SYBASE®

ユーザ定義関数ガイド

Sybase IQ 15.1

ドキュメント ID： DC01139-01-1510-01
改訂： 2009 年 5 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568

目次

はじめに

関連マニュアル、マニュアル表記規則、および Sybase IQ で可能な動作確認について説明します。

このマニュアルの内容

Sybase® IQ は、データ・ウェアハウスやデータ・マート用に特化された高パフォーマンスの意思決定支援データベース・サーバです。 『ユーザ定義関数ガイド』では、Sybase IQ で使用するユーザ定義のスカラ関数および集合関数のプログラミングに関する概念および手順について説明します。

対象読者

このマニュアルは、Sybase IQ データベースのデータにアクセスするアプリケーション開発者を対象にしています。 リレーショナル・データベース・システムの基礎知識と、Sybase IQ のユーザ・レベルの基礎的な経験があることを前提にしています。 このマニュアルは、他のマニュアルと併用するように構成されています。

関連マニュアル

追加の情報は、Sybase IQ および SQL Anywhere のマニュアルに記載されています。

Sybase IQ のマニュアル

Sybase IQ 15.1 のマニュアル・セットには、以下が含まれています。

- 『リリース・ノート』では、製品およびマニュアルに加えられた最新の変更内容について説明しています。
- 『インストールおよび設定ガイド』では、プラットフォーム固有の Sybase IQ のインストール、新バージョンへのマイグレート、特定のプラットフォームでの設定の手順について説明しています。
- 『Sybase IQ による高度なセキュリティ』では、Sybase IQ データ・レポジトリ内でのユーザによるカラムの暗号化の使用について説明しています。 このオプション製品をインストールするには、別のライセンスが必要です。

- 『エラー・メッセージ』は、Sybase エラー・コード、SQLCode、SQLState によって参照される Sybase IQ エラー・メッセージ、および SQL プリプロセッサのエラーと警告の一覧です。
- 『IMSL 数値関数ライブラリ・ユーザ・ガイド：第 2/2 巻 C 統計ライブラリ』では、IMSL C 統計ライブラリの時系列 C 関数について簡潔に説明しています。このマニュアルは、RAP – The Trading Edition™ Enterprise のユーザのみが入手できます。
- 『Sybase IQ の概要』は、Sybase IQ や Sybase Central™ データベース管理ツールの操作に慣れていない場合に参照してください。実際に操作の練習ができます。
- 『Sybase IQ によるラージ・オブジェクト管理』では、Sybase IQ データ・レポジトリ内での BLOB (バイナリ・ラージ・オブジェクト) および CLOB (キャラクタ・ラージ・オブジェクト) の格納と取得について説明しています。 このオプション製品をインストールするには、別のライセンスが必要です。
- 『Sybase IQ 15.0 の新機能』では、バージョン 15.0 の新機能と動作変更について説明しています。
- 『新機能の概要 Sybase IQ 15.1』には、最新バージョンの新機能と動作変更の概要がまとめられています。
- 『パフォーマンス&チューニング・ガイド』では、巨大なデータベースのクエリ最適化、設計、チューニングについて説明しています。
- 『クイック・スタート』には、Sybase IQ のソフトウェア・インストールを検証するために Sybase IQ に付属のデモ・データベースの構築とクエリを行う手順が記載されています。 デモ・データベースをマルチプレックスに変換するための情報も記載されています。
- 『リファレンス・マニュアル』 – Sybase IQ の 2 冊のリファレンス・ガイドで構成されています。
  - 『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』では、Sybase IQ でサポートされる SQL、ストアド・プロシージャ、データ型、およびシステム・テーブルについて説明しています。
  - 『リファレンス：文とオプション』では、Sybase IQ がサポートしている SQL 文およびオプションについて説明しています。
- 『システム管理ガイド』 – 2 巻構成です。
  - 『システム管理ガイド：第 1 巻』では、起動、接続、データベース作成、自動入力とインデックス作成、バージョン管理、照合、システムのバックアップとリカバリ、トラブルシューティング、およびデータベースの修復について説明しています。
  - 『システム管理ガイド：第 2 巻』では、プロシージャとバッチの作成および実行、OLAP でのプログラミング、リモート・データへのアクセス、Open Server としての IQ の設定、スケジューリングとイベント処理、XML でのプログラミング、およびデバッグについて説明しています。

- 『Sybase IQ マルチプレックスの使用』では、複数のノードにまたがって発生する大きなクエリの負荷を管理するために設計されているマルチプレックス機能の使用方法について説明しています。
- 『ユーティリティ・ガイド』では、Sybase IQ ユーティリティ・プログラムのリファレンス項目 (使用可能な構文、パラメータ、オプションなど) について説明しています。

Infocenter Web サイトにアクセスするには、SyBooks オンライン・ヘルプにアクセスしてください。

SQL Anywhere のマニュアル

注意： Sybase IQ は SQL Anywhere Studio® のコンポーネントである SQL Anywhere® と多くのコンポーネントを共有しているため、Sybase IQ は SQL Anywhere と同じ機能を数多くサポートしています。 IQ のマニュアル・セットは、SQL Anywhere Studio のマニュアルの該当する箇所を参照しています。

SQL Anywhere には、次のマニュアルがあります。

- 『SQL Anywhere サーバ – データベース管理』では、SQL Anywhere データベースの実行、管理、設定方法について説明します。 データベース接続、データベース・サーバ、データベース・ファイル、バックアップ手順、セキュリティ、高可用性、Replication Server での複写、管理ユーティリティおよびオプションについて説明します。
- 『SQL Anywhere サーバ – プログラミング』では、C、C++、Java、PHP、Perl、Python、および Visual Basic や Visual C# などの .NET プログラミング言語を使用した、データベース・アプリケーションの構築および展開方法について説明します。 このマニュアルでは、ADO.NET や ODBC などの各種プログラミング・インタフェースについても説明します。
- 『SQL Anywhere サーバ – SQL リファレンス』は、システム・プロシージャの参照情報およびカタログ (システム・テーブルとビュー) を提供します。 さらに、SQL 言語の SQL Anywhere での実装についても説明しています (検索条件、構文、データ型、関数)。

Product Manuals および DocCommentXchange でも、SQL Anywhere Studio 11.0 コレクションの SQL Anywhere のマニュアルを参照できます。

Sybase ソフトウェア資産管理 (SySAM) には、次のマニュアルがあります。

- 『Sybase ソフトウェア資産管理 (SySAM) 2』では資産管理の概念を紹介し、SySAM 2 のライセンスの認定および管理方法について説明します。
- 『SySAM 2 クイック・スタート・ガイド』は、SySAM を有効にした Sybase 製品を稼動させる方法について説明しています。

- 『FLEXnet ライセンス・エンド・ユーザ・ガイド』では、管理者およびエンド・ユーザ向けに FLEXnet ライセンスについて説明し、Sybase から販売される標準的な FLEXnet ライセンス配布キットに含まれているツールの使用方法について説明しています。

その他の情報

Sybase Getting Started CD、SyBooks™ CD、Sybase Product Manuals Web サイトを利用すると、製品について詳しく知ることができます。

- Getting Started CD には、PDF 形式のリリース・ノートとインストール・ガイド、SyBooks CD に含まれていないその他のマニュアルや更新情報が収録されています。 この CD は製品のソフトウェアに同梱されています。 Getting Started CD に収録されているマニュアルを参照または印刷するには、Adobe Acrobat Reader が必要です (CD 内のリンクを使用して Adobe の Web サイトから無料でダウンロードできます)。
- SyBooks CD には製品マニュアルが収録されています。この CD は製品のソフトウェアに同梱されています。 Eclipse ベースの SyBooks ブラウザを使用すれば、使いやすい HTML 形式のマニュアルにアクセスできます。

  一部のマニュアルは PDF 形式で提供されています。それらのマニュアルは SyBooks CD の PDF ディレクトリに収録されています。 PDF ファイルを開いたり印刷したりするには、Adobe Acrobat Reader が必要です。

  SyBooks をインストールして起動するまでの手順については、Getting Started CD の『SyBooks インストール・ガイド』、または SyBooks CD の `README.txt` ファイルを参照してください。
- Sybase Product Manuals Web サイトは、SyBooks CD のオンライン版であり、標準の Web ブラウザを使ってアクセスできます。 また、製品マニュアルのほか、EBFs/Maintenance、Technical Documents、Case Management、Solved Cases、ニュース・グループ、Sybase Developer Network へのリンクもあります。

  Sybase Product Manuals Web サイトにアクセスするには、Product Manuals にアクセスしてください。

Web 上の Sybase 製品の動作確認情報

Sybase Web サイトにある Sybase IQ の製品およびマニュアルの更新情報を確認してください。

製品動作確認の最新情報へのアクセス

Sybase Web サイトにある最新の製品更新情報をダウンロードします。

1. Web ブラウザで Technical Documents を指定します。
2. [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。
3. [Search] をクリックして、入手状況と動作確認レポートを表示します。

コンポーネント動作確認の最新情報へのアクセス

Sybase Web サイトにある最新のコンポーネント更新情報をダウンロードします。

1. Web ブラウザで Availability and Certification Reports を指定します。
2. [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。
3. [Search] をクリックして、入手状況と動作確認レポートを表示します。

Sybase Web サイト (サポート・ページを含む) の自分専用のビューの作成

MySybase プロファイルを設定し、Sybase Web ページの表示方法を自分専用にカスタマイズします。

1. Web ブラウザで Technical Documents を指定します。
2. [MySybase] をクリックし、MySybase プロファイルを作成します。

EBF とソフトウェア・メンテナンスの最新情報へのアクセス

MySybase アカウントを使用し、EBF とソフトウェア・メンテナンスの最新情報にアクセスします。

1. Web ブラウザで Sybase Support Page を指定します。
2. [EBFs/Maintenance] を選択します。 ユーザ名とパスワードの入力が求められたら、MySybase のユーザ名とパスワードを入力します。

3. 製品を選択します。
4. 時間枠を指定して [Go] をクリックします。 EBF/Maintenance リリースの一覧が表示されます。

   鍵のアイコンは、「Technical Support Contact」として登録されていないため、一部の EBF/Maintenance リリースをダウンロードする権限がないことを示しています。 未登録ではあるが、Sybase 担当者またはサポート・センタから有効な情報を得ている場合は、[Edit Roles] をクリックして、「Technical Support Contact」役割を MySybase プロファイルに追加します。
5. EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

SQL 構文の表記規則

このマニュアルで使用されている構文の表記規則を示します。

- キーワード：SQL キーワードは大文字で示します。 ただし、SQL キーワードは大文字と小文字の区別がないので、入力するときはどちらで入力してもかまいません。たとえば、SELECT は Select でも select でも同じです。
- プレースホルダ：適切な識別子または式で置き換えられる項目は、斜体 *italics* で表記します。
- 継続：省略記号 (...) で始まる行は、前の行から文が続いていることを表します。
- 繰り返し項目：繰り返し項目のリストは、リストの要素の後ろに省略記号を付けて表します。 複数の要素を指定できます。 複数の要素を指定する場合は、各要素間はカンマで区切る必要があります。
- オプション部分：文のオプション指定部分は、角カッコで囲みます。 次に例を示します。

  ```
  RELEASE SAVEPOINT [ savepoint-name ]
  ```

  この例では、*savepoint-name* がオプション部分です。 大カッコは入力しないでください。
- オプション：項目リストから 1 つだけ選択しなければならない場合、また何も選択する必要のない場合は、項目間を縦線で区切り、リスト全体を角カッコで囲みます。 次に例を示します。

  ```
  [ ASC | DESC ]
  ```

  この例では、ASC と DESC のどちらか 1 つを選択するか、どちらも選択しないことができます。 大カッコは入力しないでください。
- 選択肢：オプションの中の 1 つを必ず選択しなければならない場合は、選択肢を中カッコ { } で囲みます。 次に例を示します。

  ```
  QUOTES { ON | OFF }
  ```

中カッコは、ON か OFF のいずれかを含めなければならないことを示します。大カッコは入力しないでください。

書体の表記規則

このマニュアルで使用されている書体の表記規則を示します。

表 1：書体の表記規則

項目	説明
`Code`	SQL およびプログラム・コードは等幅 (固定幅) 文字フォントで表記します。
`User entry`	ユーザが入力するテキストには等幅 (固定幅) 文字フォントを使用します。
emphasis	強調する言葉は「」で囲みます。
`file names`	ファイル名は斜体で表記します。
database objects	テーブル、プロシージャなどのデータベース・オブジェクトの名前は、印刷物では太字の sans serif フォントで表記し、オンラインでは斜体で表記します。

アクセシビリティ機能

このマニュアルには、アクセシビリティを重視した HTML 版もあります。 この HTML 版マニュアルは、スクリーン・リーダで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。

Sybase IQ 15.1 とその HTML マニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。 第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。Sybase Central の Sybase IQ プラグインのアクセシビリティへの対応については、プラグイン・オンライン・ヘルプを参照してください (スクリーン・リーダの読み上げで内容を理解できる機能があります)。

注意： アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。 一部のスクリーン・リーダは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) は文字として発音し、大文字と小文字の混在したテキスト (MixedCase Text など) は単語として発音します。 構文規則を発音するようにツールを設定することをおすす

めします。 スクリーン・リーダの使用方法については、ツールのマニュアルと『Sybase IQ の概要』を参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility を参照してください。 Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報のリンクもあります。

Sybase IQ の第 508 条準拠の声明については、Sybase Accessibility を参照してください。

デモ・データベース

Sybase IQ にはデモ・データベースを作成するためのスクリプトが用意されています。

Sybase IQ にはデモ・データベース (iqdemo.db) を作成するためのスクリプトが用意されています。 このマニュアルで紹介している多くのクエリおよびコード例は、このデモ・データベースをデータ・ソースに使用しています。

デモ・データベースは、小規模会社の内部情報 (従業員、部署、財務データ) に加えて、製品と販売情報 (注文、顧客、担当者) で構成されています。

デモ・データベースの詳細については、使用しているプラットフォームの『Sybase IQ インストール・ガイド』を参照するか、システム管理者に相談してください。

不明な点があるときは

サポート契約を購入済みの Sybase 製品のインストールには、定められた 1 人以上のユーザに対して、Sybase 製品の保守契約を結んでいるサポート・センタを利用する権利が付属します。 マニュアルやオンライン・ヘルプで解決できない問題がある場合は、この担当者を通して最寄りの Sybase のサポート・センタまでご連絡ください。

ユーザ定義関数のインストールと設定

ユーザ定義関数のインストールと作成について説明します。

注意： ユーザ定義関数は、ライセンスが必要なオプションであり、使用するには IQ_UDF ライセンスが必要です。 ライセンスをインストールすると、ユーザ定義関数を使用できるようになります。

Sybase IQ の概要

外部 C/C++ ユーザ定義関数 (UDF) の作成と設定について説明します。

Sybase IQ は、高パフォーマンスのインプロセス外部 C/C++ ユーザ定義関数をサポートしています。 このような UDF は、このマニュアルで説明しているインタフェースで C または C++ コードを使用して記述された関数をサポートしています。 これらの UDF 定義は、ダイナミック・リンク・ライブラリ内でコンパイルしてリンクできます。 ダイナミック・リンク・ライブラリは、実行中の IQ サーバにロードできます。定義済みの UDF は、クエリやその他の SQL 文で直接使用できます。

これらの外部 C/C++ UDF インタフェースの使用には、IQ_UDF ライセンスが必要です。

これらの外部 C/C++ UDF は、以前のバージョンの Sybase IQ で使用可能な Interactive SQL UDF とは異なります。 Interactive SQL UDF は、変更は加えられておらず、使用に際して特別なライセンスも必要ありません。 Interactive SQL を使用して UDF を作成する手順については、『システム管理ガイド：第 2 巻』の「第 1 章 プロシージャとバッチの使用」を参照してください。

ダイナミック・リンク・ライブラリの UDF を作成して使用するには、次の手順を実行します。

1. CREATE FUNCTION または CREATE AGGREGATE FUNCTION 文を使用して、サーバに対して UDF を宣言します。 これらの文は、コマンドとして直接記述して実行できます。または、「Sybase Central の新機能ウィザード」 (13 ページ) を使用し、適切な CREATE 文を作成できます。

   CREATE FUNCTION 文の外部 C/C++ 形式には DBA または RESOURCE 権限が必要であるため、通常のユーザにはこの種類の UDF を宣言する権限がありません。
2. UDF を、C または C++ 関数のセットとして定義します。「スカラ UDF の定義」 (29 ページ) または「集計 UDF の定義」 (43 ページ) を参照してください。

3. UDF ライブラリ識別関数を記述します。 (10 ページ)
4. UDF 関数およびライブラリ識別関数をコンパイルします。 (17 ページ)
5. コンパイル済みファイルをダイナミック・リンク・ライブラリにリンクします。

これらの手順が完了すると、最初に、SQL 文の UDF への参照により、必要に応じてダイナミック・リンク・ライブラリがリンクされます。 次に、パターン呼び出し (73 ページ) が呼び出されます。

これらの高パフォーマンスの外部 C/C++ ユーザ定義関数は、サーバのプロセス領域への非サーバ・ライブラリ・コードのロードも行うため、関数の記述が不完全な場合や意図的に不正な場合、データの整合性やセキュリティ、およびサーバの堅牢性に関してリスクが発生する可能性があります。 これらのリスクを管理するために、IQ サーバごとに明示的にこの機能を有効または無効にできます (10 ページ)。

ダイナミック・ライブラリ・インタフェースの設定

ダイナミック・リンク・ライブラリで使用するインタフェース・スタイルを指定します。

動的にロードされたライブラリにはそれぞれ、次の定義のコピーが 1 つ含まれている必要があります。

```
extern "C" a_sql_uint32 extfn_use_new_api(void )
{
return EXTFN_V3_API;
}
```

この定義は、インタフェース・スタイルが使用されるサーバと、このダイナミック・リンク・ライブラリで定義される UDF へのアクセス方法を示します。 高パフォーマンスの IQ の UDF については、バージョン 3 インタフェース・スタイル (EXTFN_V3_API) のみがサポートされています。

ユーザ定義関数の有効化および無効化

Sybase IQ 15.1 には external_procedure_v3 というセキュリティ機能があり、サーバによる高パフォーマンスのインプロセス UDF の使用を有効または無効にできます。

管理者は、次のように指定すると、任意のサーバに対してバージョン 3 の UDF を有効にできます。

```
-sf -external_procedure_v3
```

これは、サーバの起動コマンドまたは設定ファイルで指定します。

管理者は、次のように指定すると、任意のサーバに対してバージョン 3 の UDF を無効にできます。

```
-sf external_procedure_v3
```

これは、サーバの起動コマンドまたは設定ファイルで指定します。

-sf フラグに関する追加の情報は、『SQL Anywhere サーバ - データベース管理』に記載されています。 SQL Anywhere のマニュアルに記載されている値は Sybase IQ 15.1 に該当しないため、使用できません。

ユーザ定義関数の作成と実行

ユーザ定義関数 (UDF) は、呼び出しを行った環境に単一の値を返すプロシージャのクラスです。 ここでは、ユーザ定義関数の作成、使用、削除について説明します。

UDF の作成および使用に必要な 4 つの手順は次のとおりです。

1. CREATE AGGREGATE FUNCTION 文または CREATE FUNCTION 文を使用して、ユーザ定義関数を宣言します。
2. C/C++ で関数のエントリ・ポイントを実装します。
3. 実装をコンパイルし、共有ライブラリとしてリンクします (17 ページ)。
4. SQL 組み込み関数を使用する SQL 文内で関数を使用します。

ユーザ定義関数の作成

ユーザ定義関数を作成するには、**CREATE FUNCTION** 文または **CREATE AGGREGATE FUNCTION** 文を使用します。 この文を実行するには、RESOURCE 権限が必要です。

SQL Anywhere ダイアレクトを使用したユーザ定義関数の作成

Watcom-SQL および Transact-SQL は SQL Anywhere がサポートする SQL ダイアレクトであり、ユーザ定義関数の作成時に使用できます。

1. Sybase Central で、DBA または RESOURCE 権限のあるユーザとしてデータベースに接続します。
2. Sybase Central の [ビュー] メニューから、[フォルダ] を選択します。
3. 左側のウィンドウ枠で、[プロシージャと関数] を右クリックして、[新規] - [関数] を選択します。
4. [ようこそ] ダイアログで、関数の名前と関数を所有するユーザを選択します。
5. 関数の SQL 構文または言語を選択します。 [次へ] をクリックします。
6. 関数の戻り値の種類を選択し、値のサイズ、単位、および位取りを指定します。
7. 戻り値の名前を入力して、[次へ] をクリックします。
8. 新しい関数の目的を説明するコメントを追加します。 [完了] をクリックします。
9. 右側のウィンドウ枠で、[SQL] タブをクリックし、プロシージャ・コードを完了します。

Sybase Central におけるスカラ・ユーザ定義関数の宣言

Sybase IQ では、SQRT 関数を使用できる場所で使用できる単純なスカラ UDF をサポートしています。 これらのスカラ UDF は決定的関数にできます。これは、指定された引数の値のセットに対して、関数が常に同じ結果値を返すことを意味します。また、Sybase IQ は、非決定的なスカラ関数もサポートしています。これは、同じ引数が異なる結果を返すことができることを意味します。

1. Sybase Central で、DBA または RESOURCE 権限のあるユーザとしてデータベースに接続します。
2. 左側のウィンドウ枠で、[プロシージャと関数] を右クリックして、[新規] - [関数] を選択します。
3. [ようこそ] ダイアログで、関数の名前を入力し、関数の所有者になるユーザを選択します。
4. ユーザ定義関数を作成するには、[外部 C/C++] を選択します。 [次へ] をクリックします。
5. [外部関数属性] ダイアログで、[スカラ] を選択します。
6. .so または .dll 拡張子を省略して、ダイナミック・リンク・ライブラリ・ファイルの名前を入力します。
7. 記述子関数の名前を入力します。 [次へ] をクリックします。
8. 関数の戻り値の種類を選択し、値のサイズ、単位、および位取りを指定します。 [次へ] をクリックします。
9. 関数が決定的であるかどうかを選択します。
10. 関数が NULL 値を尊重するか無視するかを指定します。
11. 関数の実行に使用される権限が、定義しているユーザ (定義者) の権限であるか、呼び出し側のユーザ (呼び出し者) の権限であるかを選択します。
12. 新しい関数の目的を説明するコメントを追加します。 [完了] をクリックします。
13. 右側のウィンドウ枠で、[SQL] タブをクリックし、プロシージャ・コードを完了します。

Sybase Central におけるユーザ定義の集合関数の宣言

Sybase IQ では、ユーザ定義の集合関数 (UDAF) をサポートしています。 SUM 関数は、組み込み集合関数の一例です。 単純な集合関数は、引数の値のセットを取り、その入力のセットから 1 つの結果値を生成します。 ユーザ定義の集合関数は、SUM 集合を使用できる場所で使用できるように記述できます。

1. Sybase Central で、DBA または RESOURCE 権限のあるユーザとしてデータベースに接続します。
2. 左側のウィンドウ枠で、[プロシージャと関数] を右クリックして、[新規] - [関数] を選択します。
3. [ようこそ] ダイアログで、関数の名前を入力し、関数の所有者になるユーザを選択します。
4. ユーザ定義関数を作成するには、[外部 C/C++] を選択します。 [次へ] をクリックします。
5. [集計] を選択します。
6. .so または .dll 拡張子を省略して、ダイナミック・リンク・ライブラリ・ファイルの名前を入力します。
7. 記述子関数の名前を入力します。 [次へ] をクリックします。
8. 関数の戻り値の種類を選択し、値のサイズ、単位、および位取りを指定します。 [次へ] をクリックします。
9. 関数の実行に使用される権限が、定義しているユーザ (定義者) の権限であるか、呼び出し側のユーザ (呼び出し者) の権限であるかを選択します。
10. 関数を **OVER** 句で使用できるようにするか、**OVER** 句で使用する必要があるか、または **OVER** 句で使用できないようにするかを指定します。 [次へ] をクリックします。

    関数を **OVER** 句で使用できないようにする場合は、手順 14 に進みます。
11. ウィンドウの定義に使用する場合に関数が **ORDER BY** 句のユーザを必要とするかどうかを指定します。 [次へ] をクリックします。
12. 関数を **WINDOW FRAME** 句で使用できるようにするか、**WINDOW FRAME** 句で使用する必要があるか、または **WINDOW FRAME** 句で使用できないようにするかを指定します。 [次へ] をクリックします。

    関数を **WINDOW FRAME** 句で使用できないようにする場合は、手順 14 に進みます。
13. **WINDOW FRAME** 句に対する制限を指定します。 [次へ] をクリックします。
14. 関数を呼び出す前に、重複する入力値をデータベース・サーバでフィルタする必要があるかどうかを指定します。
15. データを持たない関数が呼び出されたときに、関数の戻り値を **NULL** にするか、固定値にするかを指定します。 [次へ] をクリックします。
16. 新しい関数の目的を説明するコメントを追加します。 [完了] をクリックします。
17. 右側のウィンドウ枠で、[SQL] タブをクリックし、プロシージャ・コードを完了します。

[プロシージャと関数] に新しい関数が表示されます。

ユーザ定義関数の制限

外部 C/C++ ユーザ定義関数には、次の制限があります。

- さまざまなコンテキスト関数を受け取りながら、複数のユーザが同時に関数を呼び出すことができるように、すべての UDF を記述する必要があります。
- UDF がグローバルまたは共有のデータ構造にアクセスする場合、UDF 定義によって、データへのアクセスに関する適切なロックを実装します。これには、すべての通常のコード・パスやすべてのエラー処理状況におけるロックの解放などが含まれます。
- C++ で UDF を実装すると、そのクラスの "new" 演算子が過負荷になる可能性がありますが、グローバルな "new" 演算子を過負荷にしないようにしてください。一部のプラットフォームでは、過負荷による影響が特定のライブラリ内の定義されたコードに限定されません。
- すべての集合 UDF とすべての決定的なスカラ UDF は、同じ入力値を受け取ると必ず同じ出力値を生成するように記述する必要があります。 これに該当しないスカラ関数については、NONDETERMINISTIC と宣言して、正しくない応答の生成を回避する必要があります。

ユーザ定義関数の呼び出し

ユーザ定義関数は、パーミッションがあれば、集合関数以外の組み込み関数を使用できるどの場所でも使用できます。

次の Interactive SQL 文を実行すると、姓と名前を含んでいる 2 つのカラムから氏名が返されます。

```
SELECT fullname (GivenName, LastName)
FROM Employees;
```

fullname (Employees.GivenName,Employees.SurName)
Fran Whitney
Matthew Cobb
Philip Chin
...

次の文を実行すると、指定された姓と名前から氏名が返されます。

```
SELECT fullname ('Jane', 'Smith');
```

fullname ('Jane','Smith')
Jane Smith

fullname 関数は、この関数に対する実行パーミッションが付与されているどのユーザでも使用できます。

ユーザ定義関数の削除

作成したユーザ定義関数は、明示的に削除されないかぎり、データベースに存在します。 ユーザ定義関数の所有者または DBA 権限を持つユーザだけがデータベースから関数を削除できます。

たとえば、関数 *fullname* をデータベースから削除するには、次のように入力します。

```
DROP FUNCTION fullname
```

パーミッションの付与と取り消し

ユーザ定義関数の所有者はそれを作成したユーザです。所有者はパーミッションなしで関数を実行できます。 ユーザ定義関数の所有者は、**GRANT EXECUTE** コマンドを使用して、他のユーザにパーミッションを付与できます。

たとえば、関数 *fullname* の作成者は、次の文を発行して *another_user* に *fullname* の使用を許可できます。

```
GRANT EXECUTE ON fullname TO another_user
```

また、パーミッションを取り消すには、次の文を発行します。

```
REVOKE EXECUTE ON fullname FROM another_user
```

関数に対するユーザのパーミッションを管理する方法の詳細については、『システム管理ガイド：第 1 巻』の「第 8 章 ユーザ ID とパーミッションの管理」の「プロシージャに対するパーミッションの付与」を参照してください。

ダイナミック・リンク・ライブラリを構築するためのコンパイルおよびリンク・スイッチ

任意のユーザ定義関数のダイナミック・リンク・ライブラリを構築する場合、次のコンパイルおよびリンク・スイッチを使用します。

UDF ダイナミック・リンク・ライブラリを構築するには、次の手順を実行します。

1. UDF ダイナミック・リンク・ライブラリには、関数 "extfn_use_new_api()" の実装が必要です。 この関数のソース・コードは、「ダイナミック・ライブラリ・インタフェースの設定」 (10 ページ) に記載されています。 この関数は、ライブラリ内のすべての関数が準拠している API スタイルのサーバを通知します。 サンプル・ソース・ファイル "my_main.cxx" にはこの関数が含まれており、修正しないで使用できます。
2. UDF ダイナミック・リンク・ライブラリには、少なくとも 1 つの UDF 関数のオブジェクト・コードも必要です。 UDF ダイナミック・リンク・ライブラリは、オプションで複数の UDF を含めることができます。
3. 各 UDF のオブジェクト・コードと extfn_use_new_api() をリンクして、1 つのライブラリを構築します。

たとえば、ダイナミック・リンク・ライブラリの例 "libudfex" を構築するための手順は、次のとおりです。

- 各ソース・ファイルをコンパイルし、オブジェクト・ファイルを生成します。

```
my_main.cxx
            my_bit_or.cxx
            my_bit_xor.cxx
            my_interpolate.cxx
            my_plus.cxx
            my_plus_counter.cxx
            my_sum.cxx
```

- 生成された各オブジェクトをリンクし、1 つのライブラリを構築します。

次の項では、UDF ダイナミック・リンク・ライブラリを構築するために、ソース・ファイルをコンパイルし、オブジェクトをリンクする場合、プラットフォームに固有の推奨事項について説明します。 他のバージョンのコンパイラが動作する場合があります。次の固有の例は参考のために示します。

ダイナミック・リンク・ライブラリのコンパイルおよびリンクが完了した後、次のタスクのいずれかを完了します。

- UDF ライブラリの明示的なパス名が含まれるように、CREATE FUNCTION ... EXTERNAL NAME を更新します (推奨)。
- すべての IQ ライブラリが格納されるディレクトリに、UDF ライブラリ・ファイルを配置します。
- UDF ライブラリの場所を含むライブラリ・ロード・パスで、IQ サーバを起動します。

  Unix variant では、start_iq 起動スクリプトの LD_LIBRARY_PATH を変更することで実行できます。 すべての UNIX variant で LD_LIBRARY_PATH が一般的に使用されますが、HP では SHLIB_PATH が、AIX では LIB_PATH が優先的に使用されます。

Unix プラットフォームでは、LD_LIBRARY_PATH が使用されない場合に、外部名の指定に完全修飾名を含めることができます。 Windows プラットフォームでは、完全修飾名は使用できず、ライブラリ検索パスは PATH 環境変数で定義されます。

AIX スイッチ

AIX で共有ライブラリを構築する場合、次のコンパイルおよびリンク・スイッチを使用します。

xlC 8.0 (PowerPC)

compile switches

```
-q64 -qarch=ppc64 -qtbtable=full -qsrcmsg -qalign=natural -
qnoansialias
-qmaxmem=-1 -qenum=int -qhalt=e -qflag=w -qthreaded -
qxflags=NLOOPING
-qtmplinst=none -qthreaded
```

link switches

```
-brtl -G -lg  -lpthreads_compat -lpthreads -lm_r -ldl -bnolibpath -
v
```

HP-UX スイッチ

HP-UX で共有ライブラリを構築する場合、次のコンパイルおよびリンク・スイッチを使用します。

aCC 6.17 (Itanium)

compile switches

```
+noeh -ext +W740,749,829 +W1031 +DD64 +DSblended +FPD -Aa +ub
-U_HP_INSTANTIATE_T_IN_LIB -Wc,-ansi_for_scope,on -mt -z
```

link switches

```
 -b -Wl,+s
```

Linux スイッチ

Linux で共有ライブラリを構築する場合、次のコンパイルおよびリンク・スイッチを使用します。

g++ 4.1.1 (x86)

compile switches

```
-fPIC -fsigned-char -fno-exceptions -pthread -fno-omit-frame-
pointer
-Wno-deprecated -Wno-ctor-dtor-privacy
```

link switches

```
-ldl -lnsl -lm -lpthread -shared -Wl,-Bsymbolic -Wl,-shared
```

注意： gcc は、Linux でも使用できます。 gcc とリンクすると同時に、`-lstdc++` をリンク・スイッチに追加し、C++ ランタイム・ライブラリでリンクします。

xlC 8.0 (PowerPC)

compile switches

```
 -q64 -qarch=ppc64 -qcheck=nullptr -qinfo=gen -qtbtable=full -
qsrcmsg
-qnoansialias -qminimaltoc -qmaxmem=-1 -qenum=int -qhalt=e -qflag=w
-qthreaded
-qxflags=NLOOPING -qtmplinst=none
```

link switches

```
-qmkshrobj -ldl -lg  -qthreaded -lnsl -lm
```

Solaris スイッチ

Solaris で共有ライブラリを構築する場合、次のコンパイルおよびリンク・スイッチを使用します。

Sun Studio 12 (SPARC)

compile switches

```
-mt -noex +w -KPIC -i -instances=explicit -V -xtarget=ultra3cu -m64
-xlibmopt
-xlibmil -features=no%conststrings
-erroff=truncwarn,nokeyworddefine,diffenumtype
```

link switches

```
-z defs -G -ldl -lnsl -lsocket -ladm -lposix4 -lCrun -lCstd -lc -lm
-lefi
-liostream -lkstat
```

Sun Studio 12 (x86)

compile switches

```
+w2 -m64 -features=no%conststrings
-erroff=truncwarn,nokeyworddefine,diffenumtype,doubunder -errtags -
mt  -noex
-KPIC -instances=explicit -xlibmopt -xlibmil
```

link switches

```
-z defs -G -ldl -lnsl -lsocket -ladm -lposix4 -lCrun -lCstd -lc -lm
-lefi
-liostream -lkstat
```

Windows スイッチ

Windows で共有ライブラリを構築する場合、次のコンパイルおよびリンク・スイッチを使用します。

Visual Studio 2008 (x86)

compile and link switches

次に、my_plus 関数を含む DLL の例を示します。 DLL に含まれる UDF ごとに、記述子関数の EXPORT スイッチを含める必要があります。

```
cl /Zi /LD   /I includefilepath     my_main.cxx my_plus.cxx  /link /
map
/INCREMENTAL:NO -EXPORT:extfn_use_new_api -EXPORT:my_plus  /
out:iqudf.dll
```

SQL データ型

UDF 宣言は、特定の SQL データ型のみをサポートします。

UDF 宣言では、UDF の引数のデータ型として、または戻り値のデータ型として、次の SQL データ型を使用できます。

- **UNSIGNED BIGINT** – 記憶領域を 8 バイト必要とする符号なし 64 ビット整数です。 UDF コードで使用するデータ型の識別子は DT_UNSBIGINT であり、UDF でそのような値に使用する C/C++ データ型の typedef は "a_sql_uint64" です。 移植可能な UDF 実装をアプリケーション開発者がより簡単に記述できるようにするために、いくつかの C/C++ typedef が Sybase IQ に用意されています。
- **BIGINT** – 記憶領域を 8 バイト必要とする符号付き 64 ビット整数です。 データ型の識別子は DT_BIGINT であり、そのような値に使用する C/C++ データ型の typedef は "a_sql_int64" です。
- **UNSIGNED INT** – 記憶領域を 4 バイト必要とする符号なし 32 ビット整数です。 データ型の識別子は DT_UNSINT であり、そのような値に使用する C/C++ データ型の typedef は "a_sql_uint32" です。
- **INT** – 記憶領域を 4 バイト必要とする符号付き 32 ビット整数です。 データ型の識別子は DT_INT であり、そのような値に使用する C/C++ データ型の typedef は "a_sql_int32" です。
- **SMALLINT** – 記憶領域を 2 バイト必要とする符号付き 16 ビット整数です。 データ型の識別子は DT_SMALLINT であり、そのような値に使用する C/C++ データ型は "short" です。

- **TINYINT** – 記憶領域を 1 バイト必要とする符号なし 8 ビット整数です。 データ型の識別子は DT_TINYINT であり、そのような値に使用する C/C++ データ型は "unsigned char" です。
- **DOUBLE** – 記憶領域を 8 バイト必要とする符号付き 64 ビット倍精度浮動小数点数です。 データ型の識別子は DT_DOUBLE であり、そのような値に使用する C/C++ データ型は "double" です。
- **REAL** – 記憶領域を 4 バイト必要とする符号付き 32 ビット浮動小数点数です。 データ型の識別子は DT_FLOAT であり、そのような値に使用する C/C++ データ型は "float" です。
- **FLOAT** – SQL では、関連する精度に応じて、FLOAT は、記憶領域を 4 バイト必要とする符号付き 32 ビット浮動小数点数か、記憶領域を 8 バイト必要とする符号付き 64 ビット倍精度浮動小数点数のどちらかになります。 FLOAT データ型のオプションの精度が指定されていない場合、UDF 宣言でのみ、SQL データ型 FLOAT を使用できます。 精度が指定されていない場合、FLOAT は REAL と同義になります。そのデータ型の識別子は DT_FLOAT であり、そのような値に使用する C/C++ データ型は "float" です。
- **CHAR(<n>)** – データベースのデフォルトの文字セット内の、ブランクを埋め込まれた固定長の文字列です。 最大長 "<n>" は 32767 です。データは NULL バイトで終了できません。 データ型の識別子は DT_FIXCHAR であり、そのような値に使用する C/C++ データ型は "char *" です。
- **VARCHAR(<n>)** – データベースのデフォルトの文字セット内の、可変長の文字列です。 最大長 "<n>" は 32767 です。データは NULL バイトで終了できません。 UDF 入力引数の値が NULL でない場合、実際の長さは、an_extfn_value 構造の total_length フィールドから取得する必要があります。 同様に、この種類の UDF 結果について、実際の長さは total_length フィールドに設定する必要があります。 データ型の識別子は DT_VARCHAR であり、そのような値に使用する C/C++ データ型は "char *" です。
- **BINARY(<n>)** – NULL バイトが埋め込まれた固定長のバイナリです。値の最大バイナリ長 "<n>" は 32767 です。データは NULL バイトで終了できません。 データ型の識別子は DT_FIXBINARY であり、そのような値に通常使用する C/C++ データ型は "unsigned char *" です。
- **VARBINARY(<n>)** – 可変長のバイナリ値です。値の最大長 "<n>" は 32767 です。データは NULL バイトで終了できません。 UDF 入力引数の値が NULL でない場合、実際の長さは、an_extfn_value 構造の total_length フィールドから取得する必要があります。 同様に、この種類の UDF 結果について、実際の長さは total_length フィールドに設定する必要があります。 データは NULL バイトで終了できません。 データ型の識別子は DT_VARBINARY であり、そのような値に通常使用する C/C++ データ型は "unsigned char *" です。

- **DATE** – 符号なし整数として UDF との間で受け渡しされる、暦日値です。 UDF に渡された値は、比較／ソート操作で使用できることが保証されます。 大きい値は後の日付を示します。 実際の日付コンポーネントが必要な場合、型 DT_TIMESTAMP_STRUCT に変換するために、UDF は convert_value api を呼び出す必要があります。 この datetype は、次のような構造で日付と時刻を表します。

```
typedef struct sqldatetime {
    unsigned short    year;         /* e.g. 1992            */
    unsigned char    month;         /* 0-11                 */
   unsigned char   day_of_week;    /* 0-6  0=Sunday, 1=Monday, ...
*/
    unsigned short    day_of_year;     /* 0-365              */
    unsigned char    day;         /* 1-31                  */
    unsigned char    hour;         /* 0-23                 */
    unsigned char    minute;     /* 0-59                   */
    unsigned char    second;     /* 0-59                   */
    a_sql_uint32     microsecond;    /* 0-999999            */
} SQLDATETIME;
```

- **TIME** – 指定された日付内のある時刻を正確に記述する値です。 この値は、UNSIGNED BIGINT として UDF に渡されます。 UDF に渡された値は、比較／ソート操作で使用できることが保証されます。 大きい値は後の時刻を示します。 実際の時刻コンポーネントが必要な場合、型 DT_TIMESTAMP_STRUCT に変換するために、UDF は convert_value api を呼び出す必要があります。
- **DATETIME, SMALLDATETIME, or TIMESTAMP** – UNSIGNED BIGINT として UDF との間で受け渡しされる、暦日および時刻の値です。 UDF に渡された値は、比較／ソート操作で使用できることが保証されます。 大きい値は後の日時を示します。 実際の時刻コンポーネントが必要な場合、型 DT_TIMESTAMP_STRUCT に変換するために、UDF は convert_value api を呼び出す必要があります。

サポートされていないデータ型

UDF 宣言では、UDF の引数のデータ型として、または戻り値のデータ型として、次の SQL データ型を使用できません。

- **BIT** – 通常、TINYINT データ型として UDF 宣言で処理され、BIT からの暗黙のデータ型変換によって自動的に値変換が処理されます。
- **DECIMAL(<precision>, <scale>) or NUMERIC(<precision>, <scale>)** – 使用方法により、通常は DOUBLE データ型として処理されます。ただし、INT または BIGINT データ型を使用できるようにするには、さまざまな規則が適用されます。
- **LONG VARCHAR** – 現在サポートされていません。
- **LONG BINARY** – 現在サポートされていません。

- **TEXT** – 現在サポートされていません。

スカラ・ユーザ定義関数

Sybase IQ では、SQRT 関数を使用できる場所で使用できる単純なスカラ・ユーザ定義関数 (UDF) をサポートしています。

これらのスカラ UDF には、決定的スカラ関数、つまり特定の引数のセットに対して常に同じ結果値を返す関数、および非決定的スカラ関数、つまり同じ引数が異なる結果を返す場合のある関数のいずれを使用することもできます。

注意： この章で説明するスカラ UDF の例は IQ サーバとともにインストールされ、`$IQDIR15/samples/udf` にある .cxx ファイルに記載されています。 `$IQDIR15/lib64/libudfex` ダイナミック・リンク・ライブラリからも参照できます。

スカラ UDF の宣言

DBA、または DBA 権限を持つユーザのみがインプロセス外部 UDF を宣言できます。管理者がこのようなユーザ定義関数を有効または無効にできる、新しいサーバ起動オプションもあります。

注意： Sybase Central でユーザ定義関数宣言を作成 (14 ページ) することもできます。

デフォルトでは、すべてのユーザ定義関数へのアクセスに、UDF の所有者のアクセス・パーミッションが使用されます。

IQ のスカラ UDF を作成するためにサポートされている IQ 構文は、次のとおりです。

```
scalar-udf-declaration:
CREATE FUNCTION [ owner.]function-name
     ( [ parameter , ... ] )
RETURNS data-type
     [ routine-characteristics ... ]
EXTERNAL NAME library-and-entry-point-name-string

parameter:
    param-name data-type [ DEFAULT value ]

routine-characteristics:
     [NOT] DETERMINISTIC
  | { IGNORE | RESPECT } NULL VALUES
     | SQL SECURITY { INVOKER | DEFINER }
```

上記の構文の特性のデフォルトは、次のとおりです。

```
DETERMINISTIC
RESPECT NULL VALUES
SQL SECURITY DEFINER
```

潜在的なセキュリティの問題を最小限に抑えるため、EXTERNAL NAME 句のライブラリ名部分には安全なディレクトリの完全修飾パス名を使用することをおすすめします。

SQL Security

INVOKER (関数を呼び出しているユーザ) または DEFINER (関数を所有しているユーザ) のどちらとして関数が実行されるかを定義します。 デフォルトは DEFINER です。

SQL SECURITY INVOKER が指定された場合、プロシージャを呼び出すユーザごとに注釈が必要であるため、より多くのメモリが使用されます。 また、**SQL SECURITY INVOKER** が指定された場合、ユーザ名と INVOKER の両方で名前の決定が行われます。 適切な所有者名で、すべてのオブジェクト名 (テーブル、プロシージャなど) を修飾します。

External Name

EXTERNAL NAME 句を使用する関数は、外部ライブラリにある関数への呼び出しのラッパです。 **EXTERNAL NAME** を使用する関数は、**RETURNS** 句の後に他の句を持つことができません。 ライブラリ名にはファイル拡張子が付く場合があります。この拡張子は通常、Windows では `.dll`、UNIX では `.so` です。 拡張子が付いていなければ、ソフトウェアがプラットフォーム固有のデフォルトのファイル拡張子をライブラリに付加します。

テンポラリ関数では **EXTERNAL NAME** 句はサポートされません。『SQL Anywhere サーバ – プログラミング』の「プロシージャからの外部ライブラリの呼び出し」を参照してください。

IQ サーバは、UDF ライブラリの場所を含むライブラリ・ロード・パスで起動できます。 Unix variant では、start_iq 起動スクリプトの LD_LIBRARY_PATH を変更することで実行できます。 すべての UNIX variant で LD_LIBRARY_PATH が一般的に使用されますが、HP では SHLIB_PATH が、AIX では LIB_PATH が優先的に使用されます。

Unix プラットフォームでは、LD_LIBRARY_PATH が使用されない場合に、外部名の指定に完全修飾名を含めることができます。 Windows プラットフォームでは、完全修飾名は使用できず、ライブラリ検索パスは PATH 環境変数で定義されます。

注意： スカラ・ユーザ定義関数とユーザ定義の集合関数は、更新可能なカーソルではサポートされません。

UDF の例： my_plus 宣言

"my_plus" の例は、2 つの整数引数値を加算した結果を返す単純なスカラ関数です。

my_plus 宣言

ダイナミック・リンク・ライブラリ my_shared_lib 内に my_plus がある場合、この例の宣言は、次のようになります。

```
CREATE FUNCTION my_plus (IN arg1 INT, IN arg2 INT)
     RETURNS INT
     DETERMINISTIC
     IGNORE NULL VALUES
     EXTERNAL NAME 'describe_my_plus@my_shared_lib'
```

この宣言は、my_plus が describe_my_plus という名前の記述子ルーチンを持つ my_shared_lib にある単純なスカラ UDF であることを示します。 UDF を実装するため、その動作には実際の C/C++ エントリ・ポイントが複数必要な場合があるので、このエントリ・ポイント・セットは CREATE FUNCTION 構文に直接的には含まれません。代わりに、CREATE FUNCTION 文の EXTERNAL NAME 句がこの UDF の記述子関数を指定します。 記述子関数は呼び出されると、次の項で詳しく定義されている記述子構造を返します。 この記述子構造には、この UDF の実装を具体的に表す必須またはオプションの関数ポインタが含まれています。

この宣言は、my_plus が 2 つの INT 引数を受け入れ、1 つの INT 結果値を返すことを示します。 関数が INT 以外の引数で呼び出され、引数を暗黙的に INT に変換できる場合は、関数が呼び出される前に変換が行われます。 関数が暗黙的に INT に変換できない引数で呼び出された場合は、変換エラーが生成されます。

さらに、この宣言は、関数が決定的であることを示しています。 一般的に、決定的関数は、同じ入力値が指定された場合、同じ結果値を返します。 これは、結果が、指定された引数値以外の外部情報や以前の呼び出しに関連する動作に依存しないことを意味します。 デフォルトでは、関数は決定的であると見なされ、この特性が CREATE 文から省略された場合も結果は同じになります。

上記の宣言の最後の部分は、IGNORE NULL VALUES 特性を示します。 ほぼすべての組み込みスカラ関数は、入力引数のいずれかが NULL である場合、NULL 結果値を返します。 IGNORE NULL VALUES は、my_plus 関数がこの規則に従うことを示します。このため、入力値のいずれかが NULL である場合、この UDF ルーチンは実際には呼び出されません。 この関数のデフォルトは RESPECT NULL VALUES なので、パフォーマンスを高めるには、この特性をこの UDF の宣言で指定する必要があります。 NULL 入力値を指定された場合に NULL 以外の結果を返す可能性のあるすべての関数は、デフォルトの RESPECT NULL VALUES 特性を使用する必要があります。

次のクエリの例では、SELECT リストに my_plus が同等の算術式とともに記述されています。

```
SELECT my_plus(t.x, t.y) AS x_plus_y_one, (t.x + t.y)AS x_plus_y_two
FROM t
WHERE t.z = 2
```

次の例では、my_plus が同じクエリ内のいくつかの場所で異なる方法で使用されています。

```
SELECT my_plus(t.x, t.y), count(*)
FROM t
WHERE t.z = 2
AND my_plus(t.x, 5) > 10
AND my_plus(t.y, 5) > 10
GROUP BY my_plus(t.x, t.y)
```

UDF の例： my_plus_counter 宣言

"my_plus_counter" の例は、1 つの整数引数を取り、内部整数使用カウンタに引数値を追加した結果を返す、単純な非決定的スカラ UDF です。 入力引数値が NULL である場合、結果は、使用カウンタの現在の値になります。

my_plus_counter 宣言

my_plus_counter もダイナミック・リンク・ライブラリ my_shared_lib 内にあると仮定すると、この例の宣言は次のようになります。

```
CREATE FUNCTION my_plus_counter (IN arg1 INT DEFAULT 0)
     RETURNS INT
     NOT DETERMINISTIC
     RESPECT NULL VALUES
     EXTERNAL NAME 'describe_my_plus_counter@my_shared_lib'
```

RESPECT NULL VALUES 特性は、入力引数値が NULL である場合にもこの関数が呼び出されることを意味します。 この特性は、my_plus_counter のセマンティックに次のものが含まれるため必須です。

- 引数が NULL である場合も増加する使用カウントの内部保持。
- NULL 引数を渡した場合の NULL 以外の値の結果。

RESPECT NULL VALUES はデフォルトであるため、この句が宣言で省略されている場合でも、結果は同じです。

IQ では、すべての非決定的関数の使用を制限しています。 非決定的関数は、最上位レベルのクエリ・ブロックの SELECT リスト、または UPDATE 文の SET 句でのみ使用できます。 サブクエリ内、WHERE 句、ON 句、GROUP BY 句、または HAVING 句内では使用できません。 この制限は、GETUID、NUMBER などの非決定的な組み込み関数と同様に、非決定的な UDF に適用されます。

上記の宣言の最後の部分は、入力パラメータの DEFAULT 修飾子です。 この修飾子はサーバに対して、この関数を引数なしで呼び出せることと、その場合は欠落し

た引数に対してサーバが自動的にゼロを指定することを通知します。 DEFAULT 値が指定された場合、その引数のデータ型に暗黙的に変換する必要があります。

次の例では、最初の SELECT リスト項目によって、各ローの t.x の値に実行中のカウンタが追加されます。 2 番目および 3 番目の SELECT リスト項目は、各ローの同じ値を NUMBER 関数として返します。

```
SELECT my_plus_counter(t.x),
     my_plus_counter(0),
     my_plus_counter(),
     NUMBER()
FROM t
```

スカラ UDF の定義

スカラ・ユーザ定義関数を定義するための C/C++ コードには、次の 4 つの要素が必要です。

- **extfnapiv3.h** – UDF インタフェース定義ヘッダ・ファイルが含まれます。
- **_evaluate_extfn** – 評価関数。 すべての評価関数は、2 つの引数を取ります。
  - スカラ UDF コンテキスト構造のインスタンス。コールバック関数ポインタと、UDF が UDF 固有データを格納するポインタのセットを含む UDF の各使用に対してユニークです。
  - データ構造へのポインタ。指定されたコールバックを介して引数値および結果値にアクセスできます。
- **a_v3_extfn_scalar** – スカラ UDF 記述子構造のインスタンス。評価関数へのポインタを含みます。
- **Descriptor function** – スカラ UDF 記述子構造へのポインタを返します。

次の 2 つのオプションの要素があります。

- **_start_extfn** – 通常、SQL の使用ごとに呼び出される初期化関数。 指定した場合、この関数へのポインタも、スカラ UDF 記述子構造内に配置する必要があります。 すべての初期化関数は、UDF の使用ごとにユニークなスカラ UDF コンテキスト構造へのポインタを 1 つの引数として取るように定義されます。 渡されるコンテキスト構造は、評価ルーチンに渡されるものと同じです。
- **_finish_extfn** – 通常、SQL の使用ごとに呼び出されるシャットダウン関数。 指定した場合、この関数へのポインタも、スカラ UDF 記述子構造内に配置する必要があります。 すべてのシャットダウン関数は、UDF の使用ごとにユニークなスカラ UDF コンテキスト構造へのポインタを 1 つの引数として取るように定義されます。 渡されるコンテキスト構造は、評価ルーチンに渡されるものと同じです。

スカラ UDF 記述子構造

スカラ UDF 記述子構造 a_v3_extfn_scalar は次のように定義されます。

```
typedef struct a_v3_extfn_scalar {    //
    // Metadata descriptor for a scalar UDF
    // supplied by the UDF library to the server
    // An optional pointer to an initialize function
    void (*_start_extfn)(a_v3_extfn_scalar_context * cntxt);
    //
    // An optional pointer to a shutdown function
    void (*_finish_extfn)(a_v3_extfn_scalar_context *    cntxt);
    //
    // A required pointer to a function that will be
    // called for each invocation of the UDF on a
    // new set of argument values
    void (*_evaluate_extfn)(a_v3_extfn_scalar_context * cntxt, void
*args_handle);
    // RESERVED FIELDS MUST BE INITIALIZED TO NULL
 void  *reserved1_must_be_null;
 void  *reserved2_must_be_null;
 void  *reserved3_must_be_null;
 void  *reserved4_must_be_null;
 void  *reserved5_must_be_null;
                    …

} a_v3_extfn_scalar;
```

定義済みのスカラ UDF ごとに、**a_v3_extfn_scalar** のインスタンスが 1 つあります。オプションの初期化関数が指定されていない場合、記述子構造内の対応する値は NULL ポインタです。 同様に、シャットダウン関数が指定されていない場合、記述子構造内の対応する値は NULL ポインタになります。

初期化関数は評価ルーチンへの呼び出しの前に少なくとも 1 回呼び出されます。シャットダウン関数は最後の評価呼び出しの後に少なくとも 1 回呼び出されます。通常、初期化関数とシャットダウン関数は、使用ごとに 1 回だけ呼び出されます。

スカラ UDF コンテキスト構造

スカラ UDF 記述子構造内で指定された各関数に渡される、スカラ UDF コンテキスト構造 **a_v3_extfn_scalar_context** は、次のように定義されます。

```
typedef struct a_v3_extfn_scalar_context {
//-------- Callbacks available via the context --------
//
  short (SQL_CALLBACK *get_value)(
    void           *arg_handle,
             a_sql_uint32    arg_num,
             an_extfn_value *value
             );
 short (SQL_CALLBACK *get_piece)(
             void *          arg_handle,
```

```
            a_sql_uint32    arg_num,
            an_extfn_value *value,
            a_sql_uint32    offset
            );
 short (SQL_CALLBACK *get_value_is_constant)(
            void *          arg_handle,
            a_sql_uint32    arg_num,
            a_sql_uint32 *  value_is_constant
            );
 short (SQL_CALLBACK *set_value)(
            void *          arg_handle,
            an_extfn_value *value,
            short              append
            );
 a_sql_uint32 (SQL_CALLBACK *get_is_cancelled)(
            a_v3_extfn_scalar_context * cntxt
            );
 short (SQL_CALLBACK *set_error)(
            a_v3_extfn_scalar_context * cntxt,
            a_sql_uint32    error_number,
            const char *    error_desc_string
            );
 void (SQL_CALLBACK *log_message)(
            const char *msg,
            short msg_length
               );
 short (SQL_CALLBACK *convert_value)(
               an_extfn_value   *input,
               an_extfn_value   *output
 //---------- Data available from the context ----------
 void * _user_data;   // read-write field
 //---------- For Server Internal Use Only -------------
 void  * _for_server_internal_use;
} a_v3_extfn_scalar_context;
```

スカラ UDF コンテキスト構造内の _user_data フィールドには、UDF が要求するデータを格納できます。 通常は、_start_extfn 関数により構造が割り付けられ、_finish_extfn 関数により割り付けが解除されるヒープが格納されます。

メモリを割り付けるとき、UDF ライブラリによって C++ の new 演算子が過負荷にならないようにします。 一部のプラットフォームでは、new 演算子が過負荷になるとグローバルな影響を生じ、サーバの適正な運用に悪影響を及ぼす場合があります。

スカラ UDF コンテキスト構造の残りの部分には、ユーザの UDF 関数ごとに使用するためにエンジンによって指定される、コールバック関数のセットが格納されます。これらのコールバック関数のほとんどは、簡単な結果値で成功ステータスを返します。戻り値 true は成功を示します。 UDF 実装が正しく記述されていれば、失敗ステータスを生じることはありませんが、開発中と、可能であれば指定された UDF ライブラリのすべてのデバッグ・ビルド中は、コールバックから返されるステータス値を確認することをおすすめします。 失敗は、UDF が取得するように定義され

ているよりも多くの引数を要求するなど、UDF 実装のコーディング・エラーに起因する可能性があります。

ほとんどのコールバックで使用される引数の一般的なセットには、次のものが含まれます。

- **arg_handle** – すべての形式の評価メソッドが受け取るポインタ。このポインタにより、UDF に渡される入力引数の値が使用可能になり、UDF の結果値が設定されます。
- **arg_num** – どの入力引数にアクセスしているかを示す整数。 入力引数は 1 から始まり、昇順で左から右に番号が付けられます。
- **cntxt** – サーバがすべての UDF エントリ・ポイントに渡す、コンテキスト構造へのポインタ。
- **value** – サーバからの入力引数値の取得または関数の結果値の設定に使用される、an_extfn_value 構造のインスタンスへのポインタ。 an_extfn_value 構造の形式は、次のとおりです。

```
typedef struct an_extfn_value {
void * data;
a_SQL_uint32 piece_len;
union {
        a_SQL_uint32 total_len;
        a_SQL_uint32 remain_len;
} len;
a_SQL_data_type type;
} an_extfn_value;
```

UDF の例： my_plus 定義

my_plus の定義の例を、次に示します。

my_plus 定義

この UDF には初期化関数やシャットダウン関数が必要ないので、記述子構造内のこれらの値は 0 に設定されます。記述子関数名は、宣言で使用される EXTERNAL NAME と一致します。 この評価メソッドでは、引数が INT として宣言されているため、引数のデータ型をチェックしません。

```
#include "extfnapiv3.h"
#include <stdlib.h>


//  A simple deterministic scalar UDF that just adds
//  two integer arguments and then returns the result.
//
//  Corresponding SQL declaration:
//
//    CREATE FUNCTION my_plus(IN arg1 INT, IN arg2 INT)
//                                           RETURNS INT
//                                           DETERMINISTIC
//                                    IGNORE NULL VALUES
```

```
//                                                  EXTERNAL NAME
'my_plus@libudfex'
//
#if defined __cplusplus
extern "C" {
#endif

static void my_plus_evaluate(a_v3_extfn_scalar_context *cntxt,
                      void *arg_handle)
{
  an_extfn_value  arg;
  an_extfn_value  outval;
  a_sql_int32 arg1, arg2, result;

  //  Get first argument
  (void) cntxt->get_value( arg_handle, 1, &arg );
  if (arg.data == NULL)
  {
      return;
  }
  arg1 = *((a_sql_int32 *)arg.data);

  //  Get second argument
  (void) cntxt->get_value( arg_handle, 2, &arg );
  if (arg.data == NULL)
  {
      return;
  }
  arg2 = *((a_sql_int32 *)arg.data);

  //  Set the result value
  outval.type = DT_INT;
  outval.piece_len = sizeof(a_sql_int32);
  result = arg1 + arg2;
  outval.data = &result;
  cntxt->set_value( arg_handle, &outval, 0 );
}


static a_v3_extfn_scalar my_plus_descriptor = {
    0,
    0,
    &my_plus_evaluate,
    0,            // Reserved - initialize to NULL
    0,            // Reserved - initialize to NULL
    0,            // Reserved - initialize to NULL
    0,            // Reserved - initialize to NULL
    0,            // Reserved - initialize to NULL
        NULL                    // _for_server_internal_use
};


a_v3_extfn_scalar *my_plus()
{
  return &my_plus_descriptor;
}
```

```
#if defined __cplusplus
}
#endif
```

UDF の例： my_plus _counter 定義

この例では、入力引数値が NULL かどうかを確認するために、引数値ポインタ・データをチェックします。 また、初期化関数とシャットダウン関数も含まれ、それぞれが複数の呼び出しを受け入れます。

my_plus _counter 定義

```
#include "extfnapiv3.h"
#include <stdlib.h>

//  A simple non-deterministic scalar UDF that adds
//  an internal integer usage counter to its integer
//  argument and then returns the resulting integer.
//
//  Here, the start function creates a little structure for
//  the counter, and then the finish function deallocates it.
//
//  Corresponding SQL declaration:
//
//          CREATE FUNCTION plus_counter(IN arg1 INT)
//                      RETURNS INT
//                      NOT DETERMINISTIC
//                      RESPECT NULL VALUES
//                      EXTERNAL NAME 'my_plus_counter@libudfex'


typedef struct my_counter {
  a_sql_int32 _counter;
} my_counter;

#if defined __cplusplus
extern "C" {
#endif

static void my_plus_counter_start(a_v3_extfn_scalar_context *cntxt)
{
  my_counter *cptr = (my_counter *)cntxt->_user_data;
  //  If we have not already allocated the
  //  counter structure, then do so now
  if (!cptr) {
    cptr = (my_counter *)malloc(sizeof(my_counter));
    cntxt->_user_data = cptr;
  }
  cptr->_counter = 0;
}


static void my_plus_counter_finish(a_v3_extfn_scalar_context *cntxt)
{
```

```
  //  If we still have an allocated the
  //  counter structure, then free it now
  if (cntxt->_user_data) {
    free(cntxt->_user_data);
    cntxt->_user_data = 0;
  }
}


static void my_plus_counter_evaluate(a_v3_extfn_scalar_context
*cntxt,
                                     void *arg_handle)
{
  an_extfn_value  arg;
  an_extfn_value  outval;
  a_sql_int32 arg1, result;

  //  Increment the usage counter
  my_counter *cptr = (my_counter *)cntxt->_user_data;
  cptr->_counter += 1;

  //  Get the one argument
  (void) cntxt->get_value( arg_handle, 1, &arg );
  if (!arg.data) {
    // argument value was NULL;
    arg1 = 0;
  } else {
    arg1 = *((a_sql_int32 *)arg.data);
  }

  outval.type = DT_INT;
  outval.piece_len = sizeof(a_sql_int32);
  result = arg1 + cptr->_counter;
  outval.data = &result;
  cntxt->set_value( arg_handle, &outval, 0 );
}


static a_v3_extfn_scalar my_plus_counter_descriptor =
    {   &my_plus_counter_start,
        &my_plus_counter_finish,
        &my_plus_counter_evaluate,
        NULL,                   // Reserved - initialize to NULL
        NULL,                   // Reserved - initialize to NULL
        NULL,                   // Reserved - initialize to NULL
        NULL,                   // Reserved - initialize to NULL
        NULL,                   // Reserved - initialize to NULL
        NULL,                   // _for_server_internal_use
};


a_v3_extfn_scalar *my_plus_counter()
{
  return &my_plus_counter_descriptor;
}
```

```
#if defined __cplusplus
}
#endif
```

ユーザ定義の集合関数

Sybase IQ では、ユーザ定義の集合関数 (UDAF) をサポートしています。 SUM 関数は、組み込み集合関数の一例です。 単純な集計関数は、一連の引数値から 1 つの結果値を生成します。 SUM 集計を使用できる場所であればどこでも使用できる UDAF を記述できます。

注意： この章で説明する集合 UDF の例は IQ サーバとともにインストールされ、`$IQDIR15/samples/udf` にある .cxx ファイルに記載されています。 `$IQDIR15/lib64/libudfex` ダイナミック・リンク・ライブラリからも参照できます。

UDAF の宣言

集合 UDF は、スカラ UDF よりも作成機能がより強力で複雑です。

注意： Sybase Central でユーザ定義関数宣言を作成 (14 ページ) することもできます。

UDAF を実装する際、次のことを決定する必要があります。

- RANK のように、データ・セット全体またはパーティションでのみオンライン分析処理 (OLAP) スタイルの集合として動作するかどうか。
- 1 つの集合として動作するか、または SUM のように OLAP スタイルの集合として動作するか。
- グループ全体で 1 つの集合としてのみ動作するかどうか。

UDAF の宣言と定義には、使用法における上記の決定が反映されます。

IQ のユーザ定義の集合関数を作成する構文は、次のとおりです。

```
aggregate-udf-declaration:
    CREATE AGGREGATE FUNCTION [ owner.]function-name
          ( [ parameter , ... ] )
    RETURNS data-type
        [ aggregate-routine-characteristics ... ]
    EXTERNAL NAME library-and-entry-point-name-string

parameter:
    param-name data-type [ DEFAULT value ]

aggregate-routine-characteristics:
      DUPLICATE { SENSITIVE | INSENSITIVE }
        -- is the server allowed to eliminate DISTINCT
    | SQL SECURITY {INVOKER | DEFINER}
    | OVER restrict
```

```
    | ORDER order-restrict
        -- Must the window-spec contain an ORDER BY?
    | WINDOW FRAME
        { { ALLOWED | REQUIRED }
            [ window-frame-constraints ... ]
          | NOT ALLOWED }
    | ON EMPTY INPUT RETURNS { NULL | VALUE }
  -- Call or skip function on NULL inputs

window-frame-constraints:
      VALUES { [ NOT ] ALLOWED }
    | CURRENT ROW { REQUIRED | ALLOWED }
    | [ UNBOUNDED ] { PRECEDING | FOLLOWING } restrict

restrict:   { [ NOT ] ALLOWED } | REQUIRED

order-restrict:
{ NOT ALLOWED | SENSITIVE | INSENSITIVE | REQUIRED
```

戻り値のデータ型、引数、データ型、およびデフォルト値の処理は、スカラ UDF 定義の場合と同じです。

UDAF を 1 つの集合として使用できる場合は、DISTINCT 修飾子とともに使用できる可能性があります。 UDAF 宣言の DUPLICATE 句は、次のことを決定します。

- 結果が重複に応じて変わるため、UDAF が呼び出される前に重複した値が削除対象と見なされるかどうか (組み込みの "COUNT(DISTINCT T.A)" の場合など)。または
- 重複が存在しても結果が変わらないかどうか ("MAX(DISTINCT T.A)" の場合など)。

DUPLICATE INSENSITIVE オプションにより、オプティマイザは、結果に影響を与えずに重複の削除を検討でき、クエリの実行方法を選択できます。 UDAF は、重複の発生を前提に記述する必要があります。 重複の削除が必要な場合、_next_value_extfn 呼び出しのセットを開始する前に、サーバは削除を実行します。

スカラ UDF 構文の一部ではない、残りの句の大部分を使用して、この関数の使用を指定できます。 デフォルトでは、UDAF は、任意のウィンドウ・フレームで単純な集合としても OLAP スタイルの集合としても使用できると見なされます。

UDAF を単純な集合関数としてのみ使用するには、次の構文を使用して宣言します。

```
OVER NOT ALLOWED
```

この集合を OLAP スタイルの集合として使用しようとすると、エラーが生成されます。

UDAF で OVER 句を使用できる、または必須である場合、UDF 定義者は、"ORDER" とそれに続く制限タイプを指定することで、OVER 句内の ORDER BY 句の存在に

対して制限を指定できます。 ウィンドウ順序付けの制限タイプは、次のとおりです。

- **REQUIRED** – ORDER BY の指定は必須であり、削除できません。
- **SENSITIVE** – ORDER BY を指定するかどうかは選択できますが、指定した場合は削除できません。
- **INSENSITIVE** – ORDER BY を指定するかどうかは選択できますが、サーバは順序付けを削除して効率を高めることができます。
- **NOT ALLOWED** – ORDER BY を指定できません。

組み込みの RANK のように、順序付けされたセット全体またはパーティションに対して、OLAP スタイルの集合としてのみ UDAF を宣言するには、次のような構文を使用します。

```
OVER REQUIRED
ORDER REQUIRED
WINDOW FRAME NOT ALLOWED
```

UNBOUNDED PRECEDING のデフォルトのウィンドウ・フレームを CURRENT ROW に対して使用し、OLAP スタイルの集合としてのみ UDAF を宣言するには、次のような構文を使用します。

```
OVER REQUIRED
ORDER REQUIRED
WINDOW FRAME ALLOWED
    RANGE NOT ALLOWED
    UNBOUNDED PRECEDING REQUIRED
    CURRENT ROW REQUIRED
    FOLLOWING NOT ALLOWED
```

すべてのさまざまなオプションおよび制限のセットのデフォルトは、次のとおりです。

```
DUPLICATE SENSITIVE
SQL SECURITY DEFINER
OVER ALLOWED
ORDER SENSITIVE
WINDOW FRAME ALLOWED
CURRENT ROW ALLOWED
PRECEDING ALLOWED
UNBOUNDED PRECEDING ALLOWED
FOLLOWING ALLOWED
UNBOUNDED FOLLOWING ALLOWED
```

SQL Security

INVOKER (関数を呼び出しているユーザ) または DEFINER (関数を所有しているユーザ) のどちらとして関数が実行されるかを定義します。 デフォルトは DEFINER です。

SQL SECURITY INVOKER が指定された場合、プロシージャを呼び出すユーザごとに注釈が必要であるため、より多くのメモリが使用されます。 また、**SQL**

SECURITY INVOKER が指定された場合、ユーザ名と INVOKER の両方で名前の決定が行われます。 適切な所有者名で、すべてのオブジェクト名 (テーブル、プロシージャなど) を修飾します。

External Name

EXTERNAL NAME 句を使用する関数は、外部ライブラリにある関数への呼び出しのラッパです。 **EXTERNAL NAME** を使用する関数は、**RETURNS** 句の後に他の句を持つことができません。 ライブラリ名にはファイル拡張子が付く場合があります。この拡張子は通常、Windows では `.dll`、UNIX では `.so` です。 拡張子が付いていなければ、ソフトウェアがプラットフォーム固有のデフォルトのファイル拡張子をライブラリに付加します。

テンポラリ関数では **EXTERNAL NAME** 句はサポートされません。『SQL Anywhere サーバ– プログラミング』の「プロシージャからの外部ライブラリの呼び出し」を参照してください。

IQ サーバは、UDF ライブラリの場所を含むライブラリ・ロード・パスで起動できます。 Unix variant では、start_iq 起動スクリプトの LD_LIBRARY_PATH を変更することで実行できます。 すべての UNIX variant で LD_LIBRARY_PATH が一般的に使用されますが、HP では SHLIB_PATH が、AIX では LIB_PATH が優先的に使用されます。

Unix プラットフォームでは、LD_LIBRARY_PATH が使用されない場合に、外部名の指定に完全修飾名を含めることができます。 Windows プラットフォームでは、完全修飾名は使用できず、ライブラリ検索パスは PATH 環境変数で定義されます。

注意： スカラ・ユーザ定義関数とユーザ定義の集合関数は、更新可能なカーソルではサポートされません。

UDAF の例： my_sum 宣言

"my_sum" の例は組み込みの SUM に似ていますが、整数でのみ動作する点が異なります。

my_sum 宣言

SUM と同様に my_sum は任意のコンテキストで使用できるので、その宣言は比較的簡潔です。

```
CREATE AGGREGATE FUNCTION my_sum(IN arg1 INT)
    RETURNS BIGINT
    ON EMPTY INPUT RETURNS NULL
    EXTERNAL NAME 'describe_my_integer_sum@my_shared_lib'
```

さまざまな使用上の制限はすべてデフォルトで ALLOWED で、これにより、集合関数が許容される SQL 文のどの場所でもこの関数を使用できます。

使用上の制限がない場合は、次に示すように、my_sum は、ローのセット全体で単純な集合として使用できます。

```
SELECT MIN(t.x), COUNT (*), my_sum(t.y)
FROM t
```

使用上の制限がない場合、my_sum は、GROUP BY 句の指定に従って、グループごとに計算される単純な集合としても使用できます。

```
SELECT t.x, COUNT(*), my_sum(t.y)
FROM t
GROUP BY t.x
```

使用上の制限がないため、my_sum は、次の累積合計の例が示すように、OVER 句とともに OLAP スタイルの集合として使用できます。

```
SELECT t.x,
    my_sum(t.x)
        OVER (ORDER BY t.x ROWS BETWEEN UNBOUNDED PRECEDING AND
CURRENT ROW)
        AS cumulative_x,
    COUNT(*)
FROM t
GROUP BY t.x
ORDER BY t.x
```

UDAF の例： my_bit_xor 宣言

"my_bit_xor" の例は SQL Anywhere (SA) の組み込みの BIT_XOR に似ていますが、符号なし整数でのみ動作する点が異なります。

my_bit_xor 宣言

結果の宣言は次のようになります。

```
CREATE AGGREGATE FUNCTION my_bit_xor(IN arg1 UNSIGNED    INT)
 RETURNS UNSIGNED INT
    ON EMPTY INPUT RETURNS NULL
    EXTERNAL NAME 'describe_my_bit_xor@my_shared_lib'
```

my_sum の例と同様に、my_bit_xor には関連する使用上の制限がないため、任意のウィンドウで単純な集合としても、OLAP スタイルの集合としても使用できます。

UDAF の例： my_bit_or 宣言

"my_bit_or" の例は、SA の組み込みの BIT_OR に似ていますが、符号なし整数でのみ動作する点が異なり、単純な集合としてのみ使用できます。

my_bit_or 宣言

結果の宣言は次のようになります。

```
CREATE AGGREGATE FUNCTION my_bit_or(IN arg1 UNSIGNED INT)
     RETURNS UNSIGNED INT
```

```
    ON EMPTY INPUT RETURNS NULL
    OVER NOT ALLOWED
    EXTERNAL NAME 'describe_my_bit_or@ my_shared_lib'
```

my_bit_xor の例とは異なり、宣言の OVER NOT ALLOWED フレーズは、この関数を単純な集合としてのみ使用するように制限しています。 この使用上の制限により、my_bit_or は、ローのセット全体で単純な集合として、または、次の例に示すように GROUP BY 句の指定に従ってグループごとに計算される単純な集合としてのみ使用できます。

```
SELECT t.x, COUNT(*), my_bit_or(t.y)
FROM t
GROUP BY t.x
```

UDAF の例： my_interpolate 宣言

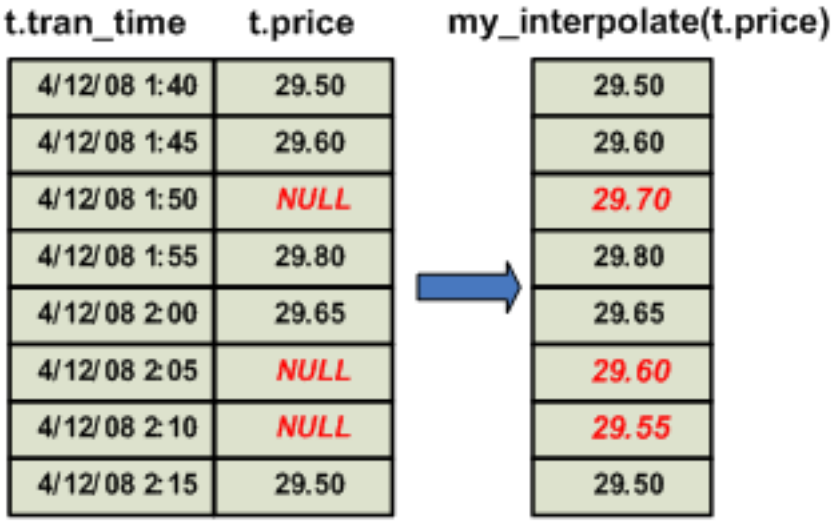
"my_interpolate" の例は、隣接する NULL 値のセットに対して、各方向で最も近い NULL 以外の値への線形補間を実行することにより、シーケンス内の NULL で示された欠落値に入力しようとする OLAP スタイルの UDAF です。

my_interpolate 宣言

特定のローの入力が NULL でない場合、そのローの結果は入力値と同じです。

次の表は、小さな入力ロー・セットにおける my_interpolate の動作を示しています。

図 1：my_interpolate の結果



相応なコストで動作するには、固定幅のローベースのウィンドウを使用して my_interpolate を実行する必要がありますが、ユーザは、隣接する NULL 値の予想

される最大数に基づいて、ウィンドウの幅を設定できます。 この関数は、倍精度浮動小数点数値のセットを取得し、double 型の結果セットを生成します。

結果の UDAF 宣言は、次のようになります。

```
CREATE AGGREGATE FUNCTION my_interpolate (IN arg1 DOUBLE)
RETURNS DOUBLE
    OVER REQUIRED
    WINDOW FRAME REQUIRED
        RANGE NOT ALLOWED
        PRECEDING REQUIRED
        UNBOUNDED PRECEDING NOT ALLOWED
        FOLLOWING REQUIRED
        UNBOUNDED FOLLOWING NOT ALLOWED
    EXTERNAL NAME 'describe_my_interpolate@my_shared_lib'
```

OVER REQUIRED は、この関数が単純な集合として使用できないことを意味します (使用された場合、ON EMPTY INPUT は関係ありません)。

WINDOW FRAME の部分は、この関数を使用したときに現在のローから前後に拡張する固定幅のローベースのウィンドウを使用する必要があることを指定します。これらの使用上の制限により、my_interpolate は、次のように、OVER 句とともに OLAP スタイルの集合としてのみ使用できます。

```
SELECT t.x,
    my_interpolate(t.x)
    OVER (ORDER BY t.x ROWS BETWEEN 5 PRECEDING AND 5 FOLLOWING)
        AS x_with_gaps_filled,
    COUNT(*)
FROM t
GROUP BY t.x
ORDER BY t.x
```

my_interpolate の OVER 句内では、ローの前後の正確な数はさまざまであり、オプションで PARTITION BY を使用できます。それ以外の場合、宣言で使用上の制限が指定されているとき、ローは、上記の例と同様である必要があります。

集計 UDF の定義

ユーザ定義の集合関数を定義するための C/C++ コードには、8 つの要素が必要です。

必要な 8 つの要素は、次のとおりです。

- **extfnapiv3.h** – UDF インタフェース定義ヘッダ・ファイル。
- **_start_extfn** – SQL の使用ごとに呼び出される初期化関数。 すべての初期化関数は、UDAF の使用ごとにユニークな集合 UDF コンテキスト構造へのポインタを 1

つの引数として取ります。 渡されるコンテキスト構造は、その使用で指定されるすべての関数に渡されるものと同じです。

- **_finish_extfn** – SQL の使用ごとに呼び出されるシャットダウン関数。 すべてのシャットダウン関数は、UDAF の使用ごとにユニークな UDAF コンテキスト構造へのポインタを 1 つの引数として取ります。
- **_reset_extfn** – 新しいグループ、新しいパーティションの開始ごとに呼び出されるリセット関数。必要に応じて、各ウィンドウ動作の開始時にも呼び出されます。 すべてのリセット関数は、UDAF の使用ごとにユニークな UDAF コンテキスト構造へのポインタを 1 つの引数として取ります。
- **_next_value_extfn** – 新しい入力引数セットごとに呼び出される関数。
  _next_value_extfn は次の 2 つの引数を取ります。

  - UDAF コンテキストへのポインタ。
  - args_handle。

  スカラ UDF の場合と同様に、arg_handle は、実際の引数値にアクセスするために指定されたコールバック関数ポインタとともに使用されます。
- **_evaluate_extfn** – スカラ UDF 評価関数と同様の評価関数。 すべての評価関数は、2 つの引数を取ります。

  - UDAF コンテキスト構造へのポインタ。
  - args_handle。
- **a_v3_extfn_aggregate** – 集合 UDF 記述子構造のインスタンス。この UDF について指定されたすべての関数へのポインタを含みます。
- **Descriptor function** – 集合 UDF 記述子構造へのポインタを返す記述子関数。

必須要素に加え、特定の使用状況のために最適化されたアクセスを可能にする、いくつかのオプションの要素があります。

- **_drop_value_extfn** – オプションの関数ポインタ。移動ウィンドウ・フレームからあふれた引数値の入力セットごとに呼び出されます。 この関数は、集合の結果を設定しません。 入力引数値にアクセスするには、get_value コールバック関数を使用します。また、必要に応じて、get_piece コールバック関数を繰り返し呼び出します。
  次の場合は、関数ポインタを NULL ポインタに設定します。

  - この集合をウィンドウ・フレームで使用できない場合。
  - 方法によっては集合に可逆性がない場合。
  - パフォーマンスの最適化が重要でない場合。

  この関数が指定されておらず、ユーザが移動ウィンドウを指定した場合、ウィンドウ・フレームが移動するたびにリセット関数が呼び出され、next_value 関数の呼び出しによりウィンドウ内の各ローが挿入され、最後に評価関数が呼び出されます。

この関数が指定された場合、ウィンドウ・フレームが移動するたびに、ウィンドウ・フレームからあふれたローごとにこの drop_value 関数が呼び出され、ウィンドウ・フレームに追加されたローごとに next_value 関数が呼び出され、最後に、評価関数が呼び出されて集合の結果が生成されます。

- **_evaluate_cumulative_extfn** – 引数値の新しい入力セットごとに呼び出すことのできる、オプションの関数ポインタ。 この関数が指定され、UNBOUNDED PRECEDING から CURRENT ROW にまたがるローベースのウィンドウ・フレームで使用される場合、next_value 関数の呼び出しの代わりにこの関数が呼び出され、直後に評価関数が呼び出されます。
  この関数は、set_value コールバックを介して、集合の結果を設定します。 入力引数値のセットへのアクセスは、通常の get_value コールバック関数を介して行われます。 次の場合は、この関数ポインタを NULL ポインタに設定します。
  - この集合がこの方法で使用されない場合。
  - パフォーマンスの最適化が重要でない場合。
- **_next_subaggregate_extfn** – 並列実行によりこの集合の一部の使用を最適化するために、_evaluate_superaggregate_extfn とともに使用する、オプションのコールバック関数ポインタ。
  一部の集合は、単純な集合として使用される場合 (すなわち、OVER 句を伴う OLAP スタイルの集合として使用されない場合)、最初の中間集合結果セットの生成により、パーティションに分割できます。この場合、中間結果はそれぞれ、入力ローの分離サブセットから計算されます。
  このような分割可能な集合の例を次に示します。
  - SUM。最終 SUM は、入力ローの各分離サブセットの SUM を実行し、サブ SUM に対して SUM を実行することにより計算されます。
  - COUNT(*)。最終 COUNT は、入力ローの各分離サブセットの COUNT を実行し、各パーティションの COUNT に対して SUM を実行することにより計算されます。

  集合が上記の条件を満たす場合、サーバは、集合の並列計算を実行できます。集合 UDF については、_next_subaggregate_extfn 関数ポインタと _evaluate_superaggregate_extfn ポインタの両方が指定されている場合にのみ、この並列最適化を適用できます。
  _reset_extfn 関数は集合の最終結果を設定しません。また、この関数は、定義により、集合 UDF の定義済みの戻り値と同じデータ型である入力引数値を 1 つだけ持ちます。
  サブ集合入力値へのアクセスは、通常の get_value コールバック関数を介して行われます。 サブ集合とスーパー集合との間の直接的なやり取りはできません。このようなやり取りはすべてサーバが処理します。 サブ集合とスーパー集合は、コンテキスト構造を共有しません。 代わりに、個々のサブ集合は、非分割

集合と同様に扱われます。 独立したスーパー集合は、次のようなパターン呼び出しを認識します。

```
_start_extfn
_reset_extfn
_next_subaggregate_extfn (repeated 0 to N times)
_evaluate_superaggregate_extfn
_finish_extfn
```

次のようなパターンも認識します。

```
_start_extfn
_reset_extfn
_next_subaggregate_extfn (repeated 0 to N times)
_evaluate_superaggregate_extfn
_reset_extfn
_next_subaggregate_extfn (repeated 0 to N times)
_evaluate_superaggregate_extfn
_reset_extfn
_next_subaggregate_extfn (repeated 0 to N times)
_evaluate_superaggregate_extfn
_finish_extfn
```

evaluate_superaggregate_extfn および _next_subaggregate_extfn のどちらも指定されていない場合、UDAF は制限され、GROUP BY CUBE または GROUP BY ROLLUP を含むクエリ・ブロック内の単純な集合としては許容されません。

- **_evaluate_superaggregate_extfn** – 並列化により単純な集合の一部の使用を最適化するために、_next_subaggregate_extfn とともに使用する、オプションのコールバック関数ポインタ。 _evaluate_superaggregate_extfn は、分割集合の結果を返すために呼び出されます。 結果値は、通常の set_value コールバック関数を使用して、a_v3_extfn_aggregate_context 構造からサーバに送信されます。

集合 UDF 記述子構造

集合 UDF 記述子構造は、次の要素で構成されています。

- **typedef struct a_v3_extfn_aggregate** – ライブラリによって指定される集合 UDF 関数のメタデータ記述子。
- **_start_extfn** – 引数のみが a_v3_extfn_aggregate_context へのポインタである、初期化関数への必須ポインタ。 通常は、一部の構造の割り付けと、a_v3_extfn_aggregate_context 内の _user_data フィールドへのアドレスの格納に使用されます。 _start_extfn は、a_v3_extfn_aggregate_context ごとに 1 回だけ呼び出されます。

  ```
  void (*_start_extfn)(a_v3_extfn_aggregate_context *);
  ```
- **_finish_extfn** – 引数のみが a_v3_extfn_aggregate_context へのポインタである、シャットダウン関数への必須ポインタ。 通常は、a_v3_extfn_aggregate_context 内の _user_data フィールドにアドレスが格納された、一部の構造の割り付け解除に使用されます。 _finish_extfn は、a_v3_extfn_aggregate_context ごとに 1 回だけ呼び出されます。

```
void (*_finish_extfn)(a_v3_extfn_aggregate_context *);
```

- **_reset_extfn** – 引数のみが a_v3_extfn_aggregate_context へのポインタである、start-of-new-group 関数への必須ポインタ。 通常は、a_v3_extfn_aggregate_context 内の _user_data フィールドにアドレスが格納された構造の一部の値をリセットするために使用されます。 _reset_extfn は繰り返し呼び出されます。

```
void (*_reset_extfn)(a_v3_extfn_aggregate_context *);
```

- **_next_value_extfn** – 引数値の新しい入力セットごとに呼び出される、必須の関数ポインタ。 この関数は、集合の結果を設定しません。 入力引数値にアクセスするには、get_value コールバック関数を使用します。また、必要に応じて、get_piece コールバック関数を繰り返し呼び出します。これは、piece_len が total_len より小さい場合に必要です。

```
void (*_next_value_extfn)(a_v3_extfn_aggregate_context *cntxt,
void *args_handle);
```

- **_evaluate_extfn** – 結果の集合結果値を返すために呼び出される、必須の関数ポインタ。 _evaluate_extfn は、set_value コールバック関数を使用してサーバに送信されます。

```
void (*_evaluate_extfn)(a_v3_extfn_aggregate_context *cntxt, void
*args_handle);
```

- **_drop_value_extfn**– 移動ウィンドウ・フレームからあふれた引数値の入力セットごとに呼び出される、オプションの関数ポインタ。 この関数を使用して集合の結果を設定しないでください。 入力引数値にアクセスするには、get_value コールバック関数を使用します。また、必要に応じて、get_piece コールバック関数を繰り返し呼び出します。ただし、アクセスは、piece_len が total_len より小さい場合に必要です。 次の場合は、_drop_value_extfn を NULL ポインタに設定します。

  - 集合をウィンドウ・フレームで使用できない場合。
  - 方法によっては集合に可逆性がない場合。
  - パフォーマンスの最適化が重要でない場合。

  この関数が指定されておらず、ユーザが移動ウィンドウを指定した場合、ウィンドウ・フレームが移動するたびにリセット関数が呼び出され、next_value 関数の呼び出しによりウィンドウ内の各ローが挿入されます。 最後に評価関数が呼び出されます。
  一方、この関数が指定された場合、ウィンドウ・フレームが移動するたびに、ウィンドウ・フレームからあふれたローごとにこの drop_value 関数が呼び出され、ウィンドウ・フレームに追加されたローごとに next_value 関数が呼び出されます。 最後に、評価関数が呼び出されて集合結果が生成されます。

```
void (*_drop_value_extfn)(a_v3_extfn_aggregate_context    *cntxt,
void *args_handle);
```

- **_evaluate_cumulative_extfn** – 引数値の新しい入力セットごとに呼び出される、オプションの関数ポインタ。 この関数が指定され、UNBOUNDED PRECEDING

から CURRENT ROW にまたがるローベースのウィンドウ・フレームで使用される場合、next_value の代わりにこの関数が呼び出され、直後に評価関数が呼び出されます。 _evaluate_cumulative_extfn は、set_value コールバックを介して、集合の結果を設定します。 入力引数値にアクセスするには、get_value コールバック関数を使用します。また、必要に応じて、get_piece コールバック関数を繰り返し呼び出します。これは、piece_len が total_len より小さい場合に必要です。

```
void (*_evaluate_cumulative_extfn)(a_v3_extfn_aggregate_context
*cntxt, void *args_handle);
```

- **_next_subaggregate_extfn** – 並列および部分的な結果集合により集合の一部の使用を最適化するために、_evaluate_superaggregate_extfn 関数とともに使用する (使用法によっては、_drop_subaggregate_extfn 関数も使用)、オプションのコールバック関数ポインタ。
  一部の集合は、単純な集合として使用された場合 (すなわち、OVER 句を伴う OLAP スタイルの集合として使用されない場合)、最初の中間集合結果のセットの生成により、パーティションに分割できます。この場合、中間結果はそれぞれ、入力ローの分離サブセットから計算されます。 このような分割可能な集合の例を次に示します。
  - SUM。最終 SUM は、入力ローの各分離サブセットの SUM を実行し、サブ SUM に対して SUM を実行することにより計算されます。
  - COUNT(*)。最終 COUNT は、入力ローの各分離サブセットの COUNT を実行し、各パーティションの COUNT に対して SUM を実行することにより計算されます。

  集合が上記の条件を満たす場合、サーバは、集合の並列計算を実行できます。集合 UDF については、_next_subaggregate_extfn コールバックと _evaluate_superaggregate_extfn コールバックの両方が指定されている場合にのみ、この最適化を適用できます。 この使用パターンには、_drop_subaggregate_extfn は不要です。
  同様に、RANGE ベースの OVER 句とともに集合を使用できるとき、_next_subaggregate_extfn、_drop_subaggregate_extfn、および _evaluate_superaggregate_extfn 関数がすべて UDAF 実装で指定されている場合にのみ、最適化を適用できます。
  _next_subaggregate_extfn は集合の最終結果を設定しません。また、この関数は、定義により、集合 UDF の戻り値と同じデータ型である入力引数値を 1 つだけ持ちます。 サブ集合入力値にアクセスするには、get_value コールバック関数を使用します。また、必要に応じて、get_piece コールバック関数を繰り返し呼び出します。これは、piece_len が total_len より小さい場合に必要です。
  サブ集合とスーパー集合との間の直接的なやり取りはできません。このようなやり取りはすべてサーバが処理します。 サブ集合とスーパー集合は、コンテキスト構造を共有しません。 個々のサブ集合は、非分割集合と同様に扱われます。独立したスーパー集合は、次のようなパターン呼び出しを認識します。

```
        _start_extfn
  _reset_extfn
        _next_subaggregate_extfn (repeated 0 to N times)
        _evaluate_superaggregate_extfn
        _finish_extfn
```

```
void (*_next_subaggregate_extfn)(a_v3_extfn_aggregate_context
*cntxt, void *args_handle);
```

- **_drop_subaggregate_extfn** – 部分的な集合により、RANGE ベースの OVER 句を含む一部の使用を最適化するために、_next_subaggregate_extfn や _evaluate_superaggregate_extfn とともに使用する、オプションのコールバック関数ポインタ。 _drop_subaggregate_extfn は、共通の順序付けキー値を共有するロー・セットが移動ウィンドウからまとめてあふれたときに必ず呼び出されます。 この最適化は、3 つの関数がすべて UDF で指定されている場合にのみ適用されます。

```
void (*_drop_subaggregate_extfn)(a_v3_extfn_aggregate_context
*cntxt,    void *args_handle);
```

- **_evaluate_superaggregate_extfn** – 並列実行により一部の使用を最適化するために、_next_subaggregate_extfn とともに使用する (場合によっては、_drop_subaggregate_extfn も使用)、オプションのコールバック関数ポインタ。 前述したように、_evaluate_superaggregate_extfn は、分割集合の結果を返すときに呼び出されます。 結果値は、set_value コールバック関数を使用して、a_v3_extfn_aggregate_contex 構造からサーバに送信されます。

```
void (*_evaluate_superaggregate_extfn)
(a_v3_extfn_aggregate_context *cntxt, void *args_handle);
```

- **NULL fields** – フィールドを NULL に初期化します。

```
void *    reserved1_must_be_null;
void *    reserved2_must_be_null;
void *    reserved3_must_be_null;
void *    reserved4_must_be_null;
void *    reserved5_must_be_null;
```

- **Status indicator bit field** – 集合の処理に使用するアルゴリズムをエンジンが最適化するためのインジケータを含む、1 ビットのフィールド。

```
a_sql_uint32  indicators;
```

- **_calculation_context_size** – サーバが UDF の計算コンテキストごとに割り付けるバイト数。 サーバは、クエリ処理時に複数の計算コンテキストを割り付けることができます。 現在アクティブなグループ・コンテキストは、a_v3_extfn_aggregate_context_user_calculation_context で使用できます。

```
short _calculation_context_size;
```

- **_calculation_context_alignment** – ユーザの計算コンテキストの配置要件を指定します。 有効な値は、1、2、4、または 8 です。

```
short _calculation_context_alignment;
```

- **External memory requirments** – 次のフィールドにより、オプティマイザは、外部割り付けメモリのコストを検討できます。 これらの値を使用し、オプティマ

イザは、複数の同時計算をどの程度実行できるかを検討できます。 これらのカウンタの見積もりは、通常のローまたはグループに基づいて行い、最大値にならないようにします。 UDF によりメモリが割り付けられていない場合は、これらのフィールドをゼロに設定します。

- external_bytes_per_group – 各集合の開始時に 1 つのグループに割り付けられるメモリの容量。 通常、これは reset() の呼び出し時に割り付けられるメモリです。
- external_bytes_per_row – グループの各ローについて、UDF により割り付けられるメモリの容量。 通常、これは next_value() 時に割り付けられるメモリの容量です。

```
double            external_bytes_per_group;
double            external_bytes_per_row;
```

- **Reserved fields for future use** – 次のフィールドを初期化します。

```
a_sql_uint64      reserved6_must_be_null;
a_sql_uint64      reserved7_must_be_null;
a_sql_uint64      reserved8_must_be_null;
a_sql_uint64      reserved9_must_be_null;
a_sql_uint64      reserved10_must_be_null;
```

- **Closing syntax** – 次の構文により、記述子を完了します。

```
//---------- For Server Internal Use Only ----------
void  * _for_server_internal_use;
} a_extfn_aggregate;
```

計算コンテキスト

_user_calculation_context フィールドにより、サーバは、複数のデータ・グループについての計算を同時に実行できます。

UDAF は、ローを処理する際、計算のために中間カウンタを保持する必要があります。 これらのカウンタを管理するための単純なモデルでは、API 関数の開始時にメモリを割り付け、集合コンテキストの _user_data フィールドにそのポインタを格納し、集合が API を完了するときにメモリを解放します。 _user_calculation_context フィールドに基づく別のメソッドを使用すると、サーバは、複数のデータ・グループについての計算を同時に実行できます。

_user_calculation_context フィールドは、同時処理グループごとにサーバが作成する、サーバ割り付けメモリ・ポインタです。 サーバは、_user_calculation_context が現在処理中のロー・グループの正しい計算コンテキストを常に示すようにします。 各 UDF API 呼び出しの間は、データに応じて、サーバは新しい _user_calculation_context 値を割り付けることができます。 クエリ処理時、サーバは、計算コンテキスト領域をディスクに保存したりリストアしたりすることができます。

UDF は、すべての中間計算値をこのフィールドに格納します。 通常の使用法を次に示します。

```
struct my_average_context
{
        int    sum;
        int    count;
};

reset(a_v3_aggregate_context *context)
{
    mycontext  = (my_average_context *) context-
>_user_calculation_context;
    mycontext->count = 0;
    mycontext->sum = 0;
}

    next_value(a_v3_aggregate_context *context, void *args_handle)
{
    mycontext  = (my_average_context *) context-
>_user_calculation_context;
    mycontext->count++;
    ..
}
```

このモデルでは、_user_data フィールドを使用できますが、中間結果計算に関する値を格納することはできません。 開始および終了エントリ・ポイントの両方で、_user_calculation_context は NULL です。

_user_calculation_context を使用して、同時処理を有効にするには、UDF は、計算コンテキスト用のサイズおよび配置要件を指定し、値を保持して a_v3_extfn_aggregate._calculation_context_size を構造の sizeof() に設定するため構造を定義する必要があります。

また、UDF は、_calculation_context_alignment を介して _user_calculation_context のデータ配置要件も指定する必要があります。 user_calculation_context メモリが文字バイト配列のみを含んでいる場合、特別な配置は不要であり、配置 1 を指定できます。同様に倍精度浮動小数点値には 8 バイト配置が必要な場合があります。 配置要件は、プラットフォームとデータ型により異なります。 必要なサイズより大きい配置を指定できますが、最小の配置を使用することでメモリの効率が向上します。

UDAF コンテキスト構造

集合 UDF コンテキスト構造 a_v3_extfn_aggregate_context には、スカラ UDF コンテキスト構造と同じコールバック関数ポインタ・セットがあります。

また、スカラ UDF コンテキストとよく似た読み込み／書き込み _user_data ポインタがあります。 現在の使用および場所を示す、読み込み専用のデータ・フィールド・セットもあります。 文中の UDF のユニークな各インスタンスには、呼び出されたときに、集合 UDF 記述子構造で指定された各関数に渡される、1 つの集合 UDF

コンテキスト・インスタンスがあります。 集合コンテキスト構造は、次のように定義されます。

- **typedef struct a_v3_extfn_aggregate_context** – クエリ内で参照される外部関数のインスタンスごとに作成されます。 クエリ内の並列サブツリーで使用する場合は、並列サブツリーのコンテキストが個別にあります。
- **Callbacks available via the context** – コールバック・ルーチンの一般的な引数は、次のとおりです。
  - **arg_handle** – サーバが提供する、関数インスタンスおよび引数へのハンドル。
  - **arg_num** – 引数値。 戻り値は、0..N です。
  - **data** – 引数データへのポインタ。

  コンテキストは get_piece の前に get_value を呼び出す必要がありますが、piece_len が total_len より小さい場合のみ、get_piece を呼び出す必要があります。

```
short (SQL_CALLBACK *get_value)(
  void *          arg_handle,
        a_sql_uint32    arg_num,
        an_extfn_value *value
        );
short (SQL_CALLBACK *get_piece)(
        void *          arg_handle,
        a_sql_uint32    arg_num,
        an_extfn_value *value,
        a_sql_uint32    offset
        );
```

- **Determining whether an argument is a constant** – UDF は、指定された引数が定数かどうかを示す情報を要求できます。 これは、たとえば、_next_value 関数を呼び出すたびに行うのではなく、_next_value 関数の最初の呼び出しで 1 回行うだけで済む場合に便利です。

```
short (SQL_CALLBACK *get_value_is_constant)(
  void *          arg_handle,
        a_sql_uint32    arg_num,
        a_sql_uint32 *  value_is_constant
        );
```

- **Returning a null value** – NULL 値を返すには、an_extfn_value で、"data" を NULL に設定します。 set_value の呼び出し時、total_len フィールドは無視されます。 append が FALSE である場合は、指定されたデータが引数値になります。それ以外の場合は、引数の現在値にデータが追加されます。 set_value は、引数の append が TRUE の呼び出しの前に、同じ引数の append が FALSE の呼び出しにより呼び出されると予期されています。 固定長データ型 (すなわち、すべての数値データ型) の場合、append フィールドは無視されます。

```
short (SQL_CALLBACK *set_value)(
  void *          arg_handle,
        an_extfn_value *value,
```

```
        short               append
        );
```

- **Determining whether the statement was interrupted** – UDF エントリ・ポイントが長期間 (かなりの秒数) にわたって動作を実行する場合、可能であれば、ユーザが現在の文を中断したかどうかを確認するために、1 または 2 秒ごとに get_is_cancelled コールバックを呼び出します。 文が中断されている場合、ゼロ以外の値が返され、UDF エントリ・ポイントはただちにリターンを実行します。 最終的には、必要なクリーンアップを実行するために _finish_extfn 関数が呼び出されますが、他の UDF エントリ・ポイントが続いて呼び出されることはありません。

```
 a_sql_uint32 (SQL_CALLBACK *get_is_cancelled)
(a_v3_extfn_aggregate_context * cntxt);
```

- **Sending error messages** – エラー・メッセージがユーザに送信され、現在の文が停止されるような何らかのエラーが UDF エントリ・ポイントに発生した場合、set_error コールバック・ルーチンが呼び出されます。set_error により、現在の文がロールバックされ、"Error from external UDF:: <error_desc_string>" と表示されます。SQLCODE は、<error_number> の否定形式です。 set_error への呼び出しの後、UDF エントリ・ポイントはただちにリターンを実行します。 最終的には、必要なクリーンアップを実行するために _finish_extfn が呼び出されますが、他の UDF エントリ・ポイントが続いて呼び出されることはありません。

```
void (SQL_CALLBACK *set_error)(
  a_v3_extfn_aggregate_context * cntxt,
        a_sql_uint32    error_number,
        // use error_number values >17000 & <100000
        const char *    error_desc_string
        );
```

- **Writing messages to the message log** – 255 バイトより長いメッセージをトランケートできます。

```
void (SQL_CALLBACK *log_message)(
  const char *msg,
        short msg_length
  );
```

- **Converting one data type to another** – 入力用です。
  - **an_extfn_value.data** – 入力データ・ポインタ。
  - **an_extfn_value.total_len** – 入力データの長さ。
  - **an_extfn_value.type** – 入力の DT_ datatype。

  出力用です。
  - **an_extfn_value.data** – UDF が提供する出力データ・ポインタ。
  - **an_extfn_value.piece_len** – 出力データの最大長。
  - **an_extfn_value.total_len** – 変換済み出力のサーバ設定長。

- **an_extfn_value.type** – 対象の出力の DT_ datatype。

```
short (SQL_CALLBACK *convert_value)(
  an_extfn_value   *input,
        an_extfn_value   *output
        );
```

- **Fields reserved for future use** – 将来使用するために予約されています。

```
void  *   reserved1;
void  *   reserved2;
void  *   reserved3;
void  *   reserved4;
void  *   reserved5;
```

- **Data available from the context** – このデータ・ポイントは、外部ルーチンが要求するコンテキスト・データの使用により、入力できます。 UDF はこのメモリを割り付けたり、割り付けを解除したりします。 文ごとに、_user_data の 1 つのインスタンスが有効です。 中間結果値には、このメモリを使用しないでください。

```
void  * _user_data;
```

- **Currently active calculation context** – UDF はこのメモリの場所を使用して、集合を計算する中間値を格納します。 このメモリは、a_v3_extfn_aggregate で要求されるサイズに基づき、サーバにより割り付けられます。 エンジンが、複数のグループについて同時計算を実行することがあるため、中間計算はこのメモリに格納される必要があります。 各 UDF エントリ・ポイントの前に、サーバは、正しいコンテキスト・データがアクティブであることを確認します。

```
void  * _user_calculation_context;
```

- **Other available aggregate information** – start_extfn を含む、すべての外部関数のエントリ・ポイントで使用できます。 ゼロは、不明な値または不適切な値を示します。 パーティションまたはグループごとに見積もられた、ローの平均数。
  - **a_sql_uint64 _max_rows_in_frame;** – ウィンドウ・フレームで定義されたローの最大数を計算します。 範囲ベースのウィンドウの場合、ユニークな値を示します。 ゼロは、不明な値または不適切な値を示します。
  - **a_sql_uint64 _estimated_rows_per_partition;** – パーティションまたはグループごとに見積もられた、ローの平均数を表示します。 ゼロは、不明な値または不適切な値を示します。
  - **a_sql_uint32 _is_used_as_a_superaggregate;** – このインスタンスが通常の集合であるか、スーパー集合であるかを指定します。 インスタンスが通常の集合である場合、結果としてゼロを返します。
- **Determining window specifications** – ウィンドウがクエリに存在する場合の、ウィンドウ指定。
  - **a_sql_uint32 _is_window_used;** – 文がウィンドウ化されるかどうかを指定します。

- **a_sql_uint32 _window_has_unbounded_preceding;** – 戻り値ゼロは、ウィンドウにバインドを解除した先行ローがないことを示します。
- **a_sql_uint32 _window_contains_current_row;** – 戻り値ゼロは、ウィンドウに現在のローが含まれないことを示します。
- **a_sql_uint32 _window_is_range_based;** – リターン・コードが 1 である場合、ウィンドウは範囲ベースです。 リターン・コードがゼロである場合、ウィンドウはローベースです。

- **Available at reset_extfn() calls** – 現在のパーティションのローの実際の数を返します。ウィンドウ化されない集合の場合は、ゼロを返します。
```
a_sql_uint64 _num_rows_in_partition;
```
- **Available only at evaluate_extfn() calls for windowed aggregates** – パーティション内の現在評価されているロー番号 (1 で始まる)。 これは、無制限ウィンドウの評価フェーズにおいて便利です。
```
a_sql_uint64 _result_row_from_start_of_partition;
```
- **Closing syntax** – 次の構文を使用し、コンテキストを完了します。
```
//---------- For Server Internal Use Only ----------
  void  * _for_server_internal_use;
} a_v3_extfn_aggregate_context;
```

UDAF の例： my_sum 定義

"my_sum" の例は、整数についてのみ動作します。

my_sum 定義

SUM と同様に my_sum は任意のコンテキストで使用できるため、最適化されたすべてのオプションのエントリ・ポイントが提供されています。 次の例では、normal _evaluate_extfn 関数も _evaluate_superaggregate_extfn 関数として使用できます。

```
#include "extfnapiv3.h"
#include <stdlib.h>
#include <assert.h>


//  Simple aggregate UDF that adds up a set of
//  integer arguments, and whenever asked returns
//  the resulting big integer total.  For int
//  arguments, the only difference between this
//  UDF and the SUM built-in aggregate is that this
//  UDF will return NULL if there are no input rows.
//
//  The start function creates a little structure for
//  the running total, and the finish function then
//  deallocates it.
//
//  Since there are no aggregate usage restrictions
//  for this UDAF, the corresponding SQL declaration
//  will look like:
//
```

```
//              CREATE AGGREGATE FUNCTION my_sum(IN arg1 INT)
//                      RETURNS BIGINT
//                      ON EMPTY INPUT RETURNS NULL
//                      EXTERNAL NAME 'my_integer_sum@libudfex'


typedef struct my_total {
  a_sql_int64   _total;
  a_sql_uint64  _num_nonnulls_seen;
} my_total;


extern "C"
void my_integer_sum_start(a_v3_extfn_aggregate_context *cntxt)
{
}


extern "C"
void my_integer_sum_finish(a_v3_extfn_aggregate_context *cntxt)
{
}


extern "C"
void my_integer_sum_reset(a_v3_extfn_aggregate_context *cntxt)
{
  my_total *cptr = (my_total *)cntxt->_user_calculation_context;
  cptr->_total = 0;
  cptr->_num_nonnulls_seen = 0;
}


extern "C"
void my_integer_sum_next_value(a_v3_extfn_aggregate_context *cntxt,
                               void *arg_handle)
{
  an_extfn_value  arg;
  a_sql_int32 arg1;

  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then add it to the total
  //
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int32 *)arg.data);
    cptr->_total += arg1;
    cptr->_num_nonnulls_seen++;
  }
}


extern "C"
void my_integer_sum_drop_value(a_v3_extfn_aggregate_context *cntxt,
                               void *arg_handle)
{
```

```
  an_extfn_value  arg;
  a_sql_int32 arg1;
  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then subtract it from the
total
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int32 *)arg.data);
    cptr->_total -= arg1;
    cptr->_num_nonnulls_seen--;
  }
}


extern "C"
void my_integer_sum_evaluate(a_v3_extfn_aggregate_context *cntxt,
                             void *arg_handle)
{
  an_extfn_value  outval;
  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Set the output result value.  If the inputs
  //  were all NULL, then set the result as NULL.
  //
  outval.type = DT_BIGINT;
  outval.piece_len = sizeof(a_sql_int64);
  if (cptr->_num_nonnulls_seen > 0) {
    outval.data = &cptr->_total;
  } else {
    outval.data = 0;
  }
  cntxt->set_value( arg_handle, &outval, 0 );
}


extern "C"
void my_integer_sum_cum_evaluate(
                       a_v3_extfn_aggregate_context *cntxt,
                       void *arg_handle)
{
  an_extfn_value  outval;
  an_extfn_value  arg;
  int arg1;
  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then add it into the
total.
  //
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int32 *)arg.data);
    cptr->_total += arg1;
    cptr->_num_nonnulls_seen++;
  }

  //  Then set the output result value.  If the inputs
  //  were all NULL, then set the result as NULL.
```

```
  //
  outval.type = DT_BIGINT;
  outval.piece_len = sizeof(a_sql_int64);
  if (cptr->_num_nonnulls_seen > 0) {
    outval.data = &cptr->_total;
  } else {
    outval.data = 0;
  }
  cntxt->set_value( arg_handle, &outval, 0 );
}


extern "C"
void my_integer_sum_next_subagg_value(
                a_v3_extfn_aggregate_context *cntxt,
                void *arg_handle)
{
  an_extfn_value  arg;
  a_sql_int64 arg1;

  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then add it to the total
  //
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int64 *)arg.data);
    cptr->_total += arg1;
    cptr->_num_nonnulls_seen++;
  }
}


extern "C"
void my_integer_sum_drop_subagg_value(
                a_v3_extfn_aggregate_context *cntxt,
                void *arg_handle)
{
  an_extfn_value  arg;
  a_sql_int64 arg1;

  my_total *cptr = (my_total *)cntxt->_user_calculation_context;

  //  Get the one argument, and if non-NULL then subtract it from the
total
  //
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_int64 *)arg.data);
    cptr->_total -= arg1;
    cptr->_num_nonnulls_seen--;
  }
}



a_v3_extfn_aggregate my_integer_sum_descriptor =
{
```

```
        &my_integer_sum_start,
        &my_integer_sum_finish,
        &my_integer_sum_reset,
        &my_integer_sum_next_value,
        &my_integer_sum_evaluate,
        &my_integer_sum_drop_value,
        &my_integer_sum_cum_evaluate,
        &my_integer_sum_next_subagg_value,
        &my_integer_sum_drop_subagg_value,
        &my_integer_sum_evaluate,
        NULL, // reserved1_must_be_null
        NULL, // reserved2_must_be_null
        NULL, // reserved3_must_be_null
        NULL, // reserved4_must_be_null
        NULL, // reserved5_must_be_null
        0, // indicators
        ( short )sizeof( my_total ), // context size
        8, // context alignment
        0.0, //external_bytes_per_group
        0.0, // external bytes per row
        0, // reserved6_must_be_null
        0, // reserved7_must_be_null
        0, // reserved8_must_be_null
        0, // reserved9_must_be_null
        0, // reserved10_must_be_null
        NULL // _for_server_internal_use
};

extern "C"
a_v3_extfn_aggregate *my_integer_sum()
{
  return &my_integer_sum_descriptor;
}
```

UDAF の例： my_bit_xor 定義

"my_bit_xor" の例は SA の組み込みの BIT_XOR に似ていますが、my_bit_xor は符号なし整数でのみ動作する点が異なります。

my_bit_xor 定義

入力および出力のデータ型が同じであるため、サブ集合値を累計し、スーパー集合結果を生成するには、通常の _next_value_extfn および _evaluate_extfn 関数を使用します。

```
#include "extfnapiv3.h"
#include <stdlib.h>
#include <assert.h>


//  Generic aggregate UDF that exclusive-ORs a set of
//  unsigned integer arguments, and whenever asked
//  returns the resulting unsigned integer result.
//
//  The start function creates a little structure for
```

```
//  the running result, and the finish function then
//  deallocates it.
//
//  Since there are no aggregate usage restrictions
//  for this UDAF, the corresponding SQL declaration
//  will look like:
//
//            CREATE AGGREGATE FUNCTION my_bit_xor(IN arg1 UNSIGNED
INT)
//                      RETURNS UNSIGNED INT
//                      ON EMPTY INPUT RETURNS NULL
//                      EXTERNAL NAME 'my_bit_xor@libudfex'


typedef struct my_xor_result {
  a_sql_uint64 _num_nonnulls_seen;
  a_sql_uint32 _xor_result;
} my_xor_result;

#if defined __cplusplus
extern "C" {
#endif

static void my_xor_start(a_v3_extfn_aggregate_context *cntxt)
{
}


static void my_xor_finish(a_v3_extfn_aggregate_context *cntxt)
{
}


static void my_xor_reset(a_v3_extfn_aggregate_context *cntxt)
{
  my_xor_result *cptr = (my_xor_result *)cntxt-
>_user_calculation_context;
  cptr->_xor_result = 0;
  cptr->_num_nonnulls_seen = 0;
}


static void my_xor_next_value(a_v3_extfn_aggregate_context *cntxt,
                              void *arg_handle)
{
  an_extfn_value  arg;
  a_sql_uint32 arg1;

  my_xor_result *cptr = (my_xor_result *)cntxt-
>_user_calculation_context;

  //  Get the one argument, and add it to the total
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_uint32 *)arg.data);
    cptr->_xor_result ^= arg1;
    cptr->_num_nonnulls_seen++;
```

```
  }
}


static void my_xor_drop_value(a_v3_extfn_aggregate_context *cntxt,
                              void *arg_handle)
{
  an_extfn_value  arg;
  a_sql_uint32 arg1;
  my_xor_result *cptr = (my_xor_result *)cntxt-
>_user_calculation_context;

  //  Get the one argument, and remove it from the total
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_uint32 *)arg.data);
    cptr->_xor_result ^= arg1;
    cptr->_num_nonnulls_seen--;
  }
}


static void my_xor_evaluate(a_v3_extfn_aggregate_context *cntxt,
                            void *arg_handle)
{
  an_extfn_value  outval;
  my_xor_result *cptr = (my_xor_result *)cntxt-
>_user_calculation_context;

  outval.type = DT_UNSINT;
  outval.piece_len = sizeof(a_sql_uint32);
  if (cptr->_num_nonnulls_seen > 0) {
    outval.data = &cptr->_xor_result;
  } else {
    outval.data = 0;
  }
  cntxt->set_value( arg_handle, &outval, 0 );
}


static void my_xor_cum_evaluate(a_v3_extfn_aggregate_context
*cntxt,
                                void *arg_handle)
{
  an_extfn_value  outval;
  an_extfn_value  arg;
  a_sql_uint32 arg1;
  my_xor_result *cptr = (my_xor_result *)cntxt-
>_user_calculation_context;

  //  Get the one argument, and include it in the result,
  //  unless that input value is null.
  //
  if (cntxt->get_value( arg_handle, 1, &arg) && arg.data) {
    arg1 = *((a_sql_uint32 *)arg.data);
    cptr->_xor_result ^= arg1;
    cptr->_num_nonnulls_seen++;
```

```
  }

  //  Then set the output result value
  outval.type = DT_UNSINT;
  outval.piece_len = sizeof(a_sql_uint32);
  if (cptr->_num_nonnulls_seen > 0) {
    outval.data = &cptr->_xor_result;
  } else {
    outval.data = 0;
  }
  cntxt->set_value( arg_handle, &outval, 0 );
}


static a_v3_extfn_aggregate my_xor_descriptor =
    {
        &my_xor_start,
        &my_xor_finish,
        &my_xor_reset,
        &my_xor_next_value,
        &my_xor_evaluate,
        &my_xor_drop_value,
        &my_xor_cum_evaluate,
        &my_xor_next_value,
        &my_xor_drop_value,
        &my_xor_evaluate,
        NULL, // reserved1_must_be_null
        NULL, // reserved2_must_be_null
        NULL, // reserved3_must_be_null
        NULL, // reserved4_must_be_null
        NULL, // reserved5_must_be_null
        0,    // indicators
        ( short )sizeof( my_xor_result ), // context size
        8,    // context alignment
        0.0,  // external_bytes_per_group
        0.0,  // external bytes per row
        0,    // reserved6_must_be_null
        0,    // reserved7_must_be_null
        0,    // reserved8_must_be_null
        0,    // reserved9_must_be_null
        0,    // reserved10_must_be_null
        NULL  // _for_server_internal_use
    };

a_v3_extfn_aggregate *my_bit_xor()
{
  return &my_xor_descriptor;
}

#if defined __cplusplus
}
#endif
```

UDAF の例： my_bit_or 定義

"my_bit_or" の例は SA の組み込みの BIT_OR に似ていますが、符号なし整数でのみ動作する点が異なり、単純な集合としてのみ使用できます。

my_bit_or 定義

"my_bit_or" の定義は、"my_bit_xor" の例よりも少し簡単です。

```
#include "extfnapiv3.h"
#include <stdlib.h>
#include <assert.h>


//  A simple (non-OLAP) aggregate UDF that ORs a set
//  of unsigned integer arguments, and whenever asked
//  returns the resulting unsigned integer result.
//
//  The start function creates a little structure for
//  the running result, and the finish function then
//  deallocates it.
//
//  The aggregate usage restrictions for this UDAF
//  only allow its use as a simple aggregate, so the
//  corresponding SQL declaration will look like:
//
//      CREATE AGGREGATE FUNCTION my_bit_or(IN arg1 UNSIGNED INT)
//            RETURNS UNSIGNED INT
//                      ON EMPTY INPUT RETURNS NULL
//            OVER NOT ALLOWED
//            EXTERNAL NAME 'my_bit_or@libudfex'


typedef struct my_or_result {
  a_sql_uint32 _or_result;
  a_sql_uint32 _non_null_seen;
} my_or_result;



#if defined __cplusplus
extern "C" {
#endif

static void my_or_start(a_v3_extfn_aggregate_context *cntxt)
{
}

static void my_or_finish(a_v3_extfn_aggregate_context *cntxt)
{
}

static void my_or_reset(a_v3_extfn_aggregate_context *cntxt)
{
```

```
  my_or_result *cptr = (my_or_result *)cntxt-
>_user_calculation_context;
  cptr->_or_result = 0;
  cptr->_non_null_seen = 0;
}


static void my_or_next_value(a_v3_extfn_aggregate_context *cntxt,
                             void *arg_handle)
{
  an_extfn_value  arg;
  a_sql_uint32 arg1;

  my_or_result *cptr = (my_or_result *)cntxt-
>_user_calculation_context;

  //  Get the one argument, and add it to the total
  if (cntxt->get_value( arg_handle, 1, &arg ) && arg.data)
  {
      arg1 = *((a_sql_uint32 *)arg.data);
      cptr->_or_result |= arg1;
      cptr->_non_null_seen = 1;
  }
}


static void my_or_evaluate(a_v3_extfn_aggregate_context *cntxt,
                           void *arg_handle)
{
  an_extfn_value  outval;
  my_or_result *cptr = (my_or_result *)cntxt-
>_user_calculation_context;

  outval.type = DT_UNSINT;
  outval.piece_len = sizeof(a_sql_uint32);
  if (cptr->_non_null_seen)
  {
      outval.data = &cptr->_or_result;
  }
  else
  {
      // Return null if no values seen
      outval.data = 0;
  }
  cntxt->set_value( arg_handle, &outval, 0 );
}



static a_v3_extfn_aggregate my_or_descriptor =
{
    &my_or_start,
    &my_or_finish,
    &my_or_reset,
    &my_or_next_value,
    &my_or_evaluate,
```

```
    NULL, // drop_val_extfn
    NULL, // cume_eval,
    NULL, // next_subaggregate_extfn
    NULL, // drop_subaggregate_extfn
    NULL, // evaluate_superaggregate_extfn
    NULL, // reserved1_must_be_null
    NULL, // reserved2_must_be_null
    NULL, // reserved3_must_be_null
    NULL, // reserved4_must_be_null
    NULL, // reserved5_must_be_null
    0, // indicators
    ( short )sizeof( my_or_result ), // context size
    8, // context alignment
    0.0, //external_bytes_per_group
    0.0, // external bytes per row
    0, // reserved6_must_be_null
    0, // reserved7_must_be_null
    0, // reserved8_must_be_null
    0, // reserved9_must_be_null
    0, // reserved10_must_be_null
    NULL // _for_server_internal_use
};


extern "C"
a_v3_extfn_aggregate *my_bit_or()
{
  return &my_or_descriptor;
}


#if defined __cplusplus
}
#endif
```

UDAF の例： my_interpolate 定義

"my_interpolate" の例は、隣接する NULL 値のセットに対して、各方向で最も近い NULL 以外の値への線形補間を実行することにより、シーケンス内の NULL 値に入力しようとする OLAP スタイルの UDAF です。

my_interpolate 定義

相応なコストで動作するには、固定幅のローベースのウィンドウを使用して my_interpolate を実行する必要がありますが、ユーザは、隣接する NULL 値の予測される最大数に基づいて、ウィンドウの幅を設定できます。 特定のローの入力が NULL でない場合、そのローの結果は入力値と同じです。 この関数は、倍精度浮動小数点数値のセットを取得し、double 型の結果セットを生成します。

```
#include "extfnapiv3.h"
#include <stdlib.h>
#include <assert.h>
```

```
//  MY_INTERPOLATE
//
//  OLAP-style aggregate UDF that accepts a double precision
//  floating point argument.  If the current argument value is
//  not NULL, then the result value is the same as the
//  argument value.  On the other hand, if the current row's
//  argument value is NULL, then the result, where possible,
//  will be the arithmetic interpolation across the nearest
//  preceding and nearest following values that are not NULL.
//  In all cases the result is also a double precision value.
//
//  The start function creates a structure for maintaining the
//  argument values within the window including their NULLness.
//  The finish function then deallocates this structure.
//
//  Since there are some strict aggregate usage restrictions
//  for this aggregate (must be used with a row-based window
//  frame that includes the current row), the corresponding
//  SQL declaration will look like:
//
//      CREATE AGGREGATE FUNCTION my_interpolate(IN arg1 DOUBLE)
//                      RETURNS DOUBLE
//                      OVER REQUIRED
//                      WINDOW FRAME REQUIRED
//                              RANGE NOT ALLOWED
//                              PRECEDING REQUIRED
//                              UNBOUNDED PRECEDING NOT ALLOWED
//                              FOLLOWING REQUIRED
//                              UNBOUNDED FOLLOWING NOT ALLOWED
//                      EXTERNAL NAME  'my_interpolate@libudfex'


typedef struct my_window {
  int      _allocated_elem;
  int      _first_used;
  int      _next_insert_loc;
  int     *_is_null;
  double  *_dbl_val;
  int      _num_rows_in_frame;
} my_window;



#if defined __cplusplus
extern "C" {
#endif


static void my_interpolate_reset(a_v3_extfn_aggregate_context
*cntxt)
{
  assert(cntxt->_user_data);
  my_window *cptr = (my_window *)cntxt->_user_data;

  cptr->_first_used = 0;
  cptr->_next_insert_loc = 0;
```

```
  cptr->_num_rows_in_frame = 0;
  for (int i=0; i<cptr->_allocated_elem; i++) {
    cptr->_is_null[i] = 1;
  }
}


static void my_interpolate_start(a_v3_extfn_aggregate_context
*cntxt)
{
  my_window *cptr = (my_window *)cntxt->_user_data;

  // Make sure function was defined correctly
  if (!cntxt->_is_window_used)
  {
      cntxt->set_error(cntxt, 20001, "Function requires window");
      return;
  }
  if (cntxt->_window_has_unbounded_preceding ||
      cntxt->_window_has_unbounded_following)
  {
      cntxt->set_error(cntxt, 20002, "Window cannot be unbounded");
      return;
  }
  if (cntxt->_window_is_range_based)
  {
      cntxt->set_error(cntxt, 20003, "Window must be row based");
      return;
  }

  if (!cptr) {
    //
    cptr = (my_window *)malloc(sizeof(my_window));
    if (cptr) {
      cptr->_is_null = 0;
      cptr->_dbl_val = 0;
      cptr->_num_rows_in_frame = 0;
      cptr->_allocated_elem = ( int )cntxt->_max_rows_in_frame;
      cptr->_is_null = (int *)malloc(cptr->_allocated_elem
                                     * sizeof(int));
      cptr->_dbl_val = (double *)malloc(cptr->_allocated_elem
                                        * sizeof(double));
      cntxt->_user_data = cptr;
    }
  }
  if (!cptr || !cptr->_is_null || !cptr->_dbl_val) {
      // Terminate this query
      cntxt->set_error(cntxt, 20000, "Unable to allocate memory");
      return;
  }
  my_interpolate_reset(cntxt);
}


static void my_interpolate_finish(a_v3_extfn_aggregate_context
*cntxt)
```

```
{
  if (cntxt->_user_data) {
    my_window *cptr = (my_window *)cntxt->_user_data;
    if (cptr->_is_null) {
      free(cptr->_is_null);
      cptr->_is_null = 0;
    }
    if (cptr->_dbl_val) {
      free(cptr->_dbl_val);
      cptr->_dbl_val = 0;
    }
    free(cntxt->_user_data);
    cntxt->_user_data = 0;
  }
}


static void my_interpolate_next_value(a_v3_extfn_aggregate_context
*cntxt,
                                      void *arg_handle)
{
  an_extfn_value  arg;
  double arg1;
  my_window *cptr = (my_window *)cntxt->_user_data;

  //  Get the one argument, and stash its value
  //  within the rotating window arrays
  //
  int curr_cell_num = cptr->_next_insert_loc % cptr-
>_allocated_elem;
  if (cntxt->get_value( arg_handle, 1, &arg ) && arg.data != NULL ) {
    arg1 = *((double *)arg.data);
    cptr->_dbl_val[curr_cell_num] = arg1;
    cptr->_is_null[curr_cell_num] = 0;
  } else {
    cptr->_is_null[curr_cell_num] = 1;
  }

  //  Then increment the insertion location and number of rows in
frame
  cptr->_next_insert_loc = ((cptr->_next_insert_loc + 1)
                            % cptr->_allocated_elem);
  cptr->_num_rows_in_frame++;
}


static void my_interpolate_drop_value(a_v3_extfn_aggregate_context
*cntxt,
                                      void * /*arg_handle*/)
{
  my_window *cptr = (my_window *)cntxt->_user_data;

  //  Drop one value from the window by incrementing past it and
  // decrement the number of rows in the frame
  cptr->_first_used = ((cptr->_first_used + 1) % cptr-
>_allocated_elem);
```

```
  cptr->_num_rows_in_frame--;
}


static void my_interpolate_evaluate(a_v3_extfn_aggregate_context
*cntxt,
                                    void *arg_handle)
{

  an_extfn_value  outval;
  my_window *cptr = (my_window *)cntxt->_user_data;
  double  result;
  int     result_is_null = 1;
  double  preceeding_value;
  int     preceeding_value_is_null = 1;
  double  preceeding_distance = 0;
  double  following_value;
  int     following_value_is_null = 1;
  double  following_distance = 0;
  int j;

  //  Determine which cell is the current cell
  int curr_cell_num =
      ((int)(cntxt->_result_row_from_start_of_partition-1))%cptr-
>_allocated_elem;
  int tmp_cell_num;

  int result_row_offset_from_start_of_frame = cptr->_first_used <=
curr_cell_num ?
          ( curr_cell_num - cptr->_first_used ) :
          ( curr_cell_num + cptr->_allocated_elem - cptr-
>_first_used );




  //  Compute the result value
  if (cptr->_is_null[curr_cell_num] == 0) {
    //
    //  If the current rows input value is not NULL, then there is
    //  no need to interpolate, just use that input value.
    //
    result = cptr->_dbl_val[curr_cell_num];
    result_is_null = 0;
    //
  } else {
    //
    //  If the current rows input value is NULL, then we do
    //  need to interpolate to find the correct result value.
    //  First, find the nearest following non-NULL argument
    //  value after the current row.
    //
    int rows_following = cptr->_num_rows_in_frame -
                 result_row_offset_from_start_of_frame - 1;
    for (j=0; j<rows_following; j++) {
      tmp_cell_num = ((curr_cell_num + j + 1) % cptr-
```

```
>_allocated_elem);
      if (cptr->_is_null[tmp_cell_num] == 0) {
          following_value = cptr->_dbl_val[tmp_cell_num];
          following_value_is_null = 0;
          following_distance = j + 1;
          break;
      }
    }
    //  Second, find the nearest preceeding non-NULL
    //  argument value before the current row.
    //
    int rows_before = result_row_offset_from_start_of_frame;
    for (j=0; j<rows_before; j++) {
     tmp_cell_num = ((curr_cell_num + cptr->_allocated_elem - j - 1)
                     % cptr->_allocated_elem);
      if (cptr->_is_null[tmp_cell_num] == 0) {
          preceeding_value = cptr->_dbl_val[tmp_cell_num];
          preceeding_value_is_null = 0;
          preceeding_distance = j + 1;
          break;
      }
    }
    //  Finally, see what we can come up with for a result value
    //
    if (preceeding_value_is_null && !following_value_is_null) {
      //
     //  No choice but to mirror the nearest following non-NULL value
      //  Example:
      //
      //    Inputs:  NULL    Result of my_interpolate:  40.0
      //             NULL                               40.0
      //             40.0                               40.0
      //
      result = following_value;
      result_is_null = 0;
      //
   } else if (!preceeding_value_is_null && following_value_is_null)
{
      //
      //  No choice but to mirror the nearest preceeding non-NULL
value
      //  Example:
      //
      //    Inputs:  10.0    Result of my_interpolate:  10.0
      //             NULL                               10.0
      //
      result = preceeding_value;
      result_is_null = 0;
      //
   } else if (!preceeding_value_is_null && !following_value_is_null)
{
      //
      //  Here we get to do real interpolation based on the
      //  nearest preceeding non-NULL value, the nearest following
      //  non-NULL value, and the relative distances to each.
      //  Examples:
```

```
      //
      //    Inputs:  10.0    Result of my_interpolate:  10.0
      //             NULL                               20.0
      //             NULL                               30.0
      //             40.0                               40.0
      //
      //    Inputs:  10.0    Result of my_interpolate:  10.0
      //             NULL                               25.0
      //             40.0                               40.0
      //
      result = (  preceeding_value
                  + (  (following_value - preceeding_value)
                       * (  preceeding_distance
                          / (preceeding_distance +
following_distance))));
      result_is_null = 0;
    }
  }

  //  And last, pass the result value out
  outval.type = DT_DOUBLE;
  outval.piece_len = sizeof(double);
  if (result_is_null) {
    outval.data = 0;
  } else {
    outval.data = &result;
  }
  cntxt->set_value( arg_handle, &outval, 0 );
}


static a_v3_extfn_aggregate my_interpolate_descriptor =
    {
        &my_interpolate_start,
        &my_interpolate_finish,
        &my_interpolate_reset,
        &my_interpolate_next_value, //( timeseries_expression )
        &my_interpolate_evaluate,
        &my_interpolate_drop_value,
       NULL, // cume_eval,
       NULL, // next_subaggregate_extfn
       NULL, // drop_subaggregate_extfn
       NULL, // evaluate_superaggregate_extfn
       NULL, // reserved1_must_be_null
       NULL, // reserved2_must_be_null
       NULL, // reserved3_must_be_null
       NULL, // reserved4_must_be_null
       NULL, // reserved5_must_be_null
       0, // indicators
       0, // context size
       0, // context alignment
       0.0, //external_bytes_per_group
       ( double )sizeof( double ), // external bytes per row
       0, // reserved6_must_be_null
       0, // reserved7_must_be_null
       0, // reserved8_must_be_null
```

```
        0, // reserved9_must_be_null
        0, // reserved10_must_be_null
        NULL // _for_server_internal_use
    };

a_v3_extfn_aggregate *my_interpolate()
  { return &my_interpolate_descriptor; }

#if defined __cplusplus
  }
#endif
```

UDF コールバック関数とパターン呼び出し

パターン呼び出しは、結果の収集時に関数が実行する手順です。

UDF および UDAF のコールバック関数

a_v3_extfn_scalar_context 構造を介してエンジンが提供し、ユーザの UDF 関数で使用されるコールバック関数のセットには、次のものが含まれます。

- **get_value** – 各入力引数の値を取得するために、評価メソッドで使用される関数。引数のデータ型の範囲が狭い場合 (256 バイト未満)、get_value の呼び出しで十分に引数値全体を取得できます。 引数のデータ型の範囲が広い場合、このコールバックに渡される an_extfn_value 構造の piece_len フィールドが total_len フィールドの値より小さい値を返すとき、残りの入力値を取得するためには、次の get_piece コールバックを使用する必要があります。
- **get_piece** – 長い引数入力値に続くフラグメントを取得するために使用される関数。
- **get_is_constant** – 指定される入力引数値が定数かどうかを決定するために使用される関数。 この関数は、たとえば、評価関数を呼び出すたびに行うのではなく、_evaluate_extfn 関数の最初の呼び出しで 1 回行うだけで済む場合など、UDF の最適化に便利です。
- **set_value** – サーバにこの呼び出しの UDF の結果値を通知するために、評価関数内で使用される関数。 結果のデータ型の範囲が狭い場合は、set_value を 1 回呼び出すだけで十分です。 一方、結果のデータ値の範囲が広い場合は、値全体を渡すには set_value を複数回呼び出す必要があり、最後の呼び出しを除き、各フラグメントについてコールバックへの append 引数が true である必要があります。NULL 結果を返すには、UDF は、結果値の an_extfn_value 構造にあるデータ・フィールドを NULL ポインタに設定する必要があります。
- **get_is_cancelled** – 文がキャンセルされているかどうかを決定するために使用される関数。 UDF エントリ・ポイントが長期間 (かなりの秒数) にわたって動作を実行する場合、可能であれば、ユーザが現在の文を中断したかどうかを確認するために、1 または 2 秒ごとに get_is_cancelled コールバックを呼び出します。文が中断されていない場合、戻り値は 0 です。
- **set_error** – サーバに (最終的にはユーザに) エラーを返すために使用される関数。エラー・メッセージがユーザに送信されるようなエラーが UDF エントリ・ポイントに発生した場合、このコールバック・ルーチンを呼び出します。 呼び出されると、set_error により現在の文がロールバックされ、ユーザは "Error from external UDF: error_desc_string" というメッセージを受け取ります。SQLCODE

は、指定された error_number の否定形式です。 既存のエラーとの衝突を避けるため、UDF は 17000 から 99999 までの error_number 値を使用します。既存の IQ 内部に基づき、"error_desc_string" の最大長は 140 文字に制限されます。

- **log_message** – サーバのメッセージ・ログにメッセージを送信するために使用される関数。 文字列は、255 バイト以内の印刷可能なテキスト文字列です。
- **convert_value** – 異なるデータ型の間のデータ変換を可能にする関数。 主に、DT_DATE、DT_TIME、および DT_TIMESTAMP、および DT_TIMESTAMP_STRUCT の間の変換に使用されます。 入力および出力の an_extfn_value は、この関数に渡されます。

スカラ UDF パターン呼び出し

一般に、スカラ UDF の場合、指定された関数ポインタについて予測されるパターン呼び出しは、次のようになります。

```
_start_extfn(if supplied)
_evaluate_extfn (repeated 0 to numerous times)
_finish_extfn(if supplied)
```

集合 UDF パターン呼び出し

ユーザが指定した集合 UDF 関数のパターン呼び出しは、スカラ関数のパターン呼び出しより、複雑で多様です。 ここでは、さまざまな SQL 文について IQ UDAF で使用するシーケンス呼び出しの例を示します。 これらの例では、次のテーブル定義を使用します。

```
create table t (a int, b int, c int)
insert into t values (1, 1, 1)
insert into t values (2, 1, 1)
insert into t values (3, 1, 1)
insert into t values (4, 2, 1)
insert into t values (5, 2, 1)
insert into t values (6, 2, 1)
```

次の省略形が使用されます。

RR = a_v3_extfn_aggregate_context. _result_row_offset_from_start_of_partition – この値は、値が計算される現在のパーティションにある、現在のロー番号を示します。この値は、ウィンドウ集合の実行時に設定され、無制限ウィンドウの評価手順の実行時に使用されます。この値は、すべての評価呼び出しで使用できます。

単純な非グループ化集合

単純な非グループ化集合のパターン呼び出しは、すべてのローの入力値を合計して、結果を生成します。

クエリ

```
select my_sum(a) from t
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)  -- input a=1
_next_value_extfn(cntxt, args)  -- input a=2
_next_value_extfn(cntxt, args)  -- input a=3
_next_value_extfn(cntxt, args)  -- input a=4
_next_value_extfn(cntxt, args)  -- input a=5
_next_value_extfn(cntxt, args)  -- input a=6
_evaluate_extfn(cntxt, args)    -- returns 21
_finish_extfn(cntxt)
```

結果

```
my_sum(a)
21
```

単純なグループ化集合

単純なグループ化集合のパターン呼び出しは、グループのすべてのローの入力値を合計して、結果を生成します。**_reset_extfn** は、グループの先頭を示します。

クエリ

```
select b, my_sum(a) from t group by b order by b
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)  -- input a=1
_next_value_extfn(cntxt, args)  -- input a=2
_next_value_extfn(cntxt, args)  -- input a=3
_evaluate_extfn(cntxt, args)    -- returns  6
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)  -- input a=4
_next_value_extfn(cntxt, args)  -- input a=5
_next_value_extfn(cntxt, args)  -- input a=6
_evaluate_extfn(cntxt, args)    -- returns 15
_finish_extfn(cntxt)
```

結果

```
b,   my_sum(a)
1,   6
2,   15
```

無制限ウィンドウにおける OLAP スタイルの集合のパターン呼び出し

"b" での分割は、"b" でのグループ化と同じパーティションを作成します。 無制限ウィンドウでは、パーティションのローごとに "a" 値が評価されます。 これは無制限クエリであるため、すべての値は最初に UDF に渡され、その後に評価サイクルが続きます。 _window_has_unbounded_preceding および _window_has_unbounded_following コンテキスト・インジケータは 1 に設定されます。

クエリ

```
select b, my_sum(a) over (partition by b rows between
unbounded preceding and
unbounded following)
from t
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)    input a=1
_next_value_extfn(cntxt, args)    input a=2
_next_value_extfn(cntxt, args)    input a=3
_evaluate_extfn(cntxt, args)         rr=1  returns 6
_evaluate_extfn(cntxt, args)         rr=2  returns 6
_evaluate_extfn(cntxt, args)      rr=3  returns 6
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)    input a=4
_next_value_extfn(cntxt, args)    input a=5
_next_value_extfn(cntxt, args)    input a=6
_evaluate_extfn(cntxt, args)      rr=1  returns 15
_evaluate_extfn(cntxt, args)      rr=2  returns 15
_evaluate_extfn(cntxt, args)      rr=3  returns 15
_finish_extfn(cntxt)
```

結果

```
b,  my_sum(a)
1,  6
1,  6
1,  6
2,  15
2,  15
2,  15
```

OLAP スタイルの非最適化累積ウィンドウの集合

_evaluate_cumulative_extfn が指定されない場合、次のパターン呼び出しにより、この累積和が評価されます。これは、_evaluate_cumulative_extfn よりも効率が低下します。

クエリ

```
select b, my_sum(a) over (partition by b
rows between unbounded preceding and current row)
from t
order by b
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)     -- input a=1
_evaluate_extfn(cntxt, args)    -- returns 1
_next_value_extfn(cntxt, args)  -- input a=2
_evaluate_extfn(cntxt, args)    -- returns 3
_next_value_extfn(cntxt, args)  -- input a=3
_evaluate_extfn(cntxt, args)    -- returns 6
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)  -- input a=4
_evaluate_extfn(cntxt, args)    -- returns 4
_next_value_extfn(cntxt, args)  -- input a=5
_evaluate_extfn(cntxt, args)    -- returns 9
_next_value_extfn(cntxt, args)  -- input a=6
_evaluate_extfn(cntxt, args)    -- returns 15
_finish_extfn(cntxt)
```

結果

```
b,  my_sum(a)
1,  1
1,  3
1,  6
2,  4
2,  9
2,  15
```

OLAP スタイルの最適化累積ウィンドウの集合

_evaluate_cumulative_extfn が指定され、next_value/evaluate シーケンスが各パーティション内の各ローの 1 つの _evaluate_cumulative_extfn 呼び出しに結合される場合、この累積和が評価されます。

クエリ

```
select b, my_sum(a) over (partition by b rows between unbounded
preceding and current row)
```

```
from t
order by b
```

パターン呼び出し

```
_start_extnfn(cntxt)
_reset_extfn(cntxt)
_evaluate_cumulative_extfn(cntxt, args) -- input a=1 returns 1
_evaluate_cumulative_extfn(cntxt, args) -- input a=2 returns 3
_evaluate_cumulative_extfn(cntxt, args) -- input a=3 returns 6
_reset_extfn(cntxt)
_evaluate_cumulative_extfn(cntxt, args) -- input a=4 returns 4
_evaluate_cumulative_extfn(cntxt, args) -- input a=5 returns 9
_evaluate_cumulative_extfn(cntxt, args) -- input a=6 returns 15
_finish_extfn(cntxt)
```

結果

```
b,  my_sum(a)
1,  1
1,  3
1,  6
2,  4
2,  9
2,  15
```

OLAP スタイルの非最適化移動ウィンドウの集合

drop_value_extfn 関数が指定されていない場合、この移動ウィンドウの和が評価されます。これは、_drop_value_extfn を使用する場合よりも効率が大幅に低下します。

クエリ

```
select b, my_sum(a) over (partition by b rows between 1 preceding and
current row)
from t
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)          input a=1
_evaluate_extfn(cntxt, args)            returns 1
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)          input a=1
_next_value_extfn(cntxt, args)          input a=2
_evaluate_extfn(cntxt, args)            returns 3
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args )         input a=2
_next_value_extfn(cntxt, args )         input a=3
_evaluate_extfn(cntxt, args)            returns 5
```

```
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)          input a=4
_evaluate_extfn(cntxt, args)               returns 4
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)          input a=4
_next_value_extfn(cntxt, args)          input a=5
_evaluate_extfn(cntxt, args)               returns 9
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)          input a=5
_next_value_extfn(cntxt, args)          input a=6
_evaluate_extfn(cntxt, args)               returns 11
_finish_extfn(cntxt)
```

結果

```
b,  my_sum(a)
1,  1
1,  3
1,  5
2,  4
2,  9
2,  11
```

OLAP スタイルの最適化移動ウィンドウの集合

_drop_value_extfn 関数が指定された場合、次のパターン呼び出しを使用して、この移動ウィンドウの和が評価されます。これは、_drop_value_extfn を使用する場合よりも効率的です。

クエリ

```
select b, my_sum(a) over (partition by b rows between 1 preceding and
current row)
from t
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)             -- input a=1
_evaluate_extfn(cntxt, args)               -- returns  1
_next_value_extfn(cntxt, args)      -- input a=2
_evaluate_extfn(cntxt, args)                  -- returns  3
_drop_value_extfn(cntxt)                            -- input a=1
_next_value_extfn(cntxt, args)      -- input a=3
_evaluate_extfn(cntxt, args)                  -- returns  5
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      -- input a=4
_evaluate_extfn(cntxt, args)                  -- returns  4
_next_value_extfn(cntxt, args)      -- input a=5
_evaluate_extfn(cntxt, args)                  -- returns 9
_drop_value_extfn(cntxt)                         -- input a=4
_next_value_extfn(cntxt, args)      -- input a=6
```

```
_evaluate_extfn(cntxt, args)              -- returns  11
_finish_extfn(cntxt)
```

結果

```
b,  my_sum(a)
1,  1
1,  3
1,  5
2,  4
2,  9
2,  11
```

後続のローのある OLAP スタイルの非最適化移動ウィンドウの集合

_drop_value_extfn 関数が指定されていない場合、次のパターン呼び出しを使用して、この移動ウィンドウの和が評価されます。 この例は前述の移動ウィンドウの例と似ていますが、評価されるローは、next_value 関数で指定された最後のローではありません。

クエリ

```
select b, my_sum(a) over (partition by b rows between 1 preceding and
1 following)
from t
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)         input a=1
_next_value_extfn(cntxt, args)      input a=2
_evaluate_extfn(cntxt, args)        returns 3
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=1
_next_value_extfn(cntxt, args)      input a=2
_next_value_extfn(cntxt, args)      input a=3
_evaluate_extfn(cntxt, args)        returns 6
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=2
_next_value_extfn(cntxt, args)      input a=3
_evaluate_extfn(cntxt, args)        returns 5
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=4
_next_value_extfn(cntxt, args)      input a=5
_evaluate_extfn(cntxt, args)        returns 9
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=4
_next_value_extfn(cntxt, args)      input a=5
_next_value_extfn(cntxt, args)      input a=6
_evaluate_extfn(cntxt, args)        returns 15
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=5
_next_value_extfn(cntxt, args)      input a=6
```

```
_evaluate_extfn(cntxt, args)        returns 11
_finish_extfn(cntxt)
```

結果

```
b,  my_sum(a)
1,  3
1,  6
1,  5
2,  9
2,  15
2,  11
```

後続のローのある OLAP スタイルの最適化移動ウィンドウの集合

_drop_value_extfn 関数が指定されている場合、次のパターン呼び出しを使用して、この移動ウィンドウの和が評価されます。 この例も前述の移動ウィンドウの例と似ていますが、評価されるローは、next_value 関数で指定された最後のローではありません。

クエリ

```
select b, my_sum(a) over (partition by b rows between 1 preceding and
1 following)
from t
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)         input a=1
_next_value_extfn(cntxt, args)      input a=2
_evaluate_extfn(cntxt, args)        returns 3
_next_value_extfn(cntxt, args)      input a=3
_evaluate_extfn(cntxt, args)        returns 6
_dropvalue_extfn(cntxt)                              input a=1
_evaluate_extfn(cntxt, args)        returns 5
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=4
_next_value_extfn(cntxt, args)      input a=5
_evaluate_extfn(cntxt, args)        returns 9
_next_value_extfn(cntxt, args)      input a=6
_evaluate_extfn(cntxt, args)        returns 15
_dropvalue_extfn(cntxt)                                 input a=4
_evaluate_extfn(cntxt, args)        returns 11
_finish_extfn(cntxt)
```

結果

```
b,  my_sum(a)
1,  3
1,  6
1,  5
```

```
2,  9
2,  15
2,  11
```

現在のローのない OLAP スタイルの非最適化移動ウィンドウ

UDF の my_sum が組み込みの SUM と同様に動作することを前提とします。 _drop_value_extfn 関数が指定されていない場合、次のパターン呼び出しを使用して、この移動ウィンドウのカウントが評価されます。 この例は前述の移動ウィンドウの例と似ていますが、現在のローがウィンドウ・フレームに含まれません。

クエリ

```
select b, my_sum(a) over (rows between 3 preceding and 1 preceding)
from t
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_evaluate_extfn(cntxt, args)          returns NULL
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=1
_evaluate_extfn(cntxt, args)          returns 1
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=1
_next_value_extfn(cntxt, args)      input a=2
_evaluate_extfn(cntxt, args)          returns 3
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=1
_next_value_extfn(cntxt, args)      input a=2
_next_value_extfn(cntxt, args)      input a=3
_evaluate_extfn(cntxt, args)          returns 6
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=2
_next_value_extfn(cntxt, args)      input a=3
_next_value_extfn(cntxt, args)      input a=4
_evaluate_extfn(cntxt, args)          returns 9
_reset_extfn(cntxt)
_next_value_extfn(cntxt, args)      input a=3
_next_value_extfn(cntxt, args)      input a=4
_next_value_extfn(cntxt, args)      input a=5
_evaluate_extfn(cntxt, args)          returns 12
_finish_extfn(cntxt)
```

結果

```
b           my_sum(a)
----------- -------------
          1          NULL
          1             1
          1             3
          2             6
```

```
          2             9
          2            12
```

現在のローのない OLAP スタイルの最適化移動ウィンドウ

_drop_value_extfn 関数が指定されている場合、次のパターン呼び出しを使用して、この移動ウィンドウのカウントが評価されます。 この例は前述の移動ウィンドウの例と似ていますが、現在のローがウィンドウ・フレームに含まれません。

クエリ

```
select b, my_sum(a) over (rows between 3 preceding and 1 preceding)
from t
```

パターン呼び出し

```
_start_extfn(cntxt)
_reset_extfn(cntxt)
_evaluate_extfn(cntxt, args)          returns NULL
_next_value_extfn(cntxt, args)      input a=1
_evaluate_extfn(cntxt, args)          returns 1
_next_value_extfn(cntxt, args)      input a=2
_evaluate_extfn(cntxt, args)          returns 3
_next_value_extfn(cntxt, args)      input a=3
_evaluate_extfn(cntxt, args)          returns 6
_dropvalue_extfn(cntxt)                 input a=1
_next_value_extfn(cntxt, args)      input a=4
_evaluate_extfn(cntxt, args)          returns 9
_dropvalue_extfn(cntxt)                 input a=2
_next_value_extfn(cntxt, args)      input a=5
_evaluate_extfn(cntxt, args)          returns 12
_finish_extfn(cntxt)
```

結果

```
b           my_sum(a)
---------- -------------
       1          NULL
       1             1
       1             3
       2             6
       2             9
       2            12
```

UDF 固有の関数と文

ユーザ定義関数では、固有の文および関数を使用します。 ここでは、関数プロトタイプの作成と、文の構文および使用法について説明します。

外部関数のプロトタイプ

API は、Sybase IQ のインストール・ディレクトリのサブディレクトリにある extfnapiv3.h という名前のヘッダ・ファイルで定義します。 このヘッダ・ファイルは、外部関数のプロトタイプのプラットフォーム依存機能を処理します。

ライブラリを記述する際に古い API が使用されていないことをデータベース・サーバに通知するため、次のような関数を使用します。

```
uint32 extfn_use_new_api( )
```

この関数は、32 ビットの符号なし整数を返します。 戻り値がゼロ以外の場合、データベース・サーバでは、新しい API が使用されているものと見なします。

DLL がこの関数をエクスポートしない場合は、データベース・サーバでは、古い API が使用されているものと見なします。 新しい API を使用している場合、戻り値は extfnapi.v3h で定義されている API バージョン番号である必要があります。

各ライブラリは、この関数の実装とエクスポートを次のように行います。

```
unsigned int extfn_use_new_api(void)
    {
        return EXTFN_V3_API;
    }
```

この関数が存在し、EXTFN_V3_API を返すことで、ライブラリがこのマニュアルで説明しているバージョン 3 の API に書き込まれる UDF を含んでいることが IQ エンジンに通知されます。

関数プロトタイプ

関数の名前は、**CREATE PROCEDURE** 文または **CREATE FUNCTION** 文で参照されるものと一致しなくてはなりません。 次のように関数を宣言します。

```
void function-name ( an_extfn_api *api, void *argument-handle )
```

関数の戻り値は void とします。引数は 2 つです。1 つは、引数の受け渡しに使用する構造、もう 1 つは、SQL プロシージャから渡された引数へのハンドルです。

an_extfn_api 構造の形式は、次のとおりです。

```
typedef struct an_extfn_api {
short (SQL_CALLBACK *get_value)(
                    void *          arg_handle,
                    a_sql_uint32    arg_num,
                    an_extfn_value *value
                    );
 short (SQL_CALLBACK *get_piece)(
                    void *          arg_handle,
                    a_sql_uint32    arg_num,
                    an_extfn_value *value,
                    a_sql_uint32    offset
                    );
 short (SQL_CALLBACK *set_value)(
                    void *          arg_handle,
                    a_sql_uint32    arg_num,
                    an_extfn_value *value
                    short          append
                    );
void (SQL_CALLBACK *set_cancel)(
                    void *    arg_handle,
                    void *    cancel_handle
                    );
} an_extfn_api;
```

an_extfn_value 構造の形式は、次のとおりです。

```
typedef struct an_extfn_value {
void *                data;
  a_sql_uint32         piece_len;
  union {
  a_sql_uint32    total_len;
  a_sql_uint32    remain_len;
} len;
  a_sql_data_type        type;
} an_extfn_value;
```

注意事項

OUT パラメータに対して get_value を呼び出すと、引数のデータ型が返り、データとして NULL が返ります。

引数に対して get_piece 関数を呼び出せるのは、同じ引数に対して get_value 関数を呼び出した直後に限られます。

NULL を返すには、an_extfn_value のデータを NULL に設定します。

set_value の append フィールドは、データを置き換えるかどうかを決定します。false の場合は、既存のデータを指定されたデータに置き換え、true の場合は、既存のデータに指定されたデータを追加します。 set_value を append=TRUE で呼び出す前に、同じ引数を append=FALSE で呼び出しておく必要があります。 固定長データ型の場合、append フィールドは無視されます。

ヘッダ・ファイル自体にも、いくつかの補足的な注意書きが記述されています。

財務管理用の関数

RAP – The Trading Edition™: Enterprise を使用している場合、ユーザ定義の集合関数を財務管理分析に使用できます。

- **ts_arma_ar**
- **ts_arma_const**
- **ts_arma_ma**
- **ts_autocorrelation**
- **ts_auto_uni_ar**
- **ts_box_cox_xform**
- **ts_difference**
- **ts_estimate_missing**
- **ts_lack_of_fit**
- **ts_lack_of_fit_p**
- **ts_max_arma_ar**
- **ts_max_arma_const**
- **ts_max_arma_ma**
- **ts_max_arma_likelihood**
- **ts_outlier_identification**
- **ts_partial_autocorrelation**
- **ts_vwap**

これらの関数の詳細については、『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』を参照してください。

ユーザ定義の集合関数の構文、パラメータ、および使用方法については、「ユーザ定義の集合関数」 (37 ページ) を参照してください。

外部ライブラリの管理

外部ライブラリは、ライブラリを必要とする UDF が最初に呼び出されたときに、サーバによりロードされます。 最初に必要になりロードされた後、そのライブラリは、サーバの使用期間中、サーバにロードされたままです。 CREATE FUNCTION 呼び出しが実行されてもロードされず、DROP FUNCTION 呼び出しが実行されても自動的にアンロードされません。

ライブラリ・バージョンの更新が必要な場合、サーバを再起動しなくても、**dbo.sa_external_library_unload** プロシージャにより、ライブラリが強制的にアンロードされます。 外部ライブラリをアンロードするための呼び出しは、対象のライブラリが現在使用されていない場合にのみ正常に完了します。 プロシージャは、

アンロードするライブラリの名前を指定する、1 つのオプションのパラメータ long varchar を取ります。 パラメータが指定されていない場合、使用されていないすべての外部ライブラリがアンロードされます。 外部関数ライブラリをアンロードするための構文は、次のとおりです。

注意： ダイナミック・リンク・ライブラリを置き換える前に、実行中の Sybase IQ サーバから既存のライブラリをアンロードする必要があります。 ライブラリのアンロードに失敗すると、サーバのクラッシュが発生する可能性があります。 ダイナミック・リンク・ライブラリを置き換える前に、Sybase IQ サーバをシャットダウンするか、sa_external_library_unload 関数を使用してライブラリをアンロードしてください。

```
call sa_external_library_unload('library.dll')
```

'/abc/def/library.dll' のような完全パスを登録済み関数が使用する場合は、関数の登録を解除します。解除しない場合、ライブラリはアンロードされません。

```
call sa_external_library_unload('/abc/def/library.dll')
```

エラー・チェックと呼び出しトレーシングの制御

external_UDF_execution_mode オプションは、外部 V3 ユーザ定義関数を含む文が評価されたときに、実行されるエラー・チェックと呼び出しトレーシングの数を制御します。

UDF の開発中、デバッグを支援するために、**external_UDF_execution_mode** を使用できます。

指定できる値

0, 1, 2

デフォルト値

0

スコープ

パブリック、一時的、またはユーザとして設定できます。

説明

デフォルトで 0 に設定されている場合、UDF を使用して文のパフォーマンスを最適化する方法で、外部 UDF が評価されます。

1 に設定されている場合、各 UDF 関数とやり取りをする情報を検証するために、外部 UDF が評価されます。

2 に設定されている場合、各 UDF とやり取りをする情報を検証するためだけでなく、UDF により指定される関数のすべての呼び出しと、それらの関数からサーバへのすべてのコールバックのログを iqmsg ファイルに記録するために、外部 UDF が評価されます。

索引

N

O

P

R

S

T

U

V

W

X

あ

い

え

こ

さ

す

せ

た

て

は

ふ

ま

ゆ

ら

り