、フィールド名に加えて使用することもできます。以下の例は、フィールド名に対する SET、ローカル変数に対する SET、フィールド名とローカル変数の両方に対する SET を示しています。

```
UPDATE table SET x=3,@v=b,@c=Count=Count+1
```

1 つ以上の実行オプションをコンマ区切りリストとして指定することによって、UPDATE の実行方法を指定できます。これらのオプションは、コメント内で以下の固有の構文を使用して指定します。

```
/* IRIS_UPDATE_HINT: option,option2 */
```

option には、%NOCHECK、%NOFPLAN、%NOINDEX、%NOLOCK、%NOTRIGGER、%PROFILE、%PROFILE_ALL を指定できます。詳細は、InterSystems SQL の "UPDATE" コマンドを参照してください。

最適化ヒントをコンマ区切りリストとして UPDATE FROM 節に指定できます。これらのヒントは、コメント内で以下の固有の構文を使用して指定します。

```
/* IRIS_UPDATEFROM_HINT: hint,hint2 */
```

hint には、%ALLINDEX、%FIRSTTABLE tablename、%FULL、%INORDER、%IGNOREINDICES、%NOFLATTEN、%NOMERGE、%NOSVSO、%NOTOPOPT、%NOUNIONOROPT、および %STARTTABLE を指定できます。詳細は、InterSystems SQL の "FROM" 節を参照してください。

FASTFIRSTROW、HOLDINDEX、INDEX(name)、NOLOCK、PAGLOCK、READCOMMITTED、READPAST、READUNCOMMITTED、REPEATABLEREAD、ROWLOCK、SERIALIZABLE、SHARED、TABLOCK、TABLOCKX、UPDLOCK、XLOCK という table_hints は解析されますが、無視されます。テーブル・ヒントの前に WITH キーワードを付けることもできます。WITH を指定した場合、括弧で囲むかどうかはオプションです。テーブル・ヒントのリストはコンマか空白のスペースで区切られます。

UPDATE では、@@ROWCOUNT システム変数に更新する行数を設定し、@@IDENTITY システム変数に最後に更新した行の IDENTITY 値を設定します。

@@ROWCOUNT が 100,000 を超える UPDATE では、自動的に UPDATE STATISTICS が呼び出されて、今後のクエリのためにテーブルが最適化されます。

以下のダイナミック SQL の例は、単純な UPDATE 操作を示しています。

ObjectScript

```
SET sql=9
SET sql(1)="CREATE TABLE #mytest (MyId INT PRIMARY KEY,"
SET sql(2)="LastName VARCHAR(20) CONSTRAINT nonull_lname NOT NULL,"
SET sql(3)="FirstName VARCHAR(20) DEFAULT '***TBD***')"
SET sql(4)="INSERT INTO #mytest(MyId,LastName,FirstName) VALUES (1224,'Smith','John')"
SET sql(5)="INSERT INTO #mytest(MyId,LastName) VALUES (1225,'Jones')"
SET sql(6)="INSERT INTO #mytest(MyId,LastName) VALUES (1226,'Brown')"
SET sql(7)="UPDATE #mytest SET FirstName='Fred' WHERE #mytest.LastName='Jones'"
SET sql(8)="SELECT FirstName,LastName FROM #mytest ORDER BY LastName"
SET sql(9)="DROP TABLE #mytest"
SET statement=##class(%SQL.Statement).%New()
SET statement.%Dialect="MSSQL"
SET status=statement.%Prepare(.sql)
WRITE status,!
SET result=statement.%Execute()
DO result.%Display()
```

以下のオプションはサポートされません。

- Sybase の `ORDER BY` 節。
- MSSQL の `OPTION` 節。
- MSSQL の `TOP` 節。
- MSSQL の行セット関数。

## 4.2.4 READTEXT

ストリーム・フィールドからデータを読み取ります。

```
READTEXT tablename.fieldname textptr offset size
```

MSSQL READTEXT 文は、テーブルのフィールドからのストリーム・データを返します。これには、以下に示すように、TEXTPTR 関数を使用して取得できる、有効なテキスト・ポインタ値が必要です。

```
DECLARE @ptrval binary(16);
SELECT @ptrval = TEXTPTR(Notes) FROM Sample.Person
READTEXT Sample.Person.Notes @ptrval 0 0
```

textptr はバイナリとして宣言する必要があります。textptr は、NULL ではないテキスト・フィールドに対してのみ定義します。INSERT 文を使用して、テキスト・フィールドに初期の非 NULL 値を指定できます。

offset には、0、正の整数値、または NULL を指定できます。0 ではテキストの先頭から読み取ります。正の整数では offset の位置から読み取ります。NULL ではテキストの末尾から読み取ります。つまり、操作は正常に完了しますが値が返されません。

size には、0、正の整数値、または NULL を指定できます。0 では offset の位置からテキストの末尾までを読み取ります。正の整数では offset の位置から size の文字数を読み取ります。NULL では操作は正常に完了しますが、値が返されません。

MSSQL HOLDLOCK キーワードは解析されますが、無視されます。

## 4.2.5 WRITETEXT

データをストリーム・フィールドに書き込み、既存のデータ値を置き換えます。

```
WRITETEXT tablename.fieldname textptr value
```

MSSQL WRITETEXT は、データをテーブルのストリーム・フィールドに書き込みます。これには、以下に示すように、TEXTPTR 関数を使用して取得できる、有効なテキスト・ポインタ値が必要です。

TSQL

```
DECLARE @ptrval binary(16);
SELECT @ptrval = TEXTPTR(Notes) FROM Sample.Person
WRITETEXT Sample.Person.Notes @ptrval 'This is the new text value'
```

textptr はバイナリとして宣言する必要があります。textptr は、NULL ではないテキスト・フィールドに対してのみ定義します。INSERT 文を使用して、テキスト・フィールドに初期の非 NULL 値を指定できます。

MSSQL BULK キーワードはサポートされません。

MSSQL WITH LOG キーワードは解析されますが、無視されます。

## 4.2.6 UPDATETEXT

ストリーム・フィールドのデータを更新します。

```
UPDATETEXT tablename.fieldname textptr offset deletelength value
```

MSSQL UPDATETEXT 文は、テーブルのフィールドからのストリーム・データを更新します。これには、TEXTPTR 関数を使用して取得できる、有効なテキスト・ポインタ値が必要です。以下の例では、既存のデータ値の先頭に「New」という単語を挿入して、Notes ストリーム・フィールドの内容を更新します。

TSQL

```
DECLARE @ptrval binary(16);
SELECT @ptrval = TEXTPTR(Notes) FROM Sample.Person
WRITETEXT Sample.Person.Notes @ptrval 0 0 'New'
```

textptr はバイナリとして宣言する必要があります。textptr は、NULL ではないテキスト・フィールドに対してのみ定義します。INSERT 文を使用して、テキスト・フィールドに初期の非 NULL 値を指定できます。

offset には、正の整数値または NULL を指定できます。0 では、既存のテキストの先頭に value を挿入します。NULL では、既存のテキストの末尾に value を挿入します。

deletelenth には、正の整数値または NULL を指定できます。0 または NULL では、value を挿入する前に offset の位置から既存の文字を削除しません。正の整数では、value を挿入する前に、offset の位置から既存の文字を指定した数だけ削除します。

MSSQL BULK キーワードはサポートされません。

MSSQL WITH LOG キーワードは解析されますが、無視されます。

### 4.2.7 TRUNCATE TABLE

テーブルからすべてのデータを削除します。

```
TRUNCATE TABLE tablename
```

InterSystems SQL の TRUNCATE TABLE コマンドを呼び出して、指定されたテーブルからすべての行を削除し、RowId (ID)、IDENTITY、および SERIAL (%Counter) の行カウンタとストリーム・フィールドの OID カウンタ値をリセットします。TRUNCATE TABLE では、削除される行の数は保持されず、@@ROWCOUNT は -1 に設定されます。

1 つ以上の実行オプションをコンマ区切りリストとして指定することによって、TRUNCATE TABLE の実行方法を指定できます。これらのオプションは、コメント内で以下の固有の構文を使用して指定します。

```
/* IRIS_DELETE_HINT: option,option2 */
```

option には、%NOCHECK と %NOLOCK を指定できます。詳細は、InterSystems SQL の "TRUNCATE TABLE" を参照してください。

## 4.3 クエリ文

### 4.3.1 SELECT

```
SELECT  [DISTINCT | ALL]
  [TOP [(]{ int | @var | ? | ALL}[)]]
  select-item {,select-item}
  [ [fieldname=IDENTITY(n)] INTO [#]copytable]
  [FROM tablename [[AS] t-alias] [,tablename2 [[AS] t-alias2]] ]
  [[WITH] [(] tablehint=val [,tablehint=val] [)] ]
  [WHERE condition-expression]
  [GROUP BY scalar-expression]
  [HAVING condition-expression]
  [ORDER BY item-order-list [ASC | DESC] ]
```

上記の SELECT 構文がサポートされています。以下の機能はサポートされていません。

- TOP nn PERCENT または TOP WITH TIES

- OPTION
- WITH CUBE
- WITH ROLLUP
- GROUP BY ALL
- GROUP WITH
- COMPUTE
- FOR BROWSE

TOP nn は取得する行数を指定します。InterSystems TSQL では、整数、?、ローカル変数、またはキーワード ALL の TOP nn をサポートしています。TOP の引数は、括弧 TOP (nn) で囲むことができます。これらの括弧は、解析前の置換を避けるために維持されます。SET ROWCOUNT が TOP nn よりも少数の行を指定する場合は、SET ROWCOUNT 値が使用されます。以下のダイナミック SQL の例は、ローカル変数を含む TOP の使用法を示しています。

ObjectScript

```
SET sql=3
SET sql(1)="DECLARE @var INT"
SET sql(2)="SET @var=4"
SET sql(3)="SELECT TOP @var Name,Age FROM Sample.Person"
SET statement=##class(%SQL.Statement).%New()
SET statement.%Dialect="MSSQL"
SET status=statement.%Prepare(.sql)
SET result=statement.%Execute()
DO result.%Display()
```

select-item リストには、以下のものを含めることができます。

- フィールド名、関数、および式。
- $IDENTITY 疑似フィールド名。これは、RowID に割り当てられているフィールド名に関係なく、常に RowID 値を返します。
- アスタリスク。SELECT * がサポートされます。アスタリスクは、指定したテーブルのすべてのフィールドを選択することを意味します。アスタリスクは、テーブル名またはテーブルのエイリアスで SELECT mytable.* のように修飾できます。
- サブクエリ。
- ストリーム・フィールド。ストリーム・フィールドでの SELECT は、開いているストリーム・オブジェクトの oref (オブジェクト参照) を返します。

INTO 節を使用して、既存のテーブルから新しいテーブルにデータをコピーできます。既定では、SELECT は、ソース・テーブルから選択されたフィールドと同じフィールド名およびデータ型で INTO テーブルを作成します。INTO テーブルが既に存在していてはいけません。以下の例に示すように、この INTO テーブルは、永続テーブルでも一時テーブルでもかまいません。

TSQL

```
SELECT Name INTO Sample.NamesA_G FROM Sample.Person WHERE name LIKE '[A-G]%'
```

TSQL

```
SELECT Name INTO #MyTemp FROM Sample.Person WHERE name LIKE '[A-G]%'
SELECT * FROM #MyTemp
```

以下の例に示すように、フィールドのエイリアスを使用して、INTO テーブルのフィールドに異なる名前を指定できます。

TSQL

```
SELECT Name AS Surname INTO Sample.NamesA_G FROM Sample.Person WHERE name LIKE '[A-G]%'
```

INTO 節には、オプションの IDENTITY フィールド定義を含めることができます。これにより、指定したフィールドが、INTO 節によって作成されるテーブルに IDENTITY フィールド (精度 n) として追加されます。

SELECT がサブクエリである場合、または UNION の一部である場合、INTO 節は使用できません。

FROM 節は必須ではありません。FROM 節なしの SELECT を使用して、以下のようにローカル変数に値を代入できます。

TSQL

```
DECLARE @myvar INT
SELECT @myvar=1234
PRINT @myvar
```

FROM 節では、以下のいずれかの構文形式を使用したテーブル・ヒントがサポートされます。

```
FROM tablename (INDEX=indexname)
FROM tablename INDEX (indexname)
```

テーブル・ヒントの前に WITH キーワードを付けたり、括弧で囲むこともできます。テーブル・ヒントのリストはコンマか空白のスペースで区切られます。次のテーブル・ヒントは解析されますが無視されます。FASTFIRSTROW、HOLDINDEX、NOLOCK、PAGLOCK、READCOMMITTED、READPAST、READUNCOMMITTED、REPEATABLEREAD、ROWLOCK、SERIALIZABLE、SHARED、TABLOCK、TABLOCKX、UPDLOCK、XLOCK。

最適化ヒントをコンマ区切りリストとして SELECT FROM 節に指定できます。これらのヒントは、コメント内で以下の固有の構文を使用して指定します。

```
/* IRIS_SELECTFROM_HINT: hint,hint2 */
```

hint には、%ALLINDEX、%FIRSTTABLE tablename、%FULL、%INORDER、%IGNOREINDICES、%NOFLATTEN、%NOMERGE、%NOSVSO、%NOTOPOPT、%NOUNIONOROPT、および %STARTTABLE を指定できます。詳細は、InterSystems SQL の ″FROM″ 節を参照してください。

WHERE 節では、AND、OR、および NOT 論理キーワードを使用できます。これにより、括弧を使用して複数の検索条件をまとめることができます。WHERE 節では、次の検索条件がサポートされています。

- 等値比較：= (等しい)、<> (等しくない)、< (より小さい)、> (より大きい)、<= (以下)、>= (以上)。
- IS NULL および IS NOT NULL 比較。
- BETWEEN 比較：`Age BETWEEN 21 AND 65` (21 および 65 は含まれる)、`Age NOT BETWEEN 21 AND 65` (21 および 65 は除外)。BETWEEN は通常、数値順に照合を行う数値の範囲に使用します。ただし、BETWEEN は、任意のデータ型の値の照合順範囲に使用できます。BETWEEN は、マッチングの対象となるフィールドと同じ照合タイプを使用します。既定では、文字列データ型の照合は大文字と小文字が区別されません。
- IN 比較：`Home_State IN ('MA','RI','CT')`。
- 引用符付き文字列として指定された、LIKE および NOT LIKE 比較。比較文字列には、_ (任意の単一の文字)、% (任意の文字列)、[abc] (項目のリストとして指定されたセット中の任意の値)、[a-c] (項目の範囲として指定されたセット中の任意の値) などのワイルドカードを指定できます。InterSystems TSQL では、^ ワイルドカードはサポートされていません。`WHERE CategoryName NOT LIKE 'D\_%' ESCAPE '\'` のように、LIKE 比較に ESCAPE 節を含めることができます。
- EXISTS 比較チェック：サブクエリが空のセットかどうかをテストするサブクエリで使用します。例えば、`SELECT Name FROM Sample.Person WHERE EXISTS (SELECT LastName FROM Sample.Employee WHERE LastName='Smith')` のように使用します。この例では、LastName='Smith' を含むレコードが Sample.Employee

に存在すれば、Sample.Person からすべての Name が返されます。存在しない場合は、Sample.Person からレコードは返されません。

- ANY および ALL 比較チェック：サブクエリおよび等値比較演算子と共に使用します。SOME キーワードは ANY の同義語です。

WHERE 節と HAVING 節の比較では、大文字と小文字は区別されません。

HAVING 節は GROUP BY 節の後に指定できます。HAVING 節は、データ・セット全体ではなく、グループに対して処理を実行できる WHERE 節に似ています。HAVING と WHERE は同じ比較を使用します。詳細は、以下の例を参照してください。

TSQL

```
SELECT Home_State, MIN(Age) AS Youngest,
  AVG(Age) AS AvgAge, MAX(Age) AS Oldest
 FROM Sample.Person
 GROUP BY Home_State
 HAVING Age < 21
 ORDER BY Youngest
```

以下のダイナミック SQL の例では、テーブル・データを選択して結果セットに入れます。

ObjectScript

```
  SET sql=7
  SET sql(1)="CREATE TABLE #mytest (MyId INT PRIMARY KEY,"
  SET sql(2)="LastName VARCHAR(20),"
  SET sql(3)="FirstName VARCHAR(20))"
  SET sql(4)="INSERT INTO #mytest(MyId,LastName,FirstName) VALUES (1224,'Smith','John')"
  SET sql(5)="INSERT INTO #mytest(MyId,LastName,FirstName) VALUES (1225,'Jones','Wilber')"
  SET sql(6)="SELECT FirstName,LastName FROM #mytest"
  SET sql(7)="DROP TABLE #mytest"
  SET statement=##class(%SQL.Statement).%New()
  SET statement.%Dialect="MSSQL"
  SET status=statement.%Prepare(.sql)
  SET result=statement.%Execute()
  DO result.%Display()
```

以下のダイナミック SQL の例では、フィールドの値を 1 つ選択してローカル変数に入れます。

ObjectScript

```
  SET sql=9
  SET sql(1)="CREATE TABLE #mytest (MyId INT PRIMARY KEY,"
  SET sql(2)="LastName VARCHAR(20),"
  SET sql(3)="FirstName VARCHAR(20))"
  SET sql(4)="INSERT INTO #mytest(MyId,LastName,FirstName) VALUES (1224,'Smith','John')"
  SET sql(5)="INSERT INTO #mytest(MyId,LastName,FirstName) VALUES (1225,'Jones','Wilber')"
  SET sql(6)="DECLARE @nam VARCHAR(20)"
  SET sql(7)="SELECT @nam=LastName FROM #mytest"
  SET sql(8)="PRINT @nam"
  SET sql(9)="DROP TABLE #mytest"
  SET statement=##class(%SQL.Statement).%New()
  SET statement.%Dialect="MSSQL"
  SET status=statement.%Prepare(.sql)
  DO statement.%Execute()
```

ORDER BY 節は昇順（ASC）や降順（DESC）を指定できます。既定は昇順です。InterSystems SQL とは異なり、ORDER BY は式に指定されるサブクエリおよびクエリで使用できます。以下はその例です。

TSQL

```
SET @var = (SELECT TOP 1 name FROM mytable ORDER BY name)
```

### 4.3.2 JOIN

JOIN (INNER JOIN と同等)、INNER JOIN、および LEFT JOIN がサポートされます。括弧は複数の JOIN の解析を合理化するために使用します。

Sybase の古い外部結合 *= および =* がサポートされます。

### 4.3.3 UNION

2 つ (またはそれ以上の) SELECT 文の結合がサポートされます。InterSystems TSQL は、UNION および UNION ALL をサポートします。UNION ALL を指定する場合、最初の SELECT だけが INTO テーブルを指定できます。この INTO テーブルには、定義されたテーブル、または SELECT フィールド・リストから生成された一時テーブルを指定できます。

### 4.3.4 FETCH カーソル

OPEN、FETCH、CLOSE、および DEALLOCATE コマンドの大部分がサポートされます。以下の機能はサポートされていません。

- `OPEN/FETCH/CLOSE @local`
- `NEXT` 以外の修飾子が続く `FETCH` (修飾子は省略可能)
- `DEALLOCATE` はサポートされますが、設計上、コードが生成されないので注意してください。

## 4.4 制御文のフロー

### 4.4.1 IF

条件が True の場合にコードのブロックを実行します。

IF コマンドは、以下に示す 4 つの構文形式でサポートされます。

IF...ELSE 構文：

```
IF condition
statement
[ELSE statement]
```

IF...THEN...ELSE 単行構文：

```
IF condition THEN statement [ELSE statement]
```

ELSEIF...END IF 構文：

```
IF condition THEN
statements
{ELSEIF condition THEN statements}
 [ELSE statements]
END IF
```

ELSE IF (SQL Anywhere) 構文：

```
IF condition THEN statement
{ELSE IF condition THEN statement}
[ELSE statement]
```

1 番目の構文形式は、TSQL 標準形式です。THEN キーワードを使用しません。空白スペースと改行は自由に使用できます。1 つの節の中で複数の statement を指定するには、BEGIN キーワードと END キーワードを使用して、文のブロックを分ける必要があります。ELSE 節はオプションです。この構文を、以下の例に示します。

ObjectScript

```
SET sql=4
SET sql(1)="DECLARE @var INT"
SET sql(2)="SET @var=RAND()"
SET sql(3)="IF @var<.5 PRINT 'The Oracle says No'"
SET sql(4)="ELSE PRINT 'The Oracle says Yes' "
SET statement=##class(%SQL.Statement).%New()
SET statement.%Dialect="MSSQL"
SET status=statement.%Prepare(.sql)
SET result=statement.%Execute()
DO result.%Display()
```

2 番目の構文形式は、単行構文です。THEN キーワードは必須です。改行の制約により `IF condition THEN statement` は、すべて同一行に置く必要があります。ただし、その行に置く必要があるのは statement の最初のキーワードのみです。それ以外では、空白スペースと改行を自由に使用できます。1 つの節の中で複数の statement を指定するには、BEGIN キーワードと END キーワードを使用して、文のブロックを分ける必要があります。ELSE 節はオプションです。この構文を、以下の例に示します。

ObjectScript

```
SET sql=3
SET sql(1)="DECLARE @var INT "
SET sql(2)="SET @var=RAND() "
SET sql(3)="IF @var<.5 THEN PRINT 'No' ELSE PRINT 'Yes' "
SET statement=##class(%SQL.Statement).%New()
SET statement.%Dialect="MSSQL"
SET status=statement.%Prepare(.sql)
SET result=statement.%Execute()
DO result.%Display()
```

3 番目の構文形式では、ELSEIF 節が使用できます。ELSEIF 節は指定しないことも、1 つまたは複数を指定することもできます。それぞれに、独自の condition テストがあります。IF 節、ELSEIF 節、ELSE 節内では、複数の文を指定できます。BEGIN キーワードと END キーワードは使用できますが、必須ではありません。改行の制約により、`IF condition THEN` と最初の statement の間に改行が必要です。それ以外では、空白スペースと改行を自由に使用できます。ELSE 節はオプションです。END IF キーワード節は必須です。この構文を、以下の例に示します。

ObjectScript

```
SET sql=14
SET sql(1)="DECLARE @var INT "
SET sql(2)="SET @var=RAND() "
SET sql(3)="IF @var<.2 THEN "
SET sql(4)="PRINT 'The Oracle' "
SET sql(5)="PRINT 'says No' "
SET sql(6)="ELSEIF @var<.4 THEN "
SET sql(7)="PRINT 'The Oracle' "
SET sql(8)="PRINT 'says Possibly' "
SET sql(9)="ELSEIF @var<.6 THEN "
SET sql(10)="PRINT 'The Oracle' "
SET sql(11)="PRINT 'says Probably' "
SET sql(12)="ELSE PRINT 'The Oracle' "
SET sql(13)="PRINT 'says Yes' "
SET sql(14)="END IF"
SET statement=##class(%SQL.Statement).%New()
SET statement.%Dialect="MSSQL"
SET status=statement.%Prepare(.sql)
SET result=statement.%Execute()
DO result.%Display()
```

4 番目の構文形式は、SQL Anywhere と互換性があります。この構文形式では、ELSE IF 節が使用できます (キーワードの間の空白に注意してください)。ELSE IF 節は指定しないことも、1 つまたは複数を指定することもできます。それぞれに、独自の condition テストがあります。1 つの節の中で複数の statement を指定するには、BEGIN キーワードと END キーワードを使用して、文のブロックを分ける必要があります。空白スペースと改行は自由に使用できます。ELSE 節はオプションです。この構文を、以下の例に示します。

ObjectScript

```
SET sql=6
SET sql(1)="DECLARE @var INT "
SET sql(2)="SET @var=RAND() "
SET sql(3)="IF @var<.2 THEN PRINT 'The Oracle says No'"
SET sql(4)="ELSE IF @var<.4 THEN PRINT 'The Oracle says Possibly'"
SET sql(5)="ELSE IF @var<.6 THEN PRINT 'The Oracle says Probably'"
SET sql(6)="ELSE PRINT 'The Oracle says Yes'"
SET statement=##class(%SQL.Statement).%New()
SET statement.%Dialect="MSSQL"
SET status=statement.%Prepare(.sql)
SET result=statement.%Execute()
DO result.%Display()
```

## 4.4.2 WHILE

条件が True の間、コードのブロックを繰り返し実行します。

```
WHILE condition BEGIN statements END
```

WHILE ループは BREAK キーワードにより終了します。

CONTINUE キーワードは直ちに WHILE ループの先頭に戻ります。

statements が複数のコマンドの場合は、BEGIN キーワードと END キーワードが必須になります。

以下の例は 4 つの結果セットを返します。これらの結果セットにはそれぞれ、レコードのペアが ID の昇順で含まれます。

```
DECLARE @n INT;
SET @n=0;
WHILE @n<8 BEGIN
      SELECT TOP 2 ID,Name FROM Sample.Person WHERE ID>@n
      SET @n=@n+2
      END;
```

## 4.4.3 CASE

指定された複数の値の最初の一致から値を返します。

```
CASE expression WHEN value THEN rtnval
[WHEN value2 THEN rtnval2] [...]
[ELSE rtndefault]
END
```

WHEN value は、単純な値である必要があります。ブーリアン式にすることはできません。

ELSE 節はオプションです。満たされる WHEN 節がなく、ELSE 節が指定されていない場合、CASE 文は expression を NULL として返します。

以下に例を示します。

SQL

```
SELECT CASE Name WHEN 'Fred Rogers' THEN 'Mr. Rogers'
                WHEN 'Fred Astare' THEN 'Ginger Rogers'
                ELSE 'Somebody Else' END
       FROM Sample.Person
```

返される値は、expression のデータ型と一致している必要はありません。

CASE は `WHEN NULL THEN rtnval` ケースを解析しますが、無視します。

## 4.4.4 GOTO とラベル

InterSystems TSQL では、GOTO コマンドとラベルをサポートします。ラベルは、後にコロン (:) が続く有効な TSQL 識別子である必要があります。ラベルへの GOTO 参照には、コロンが含まれません。

## 4.4.5 WAITFOR

特定の時間が経過するまで、または特定の時刻まで実行を遅らせる場合に使用します。

```
WAITFOR DELAY timeperiod
WAITFOR TIME clocktime
```

timeperiod は、実行を再開するまでに待機する時間を 'hh:mm[:ss[.fff]]' として表したものです。例えば、`WAITFOR DELAY '00:00:03'` は 3 秒の遅延時間を示します。`WAITFOR DELAY '00:03'` は 3 分の遅延時間を示します。`WAITFOR DELAY '00:00:00.9'` は 0.9 秒の遅延時間を示します。秒の小数部の区切り文字はコロンではなく、ピリオドです。

clocktime は、実行を再開する時刻を示します (24 時間表記の 'hh:mm[:ss[.fff]] 形式)。例えば、`WAITFOR TIME '14:35:00'` と指定すると、午後 2 時 35 分に実行が再開されます。`WAITFOR TIME '00:00:03'` と指定すると、午前 0 時 0 分 3 秒に実行が再開されます。

以下のオプションはサポートされません。

- Sybase の CHECK EVERY 節。
- Sybase の AFTER MESSAGE BREAK 節。
- MSSQL の RECEIVE 節。

# 4.5 代入文

## 4.5.1 DECLARE

ローカル変数のデータ型を宣言します。

```
DECLARE @var [AS] datatype [ = initval]
```

ローカル変数を宣言する形式のみがサポートされ、カーソル変数はサポートされません。AS キーワードはオプションです。InterSystems SQL とは異なり、ローカル変数を設定する前に、これを宣言する必要があります。

@var には、任意のローカル変数名を指定できます。Sybase のローカル変数名では、大文字と小文字が区別されます。MSSQL のローカル変数名では、大文字と小文字が区別されません。

datatype には、CHAR(12) や INT など、任意の有効なデータ型を指定できます。TEXT、NTEXT、および IMAGE データ型は許可されていません。データ型の詳細は、このドキュメントの“TSQL 構文”の章を参照してください。

オプションの initval 引数を使用すると、ローカル変数の初期値を設定できます。リテラル値に設定することも、NULL、USER、CURRENT DATE (または CURRENT_DATE)、CURRENT TIME (または CURRENT_TIME)、CURRENT TIMESTAMP (または CURRENT_TIMESTAMP)、または CURRENT_USER のいずれかに設定することもできます。DEFAULT キーワードと CURRENT_DATABASE キーワードはサポートされません。また、SET コマンドまたは SELECT コマンドを使用して、ローカル変数の値を設定することもできます。以下に例を示します。

```
DECLARE @c INT;
SELECT @c=100;
```

複数のローカル変数宣言をコンマ区切りリストとして指定することができます。それぞれの宣言に独自のデータ型と (オプションで) 独自の初期値が必要です。

```
DECLARE @a INT=1,@b INT=2,@c INT=3
```

## 4.5.2 SET

ローカル変数または環境設定に値を割り当てます。

次のようにローカル変数に値を代入します。

### TSQL

```
DECLARE @var CHAR(20)
SET @var='hello world'
```

環境設定にも使用されます。

### TSQL

```
SET option ON
```

この設定は、ストアド・プロシージャに含まれているかどうかに関係なく、解析時に即座に有効になります。別の SET コマンドによって設定が変更されるまで、設定内容は保持されます。SET をストアド・プロシージャ (SP) で指定しておき、その SP の外部や別の SP に移動しても、設定した元の値が保持されています。

以下の SET 環境設定がサポートされます。

- SET ANSI_NULLS。許可される値は、SET ANSI_NULLS ON および SET ANSI_NULLS OFF です。ANSI_NULLS OFF の場合、(a=b OR (a IS NULL) AND (b IS NULL)) の場合は、a=b は True です。システム全体の "ANSI_NULLS" TSQL 構成設定を参照してください。
- SET DATEFIRST integer は、週の最初の日と見なす曜日を指定します。許可される値は 1 ～ 7 で、1 が月曜日、7 が日曜日です。既定値は 7 です。
- SET IDENTITY_INSERT。許可される値は、SET IDENTITY_INSERT ON および SET IDENTITY_INSERT OFF です。ON の場合、INSERT 文は ID フィールドの値を指定できます。この変数は、現在のプロセスに排他的に適用され、リンク・テーブルで設定することはできません。したがって、このオプションを使用するには、SET IDENTITY_INSERT と INSERT の両方を実行するプロシージャを TSQL で定義してから、ゲートウェイ経由で InterSystems IRIS でそのプロシージャをリンクして実行する必要があります。
- SET NOCOUNT。許可される値は、SET NOCOUNT ON および SET NOCOUNT OFF です。ON に設定すると、クエリにより影響を受けた行の数を示すメッセージが抑制されます。これによりパフォーマンスを大幅に向上させることができます。
- SET QUOTED_IDENTIFIER。許可される値は、SET QUOTED_IDENTIFIER ON および SET QUOTED_IDENTIFIER OFF です。SET QUOTED_IDENTIFIER がオンの場合、二重引用符が引用符で囲まれた識別子の区切り文字として解析されます。SET QUOTED_IDENTIFIER がオフの場合、二重引用符が文字列リテラルの区切り文字として解析されます。文字列リテラルの望ましい区切り文字は一重引用符です。システム全体の "QUOTED_IDENTIFIER" TSQL 構成設定を参照してください。
- SET ROWCOUNT。整数に設定します。後続の SELECT、INSERT、UPDATE、または DELETE 文に影響を与え、影響を受ける行数を制限します。SELECT 文では、ROWCOUNT が TOP より優先されます。ROWCOUNT が TOP より小さい場合、ROWCOUNT 行数が返されます。TOP が ROWCOUNT より小さい場合、TOP 行数が返されます。ROWCOUNT は、その処理の実行中、または既定の動作に戻されるまで、設定が保持されます。既定の動作に戻すには SET ROWCOUNT 0 を指定します。小数値を指定すると、ROWCOUNT は、その値より大きい直後の整数値に設定されます。
- SET TRANSACTION ISOLATION LEVEL。以下の "トランザクション文" を参照してください。

以下の SET 環境設定は解析されますが、無視されます。

- SET TEXTSIZE integer

# 4.6 トランザクション文

InterSystems TSQL では、トランザクションをサポートしています (名前付きトランザクションも含みます)。セーブポイントはサポートされません。分散トランザクションはサポートされません。

## 4.6.1 SET TRANSACTION ISOLATION LEVEL

以下の形式のみをサポートします。

- SET TRANSACTION ISOLATION LEVEL READ COMMITTED
- SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED.

READ VERIFIED およびその他のオプションはサポートされません。

Sybase の SET TRANSACTION ISOLATION LEVEL n 整数オプション・コード (0、1、2、3) はサポートされません。

## 4.6.2 BEGIN TRANSACTION

現在のトランザクションを開始します。

```
BEGIN TRAN [name]
BEGIN TRANSACTION [name]
```

トランザクションを開始します。オプションの name 引数を使用して、名前付きトランザクション (セーブポイントとも呼ばれる) を指定できます。name 値は、リテラルで指定する必要があります。変数は使用できません。

複数の BEGIN TRANSACTION 文を発行して、入れ子になった複数のトランザクションを作成できます。@@trancount 特殊変数で、現在のトランザクション・レベルを判別できます。各トランザクション・レベルは、COMMIT 文または ROLLBACK 文で解決する必要があります。

注釈　明示的なトランザクション内にあるデータ管理言語 (DML) 文は、スキーマ検索パスを使用して未修飾テーブル名を解決することはできません。

## 4.6.3 COMMIT TRANSACTION

現在のトランザクションをコミットします。

```
COMMIT
COMMIT TRAN
COMMIT TRANSACTION
COMMIT WORK
```

この 4 つの構文形式は機能的に同じです。COMMIT キーワードは、以下に示すように、これらの任意の構文形式を指します。COMMIT 文は、現在のトランザクション中に完了したすべての処理をコミットし、トランザクション・レベル・カウンタをリセットして、設定されたすべてのロックを解除します。これによりトランザクションが完了します。コミットされた処理はロールバックできません。

複数の BEGIN TRANSACTION 文で入れ子になった複数のトランザクションを作成した場合、COMMIT は、現在の入れ子になったトランザクションを完了します。トランザクションは、トランザクションの開始を表す BEGIN TRANSACTION 文を含む操作として定義されます。COMMIT を実行すると、トランザクション・レベル・カウンタが、そのトランザクションを初期化した BEGIN TRANSACTION 文の直前の状態にリストアされます。@@trancount 特殊変数で、現在のトランザクション・レベルを判別できます。

COMMIT で名前付きトランザクションを指定することはできません。COMMIT 文の一部としてトランザクション名を指定した場合、この名前の存在が解析され、エラーは発行されませんが、トランザクション名は検証されず、無視されます。

トランザクション内でないときに COMMIT が発行された場合、Sybase ではいずれの操作も実行されず、エラーも発行されません。

## 4.6.4 ROLLBACK TRANSACTION

指定したトランザクションまたはすべての現在のトランザクションをロールバックします。

```
ROLLBACK [name]
ROLLBACK TRAN [name]
ROLLBACK TRANSACTION [name]
ROLLBACK WORK [name]
```

この 4 つの構文形式は機能的に同じです。ROLLBACK キーワードは、以下に示すように、これらの任意の構文形式を指します。オプションの name 引数では、BEGIN TRANSACTION name 文で指定された名前付きトランザクションを指定します。name 値は、リテラルで指定する必要があります。変数は使用できません。

ROLLBACK は、トランザクションをロール・バックし、実行したけれどもコミットされていない作業を元に戻し、トランザクション・レベル・カウンタをデクリメントしてロックを解除します。これを使用すると、以前の一貫性のある状態にデータベースをリストアすることができます。

- ROLLBACK は現在のトランザクション（または入れ子になった一連のトランザクション）の間に実行されたすべての作業をロールバックし、トランザクション・レベル・カウンタをゼロにリセットして、すべてのロックを解除します。これにより、データベースは、入れ子になった最も外側のトランザクションが開始される前の状態にリストアされます。
- ROLLBACK name は、指定された名前付きトランザクション（セーブポイント）以降に実行されたすべての作業をロール・バックし、元に戻したセーブポイントの数だけトランザクション・レベル・カウンタをデクリメントします。すべてのセーブポイントがロール・バックまたはコミットされ、トランザクション・レベル・カウンタがゼロにリセットされると、トランザクションは完了します。名前付きトランザクションが存在しない場合、または既にロール・バックされている場合、ROLLBACK は現在のトランザクション全体をロール・バックします。

トランザクション内でないときに ROLLBACK が発行された場合、Sybase ではいずれの操作も実行されず、エラーも発行されません。

### 4.6.5 SAVE TRANSACTION

InterSystems TSQL では、SAVE TRANSACTION [savepoint-name] 文は解析されますが無視されます。指定によって処理が実行されるわけではありません。

### 4.6.6 LOCK TABLE

現在のユーザがテーブルをロックできます。

```
LOCK TABLE tablename IN {SHARE | EXCLUSIVE} MODE [WAIT  numsecs | NOWAIT]
```

LOCK TABLE 文は、指定されたテーブル内のすべてのレコードをロックします。テーブルは、SHARE MODE または EXCLUSIVE MODE でロックできます。オプションの WAIT 節は、テーブル・ロックが取得されるまで待つ秒数を指定します。LOCK TABLE 文は、現在のユーザが指定されたテーブルに対してそれまで保持していたロックを直ちに解放します。

LOCK TABLE は、トランザクション内でのみ意味を持ちます。これは、現在のトランザクションが継続している間はテーブルをロックします。トランザクション内ではない場合、LOCK TABLE は何の処理も実行しません。

"テーブル参照" の説明に従って tablename を指定します。LOCK TABLE では、1 つのテーブルのロックがサポートされます。複数のテーブルのロックはサポートされません。

LOCK TABLE では、SHARE モードと EXCLUSIVE モードがサポートされます。WRITE モードはサポートされません。

LOCK TABLE では、WITH HOLD 節はサポートされません。

WAIT の時間は、秒単位の整数で指定します。LOCK TABLE では、時刻として指定された WAIT の時間はサポートされません。

## 4.7 プロシージャ文

以下の標準的な Transact-SQL 文がサポートされます。

## 4.7.1 CREATE PROCEDURE / CREATE FUNCTION

名前付きの実行可能なプロシージャを作成します。

```
CREATE PROCEDURE procname [[@var [AS] datatype [= | DEFAULT value] [,...]] [RETURNS
 datatype] [AS] code
CREATE PROC procname [[@var [AS] datatype [= | DEFAULT value] [,...]] [RETURNS
datatype] [AS] code
CREATE FUNCTION procname [[@var [AS] datatype [= | DEFAULT value] [,...]] [RETURNS
datatype] [AS] code
```

PROCEDURE または FUNCTION のいずれかの結果として単一スカラ値を返すことができます。OUTPUT パラメータと既定値もサポートされます。これらのコマンドは返り値の型を TSQL 型宣言から InterSystems IRIS 型記述子へ変換します。現在のところ、結果セットとテーブルを返すことはできません。

CREATE PROCEDURE または CREATE PROC のいずれかとしてサポートされます。CREATE FUNCTION は CREATE PROCEDURE によく似ていますが、ルーチン・タイプの引数値が "PROCEDURE" でなく "FUNCTION" です。

- CREATE FUNCTION では任意の文を使用できます。
- RETURN キーワードは CREATE PROCEDURE で使用できます。プロシージャが RETURN または RAISERROR 文を呼び出すことなく終了する場合、整数値 0 を返します。
- WITH EXECUTE キーワード節は CREATE PROCEDURE および CREATE FUNCTION で使用できます。このキーワードは RETURN キーワードの後に記述する必要があります。

CREATE PROCEDURE は仮パラメータ・リストを指定できます。仮パラメータはコンマ区切りのリストで指定します。括弧で囲むこともできます (オプション)。パラメータ変数とそのデータ型の間の AS キーワードはオプションです。必要に応じて、DEFAULT キーワードまたは = 記号を使用して、既定値を仮パラメータに割り当てることができます。実際のパラメータ値が指定されていない場合は、この既定値が使用されます。TSQL には、入力仮パラメータにキーワード・インジケータがありません。出力仮パラメータは、OUTPUT キーワードをデータ型に続けることで指定できます。その代わりに、これらの仮パラメータの前にはオプションのキーワード IN、OUT、または INOUT を指定できます。

以下の例は、2 つの仮パラメータを使用するプロシージャ AvgAge の作成方法を示しています。

TSQL

```
CREATE PROCEDURE AvgAge @min INT, @max INT
AS
BEGIN TRY
  SELECT AVG(Age) FROM Sample.Person
  WHERE Age > @min AND Age < @max
END TRY
BEGIN CATCH
  PRINT 'error!'
END CATCH
```

このプロシージャは、以下の文によって実行されます。この場合、指定された実際のパラメータ値が平均の年齢を 21 から 65 に制限します。

TSQL

```
EXEC AvgAge 20,66
```

以下の例では、除算演算の結果を返すプロシージャが作成されます。RETURNS キーワードは、返り値の小数桁数を制限しています。

```
CREATE PROCEDURE SQLUser.MyDivide @a INTEGER, @b INTEGER, OUT @rtn INTEGER RETURNS DECIMAL(2,3)
BEGIN
SET @rtn = @a / @b;
RETURN @rtn;
END
```

このプロシージャは、以下の文によって実行されます。

TSQL

```
SELECT SQLUser.MyDivide(7,3)
```

以下の例は、プロシージャ OurReply の作成方法を示しています。

TSQL

```
CREATE PROCEDURE OurReply @var CHAR(16) DEFAULT 'No thanks' AS PRINT @var
```

パラメータなしで実行すると、OurReply により既定のテキスト (“No thanks“) が出力されます。パラメータありで実行した場合は、OurReply により、EXEC 文で指定した実際のパラメータ値が出力されます。

CREATE FUNCTION と CREATE PROCEDURE は、ストアド・プロシージャから発行できないことに注意してください。

ALTER PROCEDURE 文はサポートされません。

#### 4.7.1.1 CREATE PROCEDURE のインポート

インポートされた TSQL ソースに CREATE PROC 文が含まれる場合は、CREATE PROC ソースを含むクラス・メソッドが作成されます。このクラス・メソッドは名前がスキーマおよびプロシージャ名に基づく既存のクラスまたは新しいクラスのいずれかに配置されます。

プロシージャが既に存在する場合は、既存の実装が置き換えられます。スキーマおよびプロシージャから生成されたクラス名に一致するクラスが既に存在する場合は、既存のクラス名が使用されます (以前に TSQL ユーティリティによって生成されている場合)。一致するクラスが存在しない場合は、スキーマおよびプロシージャ名に基づく一意のクラス名が生成されます。スキーマは既定でシステム構成で定義された既定スキーマになります。プロシージャが正常に作成されると、結果のクラスがコンパイルされます。

ロギングが要求される場合は、ソース文が、それを包むクラス名、クラス・メソッド、および生成された仮引数と共に記録されます。プロセスで発生したエラーもログに記録されます。CREATE PROC 処理中にエラーが検出され、新しいクラスが生成された場合、そのクラスは削除されます。

### 4.7.2 ALTER FUNCTION

サポートされます。WITH EXECUTE キーワード節がサポートされます。

### 4.7.3 DROP FUNCTION

1 つの関数または関数のコンマ区切りリストを削除します。

```
DROP FUNCTION funcname [,funcname2 [,...] ]
```

IF EXISTS 節はサポートされません。

### 4.7.4 DROP PROCEDURE

プロシージャまたはプロシージャのコンマ区切りのリストを削除します。

```
DROP PROCEDURE [IF EXISTS] procname [,procname2 [,...] ]
DROP PROC [IF EXISTS] procname [,procname2 [,...] ]
```

オプションの IF EXISTS 節は、存在しない procname を指定した場合のエラーを抑制します。この節を指定せずに、存在しない procname を指定すると、SQLCODE -362 エラーが生成されます。DROP PROCEDURE はアトミック処理です。指定されたすべてのプロシージャが正常に削除されるか、どれも削除されないかのどちらかになります。

## 4.7.5 RETURN

クエリまたはプロシージャの実行を停止します。引数なしとするか、もしくは引数を付けることができます。引数なしの RETURN は TRY または CATCH ブロックの終了時に使用する必要があります。プロシージャから返す場合、RETURN はオプションで整数ステータス・コードを返すことができます。ステータス・コードを指定しない場合は、空の文字列 (″″) を返します。

## 4.7.6 EXECUTE

プロシージャを実行するか、TSQL コマンドの文字列を実行します。

```
EXECUTE [@rtnval = ] procname [param1 [,param2 [,...] ] ]
EXECUTE ('TSQL_commands')
```

EXEC は EXECUTE の同義語です。

- EXECUTE procname を使用して、ストアド・プロシージャを実行できます。パラメータはコンマ区切りリストとして指定します。このパラメータ・リストは括弧で囲みません。名前付きパラメータがサポートされます。

  EXECUTE procname では、`EXECUTE @rtn=Sample.MyProc param1,param2` という構文を使用して、オプションで RETURN 値を受け取ることができます。

  EXECUTE procname は CALL 文と似ています。CALL 文を使用してストアド・プロシージャを実行することもできます。CALL では、まったく異なる構文が使用されます。

  TSQL

  ```
  CREATE PROCEDURE Sample.AvgAge @min INT, @max INT
    AS
    SELECT Name,Age,AVG(Age) FROM Sample.Person
    WHERE Age > @min AND Age < @max
    RETURN 99
  ```

  TSQL

  ```
  DECLARE @rtn INT;
  EXECUTE @rtn=Sample.AvgAge 18,65
  SELECT @rtn
  ```

  指定されたプロシージャが存在しない場合、SQLCODE -428 エラー (ストアド・プロシージャが見つかりません) が発行されます。

  `WITH RECOMPILE` 節は解析されますが、無視されます。

  EXECUTE procname の機能であるプロシージャ変数とプロシージャ番号 (つまり ';n') はサポートされません。

- EXECUTE (TSQL commands) を使用して、ダイナミック SQL を実行できます。TSQL コマンドは括弧で囲みます。実行する TSQL コマンドを、一重引用符で囲んだ文字列として指定します。TSQL コマンド文字列には、改行と空白を含めることができます。ダイナミック TSQL は現在のコンテキストで実行されます。

  TSQL

  ```
  EXECUTE('SELECT TOP 4 Name,Age FROM Sample.Person')
  ```

  または

TSQL

```
DECLARE @DynTopSample VARCHAR(200)
SET @DynTopSample='SELECT TOP 4 Name,Age FROM Sample.Person'
EXECUTE (@DynTopSample)
```

以下の例は、複数の結果セットを返す EXECUTE を示しています。

TSQL

```
EXECUTE('SELECT TOP 4 Name FROM Sample.Person
         SELECT TOP 6 Age FROM Sample.Person')
```

### 4.7.7 EXECUTE IMMEDIATE

TSQL コマンドの文字列を実行します。

```
EXECUTE IMMEDIATE "TSQL_commands"
```

WITH QUOTES ON/OFF、WITH ESCAPES ON/OFF、および WITH RESULT SET ON/OFF の各ブーリアン・オプションはサポートされません。

### 4.7.8 CALL

プロシージャを実行します。

```
[@var = ] CALL procname ([param1 [,param2 [,...] ] ])
```

CALL 文は、機能的には EXECUTE procname 文と同じです。構文的には異なります。

プロシージャ・パラメータはオプションです。括弧で囲む必要があります。

オプションの @var 変数は、RETURN 文から返された値を受け取ります。ストアド・プロシージャの実行が RETURN 文で終わらない場合、@var は 0 に設定されます。

以下の例は、ストアド・プロシージャを呼び出し、2 つの入力パラメータを渡します。そして、プロシージャの RETURN 文から値を受け取ります。

```
DECLARE @rtn INT
@rtn=CALL Sample.AvgAge(18,34)
SELECT @rtn
```

## 4.8 その他の文

### 4.8.1 CREATE USER

CREATE USER は新しいユーザを作成します。

```
CREATE USER username
```

この文を実行すると、InterSystems IRIS ユーザが作成され、パスワードが指定したユーザ名に設定されます。その後、管理ポータルの **[システム管理]** インタフェースを使用して、パスワードを変更できます。CREATE USER を使用して、パスワードを明示的に設定することはできません。

ユーザ名は、大文字と小文字が区別されません。InterSystems TSQL と InterSystems SQL は、どちらも同じ定義済みユーザ名のセットを使用します。既に存在するユーザを作成しようとすると、InterSystems IRIS はエラー・メッセージを発行します。

既定では、ユーザはいずれの特権も持っていません。ユーザに特権を付与するには、GRANT コマンドを使用します。

DROP USER 文はサポートされません。

## 4.8.2 GRANT

ユーザまたは複数のユーザのリストに特権を与えます。

```
GRANT privilegelist ON tablelist TO granteelist
```

```
GRANT EXECUTE ON proclist TO granteelist
GRANT EXEC ON proclist TO granteelist
```

- privilegelist : 単一の特権または特権のコンマ区切りリスト。使用可能な特権は SELECT、INSERT、DELETE、UPDATE、REFERENCES、および ALL PRIVILEGES です。ALL は ALL PRIVILEGES の同義語です。ALTER 特権は直接的にはサポートされませんが、ALL PRIVILEGES によって付与される特権の 1 つです。
- tablelist : 単一のテーブル名 (またはビュー名) あるいはテーブル名およびビュー名のコンマ区切りリスト。"テーブル参照" の説明に従ってテーブル名を指定します。
- proclist : 単一の SQL プロシージャまたは複数の SQL プロシージャ名のコンマ区切りリスト。記述されたすべてのプロシージャが存在する必要があります。存在しないと、SQLCODE -428 エラーが返されます。
- granteelist : 単一の grantee (特権を割り当てられるユーザ) または grantee のコンマ区切りリスト。grantee のユーザ名は "PUBLIC" または "*" になります。* を指定すると、指定した特権が既存のすべてのユーザに付与されます。CREATE USER を使用して作成されたユーザは、最初はいずれの特権も持っていません。存在しないユーザを grantee のコンマ区切りリストで指定しても、何の影響もありません。GRANT はそのユーザを無視し、指定された特権をリスト内の存在するユーザに付与します。

指定したフィールドに対する特権の指定はサポートされません。

WITH GRANT OPTION 節は解析されますが、無視されます。

特権を、既にその特権を持っているユーザに付与しても何の影響もなく、エラーが発行されることもありません。

## 4.8.3 REVOKE

ユーザまたは複数のユーザのリストから、付与した特権を取り消します。

```
REVOKE privilegelist ON tablelist FROM granteelist CASCADE
```

```
REVOKE EXECUTE ON proclist FROM granteelist
REVOKE EXEC ON proclist FROM granteelist
```

特権を、その特権を持っていないユーザから取り消しても何の影響もなく、エラーが発行されることもありません。

詳細は "GRANT" を参照してください。

## 4.8.4 PRINT

指定されたテキストを現在のデバイスに表示します。

```
PRINT expression [,expression2 [,...]]
```

expression には、一重引用符で囲んだリテラル文字列、数値、あるいは文字列または数値に解決される変数または式を指定できます。任意の数の式をコンマで区切って指定できます。

PRINT では、Sybase の arg-list 構文はサポートされません。expression 文字列内の %3! のようなプレースホルダは置換されず、リテラルとして表示されます。

## 4.8.5 RAISERROR

```
RAISERROR errnum 'message'
RAISERROR(error,severity,state,arg) WITH LOG
```

構文形式 (括弧が付く場合と付かない場合) の両方がサポートされます。RAISERROR と RAISEERROR の両方のスペルがサポートされ、同義として扱われます。RAISERROR は、@@ERROR の値を指定のエラー番号およびエラー・メッセージに設定し、%SYSTEM.Error.FromXSQL() メソッドを呼び出します。

Sybase 互換構文 (括弧なし) では、errnum エラー番号が必要です。その他の引数はオプションです。

### TSQL

```
RAISERROR 123 'this is a big error'
PRINT @@ERROR
```

RAISERROR コマンドはエラー状態を起こします。このエラーの検出は、ユーザ・コードに任されています。ただし、TRY ブロックの本文に RAISERROR を配置すると、制御は CATCH ブロックのペアに移ります。CATCHブロックに RAISERROR を配置すると、制御は外側の CATCH ブロック (存在する場合) またはプロシージャの出口に移ります。RAISERROR は、プロシージャの外部の例外をトリガしません。エラーのチェックは呼び出し側に任されます。

AFTER 文レベルのトリガにより RAISEERROR を実行すると、返される %msg 値に、errnum および message の値が、コンマ区切りのメッセージ文字列コンポーネントとして格納されます (%msg="errnum,message")。

Microsoft 互換構文 (括弧付き) では、error が必要です (エラー番号でも引用符付きエラー・メッセージでもかまいません)。エラー番号は、特に指定しない場合、既定で 50000 に設定されます。オプションの severity 引数および state 引数には、整数値を指定します。

### TSQL

```
RAISERROR('this is a big error',4,1) WITH LOG
PRINT @@ERROR
```

## 4.8.6 UPDATE STATISTICS

指定テーブルのクエリ・アクセスを最適化します。指定するテーブルは、標準テーブルでも # 一時テーブルでもかまいません (詳細は、"CREATE TABLE" を参照してください)。InterSystems IRIS では最適化のために、指定されたテーブル名の引数が $SYSTEM.SQL.Stats.Table.GatherTableStats() メソッドへ渡されます。UPDATE STATISTICS は GatherTableStats() を呼び出します。他の UPDATE STATISTICS 構文はすべて、互換性維持のためだけに解析されますが、無視されます。バッチまたはストアド・プロシージャでは、指定されたテーブルの最初の UPDATE STATISTICS 文のみがテーブル・チューニングの呼び出しを生成します。詳細は、"SQL 最適化ガイド" の "テーブルのチューニング" を参照してください。

INSERT、UPDATE、または DELETE の @@ROWCOUNT が 100,000 を超える場合、そのコマンドを実行すると自動的にテーブルの UPDATE STATISTICS (テーブル・チューニング) が呼び出されます。

TSQL の TRACE 構成オプションが設定されている場合、チューニングされたテーブルのレコードがトレース・ログ・ファイルに含まれます。

### 4.8.7 USE データベース

サポートされます。また、拡張機能 `USE NONE` を使用して、データベースなしを選択することもできます。作成時に有効となり、変換オブジェクトが (シェルやバッチのロードなどに) 存在する限り保持されます。

## 4.9 InterSystems 拡張機能

TSQL では、Transact-SQL 向けの多くの InterSystems 拡張機能をサポートします。移植可能なコードにインターシステムズ独自の文を記述しても問題がないようにするため、InterSystems TSQL では特殊な形式の単独行のコメントをサポートしています。これは、2 つのハイフンと垂直バーで表記します。この演算子は、Transact-SQL 実装ではコメントとして解析されますが、InterSystems TSQL では実行可能な文として解析されます。詳細は、このドキュメントの "TSQL 構文" の章の "コメント" セクションを参照してください。

TSQL には、次の InterSystems 拡張機能が含まれています。

### 4.9.1 OBJECTSCRIPT

この拡張機能により、コンパイル出力に ObjectScript コードまたは InterSystems SQL コードを含めることができます。1 行以上のインターシステムズのコードを中括弧で囲みます。

TSQL は InterSystems SQL の %STARTSWITH 述語をサポートしないので、以下のダイナミック SQL の例では OBJECTSCRIPT を使用しています。

**TSQL**

```
  SET myquery = "OBJECTSCRIPT {SELECT Name FROM Sample.Person WHERE Name %STARTSWITH 'A'}"
  SET tStatement = ##class(%SQL.Statement).%New(,,"Sybase")
  WRITE "language mode set to ",tStatement.%Dialect,!
  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"
```

以下のダイナミック SQL の例では、OBJECTSCRIPT を使用して TSQL ルーチンに ObjectScript コードを含めます。

**TSQL**

```
  SET newtbl=2
  SET newtbl(1)="CREATE TABLE Sample.MyTest(Name VARCHAR(40),Age INTEGER)"
  SET newtbl(2)="OBJECTSCRIPT {DO $SYSTEM.SQL.Stats.Table.GatherTableStats(""Sample.MyTest"") WRITE
""TuneTable Done"",!}"
  SET tStatement = ##class(%SQL.Statement).%New(,,"Sybase")
  WRITE "language mode set to ",tStatement.%Dialect,!
  SET qStatus = tStatement.%Prepare(.newtbl)
   IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
  SET rset = tStatement.%Execute()
  DO rset.%Display()
  WRITE !,"End of data"
```

上記の例では、WRITE コマンドで新しい行 (,!) を指定しています。OBJECTSCRIPT 拡張機能では、実行後に新しい行が発行されないため、ここで指定しておく必要があります。

## 4.9.2 IMPORTASQUERY

この拡張機能では、ストアド・プロシージャをクラス・メソッドではなくクエリとして強制的にインポートします。この機能は、EXEC 文のみを含むストアド・プロシージャで使用すると便利です。InterSystems IRIS では、そのようなストアド・プロシージャがクエリかどうかをインポート時に判断できなくなるためです。

# 5

# TSQL 設定

設定は、コンパイラとカラー表示機能の動作を調整するために使用します。TSQL 構成オプションは、標準の InterSystems IRIS 構成の一部です。

InterSystems IRIS® データ・プラットフォームでは、以下の TSQL 設定がサポートされています。

- DIALECT
- ANSI_NULLS
- CASEINSCOMPARE (文字列比較で大文字と小文字を区別しない。)
- QUOTED_IDENTIFIER
- 等値リテラル置換
- TRACE

これらの値は、対応する ^%SYS("tsql","SET",...) グローバル配列値の設定に使用します。

InterSystems IRIS の管理ポータルや、**%SYSTEM.TSQL** Get および Set クラス・メソッドを使用して、これらの設定を表示および変更できます。

- InterSystems IRIS 管理ポータルに移動します。**[システム管理]**、**[構成]**、**[SQL およびオブジェクトの設定]**、**[TSQL 互換性]** の順に移動します。ここでは、**DIALECT** (Sybase または MSSQL。既定では Sybase) を指定したり、**ANSI_NULLS**、**CASEINSCOMPARE** および **QUOTED_IDENTIFIER** をオンまたはオフに設定したりできます。

  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、[ ] > [ ] > [TSQL ] ( )* のように表示されます。構成の変更を有効にするには、**[保存]** ボタンをクリックする必要があります。

- $SYSTEM.TSQL.CurrentSettings() メソッドを呼び出して次の設定を表示します。

  ObjectScript

  ```
  DO ##class(%SYSTEM.TSQL).CurrentSettings()
  ```

  **%SYSTEM.TSQL** クラス・メソッドを使用して、これらの設定を取得または設定できます。これらのメソッドは言語文字列を取り、現在の言語と指定された設定の両方を変更します。TSQL 言語ごとに個別の設定はありません。例えば、CaseInsCompare を変更すると、Sybase および MSSQL の両方に対する構成設定が変更されます。

InterSystems IRIS の構成設定を変更して、TSQL との互換性やパフォーマンスを向上させることもできます。この章で説明する構成可能なオプションは以下のとおりです。

- クエリキャッシュのソースの保存
- データの照合と文字列の切り捨て

- タイムスタンプのデータ型と時刻精度
- 一時データベースの設定

# 5.1 DIALECT

DIALECT 構成オプションでは、Transact-SQL 言語を選択できます。選択可能なオプションは Sybase および MSSQL です。既定値は Sybase です。現在の設定を返すには、$SYSTEM.TSQL.GetDialect() を使用します。このオプションをシステム全体で設定するには、InterSystems IRIS の管理ポータルまたは $SYSTEM.TSQL.SetDialect() メソッドを使用します。

ObjectScript

```
WRITE ##class(%SYSTEM.TSQL).SetDialect("Sybase")
```

このメソッドは、以前の言語設定を返します。

DIALECT=MSSQL の場合、DECLARE 文はホスト変数値を結合します。

DIALECT=Sybase の場合、ホスト変数値は更新され、各カーソルを OPEN にします。

注釈　以下のように InterSystems SQL の既定値を上書きすることにより、Transact-SQL のソース・コードを処理するように InterSystems SQL を設定することもできます。

InterSystems ダイナミック SQL で Transact-SQL 言語を設定する

管理ポータルの SQL インタフェースで Transact-SQL 言語を設定する

InterSystems SQL シェルで Transact-SQL 言語を設定する

JDBC で Transact-SQL 言語を設定する

# 5.2 ANSI_NULLS

ANSI_NULLS 構成オプションを使用すると、NULL 値に対する比較で True と False のどちらが返されるかを指定できます。既定値は OFF です。

- ON：NULL 値に対する比較はすべて不明と評価されます。例えば、Age が NULL の場合でも、Age = NULL は False を返します。NULL は不明になります。そのため、NULL=NULL の指定は False/不明になります。
- OFF：それぞれの行について、フィールドに値が含まれていない場合、フィールド値と NULL の比較は True と評価されます。例えば、Age に値が含まれていないそれぞれの行について、Age = NULL は True を返します。ただし、ANSI_NULLS OFF を使用して、2 つの異なるフィールドの NULL 値を比較することはできません。値が含まれていない 2 つのフィールドの比較は常に False です。例えば、Age = DateOfBirth と Age != DateOfBirth は、両方のフィールドに値が含まれていない場合はどちらも False を返します。

以下のように、%SYSTEM.TSQL クラス・メソッドを使用するか、または TSQLAnsiNulls プロパティから、現在の ANSI_NULLS 設定を確認できます。

ObjectScript

```
SET context=##class(%SYSTEM.Context.SQL).%New()
WRITE "ANSI_NULLS is = ",context.TSQLAnsiNulls
```

現在の設定を返すには、$SYSTEM.TSQL.GetAnsiNulls() を使用します。このメソッドは、現在の既定の言語と現在の ANSI_NULLS 設定の両方をコンマ区切りの文字列 (例 : MSSQL, ON) として返します。

ANSI_NULLS をシステム全体で有効 (ON) または無効 (OFF) にするには、InterSystems IRIS の管理ポータルまたは $SYSTEM.TSQL.SetAnsiNulls() メソッドを使用します。

ObjectScript

```
WRITE ##class(%SYSTEM.TSQL).SetAnsiNulls("Sybase","OFF")
```

このメソッドは既定の言語と ANSI_NULLS 設定の両方を設定し、以前の設定をコンマ区切りの文字列 (例 : MSSQL, ON) として返します。

# 5.3 CASEINSCOMPARE

CASEINSCOMPARE 設定では、'A'='a' のように大文字と小文字を区別せずに、同様に比較します。既定値は OFF です。このオプションを ON に設定すると、比較演算子 = および <> を使用した演算では、ほとんどのコンテキストで大文字と小文字が区別されなくなります。ただし、次の場合は大文字と小文字が区別されます。

- JOIN で比較条件が ON のとき。
- いずれかのオペランドがサブクエリのとき。

InterSystems SQL では上記の場合に %SQLUPPER 演算子を受け付けないため、これらは例外になっています。

以下のように、%SYSTEM.TSQL クラス・メソッドを使用するか、または TSQLCaseInsCompare プロパティから、現在の CASEINSCOMPARE 設定を確認できます。

ObjectScript

```
SET context=##class(%SYSTEM.Context.SQL).%New()
WRITE "ANSI_NULLS is = ",context.TSQLCaseInsCompare
```

現在の設定を返すには、$SYSTEM.TSQL.GetCaseInsCompare() を使用します。CASEINSCOMPARE をシステム全体で有効 (ON) または無効 (OFF) にするには、InterSystems IRIS の管理ポータルまたは $SYSTEM.TSQL.SetCaseInsCompare() メソッドを使用します。

ObjectScript

```
WRITE ##class(%SYSTEM.TSQL).SetCaseInsCompare("Sybase","OFF")
```

このメソッドは、以前の CASEINSCOMPARE 設定を返します。

# 5.4 QUOTED_IDENTIFIER

QUOTED_IDENTIFIER 構成オプションにより、引用符で囲まれた識別子をサポートするかどうかを選択できます。既定はオフ (サポートしない) です。このオプションは、InterSystems IRIS 管理ポータルを使用して設定されます。QUOTED_IDENTIFIER がオンの場合、二重引用符が識別子の区切り文字として解析されます。QUOTED_IDENTIFIER がオフの場合、二重引用符が文字列リテラルの代替の区切り文字として解析されます。文字列リテラルの望ましい区切り文字は一重引用符です。

以下のように、%SYSTEM.TSQL クラス・メソッドを使用するか、または TSQLQuotedIdentifier プロパティから、現在の QUOTED_IDENTIFIER 設定を確認できます。

ObjectScript

```
SET context=##class(%SYSTEM.Context.SQL).%New()
WRITE "ANSI_NULLS is = ",context.TSQLQuotedIdentifier
```

現在の設定を返すには、$SYSTEM.TSQL.GetQuotedIdentifier() を使用します。QUOTED_IDENTIFIER をシステム全体で有効 (ON) または無効 (OFF) にするには、InterSystems IRIS の管理ポータルまたは $SYSTEM.TSQL.SetQuotedIdentifier() メソッドを使用します。

ObjectScript

```
WRITE ##class(%SYSTEM.TSQL).SetQuotedIdentifier("Sybase","OFF")
```

このメソッドは、以前の QUOTED_IDENTIFIER 設定を返します。

# 5.5 等値リテラル置換

等値リテラル置換構成オプションは、メソッドを使用して設定します。管理ポータルから設定することはできません。これは、TSQL コンパイラの動作を制御します。現在の設定を返すには、$SYSTEM.TSQL.GetEqualLiteralReplacement() を使用します。既定値は ON です。

等値リテラル置換をシステム全体で有効 (ON) または無効 (OFF) にするには、$SYSTEM.TSQL.SetEqualLiteralReplacement() メソッドを使用します。

ObjectScript

```
WRITE ##class(%SYSTEM.TSQL).SetEqualLiteralReplacement("Sybase","OFF")
```

SetEqualLiteralReplacement("Sybase","OFF") を設定すると、WHERE 節の等号 (=) 述語または IN(...) 述語を含む TSQL クエリは、等号の左側や右側のリテラル値、または IN 述語内のリテラル値についてリテラル置換を実行しません。これは、異常値があるフィールドが条件に含まれている場合にクエリ・オプティマイザがより効率的なプランを選択するうえで役立つことがあります。

# 5.6 TRACE

TRACE 構成オプションでは、TSQL プロシージャの実行ログ・ファイルを作成します。TRACE をアクティブにして TSQL ストアド・プロシージャ (メソッドまたはシステム・ストアド・プロシージャ) をコンパイルした場合、TSQL プロシージャを実行すると、アクティブな tsql ログ・ファイルにトレース・メッセージが記録されます。

TSQL プロシージャの実行元のプロセスごとに別個の tsql トレース・ログ・ファイルが作成されます。トレースはシステム全体でアクティブになります。トレース・ログ・ファイルはネームスペース固有です。

TRACE の設定に管理ポータルは使用しません。このオプションは、$SYSTEM.TSQL.SetTrace() メソッドを使用してシステム全体について設定できます。言語は指定されません。

ObjectScript

```
WRITE ##class(%SYSTEM.TSQL).SetTrace("ON")
```

現在の設定を返すには、$SYSTEM.TSQL.GetTrace() を使用します。

以下の ObjectScript コマンドを使用すると、システム全体で TRACE をアクティブ (=1) または非アクティブ (=0) に設定できます。

ObjectScript

```
SET ^%SYS("tsql","TRACE")=1
```

現在のトレース設定を返すには、以下のコマンドを使用します。

ObjectScript

```
WRITE ^%SYS("tsql","TRACE")
```

TRACE ログ・ファイルには、各操作のタイムスタンプ、各操作の経過時間、グローバル参照の数、および %ROWCOUNT (該当する場合) が記録されます。TRUNCATE TABLE は常に %ROWCOUNT -1 を返します。操作にシャード・テーブルが含まれる場合、グローバル参照の数は、プロシージャが実行されたプロセスのみが対象です。その他のシャードに送信された作業は、グローバル参照の数に含まれません。

TRACE ログ・ファイルは、内部一時テーブル名を使用する一時テーブルを表します。ユーザが指定した対応する #TempTable 名が /* mytemptable */ コメントに表示されます。

TRACE ログ・ファイルは、InterSystems IRIS インスタンスの mgr ディレクトリにある、現在のネームスペースを表すサブディレクトリ内に作成されます。ファイルには現在のプロセス番号を使用して名前が付けられます。例えば、IRIS/mgr/user/tsql16392.log のようになります。以下に一般的な TRACE ログ・ファイルを示します。

```
IRIS TSQL Log, created 07/06/2020 13:44:41.020101 by process 16392
   Version: IRIS for Windows (x86-64) 2020.2 (Build 211U) Fri Jun 26 2020 13:19:52 EDT
     User: glenn

   07/06/2020 13:44:41.020488
PREPARE EXECUTEPROC: Sample.StuffProc
   07/06/2020 15:02:44.270773
PREPARE EXECUTEPROC: sp_addtype
   07/06/2020 15:04:50.625108
PREPARE EXECUTEPROC: sp_addtype

Log restarted: 07/06/2020 15:15:42
   07/06/2020 15:15:42.623033
CALLSP:: CreateMyTableProc()
   07/06/2020 15:15:42.624807
EXECUTE CREATE TABLE Sample.MyTable (Name SHORTSTR, BigName MIDSTR):
       Elapsed time = .313114s        # Global Refs = 17,446
RETURN:: CreateMyTable with value = 0
   07/06/2020 15:15:42.938084
   context object: 154@%Library.ProcedureContext

       Context status is OK

   07/06/2020 15:23:42.171761
CALLSP:: CreateMyTable()
   07/06/2020 15:23:42.174175
EXECUTE CREATE TABLE Sample.MyTable (Name SHORTSTR, BigName MIDSTR):
       ERROR: -201 Table 'Sample.MyTable' already exists
       SQLCODE = -400        Elapsed time = .002356s        # Global Refs = 151
RETURN:: CreateMyTable with value = 0
   07/06/2020 15:23:42.176979
   context object: 485@%Library.ProcedureContext


           Error:
ERROR #5540: SQLCODE: -201 Message: Table 'Sample.MyTable' already exists
```

## 5.6.1 クエリ・キャッシュのソース

デバッグに役立つように、クエリ・キャッシュのソース・コードと生成されたクエリ・キャッシュを保持することも推奨します。このオプションは次のように構成できます。

ObjectScript

```
SET status=$SYSTEM.SQL.Util.SetOption("CachedQuerySaveSource",1,.oldval)
```

# 5.7 データの照合と文字列の切り捨て

InterSystems SQL の既定の照合は SQLUPPER です。これは、TSQL のネイティブの照合順に最適には一致しません。

Sybase では、複数のソート順がサポートされています。既定値はバイナリです。この既定値の説明は次のとおりです。「すべてのデータをその文字セットの数値のバイト値に従ってソートします。バイナリ順では、ASCII の大文字はすべて小文字より先にソートされます。アクセント付き文字または表意 (マルチバイト) 文字はそれぞれの標準の順序でソートされ、これは任意の順序である可能性があります。すべての文字セットでバイナリ順が既定値になっています」

このバイナリ照合順は InterSystems IRIS の SQLSTRING の照合順に最適に一致します。したがって、SQL の照合順を SQLUPPER から SQLSTRING に変更するのが最適な互換性オプションであると考えられますが、すべての文字で正しいとは保証されません。

- 現在のネームスペースの既定の照合を設定するには、`SET status=$$SetEnvironment^%apiOBJ("COLLATION","%Library.String","SQLSTRING")` とします。
- システム全体の既定の照合を設定するには、`SET ^%oddENV("collation","%Library.String")="SQLSTRING"` とします。

SQLSTRING 照合では、データがデータベースにロードされる際に文字列内の末尾の空白スペースが保持されます。このような文字列をストアド・プロシージャで処理する場合、末尾に空白スペースがあることでエラーが発生する可能性があります。したがって、読み取り時に文字列を切り捨てるように文字列データ型を構成することをお勧めします。文字列データ型は次のように構成できます。

ObjectScript

```
SET status=$$SetSysDatatypes^%SYS.CONFIG(,"CHAR","%Library.String(MAXLEN=1,TRUNCATE=1)")
SET status=$$SetSysDatatypes^%SYS.CONFIG(,"CHAR(%1)","%Library.String(MAXLEN=%1,TRUNCATE=1)")
SET status=$$SetSysDatatypes^%SYS.CONFIG(,"VARCHAR","%Library.String(MAXLEN=1,TRUNCATE=1)")
SET status=$$SetSysDatatypes^%SYS.CONFIG(,"VARCHAR(%1)","%Library.String(MAXLEN=%1,TRUNCATE=1)")
SET status=$$SetSysDatatypes^%SYS.CONFIG(,"VARCHAR(%1,%2)","%Library.String(MAXLEN=%1,TRUNCATE=1)")
```

# 5.8 タイムスタンプと時刻精度

InterSystems SQL の既定のタイムスタンプ・データ型は %TimeStamp です。TSQL データベースに日時フィールドが多数ある場合 (金融データベースによく見られます)、%PosixTime を既定のタイムスタンプ・データ型として使用すると、必要なディスク容量を減らすことができるため、読み取りと書き込みの両方で行アクセス速度を向上させることができます。タイムスタンプ・データ型は次のように構成できます。

ObjectScript

```
SET status=$$SetSysDatatypes^%SYS.CONFIG(,"DATETIME","%Library.PosixTime")
SET status=$$SetSysDatatypes^%SYS.CONFIG(,"DATETIME2","%Library.PosixTime")
SET status=$$SetSysDatatypes^%SYS.CONFIG(,"TIMESTAMP","%Library.PosixTime")
```

また、以下の例に示すように、既定の時刻精度を、任意の小数点以下の有効桁数に構成することもできます。

ObjectScript

```
SET status=$SYSTEM.SQL.Util.SetOption("DefaultTimePrecision",6,.oldval)
```

# 5.9 一時データベースの設定

パフォーマンスを向上させるために、一時データベースと作業データベースの設定を変更して標準のチェックを解除すると効果的な場合があります。これらの変更は一部の TSQL 環境には適さない場合があります。作業テーブルは本質的に一時的ですが、あるストアド・プロシージャで使われた後に別のストアド・プロシージャで使われて存在し続ける可能性があることに注意してください。

# 6

# TSQL 関数

## 6.1 サポートしている関数

InterSystems IRIS® データ・プラットフォームには、以下の TSQL 関数が実装されています。

### 6.1.1 ABS

```
ABS(num)
```

num の絶対値を返します。したがって、123.99 も -123.99 も 123.99 を返します。

### 6.1.2 ACOS

```
ACOS(float)
```

アーク・コサイン：コサインが float である角度をラジアン単位で返します。したがって、1 は 0 を返します。

### 6.1.3 ASCII

```
ASCII(char)
```

文字列 char の最初の文字に対応する整数値を返します。したがって、`ASCII('A')` は 65 を返します。

ASCII は UNICODE と機能的に同じです。この逆の関数は CHAR になります。

### 6.1.4 ASIN

```
ASIN(float)
```

アーク・サイン：サインが float である角度をラジアン単位で返します。したがって、1 は 1.570796326... を返します。

## 6.1.5 ATAN

```
ATAN(float)
```

アーク・タンジェント：タンジェントが float である角度をラジアン単位で返します。したがって、1 は 0.785398163... を返します。

## 6.1.6 AVG

```
AVG(numfield)
AVG(DISTINCT numfield)
```

集約関数：numfield 列の平均値を返すためにクエリで使用されます。例えば、`SELECT AVG(Age) FROM Sample.Person` のように使用します。AVG(DISTINCT numfield) では、numfield 列内の一意の値の数の平均値が計算されます。NULL フィールドは無視されます。

## 6.1.7 CAST

```
CAST(expression AS datatype)
```

指定した datatype に変換した expression を返します。CAST はサポートされるデータ型で使用できます。詳細は、"InterSystems SQL リファレンス" の "データ型" を参照してください。CAST では、sp_addtype ストアド・プロシージャを使用して作成されたユーザ定義データ型がサポートされます。

expression が '2004-11-23' のような日付の文字列で、datatype が TIMESTAMP または DATETIME の場合、'00:00:00' の時間値が返されます。

expression が '1:35PM' のような時間の文字列で、datatype が TIMESTAMP または DATETIME の場合、時刻は 24 時間形式に変換され、AM や PM の接尾語は削除され、秒単位の部分はゼロで埋められます。'1900-01-01' が既定の日付値として返されます。したがって、'1:35PM' は '1900-01-01 13:35:00' に変換されます。

expression が '2004-11-23' のような日付の文字列で、datatype が DATE の場合、日付は `60703` (2007 年 3 月 14 日) のような InterSystems IRIS $HOROLOG 日付形式で返されます。

InterSystems TSQL では、データ型 XML はサポートされていません。しかし、コンパイル中のエラーを生成する代わりに、SQL モードの CAST(x AS XML) が CAST(x AS VARCHAR(32767)) を生成します。プロシージャ・モードでは、CAST(x AS XML) は変換の生成は行いません。

"CONVERT" を参照してください。

## 6.1.8 CEILING

```
CEILING(num)
```

num 以上の最も近い整数が返されます。したがって、123.99 では 124、-123.99 では -123 を返します。

Sybase の CEIL 同義語はサポートされません。

## 6.1.9 CHAR

```
CHAR(num)
```

整数値 num に対応する文字を返します。したがって、`CHAR(65)` は A を返します。

CHAR は NCHAR と機能的に同じです。この逆の関数は ASCII になります。

## 6.1.10 CHAR_LENGTH / CHARACTER_LENGTH

```
CHAR_LENGTH(string)
CHARACTER_LENGTH(string)
```

string の文字数を返します。

## 6.1.11 CHARINDEX

```
CHARINDEX(seekstring,target[,startpoint])
```

target の中の seekstring の最初のオカレンスの最初の文字に相当する場所 (1 からカウントします) を返します。オプションの startpoint 整数を使用して、どこから検索を開始するかを指定できます。返り値は、startpoint に関係なく、target の最初からカウントされます。startpoint を指定しない場合および 0、1 または負の値を指定した場合、target は最初から検索されます。seekstring が見つからない場合、CHARINDEX は 0 を返します。

## 6.1.12 COALESCE

```
COALESCE(expression1,expression2,...)
```

指定された式のリストから、最初の非 NULL 式を返します。

## 6.1.13 COL_NAME

```
COL_NAME(object_id,column_id)
```

列名を返します。プロシージャ・コードまたはトリガ・コード内で使用されます。

TSQL では、この関数に対して 2 つの引数をサポートします。3 番目の引数はサポートされません。

以下の例は、Sample.Person の 4 番目の列名を返します。

ObjectScript

```
SET sql=2
SET sql(1)="SELECT 'column name'=COL_NAME(id,4) FROM Sample.Person"
SET sql(2)="WHERE id=OBJECT_ID('Sample.Person')"
SET statement=##class(%SQL.Statement).%New()
SET statement.%Dialect="MSSQL"
SET status=statement.%Prepare(.sql)
SET result=statement.%Execute()
DO result.%Display()
```

COL_NAME では、Sybase の 3 番目の引数はサポートされません。

## 6.1.14 CONVERT

```
CONVERT(datatype,expression [,style])
```

指定した datatype に変換した expression を返します。

- BIT : datatype が BIT で、expression がブーリアン値の場合、入力値が 0 以外の数字ならば結果が 1 となり、入力値が 0 ならば結果が 0 となります。また、入力値が文字列 'TRUE' (大文字と小文字の区別なし) ならば結果が

1 となり、入力値が文字列 'FALSE' (大文字と小文字の区別なし) ならば結果が 0 となります。さらに、入力値が NULL ならば結果が NULL となります。入力値がこれ以外の場合には、SQLCODE -141 エラーが生成されます。

- CHAR : CHAR の長さが指定されていない場合、30 文字の長さに変換します。
- DATETIME : datatype が日時またはタイムスタンプの場合、以下のようになります。
  - expression が '2004-11-23' のような日付の文字列の場合、'00:00:00' という時間値が指定されます。
  - expression が '1:35PM' のような時間の文字列で、datatype が日時またはタイムスタンプの場合、時刻は 24 時間形式に変換され、AM や PM の接尾語は削除され、秒単位の部分はゼロで埋められます。'1900-01-01' が既定の日付値として返されます。したがって、'1:35PM' は '1900-01-01 13:35:00' に変換されます。
  - expression が空の文字列または 1 つ以上の空白スペースの文字列である場合、CONVERT は '1900-01-01 00:00:00' を返します。

既定では、DATETIME データ型は **%Library.TimeStamp** にマップされます。CONVERT では、**%Library.PosixTime** への DATETIME のデータ型マッピングもサポートされます。

CONVERT は DATETIME2 データ型をサポートします。InterSystems IRIS は、DATETIME2 をシステム定義の DDL マッピング **%Library.TimeStamp** にマップします。このマッピングは新規インストールで提供されます。アップグレード・インストールを使用している場合は、このマッピングの作成が必要になることもあります。

CONVERT では、sp_addtype ストアド・プロシージャを使用して作成されたユーザ定義データ型がサポートされます。

オプションの style 引数を使用して、日時またはタイムスタンプの値を文字列に変換するときの日付/時刻の形式を指定できます。さまざまな style コードを指定することにより、異なる複数の形式で日付と時刻を返すことができます。使用可能な style コードは、100 から 116、120 から 123、126、130、131、136 から 140 です (対応するコード 0 から 7 および 10 から 40 は、同じ値を 2 桁の年で返します)。日時の既定の style は 0 です。

```
mon dd yyyy hh:mmAM
```

Sybase についてサポートされている日時スタイルの一部を以下に示します。

```
15 / 115 = format dd/[yy]yy/mm
16 / 116 = format mon dd yyyy HH:mm:ss
22 / 122 = format [yy]yy/mm/dd HH:mm AM (or PM)
23 / 123 = format [yy]yy-mm-ddTHH:mm:ss
36 / 136 = format hh:mm:ss.zzzzzzAM(PM)
37 / 137 = format hh:mm.ss.zzzzzz
38 / 138 = format mon dd [yy]yy hh:mm:ss.zzzzzzAM(PM)
39 / 139 = format mon dd [yy]yy HH:mm:ss.zzzzzz
40 / 140 = format yyyy-mm-dd hh:mm:ss.zzzzzz
```

20 および 21 (120 および 121) の style コードは、ODBC のタイムスタンプ形式を返します。20 は、整数秒に切り捨てます。21 は、秒の小数部を返します。

```
yyyy-mm-dd hh:mm:ss.fff
```

詳細は、"InterSystems SQL リファレンス" の InterSystems SQL "CONVERT" 関数を参照してください。機能的にはこれと同じです。

"CAST" を参照してください。

## 6.1.15 COS

```
COS(float)
```

コサイン : float で指定された角度のコサインを返します。したがって、1 は 0.540302305... を返します。

## 6.1.16 COT

```
COT(float)
```

コタンジェント：float で指定された角度のコタンジェントを返します。したがって、1 は .64209261593... を返します。

## 6.1.17 COUNT

```
COUNT(field)
COUNT(DISTINCT field)
COUNT(*)
COUNT(1)
```

集約関数：field 列の値の数を返すためにクエリで使用されます。NULL フィールドはカウントされません。例えば、`SELECT COUNT(Name) FROM Sample.Person` のように使用します。COUNT(*) および COUNT(1) は同義語で、すべての行をカウントします。COUNT(DISTINCT field) では、field 列内の一意の値の個数がカウントされます。NULL フィールドはカウントされません。

## 6.1.18 CURRENT_DATE

```
CURRENT_DATE
CURRENT DATE
```

以下の形式で現在のローカル日付を返します。

```
yyyy-mm-dd
```

2 つの構文形式 (アンダースコアありとなし) は同じです。この関数では括弧を使用しないため注意してください。

この関数は、Sybase 言語および MSSQL 言語でサポートされる SQL Anywhere との互換性のために提供されています。

## 6.1.19 CURRENT_TIME

```
CURRENT_TIME
CURRENT TIME
```

以下の形式で現在のローカル時間を返します。

```
hh:mm:ss
```

時刻は 24 時間形式で指定されます。秒の小数部は返されません。

2 つの構文形式 (アンダースコアありとなし) は同じです。この関数では括弧を使用しないため注意してください。

この関数は、Sybase 言語および MSSQL 言語でサポートされる SQL Anywhere との互換性のために提供されています。

## 6.1.20 CURRENT_TIMESTAMP

```
CURRENT_TIMESTAMP
CURRENT TIMESTAMP
```

次の形式で現在のローカルな日時を返します。

```
yyyy-mm-dd hh:mm:ss
```

時刻は 24 時間形式で指定されます。秒の小数部は返されません。

2 つの構文形式 (アンダースコアありとなし) は同じです。この関数では括弧を使用しないため注意してください。

## 6.1.21 CURRENT_USER

```
CURRENT_USER
```

現在のユーザ名を返します。

この関数では括弧を使用しないため注意してください。

## 6.1.22 DATALENGTH

```
DATALENGTH(expression)
```

expression を表現するのに使用したバイト数を示す整数を返します。したがって、'fred' は 4、+007.500 は 3 を返します。

## 6.1.23 DATEADD

```
DATEADD(code,num,date)
```

code で指定された間隔を num で指定された回数分加えることによって修正された date の値を返します。date は、さまざまな形式の日付、時刻または日付/時刻の文字列を取ることができます。以下のいずれかの code 値を指定できます。左の列は省略形、右の列は正式名称で、どちらで指定してもかまいません。

| 省略形 | 名前 |
|---|---|
| yy | Year |
| qq | Quarter |
| mm | Month |
| dy | DayofYear |
| dd | Day |
| dw、w | Weekday |
| wk | Week |
| hh | Hour |
| mi | Minute |
| ss | Second |
| ms | Millisecond |

コード値は、大文字と小文字を区別しません。Day、DayofYear、および Weekday はすべて同じ値を返します。

DATEADD で返される値には、常に次の形式で日付と時刻の両方が含められます。

```
yyyy-mm-dd hh:mm:ss.n
```

秒の小数部は、元の秒に小数部が含まれていた場合のみ返されます。

日付を指定しない場合 (つまり、date に時刻の値しか含まれていない場合)、日付は既定により 1/1/1900 となります。

date で時刻を指定しない場合、既定により 00:00:00 となります。時刻は常に 24 時間形式で返されます。

## 6.1.24 DATEDIFF

```
DATEDIFF(code,startdate,enddate)
```

startdate と enddate の間の code 間隔の数を返します。2 つの日付は、次の形式の日付、時刻または日付/時刻の文字列を取ることができます。

```
yyyy-mm-dd hh:mm:ss.n
```

以下のいずれかの code 値を指定できます。左の列は省略形、右の列は正式名称で、どちらで指定してもかまいません。

| 省略形 | 名前 |
|---|---|
| yy | Year |
| mm | Month |
| dd | Day |
| dw、w | Weekday |
| wk | Week |
| hh | Hour |
| mi | Minute |
| ss | Second |
| ms | Millisecond |

コード値は、大文字と小文字を区別しません。Day、DayofYear、および Weekday はすべて同じ値を返します。

日付を指定しない場合 (つまり、date または enddate に時刻の値しか含まれていない場合)、日付は既定により 1/1/1900 となります。

startdate や enddate で時刻を指定しない場合、既定により 00:00:00 となります。

## 6.1.25 DATENAME

```
DATENAME(code,date)
```

code で指定した日付の部分的な値を文字列として返します。date には、さまざまな形式の日付、時刻、または日付/時刻の文字列を指定できます。date に指定する文字列は引用符で囲む必要があります。code の値は引用符で囲んでも囲まなくてもかまいません。使用可能な code の値は、以下のとおりです。

| 値 | 説明 |
| --- | --- |
| yyyy, yy<br>year | 年。4 桁の年数を返します。2 桁で指定した場合、DATENAME には先頭に '19' が付加されます。 |
| qq, q<br>quarter | 四半期。1 から 4 の整数を返します。 |
| mm, m<br>month | 月。月の正式名称を返します。例えば、'December' です。 |
| dy, y<br>dayofyear | 年間通算日。1 から 366 の日付を整数で返します。 |
| dd, d<br>day | 月間通算日。1 から 31 でカウントした整数を返します。 |
| wk, ww<br>week | 通算週。1 から 53 でカウントした整数を返します。 |
| dw、w<br>weekday | 曜日。日曜日から数えた曜日の数を返します。例えば 3 は火曜日です。 |
| hh<br>hour | 時。(24 時間形式で) 1 日のうちの時刻を 0 から 23 までの整数として返します。 |
| mi, n<br>minute | 分。0 から 59 の整数を返します。 |
| ss, s<br>second | 秒。0 から 59 の整数を返します。ミリ秒を表す小数部が含まれることがあります。 |
| ms<br>millisecond | ミリ秒。秒の小数部を整数で返します。 |

コード値は、大文字と小文字を区別しません。

日付を指定しない場合、既定により 1/1/1900 となります。2 桁の年数は既定により 19xx となります。

時刻を指定しない場合、既定により 00:00:00 となります。時刻は常に 24 時間形式で返されます。秒の小数部が定義されている場合には必ず秒の小数部を含めて返されます。ミリ秒は、小数部ではなく、整数として返されます。

## 6.1.26 DATEPART

```
DATEPART(code,date)
```

code で指定した日付の部分的な値を整数として返します。date は、さまざまな形式の日付、時刻または日付/時刻の文字列を取ることができます。利用可能な code 値は DATENAME に示しています。

## 6.1.27 DAY

```
DAY(date)
```

指定した日付または日付/時刻の文字列の日付部分を返します。date は、次の ODBC タイムスタンプ形式で指定します。

```
yyyy-mm-dd hh:mm:ss.n
```

date には日付コンポーネントを含める必要があります。日付の区切り文字にはハイフン (-) を使用してください。

date は、60703 (2007 年 3 月 14 日) のように InterSystems IRIS $HOROLOG 日付形式で指定することもできます。

## 6.1.28 DB_NAME

```
DB_NAME(database-id)
```

現在のネームスペース名を返します。database-id 引数はオプションです。

## 6.1.29 DEGREES

```
DEGREES(float)
```

ラジアン単位の角度測定値を、対応する度単位の測定値に変換します。

## 6.1.30 ERROR_MESSAGE

CATCH ブロック内から呼び出されたときには、現在のエラーメッセージを返します。そうでない場合は、NULL を返します。

## 6.1.31 ERROR_NUMBER

CATCH ブロック内から呼び出されたときには、現在の SQLCODE エラーを返します。そうでない場合は、NULL を返します。

## 6.1.32 EXEC

```
EXEC(@var)
```

以下の例に示すように、実行時にダイナミック SQL を実行します。

TSQL

```
DECLARE @dyncode VARCHAR(200)
SELECT @dyncode='SELECT TOP 4 Name,Age FROM Sample.Person'
EXEC(@dyncode)
```

この動的な実行と、ストアド・プロシージャを実行する EXECUTE コマンドを比較してください。

## 6.1.33 EXP

```
EXP(num)
```

num の指数を返します。これは定数 e (2.71828182) の num 乗です。したがって、EXP(2) は 7.3890560989 を返します。

## 6.1.34 FLOOR

```
FLOOR(num)
```

num 以下の最も近い整数が返されます。したがって、123.99 では 123、-123.99 では -124 を返します。

## 6.1.35 GETDATE

```
GETDATE()
```

次の形式で現在のローカルな日時を返します。

```
yyyy-mm-dd hh:mm:ss.n
```

時刻は 24 時間形式で指定されます。秒の小数部が返されます。

## 6.1.36 GETUTCDATE

```
GETUTCDATE()
```

次の形式で現在の UTC (グリニッジ標準時間) の日時を返します。

```
yyyy-mm-dd hh:mm:ss.n
```

時刻は 24 時間形式で指定されます。秒の小数部が返されます。

## 6.1.37 HOST_NAME

```
HOST_NAME()
```

現在のホスト・システムのシステム名を返します。

## 6.1.38 INDEX_COL

```
INDEX_COL(table_name,index_id,key,[,user_id])
```

指定したテーブル内のインデックス付けされた列の名前を返します。table_name には完全修飾名を指定できます。index_id はテーブル内のインデックス数です。key はインデックス内のキーで、1 ～ sysindexes.keycnt (クラスタ化インデックスの場合) または sysindexes.keycnt+1 (非クラスタ化インデックスの場合) の値を指定します。user_id は解析されますが無視されます。

## 6.1.39 ISNULL

```
ISNULL(expr,default)
```

expr が NULL の場合、default が返されます。expr が NULL でない場合、expr が返されます。

## 6.1.40 ISNUMERIC

```
ISNUMERIC(expression)
```

expression が有効な数値の場合は 1、それ以外は 0 を返すブーリアン関数です。

指定された expression が NULL 値のフィールドである場合、ISNUMERIC は NULL を返します。

## 6.1.41 LEFT

```
LEFT(string,int)
```

string からの文字を左からカウントした int 数を返します。int が string より大きい場合、完全な文字列が返されます。"RIGHT" を参照してください。

## 6.1.42 LEN

```
LEN(string)
```

string の文字数を返します。

## 6.1.43 LOG

```
LOG(num)
```

num の自然対数を返します。したがって、LOG(2) は .69314718055 を返します。

## 6.1.44 LOG10

```
LOG10(num)
```

num の常用対数を返します。したがって、LOG10(2) は .301029995663 を返します。

## 6.1.45 LOWER

```
LOWER(string)
```

大文字をすべて小文字に変換して string を返します。"UPPER" を参照してください。

## 6.1.46 LTRIM

```
LTRIM(string)
```

string から先頭の空白を削除します。

string がすべて空白スペースで構成される場合、言語によって動作が決まります。

- Sybase：NULL を返します。
- MSSQL：空の文字列を返します。

"RTRIM" を参照してください。

## 6.1.47 MAX

```
MAX(numfield)
```

集約関数：numfield 列の最大値を返すためにクエリで使用されます。以下はその例です。

TSQL

```
SELECT MAX(Age) FROM Sample.Person
```

NULL フィールドは無視されます。

## 6.1.48 MIN

```
MIN(numfield)
```

集約関数：numfield 列の最小値を返すためにクエリで使用されます。以下はその例です。

TSQL

```
SELECT MIN(Age) FROM Sample.Person
```

NULL フィールドは無視されます。

## 6.1.49 MONTH

```
MONTH(date)
```

指定した日付または日付/時刻の文字列の月部分を返します。date は、次の ODBC タイムスタンプ形式で指定します。

```
yyyy-mm-dd hh:mm:ss.n
```

日付の区切り文字にはハイフン (-) を使用してください。それ以外の形式の日付は 0 を返します。

date は、`60703` (2007 年 3 月 14 日) のように InterSystems IRIS $HOROLOG 日付形式で指定することもできます。

## 6.1.50 NCHAR

```
NCHAR(num)
```

整数値 num に対応する文字を返します。したがって、NCHAR(65) は A を返します。

NCHAR は CHAR と機能的に同じです。この逆の関数は ASCII になります。

## 6.1.51 NEWID

```
NEWID()
```

SQL Server の UNIQUEIDENTIFIER データ型と互換性のある型の一意の値を返します。UNIQUEIDENTIFIER は、システムで生成される 16 バイトのバイナリ文字列で、グローバル一意識別子 (GUID) としても知られています。GUID は、不定期に接続されるシステム上のデータベースを同期するのに使用されます。GUID は 36 文字の文字列で、32 文字の 16 進数が、ハイフンで 5 つのグループに分割されて構成されています。InterSystems TSQL では、UNIQUEIDENTIFIER はサポートされていません。グローバル一意識別子のデータ型として VARCHAR(36) が代わりに使用されます。

NEWID 関数は引数を取りません。引数の括弧は必須です。

フィールドを定義する際は、NEWID() を使用して DEFAULT 値を指定できます。

対応する InterSystems SQL 関数は $TSQL_NEWID です。

SQL

```
SELECT $TSQL_NEWID()
```

## 6.1.52 NOW

```
NOW(*)
```

次の形式で現在のローカルな日時を返します。

```
yyyy-mm-dd hh:mm:ss
```

時刻は 24 時間形式で指定されます。秒の小数部は返されません。

括弧内にアスタリスクが必要になることに注意してください。

## 6.1.53 NULLIF

```
NULLIF(expr1,expr2)
```

expr1 が expr2 と等しい場合、NULL を返します。それ以外の場合、expr1 を返します。

## 6.1.54 OBJECT_ID

```
OBJECT_ID(objname,objtype)
```

引用符付き文字列 (必要に応じてオブジェクト・タイプ) としてオブジェクト名を取り、指定されたオブジェクトの対応するオブジェクト IDを整数として返します。利用可能な objtype の値は、RI = FOREIGN KEY 制約、K = PRIMARY KEY または UNIQUE 制約、P = ストアド・プロシージャ、S = システム・テーブル、TR = トリガ、U = ユーザ・テーブル、V = ビュー

です。objtype が省略されている場合、OBJECT_ID はすべてのオブジェクト・タイプをテストし、最初に一致したものを返します。

TSQL

```
CREATE PROCEDURE GetObjIds
AS SELECT OBJECT_ID('Sample.Person','U'),OBJECT_ID('Sample.Person_Extent','P')
GO
```

objname が存在しない場合、またはオプションの objtype が指定されていて、objname と一致しない場合は、NULL を返します。プロシージャ・コードまたはトリガ・コード内で使用されます。OBJECT_NAME の逆関数です。

## 6.1.55 OBJECT_NAME

```
OBJECT_NAME(id)
```

オブジェクト ID (整数) を取り、指定されたオブジェクトの対応するオブジェクト名を返します。id が存在しない場合、空文字列を返します。プロシージャ・コードまたはトリガ・コード内で使用されます。OBJECT_ID の逆関数です。

TSQL

```
CREATE PROCEDURE GetObjName
AS SELECT OBJECT_NAME(22)
GO
```

## 6.1.56 PATINDEX

```
PATINDEX(pattern,string)
```

string の中で最初に出現した pattern の開始位置を 1 からカウントして整数を返します。string に pattern がない場合、0 を返します。pattern は、引用符付きの文字列として指定します。比較では大文字と小文字が区別されます。pattern には以下のワイルドカード文字を含めることができます。

| 文字 | 説明 |
|---|---|
| % | ゼロ個以上の文字。例えば、'%a%' は、string の中の最初の 'a' の出現位置を返します。string の最初の文字が 'a' の場合も含まれます。 |
| _ | 単独の文字。例えば、'_l%' を指定したときに、string が 'Al'、'el'、'il' などの部分文字列で始まる場合、1 を返します。 |
| [xyz] | 指定した文字のリストの中にある任意の 1 文字。例えば、'[ai]l%' を指定したときに、string が 'el' や 'Al' ではなく 'al' や 'il' などの部分文字列で始まる場合、1 を返します。 |
| [a-z] | 指定した文字の範囲の中にある任意の 1 文字。例えば、'%s[a-z]t%' には 'sat'、'set'、'sit' などが一致します。範囲は ASCII の昇順で指定する必要があります。 |

キャレット (^) 文字はワイルドカード文字ではありません。角括弧に含めるとリテラルとして処理されます。一般に pattern はパーセント (%) 文字で囲まれた検索文字列 ('%Chicago%') で構成され、string 全体を検索することを示します。

PATINDEX は、シャード・クエリ、並列クエリ、およびリンク・テーブル・クエリでサポートされます。

## 6.1.57 PI

```
PI()
```

定数 pi を返します。括弧は必須ですが、引数は指定しなくてもかまいません。したがって、PI() は 3.141592653589793238 を返します。

## 6.1.58 POWER

```
POWER(num,exponent)
```

exponent にした値 num を返します。

## 6.1.59 QUOTENAME

```
QUOTENAME(value)
```

value を区切り文字付き識別子として返します。TSQL は、二重引用符 (″value″) を区切り文字としてサポートします。以下はその例です。

TSQL

```
PRINT 123
  // returns 123
PRINT QUOTENAME(123)
  // returns "123"
```

## 6.1.60 RADIANS

```
RADIANS(float)
```

度単位の角度測定値を、対応するラジアン単位の測定値に変換します。

## 6.1.61 RAND

```
RAND([seed])
```

1 未満の小数値として乱数を返します。オプションの seed 整数引数は無視されます。これは互換性のために提供されています。クエリで RAND を複数回使用する場合は、別の乱数が返されます。

## 6.1.62 REPLACE

```
REPLACE(target,search,replace)
```

target 文字列に含まれる search 文字列の各インスタンスを検索し、replace 文字列に置き換えて、結果の文字列を返します。target 文字列から search 文字列を削除するには、replace に空白の文字列を指定します。

## 6.1.63 REPLICATE

```
REPLICATE(expression,repeat-count)
```

REPLICATE は、expression を repeat-count 回連結した文字列を返します。

expression が NULL の場合、REPLICATE は NULL を返します。expression が空文字列の場合、REPLICATE は空文字列を返します。

repeat-count が小数の場合、整数部のみを使用します。repeat-count が 0 の場合、REPLICATE は空文字列を返します。repeat-count が負数、NULL、または非数値文字列の場合、REPLICATE は NULL を返します。

## 6.1.64 REVERSE

```
REVERSE(string)
```

string の文字の順序を反対にします。

## 6.1.65 RIGHT

```
RIGHT(string,int)
```

左からカウントした string からの文字の int 数を返します。int が string より大きい場合、完全な文字列が返されます。"LEFT" を参照してください。

## 6.1.66 ROUND

```
ROUND(num,length)
```

整数の length で指定した小数桁数に丸めた num を返します。length が小数桁数より大きい場合は丸められません。length が 0 の場合、num は整数に丸められます。length 引数を省略すると、既定の 0 となります。length が負の整数の場合、num は小数点より左の値に丸められます。3 つ目の引数は ROUND では許可されません。

## 6.1.67 RTRIM

```
RTRIM(string)
```

string から末尾の空白を削除します。

string がすべて空白スペースで構成される場合、言語によって動作が決まります。

- Sybase : NULL を返します。
- MSSQL : 空の文字列を返します。

"LTRIM" を参照してください。

## 6.1.68 SCOPE_IDENTITY

同じスコープの IDENTITY 列に挿入された最後の ID 値を返します。ただし、最後の IDENTITY は現在のプロシージャのスコープに制限されません。したがって、現在のプロシージャ内の文が IDENTITY 値を生成したことがわかっている

場合のみ、SCOPE_IDENTITY を使用する必要があります。例えば、SCOPE_IDENTITY は同じプロシージャ内の INSERT コマンドの後で使用する必要があります。

以下のダイナミック SQL の例では、2番目の INSERT から IDENTITY 値を返します。

ObjectScript

```
SET sql=6
SET sql(1)="CREATE TABLE #mytest (MyId INT IDENTITY(1,1),"
SET sql(2)="Name VARCHAR(20))"
SET sql(3)="INSERT INTO #mytest(Name) VALUES ('John Smith')"
SET sql(4)="INSERT INTO #mytest(Name) VALUES ('Walter Jones')"
SET sql(5)="PRINT SCOPE_IDENTITY()"
SET sql(6)="DROP TABLE #mytest"
SET statement=##class(%SQL.Statement).%New()
SET statement.%Dialect="MSSQL"
SET status=statement.%Prepare(.sql)
SET result=statement.%Execute()
DO result.%Display()
```

## 6.1.69 SIGN

```
SIGN(num)
```

num の記号を示す値を返します。num が負の値 (例えば、-32) のときは -1 を返します。num が正の値 (例えば、32 または +32) のときは 1 を返します。num がゼロ (例えば、0 または -0) のときは 0 を返します。

## 6.1.70 SIN

```
SIN(float)
```

サイン：float で指定された角度のサインを返します。したがって、1 は .841470984807... を返します。

## 6.1.71 SPACE

```
SPACE(num)
```

長さ num の空白スペースの文字列を返します。

## 6.1.72 SQRT

```
SQRT(num)
```

num の平方根の値を返します。したがって、SQRT(9) は 3 を返します。

## 6.1.73 SQUARE

```
SQUARE(num)
```

num の二乗の値を返します。したがって、SQUARE(9) は 81 を返します。

## 6.1.74 STR

```
STR(num,[length[,precision]])
```

length 文字列を返します。length が数値 num (小数点や記号を含む) の文字数以上の場合、STR では、文字列に変換し、先頭の空白を埋め込んで length 文字の結果文字列とした数値 num を返します。

オプション整数 precision を指定すると、num は、文字列を変換する前に指定した小数桁数に切り捨てられます。precision を省略すると、num は整数部分まで切り捨てられます。precision が小数桁数より大きいときには、文字列の変換前に num には末尾のゼロが埋められます。

length を省略すると、既定により 10 となります。length が (precision での調整後の) num の文字数未満のとき、文字の length 数がすべてアスタリスクで構成されたダミーの文字列を返します。

## 6.1.75 STUFF

```
STUFF(string,start,length,replace)
```

削除した文字の length 数と挿入した replace 文字列を含む string を返します。削除および挿入の地点は、start 整数で指定されます。これは、string の最初からカウントされます。length が 0 のとき、文字は削除されません。replace が空の文字列のとき、文字は挿入されません。

start が string 内の文字数よりも大きい場合、値は返されません。start が 1 のとき、文字の length 数が string の最初から削除され、replace 文字列が挿入されます。start が 0 のとき、文字の length から 1 文字引いたものが string の最初から削除され、replace 文字列が挿入されます。

length が string の文字数以上のとき、replace 文字列が返されます。replace 文字列の長さは string または length の長さに制限されません。

## 6.1.76 SUBSTRING

```
SUBSTRING(string,start,length)
```

start の場所で始まる string の下位文字列を文字の length 数で返します。start が string の長さより大きいとき、または length が 0 のとき、文字列は返されません。

## 6.1.77 SUM

```
SUM(numfield)
SUM(DISTINCT numfield)
```

集約関数：numfield 列の合計値を返すためにクエリで使用されます。以下はその例です。

TSQL

```
SELECT SUM(Age) FROM Sample.Person
```

SUM(DISTINCT numfield) は、numfield 列内の一意の値を合計します。NULL フィールドは無視されます。

## 6.1.78 SUSER_NAME

```
SUSER_NAME()
```

現在の OS ユーザの名前を返します。括弧は必須ですが、引数は指定しなくてもかまいません。TSQL の USER_NAME() 関数、InterSystems SQL の USER 関数、および ObjectScript の $USERNAME 特殊変数と同等です。

## 6.1.79 SUSER_SNAME

```
SUSER_SNAME()
```

現在の OS ユーザの名前を返します。括弧は必須ですが、引数は指定しなくてもかまいません。TSQL の USER_NAME() 関数、InterSystems SQL の USER 関数、および ObjectScript の $USERNAME 特殊変数と同等です。

## 6.1.80 TAN

```
TAN(float)
```

タンジェント：float で指定された角度のタンジェントを返します。したがって、1 は 1.55740772465... を返します。

## 6.1.81 TEXTPTR

```
TEXTPTR(field)
```

field で指定したイメージまたはテキスト列データへの内部ポインタを返します。このポインタのデータ型は VARBINARY(16) です。

## 6.1.82 TEXTVALID

```
TEXTVALID('table.field',textpointer)
```

TEXTPTR からイメージ列またはテキスト列への内部ポインタを受け取り、table.field で指定された値と比較します。このポインタが指定された table.field を指している場合は、1 を返します。そうでない場合は、0 を返します。

## 6.1.83 UNICODE

```
UNICODE(char)
```

文字列 char の最初の文字に対応する Unicode 整数値を返します。したがって、`UNICODE('A')` は 65 を返します。

UNICODE は ASCII と機能的に同じです。この逆の関数は CHAR になります。

## 6.1.84 UPPER

```
UPPER(string)
```

小文字をすべて大文字に変換して string を返します。"LOWER" を参照してください。

### 6.1.85 USER

```
USER
```

現在のユーザ名を返します。

この関数では括弧を使用しないため注意してください。

### 6.1.86 USER_NAME

```
USER_NAME([userid])
```

ユーザ ID で指定したユーザ名を返します。オプションの userid が省略されると、現在のユーザ名を返します。引数はオプションで、括弧は必須です。

### 6.1.87 YEAR

```
YEAR(date)
```

指定した日付または日付/時刻の文字列の年部分を返します。date は、次の ODBC タイムスタンプ形式で指定します。

```
yyyy-mm-dd hh:mm:ss.n
```

日付の区切り文字にはハイフン (-) またはスラッシュ (/) のいずれかを使用できます。

date は、`60703` (2007 年 3 月 14 日) のように InterSystems IRIS $HOROLOG 日付形式で指定することもできます。

## 6.2 サポートしていない関数

以下の Microsoft Transact-SQL 関数は、現時点の TSQL ではサポートしていません。APP_NAME、ATN2、BINARY_CHECKSUM、CHECKSUM、COL_LENGTH、COLLATIONPROPERTY、COLUMNPROPERTY、CURSOR_STATUS、DATABASEPROPERTY、DATABASEPROPERTYEX、DB_ID、DIFFERENCE、FILE_ID、FILE_NAME、FILEGROUP_ID、FILEGROUP_NAME、FILEGROUPPROPERTY、FILEPROPERTY、FORMATMESSAGE、FULLTEXTCATALOGPROPERTY、FULLTEXTSERVICEPROPERTY、GETANSINULL、HOST_ID、IDENT_CURRENT、IDENT_INCR、IDENT_SEED、IDENTITY、INDEXKEY_PROPERTY、INDEXPROPERTY、ISDATE、IS_MEMBER、IS_SRVROLEMEMBER、OBJECTPROPERTY、PARSENAME、PERMISSIONS、ROWCOUNT_BIG、SERVERPROPERTY、SESSIONPROPERTY、SESSION_USER、SOUNDEX、SQL_VARIANT_PROPERTY、STATS_DATE、STDEV、STDEVP、SYSTEM_USER、TYPEPROPERTY。

# 7

# TSQL 変数

## 7.1 ローカル変数

既定では、TSQL ローカル変数は接頭辞にアット・マーク (@) を使用して指定します。例えば、@myvar のようにします。この既定は、PLAINLOCALS も許容するようにオーバーライドできます。これにより、接頭辞のアット・マーク (@) なしで指定した TSQL ローカル変数を使用できます。例えば、myvar のようにします。

### 7.1.1 ローカル変数の宣言

ローカル変数は、使用前に (DECLARE を使用するかまたは仮パラメータとして) 宣言する必要があります。変数名は、有効な識別子である必要があります。ローカル変数名は、大文字と小文字が区別されません。InterSystems TSQL では厳密なデータ型の指定は必須ではありませんが、宣言では何らかのデータ型を指定する必要があります。サポートされているデータ型のリストは、このドキュメントの“TSQL 構文”の章を参照してください。

DECLARE コマンドの構文は以下のとおりです。

```
DECLARE @var [AS] datatype [ = initval]
```

宣言している変数が不適切な場合は、NDC 設定を使用して、このチェックをオフに切り替えることができます。ただし、NDC を使用する場合でもカーソルを宣言する必要があります。

ストアド・プロシージャの引数は、自動的にローカル変数として宣言されます。

### 7.1.2 ローカル変数の設定

ローカル変数は、SET コマンドまたは SELECT コマンドのどちらでも設定できます。ローカル変数は、PRINT コマンドまたは SELECT コマンドのどちらを使用しても表示できます。以下のダイナミック SQL の例は、宣言、設定、および表示される 2 つのローカル変数を示しています。

ObjectScript

```
SET myquery = 3
SET myquery(1) = "DECLARE @a CHAR(20),@b CHAR(20) "
SET myquery(2) = "SET @a='hello ' SET @b='world!' "
SET myquery(3) = "PRINT @a,@b"
SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
SET qStatus = tStatement.%Prepare(.myquery)
 IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
SET rset = tStatement.%Execute()
DO rset.%Display()
```

ObjectScript

```
SET myquery = 3
SET myquery(1) = "DECLARE @a CHAR(20),@b CHAR(20) "
SET myquery(2) = "SELECT @a='hello ', @b='world!'"
SET myquery(3) = "SELECT @a,@b"
SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
SET qStatus = tStatement.%Prepare(.myquery)
 IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
SET rset = tStatement.%Execute()
DO rset.%Display()
```

### 7.1.3 初期値および既定値

既定では、DECLARE はローカル変数を "" (SQL NULL) に初期化します。必要な場合は、DECLARE コマンドに、ローカル変数の初期値 (initval) を指定できます。

宣言された変数がスカラのサブクエリの結果に設定される場合、サブクエリが行を返さないと、InterSystems TSQL はその変数を "" (SQL NULL) に設定します。この既定は、MS SQLServer と互換性がありますが、Sybase とは互換性がありません。

### 7.1.4 プレーンなローカル変数

既定では、ローカル変数には接頭辞 @ が必要です。しかし、プレーンなローカル変数 (接頭辞 @ を必要としないローカル変数) も指定できます。以下のコマンドは、プレーンなローカル変数を有効化します。

TSQL

```
SET PLAINLOCALS ON
```

プレーンなローカル変数は、宣言する前に有効化する必要があります。有効化されたプレーンなローカル変数で、接頭辞 @ のあるローカル変数も、接頭辞 @ のないローカル変数も宣言できます。ただし、接頭辞 @ のみが異なる 2 つの変数は宣言できません。例えば、@myvar と myvar は同じ変数と見なされます。プレーンなローカル変数を宣言、選択、または出力するときには、同じ変数で接頭辞 @ のあるもの、またはないものを指定できます。

プレーンなローカル変数は、その他すべての TSQL 変数規約に従います。

以下の TSQL クラス・メソッドは PLAINLOCALS ON を指定し、@ ローカル変数とプレーンなローカル変数の両方を宣言して使用します。

```
ClassMethod Hello() As %String [Language=tsql,ReturnResultsets,SqlProc ]
 { SET PLAINLOCALS ON;
   DECLARE @a CHAR(20),b CHAR(20);
   SET @a='hello ' SET b='world!';
   PRINT @a,b;
  }
```

## 7.2 @@ 特殊変数

TSQL 特殊変数は、接頭辞 @@ で識別されます。@@ 変数はシステム側で定義されています。ユーザ・プロセスでは作成も変更もできません。スコープでは、@@ 変数はグローバルなので、すべてのプロセスで利用できます。そのため、他の Transact-SQL のドキュメントでは“グローバル変数”と呼ばれることもあります。InterSystems IRIS では“グローバル変数”という用語をまったく別の意味で広く使用しているので、混乱を避ける意味で、これらの TSQL @@ 変数をここでは“特殊変数”と呼ぶことにします。

実装されている特殊変数は以下のとおりです。未実装の特殊変数を呼び出すと、#5001 '@@nnn' unresolved symbol エラーまたは #5002 <UNDEFINED> エラーが生成されます。各特殊変数に対し、対応する ObjectScript および InterSystems SQL で生成したコードが提供されます。

## 7.2.1 @@ERROR

最新の TSQL エラーのエラー番号が含まれます。0 はエラーが発生していないことを示します。値 0 が返されるのは、SQLCODE=0 (正常に終了) の場合、または SQLCODE=100 (値が存在しない、または値がこれ以上存在しない) の場合です。これらの 2 つの結果を区別するには、@@SQLSTATUS を使用します。

ObjectScript `SQLCODE`

SQL `:SQLCODE`

## 7.2.2 @@FETCH_STATUS

直前の FETCH カーソル文のステータスを指定する整数が含まれます。使用可能なオプションは以下のとおりです。0 = 行が正常にフェッチされた。-1 = データをフェッチできなかった。-2 = フェッチする行が存在しなかったか、何らかのエラーが発生した。-1 の値は、FETCH するデータがない、または FETCH コマンドがデータの最後に達したことを示します。

### ObjectScript

```
SET myquery = "SELECT @@FETCH_STATUS AS FetchStat"
SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
SET qStatus = tStatement.%Prepare(myquery)
 IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
SET rset = tStatement.%Execute()
DO rset.%Display()
```

対応する InterSystems SQL 関数は以下のとおりです。

### SQL

```
SELECT $TSQL_FETCH_STATUS()
```

ObjectScript `$Case($Get(SQLCODE,0),0:0,100:-1,:-2)`

SQL `CASE :SQLCODE WHEN 0 THEN 0 WHEN 100 THEN -1 ELSE -2 END`

## 7.2.3 @@IDENTITY

最後に挿入、更新または削除された行の IDENTITY フィールドの値が含まれます。

ObjectScript `%ROWID`

SQL `:%ROWID`

## 7.2.4 @@LOCK_TIMEOUT

ロックのタイムアウト値を秒単位で指定する整数が含まれます。ロック・タイムアウトは、リソースでの挿入、更新、削除、および選択に排他的にロックをかける必要がある際に使用します。既定値は 10 です。

### ObjectScript

```
SET myquery = "SELECT @@LOCK_TIMEOUT AS LockTime"
SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
SET qStatus = tStatement.%Prepare(myquery)
 IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
SET rset = tStatement.%Execute()
DO rset.%Display()
```

対応する InterSystems SQL 関数は以下のとおりです。

SQL

```
SELECT $TSQL_LOCK_TIMEOUT()
```

ObjectScript LOCK コマンド

SQL SET OPTION LOCK_TIMEOUT

## 7.2.5 @@NESTLEVEL

現在のプロセスの入れ子レベルを指定する整数が含まれます。

ObjectScript

```
  SET myquery = "PRINT @@NESTLEVEL"
  SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
  SET qStatus = tStatement.%Prepare(myquery)
   IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
  SET rset = tStatement.%Execute()
  DO rset.%Display()
```

対応する InterSystems SQL 関数は以下のとおりです。

SQL

```
SELECT $TSQL_NESTLEVEL()
```

ObjectScript $STACK

## 7.2.6 @@ROWCOUNT

直前の SELECT、INSERT、UPDATE、または DELETE コマンドの影響を受ける行数が含まれます。単一行の SELECT は、常に 0 (行が選択されていない) または 1 のいずれかの @@ROWCOUNT 値を返します。

AFTER 文レベルのトリガを起動する場合、トリガ入力時の @@ROWCOUNT の値は、トリガの直前の @@ROWCOUNT です。トリガ・コードのスコープ内で影響を受けた行は、@@ROWCOUNT 値に反映されます。トリガ・コードの完了時、@@ROWCOUNT はトリガ呼び出しの直前の値に戻されます。

ObjectScript %ROWCOUNT

SQL :%ROWCOUNT

## 7.2.7 @@SERVERNAME

InterSystems IRIS インスタンス名が含まれます。

ObjectScript

```
  SET myquery = "SELECT @@SERVERNAME AS CacheInstance"
  SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
  SET qStatus = tStatement.%Prepare(myquery)
   IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
  SET rset = tStatement.%Execute()
  DO rset.%Display()
```

対応する InterSystems SQL 関数は以下のとおりです。

SQL

```
SELECT $TSQL_SERVERNAME()
```

ObjectScript `$PIECE($system,":",2)`

## 7.2.8 @@SPID

現在のプロセスのサーバ・プロセス ID が含まれます。

### ObjectScript

```
SET myquery = "SELECT @@SPID AS ProcessID"
SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
SET qStatus = tStatement.%Prepare(myquery)
 IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
SET rset = tStatement.%Execute()
DO rset.%Display()
```

対応する InterSystems SQL 関数は以下のとおりです。

### SQL

```
SELECT $TSQL_SPID()
```

ObjectScript `$JOB`

## 7.2.9 @@SQLSTATUS

直前の SQL 文の完了ステータスを指定する整数が含まれます。0 = 成功、1 = 失敗、2 = 有効な値がまったく（またはそれ以上）ない、のいずれかの値を取ることができます。

### ObjectScript

```
SET myquery = "SELECT @@SQLSTATUS AS SQLStatus"
SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
SET qStatus = tStatement.%Prepare(myquery)
 IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
SET rset = tStatement.%Execute()
DO rset.%Display()
```

対応する InterSystems SQL 関数は以下のとおりです。

### SQL

```
SELECT $TSQL_SQLSTATUS()
```

ObjectScript `$Case($Get(SQLCODE,0),0:0,100:2,:1)`

SQL `CASE :SQLCODE WHEN 0 THEN 0 WHEN 100 THEN 2 ELSE 1 END`

## 7.2.10 @@TRANCOUNT

現在アクティブなトランザクションの数が含まれます。

### ObjectScript

```
SET myquery = "SELECT @@TRANCOUNT AS ActiveTransactions"
SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
SET qStatus = tStatement.%Prepare(myquery)
 IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
SET rset = tStatement.%Execute()
DO rset.%Display()
```

対応する InterSystems SQL 関数は以下のとおりです。

SQL

```
SELECT $TSQL_TRANCOUNT()
```

ObjectScript $TLEVEL

## 7.2.11 @@VERSION

InterSystems IRIS バージョン番号およびインストールした日付と時刻が含まれます。

ObjectScript

```
  SET myquery = "SELECT @@VERSION AS CacheVersion"
  SET tStatement = ##class(%SQL.Statement).%New(,,"MSSQL")
  SET qStatus = tStatement.%Prepare(myquery)
   IF qStatus'=1 { WRITE "%Prepare failed",$System.Status.DisplayError(qStatus) QUIT}
  SET rset = tStatement.%Execute()
  DO rset.%Display()
```

対応する InterSystems SQL 関数は以下のとおりです。

SQL

```
SELECT $TSQL_VERSION()
```

ObjectScript $ZVERSION

# 8

# TSQL システム・ストアド・プロシージャ

InterSystems IRIS には、システム内のデータベース・オブジェクトの管理と追跡に役立つ TSQL システム・ストアド・プロシージャが用意されています。TSQL システム・ストアド・プロシージャは、任意のネームスペースやスキーマで実行できます。すべてのストアド・プロシージャのスコープは現在のネームスペースです。

ストアド・プロシージャを実行するには、TSQL の EXECUTE コマンドまたは EXEC コマンドを使用します。sp_xxx のような名前のストアド・プロシージャの場合、この execute コマンドは明示的または暗黙的にすることができます。したがって、以下の TSQL 文は機能的に同じです。

```
EXECUTE sp_addtype 'shortstr','varchar(6)','not null'
EXEC sp_addtype 'shortstr','varchar(6)','not null'
sp_addtype 'shortstr','varchar(6)','not null'
```

InterSystems TSQL では、以下のシステム・ストアド・プロシージャがサポートされています。

- sp_addtype
- sp_droptype
- sp_procxmode

## 8.1 sp_addtype

このシステム・ストアド・プロシージャは、ユーザ定義データ型を追加します。

```
sp_addtype typename, phystype [(length) | (precision [, scale])]
    [, "identity" | nulltype]
```

typename は、ユーザ定義データ型の名前です。phystype は、ユーザ定義データ型の基になる物理データ型です。オプションで、IDENTITY または nulltype を指定できます。オプションの IDENTITY キーワードは、ユーザ定義データ型に IDENTITY プロパティがあることを指定します。このプロパティは定義上、NOT NULL です。オプションの nulltype は、このデータ型の列に NOT NULL 制約が必要かどうかを指定します。オプションは、NULL (NULL を許可する) と NOT NULL または NONULL (NULL を許可しない) です。

typename は、テーブル列にのみ使用できます。typename は、現在のネームスペース内で定義されます。複数のネームスペースにわたる typename のマッピングはサポートされていません。

SQL コンパイラは、システム構成の DDL データ型マッピング定義で型マッピングを検索してから、typename テーブルで検索します。したがって、sp_addtype を使用して My_Type という名前の typename を定義しても、ユーザ定義の DDL データ型マッピングでも My_Type を定義している場合は、ユーザ定義の DDL データ型マッピング定義から My_Type のマッピングが取得されます。

この typename の検索は実行時に行われるので、DDL のコンパイル時に typename を定義する必要はありません。

sp_addtype を、指定した nulltype と共に使用して定義したデータ型の動作は以下のようになります。

- DDL フィールドに NULL を指定した場合、sp_addtype で NOT NULL が指定されていても、フィールドの値は必須ではありません。
- DDL フィールドに NOT NULL を指定した場合、sp_addtype で NULL が指定されていても、フィールドの値は必須です。
- DDL フィールドに NULL または NOT NULL を指定していない場合、sp_addtype で NOT NULL が指定されていても、フィールドの値は必須です。
- DDL フィールドに NULL または NOT NULL を指定していない場合、sp_addtype で NOT NULL が指定されているか、または nulltype が指定されていなくても、フィールドの値は必須ではありません。

以下の例では、データ型 shortstr を作成します。このデータ型には値が必要であり (NOT NULL)、この値は 6 文字以下でなければなりません。

```
EXEC sp_addtype 'shortstr','varchar(6)','not null'
```

# 8.2 sp_droptype

このシステム・ストアド・プロシージャは、ユーザ定義データ型を削除します。

```
sp_droptype typename
```

typename は、現在のネームスペース内で定義されているユーザ定義データ型の名前です。

"sp_addtype" を参照してください。

# 8.3 sp_procxmode (Sybase のみ)

このシステム・ストアド・プロシージャは、ストアド・プロシージャに関連付けられている実行モードを表示または変更します。

```
sp_procxmode [procname [, tranmode]]
```

procname は、ストアド・プロシージャの名前です。tranmode は、トランザクション実行モードです。値は、"chained"、"unchained" (既定値)、および "anymode" です。

引数を指定せずに呼び出された場合、sp_procxmode は、ネームスペースに対して定義されているすべてのプロシージャ、それらのユーザ名およびトランザクション・モード (1=unchained) で構成される結果セットを返します。procname 引数のみを指定して呼び出された場合、sp_procxmode は、指定されたプロシージャのユーザ名とトランザクション・モードを返します。

以下の tranmode 値がサポートされています。

- chained : データ取得文またはデータ変更文 (delete、insert、open、fetch、select、または update) の前にトランザクションを暗黙的に開始します。この場合でも、commit transaction または rollback transaction を使用して、トランザクションを明示的に終了する必要があります。chained モードで実行するように定義されたプロシージャは、プロシージャの開始時に autocommit_off を設定し、プロシージャの終了時に以前の設定をリストアします。

- unchained : (既定値) データ取得文またはデータ変更文の前にユーザがトランザクションを明示的に開始する必要があります。commit transaction または rollback transaction を使用して、トランザクションを明示的に終了する必要があります。unchained モードで実行するように定義されたプロシージャは、プロシージャの開始時に autocommit_on を設定し、プロシージャの終了時に以前の設定をリストアします。
- anymode : モードが定義されていない場合、または anymode として定義された場合、自動コミット設定は変更されません。

InterSystems IRIS では、プロセス設定を変更する SET [UN]CHAINED オプションはサポートされていません。使用される設定は、プロセスの自動コミット・モードの現在の設定です。

chained として定義されたプロシージャが autocommit_on モードのプロセスによって呼び出された場合、InterSystems IRIS はエラーを報告しません。また、unchained として定義されたプロシージャが autocommit_off モードのプロセスによって呼び出された場合もエラーを報告しません。

tranmode メタデータは、実際のメソッド定義の一部ではありません。つまり、tranmode を変更してもリコンパイルの必要はありません。また、TSQL (Sybase) ストアド・プロシージャを含むクラスをエクスポート/インポートしたときに、プロシージャの tranmode 設定がクラス定義と共にエクスポートされることはありません。インポート時に、プロシージャを chained モードで定義する必要がある場合は、プロシージャについて `EXEC sp_procxmode 'procname', 'chained'` を呼び出す必要があります。