バージョン 9 リリース 1

IBM InfoSphere Optim Optim Designer の使用

バージョン 9 リリース 1

IBM InfoSphere Optim
Optim Designer の使用

ご注意

本書および本書で紹介する製品をご使用になる前に、 131 ページの『特記事項』に記載されている情報をお読みください。

本書は、Optim Designer バージョン 2、リリース 2、モディフィケーション 3、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： Version 9 Release 1
IBM InfoSphere Optim
Using Optim Designer

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1版第1刷 2012.9

目次

第 1 章 InfoSphere Optim Designer の使用法

データ・モデル、データ・プライバシー・ポリシー、データ管理サービス、およびデータ・ストアを定義するには、 IBM® InfoSphere® Optim™ Designer を使用します。Optim Designer でオブジェクトを定義して、Optim リポジトリーで共有することができます。リポジトリー内のオブジェクトを管理するには、Repository Explorer を使用します。 Optim ディレクトリーでオブジェクトを表示し、要求をトランスフォームするには、Directory Explorer を使用します。

データ管理サービス
データを抽出、変換、ロード、または挿入するには、データ管理サービスを使用します。サービスではアクセス定義を使用して、抽出するデータを定義します。またサービスでは、表マップと列マップを使用して、ソース・データおよびターゲット・データをマップします。Optim Directory からデータ管理サービスに処理要求をトランスフォームすることもできます。埋め込みモードでサービスをテストするには、Optim Manager を使用します。

データ・マスキング
データ管理サービスで処理されるデータをマスクすることができます。データ・マスキングを使用すると、国民 ID 番号、クレジット・カード番号、日付、数値、個人情報などのデータが変換されます。データ・マスキング関数を入力するか、LUA スクリプトを使用して列マップ・プロシージャーを作成するには、列マップを使用します。

アクセス定義
一致するスキーマまたは異なるスキーマでデータ・モデルおよびデータのサブセットを定義するには、アクセス定義を使用します。アクセス定義では、データ管理サービスで使用される選択基準と関係が特定されます。DBMS の外部で関係を定義し、SQL を使用して選択基準を定義することができます。

Optim Manager
Optim Designer (埋め込みモード) から Optim Manager を開き、データ管理サービスをテストすることができます。

始めに

最初に、Repository Explorer を使用して Optim リポジトリーに接続する必要があります。

Optim パースペクティブ

Optim Designer の Optimパースペクティブでは、データ・モデルおよびデータ管理サービスを定義し、Optim ディレクトリーに接続するために必要なツールを使用できます。 Optim Designer を最初に開いたときには、Optim パースペクティブがデフォルトのパースペクティブになっています。

Optim パースペクティブには以下のビューがあります。

リポジトリー・エクスプローラー
リポジトリー・エクスプローラーを使用して、データ・ストア別名、アクセス定義、データ管理サービス、および Optim リポジトリーに格納されているその他のオブジェクトを定義します。

Directory Explorer
Directory Explorer を使用して Optim ディレクトリーに接続します。

Optim パースペクティブを開くには、**「ウィンドウ」** > **「パースペクティブを開く」** > **「その他」**をクリックします。「パースペクティブを開く」ウィンドウで**「Optim」**を選択します。

リポジトリー・エクスプローラー

リポジトリー・エクスプローラーを使用して、データ・ストア別名、アクセス定義、データ管理サービス、および Optim リポジトリーに格納されているその他のオブジェクトを定義します。

Optim Designer が Optim リポジトリーに接続されている場合、リポジトリー・エクスプローラーには次のオブジェクトが含まれています。

データ・ストア別名

データ・ストア別名とは、データベースに関連付けられているユーザー定義のオブジェクトのことです。データ・ストア別名を定義する場合には、Optim がデータベースとの通信に使用するパラメーターを指定します。これらのパラメーターには、データベース管理システム (DBMS) のタイプとバージョン、およびクライアント接続と JDBC 接続の両方のプロパティーが含まれています。

データ・ストア別名は、特定のデータベースにアクセスして要求された関数を実行できるようにする高位修飾子の役割を果たします。例えば、アクセス定義で、表の名前をデータ・ストア別名で修飾する必要があります。参照されたデータ・ストア別名は、表が存在するデータベースへの接続に必要なパラメーターを提供します。

Optim 主キー

主キーとは、表の各行を一意的に定義する値が含まれている列のことです。データベース表は Optim がデータベース表のデータを挿入、更新、リストア、または削除するための主キーを持っている必要があります。Optim は、データベースに定義されている主キーを使用します。ただし、Optim 主キーを定義して、データベース内の主キーを補足することができます。

Optim 関係

Optim は、関係を使用して、関係する表から取得されるデータを決定し、データベースに定義される関係を利用できる場合にはそれに基づきます。ただし、Optim 関係を定義して、データベース内の関係を補足することができます。

アクセス定義

処理するデータに対して、表、関係トラバーサル、および選択基準を指定するには、アクセス定義を使用します。

列マップ

列マップにより、データ管理サービスでの処理について列を一致させるまたは除外するために必要な仕様が得られます。変換、挿入、およびロード・サービスは、1 つ以上の列マップを参照する表マップを参照する必要があります。列マップを使用すると、関数または列マップ・プロシージャーによってデータ変換を定義することができます。

表マップ

表マップは、互換性のあるデータのソース表と宛先表を関連させるための仕様を定義します。名前が異なる表をマップし、表の名前を変更し、処理から表を除外し、または列マップを含めることで、データに対するコントロールを向上させることができます。

サービス

データを抽出、変換、ロード、または挿入するには、データ管理サービスを使用します。サービスではアクセス定義を使用して、抽出するデータを定義します。またサービスでは、表マップと列マップを使用して、ソース・データおよびターゲット・データをマップします。サービスが処理するエンティティーにデータ・プライバシー・ポリシーを適用することにより、データをマスクするこ

とができます。 Optim Directory からデータ管理サービスに処理要求をトランスフォームすることもできます。埋め込みモードでサービスをテストするには、Optim Manager を使用します。

フォルダー

リポジトリー・エクスプローラーのフォルダーには、サービス、アクセス定義、列マップ、表マップが含まれます。フォルダーを使用して、これらのオブジェクトを編成します。

リポジトリー・エクスプローラーのフォルダーの作成:

リポジトリー・エクスプローラー内でフォルダーを作成するには、「新規フォルダー」ウィンドウを使用します。

リポジトリー・エクスプローラーのフォルダーには、サービス、アクセス定義、列マップ、表マップが含まれます。フォルダーを使用して、これらのオブジェクトを編成します。

リポジトリー・エクスプローラーのフォルダーを作成するには、次のようにします。

1. フォルダーが既存かどうかに応じて、次のいずれかのステップを行ってください。
   a. 初期フォルダーを作成するには、リポジトリー・エクスプローラーで「**新規フォルダーを作成する場合にクリックします**」をクリックします。
   b. 追加のフォルダーを作成するには、リポジトリー・エクスプローラー・ビューを右クリックし、「**新規**」 > 「**フォルダー**」をクリックします。

   「新規フォルダー」ウィンドウが開きます。
2. フォルダー名を入力し、「**OK**」をクリックします。

リポジトリー・エクスプローラーに新しいフォルダーが表示されます。

Directory Explorer

Directory Explorer を使用して Optim ディレクトリーに接続します。リポジトリーに関連付けることができる Optim ディレクトリーは 1 つだけです。

処理要求をデータ管理サービスにトランスフォームするには、Directory Explorer を使用します。

使用可能な定義は以下のとおりです。

アイコン	定義
	アクセス定義
	アーカイブ要求
	列
	列マップ
	変換要求
	作成者 ID
	DB 別名

アイコン	定義
	削除要求
	抽出要求
	挿入要求
	ロード要求
	主キー
	関係
	復元要求
	表
	表マップ
	変数

Optim Designer からの Optim Manager の使用

データ管理サービスと Optim 相互運用性サービスをテストするには、Optim Designer から Optim Manager を使用します。Optim Designer から Optim Manager を使用する場合、これは埋め込みモードの Optim Manager とも呼ばれます。

Optim Manager は、サービスのテストに使用できる Web アプリケーションです。Optim Manager は、Optim Designer により提供される内部ブラウザーまたは外部ブラウザーに表示されます。 Optim Designer で Optim Manager 用に使用するブラウザーを選択するには、Optim Designer で、**「ウィンドウ」** > **「設定」** > **「一般」** > **「Web ブラウザー」**をクリックします。

Optim Designer から Optim Manager を開く

Optim Manager を開くには、サービスをテストします。Optim Manager は、Web ブラウザーに `http://localhost:portnumber/console` という URL を入力して開くこともできます。ここで、*portnumber* は Optim Manager に割り当てられたポート番号です。デフォルトのポート番号は 60000 です。

Optim Manager を開いたときに、ページが見つからなかったというメッセージがブラウザーに表示される場合は、ポートの競合が存在している可能性があるため、Optim Manager ポート番号を変更する必要があります。

Optim Designer のツールバーから Optim Manager を開くこともできます。

Optim Manager ポート番号の変更

ポート番号を変更するには、Optim Designer のインストール・ディレクトリーにある `eclipse.ini` ファイル内の以下のプロパティーを編集する必要があります。

```
-Dorg.eclipse.equinox.http.jetty.http.port=portnumber
```

ここで、*portnumber* は新しい Optim Manager ポート番号です。 Optim Designer が開いている場合は、新しいポート番号を適用するために、アプリケーションを再始動する必要があります。

サンプル・データ

Optim は、事前定義のソース/ターゲット・データ・ソースを含む、データ・マスキング用のサンプル・データを提供しています。

サンプル・データベース表と構造

いくつかのデータベース表はグループとして、顧客と注文に関する情報、および配送についての指示を含んでいます。また、サンプル表には、販売と在庫に関する情報も含まれています。サンプル表のインストールに使用する DBMS に応じて、データ・タイプには若干の違いがあります。次の図には、サンプル・データベースの基本構造が示されています。

いくつかのデータベース表はグループとして、顧客と注文に関する情報、および配送についての指示を含んでいます。また、サンプル表には、販売と在庫に関する情報も含まれています。サンプル表のインストールに使用する DBMS に応じて、データ・タイプには若干の違いがあります。

サンプル・データベースと共に、追加的な表のセットもインストールされます。追加セットに含まれる表の名前は、最初のセットの表と同じ名前に接尾部「2」が付いたものになります。この追加セットの 4 つの表は、以下のとおりです。

- OPTIM_CUSTOMERS2
- OPTIM_ORDERS2
- OPTIM_DETAILS2
- OPTIM_ITEMS2

追加セットの表には、データが含まれていません。これらの表は、Optim の機能を例示する目的で使用されます。

OPTIM_SALES 表:

OPTIM_SALES 表は、各販売担当員を名前、ID 番号、管理者によって識別します。

OPTIM_SALES 表には、以下の列があります。

SALESMAN_ID
CHAR。最大 6 文字。NULL を含めることはできません。

FIRST_NAME
VARCHAR。最大 15 文字。NULL を含めることはできません。

LAST_NAME
VARCHAR。最大 15 文字。NULL を含めることはできません。

NATIONALITY
VARCHAR。最大 30 文字

NATIONAL_ID
VARCHAR。最大 30 文字

PHONE_NUMBER
VARCHAR。最大 20 文字。NULL 値を含めることはできません。

EMAIL_ADDRESS
VARCHAR。最大 70 文字。NULL を含めることはできません。

AGE SMALLINT。NULL を含めることはできず、デフォルト値があります。

SEX CHAR。1 文字。NULL を含めることはできず、デフォルト値があります。

TERRITORY
VARCHAR。最大 14 文字。NULL を含めることはできません。

MANAGER_ID
VARCHAR。最大 6 文字。

主キー

SALESMAN_ID 列は、主キー列です。

他の表との関係

OPTIM_SALES 表は、以下の表の親です。

- OPTIM_CUSTOMERS 表 (SALESMAN_ID 列の外部キーを介して)。
- OPTIM_MALE_RATES 表 (SEX ='M' の場合に、列 AGE の Optim データ駆動型関係を介して)。
- OPTIM_FEMALE_RATES 表 (SEX ='F' の場合に、列 AGE の Optim データ駆動型関係を介して)。
- OPTIM_STATE_LOOKUP 表。SUBSTR(SALESMAN_ID,1,2) を使った Optim サブストリング関係を介して。

OPTIM_CUSTOMERS 表:

OPTIM_CUSTOMERS 表には、顧客の名前、ID 番号、および住所が含まれます。

OPTIM_CUSTOMERS 表には、以下の列があります。

CUST_ID
CHAR。最大 5 文字で、NULL を含めることはできず、チェック制約が含まれます。

CUSTNAME
CHAR。最大 20 文字で、NULL を含めることはできません。

ADDRESS1
VARCHAR。最大 100 文字で、NULL を含めることはできません。

ADDRESS2
VARCHAR。最大 100 文字で、NULL を含めることはできません。

LOCALITY
VARCHAR。最大 56 文字

CITY VARCHAR。最大 60 文字

STATE
VARCHAR。最大 30 文字

COUNTRY_CODE
CHAR。最大 2 文字

POSTAL_CODE
VARCHAR。最大 15 文字

POSTAL_CODE_PLUS4
CHAR。最大 4 文字で、NULL 値を含めることが可能です。

EMAIL_ADDRESS
VARCHAR。最大 70 文字

PHONE_NUMBER
VARCHAR。最大 20 文字

YTD_SALES
DECIMAL。金額 (ドル)。NULL を含めることはできず、デフォルト値があります。

SALESMAN_ID
CHAR。最大 6 文字

NATIONALITY
VARCHAR。最大 30 文字

NATIONAL_ID
VARCHAR。最大 30 文字

CREDITCARD_NUMBER
VARCHAR。19 文字

CREDITCARD_TYPE
VARCHAR。最大 30 文字

CREDITCARD_EXP
CHAR。4 文字

CREDITCARD_CVV
VARCHAR。最大 4 文字

DRIVER_LICENSE
VARCHAR。最大 30 文字

CUSTOMER_INFO
XMLTYPE

主キー

CUST_ID 列は、主キー列です。

他の表との関係

OPTIM_CUSTOMERS 表は、以下の表の親です。

- OPTIM_ORDERS 表 (CUST_ID 列の外部キーを介して)。
- OPTIM_SHIP_TO 表 (CUST_ID 列の Optim 関係を介して)。

OPTIM_CUSTOMERS 表は、以下の表の子です。

- OPTIM_SALES 表 (SALESMAN_ID 列の外部キーを介して)。

OPTIM_ORDERS 表:

OPTIM_ORDERS 表には、注文に関する情報 (注文番号、顧客 ID、販売担当員など) が含まれます。

OPTIM_ORDERS 表には、以下の列があります。

ORDER_ID
DECIMAL。注文 ID 番号。NULL を含めることはできません。

CUST_ID
CHAR。顧客 ID 番号。最大 5 文字で、NULL を含めることはできません。

ORDER_DATE
TIMESTAMP。注文日。NULL を含めることはできず、デフォルト値があります。

ORDER_TIME
TIMESTAMP。時刻。NULL を含めることはできず、デフォルト値があります。

FREIGHT_CHARGES
DECIMAL。金額 (ドル)。

ORDER_SALESMAN
CHAR。最大 6 文字

ORDER_POSTED_DATE
TIMESTAMP。NULL を含めることはできず、デフォルト値があります。

ORDER_SHIP_DATE
CHAR。注文の出荷日。最大 8 文字で、NULL を含めることはできず、デフォルト値があります。

主キー

ORDER_ID 列は、主キー列です。

他の表との関係

OPTIM_ORDERS 表は、ORDER_ID 列の外部キーを介した OPTIM_DETAILS 表の親です。

OPTIM_ORDERS 表は、CUST_ID 列の外部キーを介した OPTIM_CUSTOMERS 表の子です。

OPTIM_DETAILS 表:

OPTIM_DETAILS 表には、OPTIM_ORDERS 表のそれぞれの注文に関する追加情報が含まれます。

OPTIM_DETAILS 表には、以下の列があります。

ORDER_ID
DECIMAL。注文 ID 番号。NULL を含めることはできません。

ITEM_ID
CHAR。最大 5 文字の品目 ID 番号。NULL を含めることはできません。

ITEM_QUANTITY
DECIMAL。品目の数。NULL を含めることはできません。

DETAIL_UNIT_PRICE
DECIMAL。単価を示す金額 (ドル)。NULL を含めることはできません。

主キー

ORDER_ID 列および ITEM_ID 列は、主キー列です。

他の表との関係

OPTIM_DETAILS 表は、以下の表の子です。

- OPTIM_ORDERS 表 (ORDER_ID 列の外部キーを介して)。
- OPTIM_ITEMS 表 (ITEM_ID 列の外部キーを介して)。

OPTIM_ITEMS 表:

OPTIM_ITEMS 表には、1 つの注文の各品目に関する情報 (説明、価格、在庫数など) が含まれます。

OPTIM_ITEMS 表には、以下の列があります。

ITEM_ID
CHAR。最大 5 文字。NULL を含めることはできません。

ITEM_DESCRIPTION
VARCHAR。最大 72 文字。NULL を含めることはできません。

CATEGORY
VARCHAR。最大 14 文字。NULL を含めることはできません。

RATING
CHAR。最大 4 文字。NULL を含めることはできません。

UNIT_PRICE
DECIMAL。金額 (ドル)。NULL を含めることはできません。

ON_HAND_INVENTORY
INTEGER。NULL を含めることはできません。

主キー

ITEM_ID 列は、主キー列です。

他の表との関係

OPTIM_ITEMS 表は、ITEM_ID 列の外部キーを介した OPTIM_DETAILS 表の親です。

OPTIM_SHIP_TO 表:

OPTIM_SHIP_TO 表には、注文配送に関する情報が含まれます。

OPTIM_SHIP_TO 表には、以下の列があります。

CUST_ID
CHAR。最大 5 文字。NULL を含めることはできません。

SHIP_ID
DECIMAL。NULL を含めることはできません。

ADDRESS1
VARCHAR。最大 100 文字

ADDRESS2
VARCHAR。最大 100 文字

LOCALITY
VARCHAR。最大 56 文字

CITY VARCHAR。最大 30 文字

STATE
VARCHAR。最大 30 文字

COUNTRY_CODE
CHAR。2 文字の略語

POSTAL_CODE
VARCHAR。最大 15 文字

POSTAL_CODE_PLUS4
CHAR。4 文字

IN_CARE_OF
VARCHAR。最大 31 文字

SHIPPING_CHANGE_DT
TIMESTAMP。NULL を含めることはできず、デフォルト値があります。

主キー

SHIP_ID 列は、主キー列です。

他の表との関係

OPTIM_SHIP_TO 表は、SHIP_ID 列の Optim 関係を介した OPTIM_SHIP_INSTR 表の親です。

OPTIM_SHIP_TO 表は、CUST_ID 列の Optim 関係を介した OPTIM_CUSTOMERS 表の子です。

OPTIM_SHIP_INSTR 表:

OPTIM_SHIP_INSTR 表には、注文配送に関する詳細情報が含まれます。

OPTIM_SHIP_INSTR 表には、以下の列があります。

SHIP_ID
DECIMAL

SHIP_INSTR_ID
INTEGER

ORDER_SHIP_INSTR
VARCHAR。最大 254 文字

SHIP_UPDATED
TIMESTAMP。NULL を含めることはできず、デフォルト値があります。

主キー

SHIP_INSTR_ID 列は、主キー列です。

他の表との関係

OPTIM_SHIP_INSTR 表は、SHIP_ID 列の Optim 関係を介した OPTIM_SHIP_TO 表の子です。

OPTIM_MALE_RATES 表:

OPTIM_MALE_RATES 表には、年齢に基づく保険料率が含まれます。

OPTIM_MALE_RATES 表には、以下の列があります。

AGE SMALLINT

RATE_PER_1000
DECIMAL。金額 (ドル) による料率。

主キー

RATE_PER_1000 列は、主キー列です。

他の表との関係

OPTIM_MALE_RATES 表は、AGE 列の Optim データ駆動型関係を介した OPTIM_SALES 表の子です。

OPTIM_FEMALE_RATES 表:

OPTIM_FEMALE_RATES 表には、年齢に基づく保険料率が含まれます。

OPTIM_FEMALE_RATES 表には、以下の列があります。

AGE SMALLINT

RATE_PER_1000
DECIMAL。金額 (ドル) による料率。

主キー

RATE_PER_1000 列は、主キー列です。

他の表との関係

OPTIM_FEMALE_RATES 表は、AGE 列の Optim データ駆動型関係を介した OPTIM_SALES 表の子です。

OPTIM_STATE_LOOKUP 表:

OPTIM_STATE_LOOKUP 表には、州のコードおよび対応する略語が含まれます。

OPTIM_STATE_LOOKUP 表には、以下の列があります。

DIST_CODE
CHAR。3 文字。NULL 値を含めることはできません。

DISTRICT
CHAR。2 文字。NULL 値を含めることはできません。

主キー

OPTIM_STATE_LOOKUP 表には、主キーがありません。

他の表との関係

OPTIM_STATE_LOOKUP 表は OPTIM_SALES 表の子です。 SUBSTR(SALESMAN_ID,1,2) を使用した列 DISTRICT でのサブストリング関係を介しています。

データ・プライバシー表

Optim データ・プライバシー・ライセンスを保有するクライアントは、データ・プライバシー表を使用できます。これらの表を使用して、企業や個人のデータ (従業員名、顧客名、社会保障番号、クレジット・カード番号、E メール・アドレスなど) をマスクします。これらの表を使用すると、アプリケーションのコンテキスト内で固有かつ有効な変換データを生成することができます。

データ・プライバシー表を使用して、以下のことを行えます。

- 開発者、品質保証テスター、その他の人員が使用することのできるデータを識別不可にする (またはマスキングする) ことで、内部プライバシー違反を防ぎます。
- 顧客データをコンテキスト上正しい、しかし架空のデータに置き換えることで、プライバシー準拠イニシアチブを向上させます。
- アプリケーションの開発環境およびテスト環境において、機密の顧客情報および従業員データを保護します。
- マスクしたエレメントを関連表に伝搬し、データベースの参照整合性を確保することで、有効なテスト結果が確実に得られるようにします。

個人データの各カテゴリーは、オーストラリア (AU)、カナダ (CA)、フランス (FR)、ドイツ (DE)、イタリア (IT)、日本 (JP)、スペイン (ES)、英国 (UK)、米国 (US) 用にそれぞれ別個の表になっています (括弧内は略語)。各表には連続番号の入った列が含まれており、この列はルックアップ表の行の選択にハッシュ値を使用するルックアップ・ポリシーと共に使用されます。

それぞれの表名は、国の略語を示す接頭部とカテゴリーから成ります (*countryabbreviation_category*)。例えば、カナダ用の住所表の名前は CA_ADDRESSES、ドイツ用の住所表の名前は DE_ADDRESSES です。

スキーマには以下のカテゴリーが含まれます。

ADDRESSES
番地、市区町村、地域 (例えば州などの行政区分)、郵便番号の列を含む表。

FIRSTNAME
男性と女性のファーストネームの列を含む表。

FIRSTNAME_F
女性のファーストネームの列を含む表。

FIRSTNAME_M
男性のファーストネームの列を含む表。

LASTNAME
ラストネーム (姓) の列を含む表。

PERSON
生年月日、ファーストネーム、ラストネーム、性別、電話番号、国民識別番号、会社名、および E メール・アドレスの列を含む表。

CCN 関連する発行者 (MasterCard、VISA など) のクレジット・カード番号を含む表。

DOMAIN_NAMES
E メール・アドレスのマスキング用のドメイン・ネームを含む表。

サンプル・データ表の作成

サンプル・データ表を作成するには、Configuration ユーティリティーを使用します。

サンプル・データ表を作成するには、以下のようにします。

1. Optim Designer から、「**ユーティリティー**」 > 「**構成**」をクリックします。 Configuration ユーティリティーが開きます。
2. 「**タスク**」 > 「**サンプル・データのロード/ドロップ**」をクリックします。 「サンプル・データのロード/ドロップ」ウィザードが開きます。
3. ウィザードのステップを完了します。

   Optim ディレクトリーを選択しなければなりません。Optim リポジトリーのデフォルトの Optim ディレクトリー名は PODREPO です。

   DB 別名を選択するよう求めるプロンプトが出されたら、データ・ストア別名の名前を選択します。

データ・プライバシー表の作成

データ・プライバシー表を作成するには、Configuration ユーティリティーを使用します。

データ・プライバシー表を作成するには、以下のようにします。

1. Optim Designer から、「**ユーティリティー**」 > 「**構成**」をクリックします。 Configuration ユーティリティーが開きます。
2. 「**タスク**」 > 「**データ・プライバシー・データのロード/ドロップ**」をクリックします。 「データ・プライバシー・データのロード/ドロップ」ウィザードが開きます。
3. ウィザードのステップを完了します。

   Optim ディレクトリーを選択しなければなりません。 Optim リポジトリーのデフォルトの Optim ディレクトリー名は PODREPO です。

   DB 別名を選択するよう求めるプロンプトが出されたら、データ・ストア別名の名前を選択します。

データベース・サポート

Optim Designer は、複数のデータベース管理システムをサポートします。

Optim Designer は、以下のデータベースをサポートします。

- DB2® for z/OS® V8.1、V9.1、V10.1
- DB2 for Linux、UNIX、および Windows V8.2、V9.1、V9.5、V9.7
- DB2 for i V5.4
- Informix® V10
- Microsoft SQL Server 2005、2008
- Oracle V10.2、V11、V11.2
- Sybase V12.5、V15、V15.5
- Teradata V2.6、V12、V13
- IBM Informix V11.5、V.11.7

DB2 の前提条件

Optim を使用して、DB2 z/OS のインスタンスから完全な JDBC メタデータを取得するには、ZPARMS の DESCSTAT 値を YES に設定する必要があります。 また、JDBC で必要なストアー

ド・プロシージャーをインストールし、必要なパッケージをバインドし、セキュリティー権限を設定するために、ジョブ DSNTIJMS を実行する必要もあります。さらに、DB2 から要求された場合に、ワークロード・マネージャー (WLM) でストアード・プロシージャー・アドレス・スペースを開始できるようにするための WLM を定義しておく必要があります。

アクセシビリティー機能

アクセシビリティー機能は、運動障害や視覚障害など身体に障害を持つユーザー、また特別な支援を必要とするユーザーが、ソフトウェア・プロダクトを快適に使用できるようにサポートします。

Optim Designer は、Eclipse 環境で使用可能なアクセシビリティー機能を使用します。

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーまたは特別な支援を必要とするユーザーがソフトウェア・プロダクトを快適に使用できるようにサポートします。

第 2 章 データ・ソース接続の管理

データ・ストア別名を作成するには、Repository Explorer を使用します。 Directory Explorer を使用して Optim ディレクトリーに接続します。

データ・ストア別名での作業

データ・ストア別名とは、データベースに関連付けられているユーザー定義のオブジェクトのことです。データ・ストア別名を定義する場合には、Optim がデータベースとの通信に使用するパラメーターを指定します。これらのパラメーターには、データベース管理システム (DBMS) のタイプとバージョン、およびクライアント接続と JDBC 接続の両方のプロパティーが含まれています。

データ・ストア別名は、特定のデータベースにアクセスして要求された関数を実行できるようにする高位修飾子の役割を果たします。例えば、アクセス定義で、表の名前をデータ・ストア別名で修飾する必要があります。参照されたデータ・ストア別名は、表が存在するデータベースへの接続に必要なパラメーターを提供します。

データ・ストア別名は、特定のデータベースを識別し、主キー、データベース表、および関係の完全修飾名の接頭部の役割を果たします。データ・ストア別名は、データベースを管理する上で重要な要素です。データ・ストア別名には、以下のルールが適用されます。

- 各データベースにはデータ・ストア別名を 1 つだけ指定できる。
- 各データ・ストア別名は固有でなければならない。
- リポジトリー内のオブジェクトには、データ・ストア別名と同じ名前を付けることはできない。

データ・ストア別名の定義

新しいデータ・ストア別名を定義するには、「新規データ・ストア別名」ウィザードを使用します。

データ・ストア別名を定義するには、クライアント接続の接続ストリングと JDBC 接続の .jar ファイルの両方が必要です。

データ・ストア別名を定義するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**データ・ストア別名**」を右クリックし、「**新規データ・ストア別名**」をクリックします。 「新規データ・ストア別名」ウィザードが開きます。
2. ウィザードのステップを完了します。

   クライアント接続と JDBC 接続の両方のプロパティーを入力する必要があります。また、データベースで使用される文字セット、および Optim オブジェクトにアクセスするパッケージの管理方法も特定する必要があります。

データ・セット別名の編集

データ・ストア別名を編集するには、「接続プロパティー」ウィザードを使用します。

データ・ストア別名を編集するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Data Store Aliases**」ノードを展開してデータ・ストア別名を右クリックし、「**開く**」をクリックします。 「接続プロパティー」ウィザードが開きます。

2. ウィザードのステップを完了します。

データ・ストア別名への接続

データ・ストア別名に接続するには、Repository Explorer を使用します。

データ・ストア別名に接続するには、以下のようにします。

Repository Explorer で「**Data Store Aliases**」ノードを展開し、データ・ストア別名を右クリックして「**接続**」をクリックします。

Optim リポジトリーでの作業

Optim リポジトリーには、Optim コンポーネント間で共有される、データ・ストア別名、サービス、アクセス定義などの定義が含まれています。

リポジトリーに関連付けることができる Optim ディレクトリーは 1 つだけです。

Optim リポジトリーへの接続

Optim リポジトリーへの接続を定義するには、「新規リポジトリー接続」ウィンドウを使用します。

接続を作成するには、Optim リポジトリーが実行されている必要があります。

Optim リポジトリーへの接続を定義するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Optim リポジトリー**」を右クリックし、「**新規**」 > 「**リポジトリー接続**」をクリックします。 「新規リポジトリー接続」ウィンドウが開きます。
2. リポジトリーの接続情報とユーザー資格情報を入力します。
3. 「**接続のテスト**」をクリックして接続を検査します。
4. 「**OK**」をクリックします。

Optim Designer がリポジトリーに接続されると、Repository Explorer にリポジトリーの内容が表示されます。

リポジトリー接続の編集

Optim リポジトリーへの接続を定義するには、「リポジトリー接続の編集」ウィンドウを使用します。

Optim リポジトリーへの接続を定義するには、以下のようにします。

1. Repository Explorer で「**Optim リポジトリー**」を右クリックし、「**開く**」をクリックします。 「リポジトリー接続の編集」ウィンドウが開きます。
2. リポジトリーの接続情報とユーザー資格情報を入力します。
3. 「**接続のテスト**」をクリックして接続を検査します。
4. 「**OK**」をクリックします。

Optim Designer がリポジトリーに接続されると、Repository Explorer にリポジトリーの内容が表示されます。

リポジトリー接続の変更

別の Optim リポジトリーに接続するには、「リポジトリーの切り替え」ウィンドウを使用します。

接続を変更するには、Optim リポジトリーが実行されている必要があります。

Optim リポジトリーへの接続を変更するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Optim リポジトリー**」を右クリックし、「**リポジトリーの切り替え**」をクリックします。 「リポジトリーの切り替え」ウィンドウが開きます。
2. リポジトリー接続を選択します。
3. 「**OK**」をクリックします。

Optim Designer がリポジトリーに接続されると、Repository Explorer にリポジトリーの内容が表示されます。

Optim ディレクトリーとリポジトリーの関連付け

Optim ディレクトリーとリポジトリーを関連付けるには、「ディレクトリーの関連付け」ウィンドウを使用します。

Optim ディレクトリー接続は、Optim Designer コンピューターのレジストリーで定義する必要があります。レジストリー内の接続は Optim 構成プログラムを使用して定義します。

リポジトリーに関連付けることができる Optim ディレクトリーは 1 つだけです。

Optim ディレクトリーとリポジトリーを関連付けるには、次のようにします。

1. Directory Explorer で 「**リポジトリー接続に関連付ける Optim ディレクトリーをクリックしてください。**」をクリックします。 「ディレクトリーの関連付け」ウィンドウが開きます。
2. Optim ディレクトリーを選択します。
3. 「**完了**」をクリックします。

Optim Designer が Optimディレクトリーに接続されると、Directory Explorer にOptim ディレクトリーの内容が表示されます。

要求をサービスに変換する

Optim Service ウィザードに対する Transform Request を使用して、Optim ディレクトリー内の要求をデータ管理サービスに変換します。

Optim ディレクトリーは、リポジトリーに関連付けられている必要があります。

Optim ディレクトリー内の要求をデータ管理サービスに変換するには、以下のようにします。

1. ディレクトリー・エクスプローラーで、「**変換**」をクリックします。 「要求を Optim Service に変換」ウィザードが開きます。
2. 「DB 別名とデータベース接続」ページで接続情報を確認します。要求に関連付けられている DB 別名に対してデータ・ストア別名が定義されていない場合には、「**接続の編集**」をクリックしてデータ・ストア別名を定義します。「**次へ**」をクリックします。
3. 変換が成功した場合、「**終了**」をクリックします。変換が失敗した場合は、「**次へ**」をクリックしてエラー情報を確認します。

Repository Explorer に、要求が含まれている Optim ディレクトリーの名前が付いたフォルダーが作成されます。変換された要求が、フォルダー以下のサービスとしてリストされます。

第 3 章 データ・モデルの管理

抽出するデータを定義してデータ管理サービスにターゲット・データをマップする、データ・モデルを作成できます。また、Optim 関係と主キーを作成して、データベース内の関係と主キーを補足することができます。

アクセス定義での作業

処理するデータに対して、表、関係トラバーサル、および選択基準を指定するには、アクセス定義を使用します。

アクセス定義には以下の項目が含まれています。

表
アクセス定義は、少なくとも 1 つの表、ビュー、別名、またはシノニムを参照する必要があります。行が最初に選択される表、ビュー、別名、またはシノニムは、開始表と呼ばれます。開始表の名前を入力して、開始表に関連するすべての表の名前 (最大 24,000 個の表) を簡単に含めることができます。

開始表 開始表は、データを抽出する場合に最初に使用する表です。アクセス定義では、参照表以外の任意の表を開始表に指定できます。開始表を明示的に指定しない場合は、テーブル・リスト内の最初の表が開始表になります。

参照表 参照表の選択基準を指定しない限り、この表からすべての行が選択されます。開始表を除く任意の表を参照表として指定します。

リレーションシップ
リレーションシップによって、表からデータを選択するためのトラバーサル・パスが決定されます。デフォルトでは、リレーションシップは親から子にトラバースされますが、アクセス定義内の設定を使用してトラバーサル方向をコントロールすることができます。アクセス定義によって参照される表のリレーションシップが、リレーションシップ・タブにリストされます (最大 24,000 の表)。処理に使用するリレーションシップと、トラバースされる方向を選択できます。

選択基準
選択基準によって、アクセス定義内の表から使用する特定のデータ・セットを定義することができます。SQL 演算子と値を指定し、デフォルト値によって置換変数を使用することができます。

ポイント・アンド・シュート
ポイント・アンド・シュート・リストを使用すると、サービスに含まれる開始表から特定の行を選択できます。

変数 変数は、アクセス定義で指定されるユーザー定義のデフォルト値です。これらの置換変数を使用すると、列の選択基準の指定、または SQL WHERE 節の作成を行うことができます。

その他のパラメーター
開始表の特定の列の値に対応する行を抽出したり、指定された抽出率を使用 (n 番目の行ごと) したりするための追加パラメーターを使用します。

命名規則

完全修飾アクセス定義名の形式は、*identifier.name* です。

identifier
アクセス定義に割り当てられている修飾子 (1 から 8 文字)。

name アクセス定義に割り当てられるベース名 (1 から 12 文字)。

命名規則の論理セットを使用して、それぞれのアクセス定義の使用を識別し、それらを編成することにより、アクセスが容易になります。

アクセス定義の作成

アクセス定義を作成するには、「新規アクセス定義」ウィザードを使用します。

アクセス定義を作成する前に、開始表が含まれるデータベースのデータ・ストア別名が存在していなければなりません。

アクセス定義を作成するには、以下のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**アクセス定義**」を右クリックして「**新規アクセス定義**」をクリックします。 「新規アクセス定義」ウィザードが開きます。
2. ウィザードのステップを完了します。

   データ・ストア別名を選択し、別名から開始表を選択する必要があります。開始表に関連する表を追加することもできます。

Access Definition Editor に新しいアクセス定義が表示されます。

アクセス定義での表の管理

アクセス定義内で表を追加または削除するには、 Access Definition Editor の **Tables** タブを使用します。表を変更して、関連表または参照表にすることもできます。

アクセス定義への表の追加

データ・ストア別名の表をアクセス定義に追加するには、「表の追加」ウィザードを使用します。

アクセス定義に表を追加するには、次のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、**Access Definitions** ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**表**」 タブを選択します。
3. 「**表の追加**」をクリックします。 「表の追加」ウィザードが開きます。
4. ウィザードのステップを完了します。

   追加する表が含まれているデータ・ストア別名を選択する必要があります。参照表、または選択した表に関連する表を追加できます。
5. アクセス定義を保存します。

参照表または関連する表への変更

アクセス定義内の表を関連する表または参照表に変更するには、アクセス定義エディターを使用します。

表を参照表または関連する表に変更するには、次のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**Access Definitions**」 ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**表**」 タブを選択します。
3. 関連する表または参照表に変更する表を選択します。
4. 「**参照表に変更**」または「**関連する表に変更**」をクリックします。 「**タイプ**」列に、新しい表タイプが表示されます。
5. アクセス定義を保存します。

アクセス定義から表を削除する

アクセス定義から表を削除するには、Access Definition Editor を使用します。

アクセス定義から表を削除するには、以下のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**Access Definitions**」 ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**表**」 タブを選択します。
3. 削除する表を選択します。
4. 「**表の除去**」をクリックします。
5. アクセス定義を保存します。

トラバーサル・ステップの表示および編集

アクセス定義のトラバーサル・ステップを表示して編集するには、「トラバーサル・ステップ」ウィンドウを使用します。

表、関係、およびデータ選択のために実行されるステップを評価できます。この評価によって、目的のデータ集合を確実に取得できます。

処理のステップは、選択する関係および指定する基準に従い、任意の回数繰り返すことができます。

任意の表が後続のステップで複数回再アクセスされる場合があります。また、循環が含まれる場合もあります。循環によって、循環を完全に一周しても追加の行が選択されなくなるまで、表集合が繰り返しトラバースされます。

既に選択された子行に対して親行を選択するために関係がトラバースされる場合、親表に対する選択基準は無視されます。

トラバーサル・ステップを表示および編集するには、以下のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**Access Definitions**」ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**表**」 タブを選択します。
3. 「**ステップの表示**」をクリックします。 「トラバーサル・ステップ」ウィンドウが開きます。
4. ステップを確認して、表が選択される順序を編集します。
5. 「**OK**」をクリックします。
6. アクセス定義を保存します。

選択基準の管理

SQL WHERE 節を定義してデフォルト値で置換変数を使用することで、選択基準を使用して特定の関連するデータ集合にフォーカスすることができます。

選択基準は、SQL 構文に準拠していること、および関係演算子または論理演算子を含むことが必要です。論理演算子および構文は、DBMS によって異なります。詳しくは、該当する DBMS 資料を参照してください。

表の必要なデータ集合を選択するには、論理演算子 AND および OR の組み合わせが必要な場合があります。

ポイント・アンド・シュート・リストも使用されている場合には、論理演算子 OR によって、他の基準と合わせてインクルードされます。

関係が子から親にトラバースされる場合、親表の選択基準は無視されます。

日付基準

ユニーク演算子 **BEFORE** によって、DATE 列の値に基づきデータを選択できます。 この演算子の構文は、以下のとおりです。

BEFORE (*n***D** *n***W** *n***M** *n***Y**)

D、**W**、**M** および **Y** の引数を任意に組み合わせて使用して、実行時の日付から減算された日、週、月、年の数を示します。引数が指定されない場合、現在の日付が使用されます。計算された日付より古い日付の行が、抽出またはアーカイブされます。*n* の乗数は整数で、オプションで + または - を先頭に付けることができます。

選択基準の定義

SQL WHERE 節による選択基準を定義するには、「選択基準の編集」ウィンドウを使用します。

選択基準を定義するには、次のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**Access Definitions**」ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**表**」 タブを選択します。
3. 選択基準を追加する表を選択します。
4. 「**選択基準の編集**」をクリックします。 「選択基準の編集」ウィンドウが開き、表の WHERE 節が表示されます。
5. 選択基準を入力します。

   列と演算子のリストを表示するには、WHERE 節を右クリックして「**コンテンツ・アシスト**」をクリックします。

   変数を追加するには、「**変数区切り文字**」を選択して「**変数の挿入**」をクリックします。

   「**構文チェック**」をクリックして、構文エラーおよび確認エラーを特定します。
6. 「**OK**」をクリックします。

   構文が無効である場合は、エラーを確認するプロンプトが開きます。エラーが含まれている基準は保存できません。

7. アクセス定義を保存します。

データ・グループ・オプションの指定

グループ選択を指定するには、Access Definition Editor の「**データ・グループのプロパティー**」タブを使用します。この指定を行うと、開始表内の特定の列にある値に基づいていくつかの行が抽出されます。各グループ内の行の値は選択した列の値と同じになります。

グループ選択パラメーターと基準を組み合わせて使用すると、最初に基準が適用され、その結果にグループ選択パラメーターが適用されます。

データ・グループ・オプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**アクセス定義**」ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**データ・グループのプロパティー**」タブを選択します。
3. 列を選択し、グループ・オプションを指定します
4. アクセス定義を保存します。

データ・サンプリング・オプションと行制限オプションの指定

行のサンプリングを抽出するオプションや、抽出する行の数を制限するオプションを指定するには、Access Definition Editor の「**表**」タブを使用します。

ポイント・アンド・シュート・リストを使用して開始表の行を選択した場合、抽出サービスは開始表のデータ・サンプリング・パラメーターや行制限パラメーターを無視します。

データのサンプリングや行の制限に関するオプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**アクセス定義**」ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**表**」 タブを選択します。
3. 以下のオプションを入力します。

   N 番目の行ごと
   表のサンプリング係数を指定する数値を入力します。例えば、5 を入力すると、プロセスは、表内の行の抽出を 5 番目の行から開始して 5 行ごとに実行します。有効な値は 1 から 9999 です。

   行の制限
   数値を入力すると、表から抽出される行数が制限されます。有効な値は 1 から 999999999999 です。

4. アクセス定義を保存します。

アクセス定義での関係の管理

アクセス定義で関係を追加または削除するには、Access Definition Editor の **Relationships** タブを使用します。関係トラバーサルと表アクセスのオプションを指定することもできます。

関係トラバーサル・オプションの選択

アクセス定義内の関係トラバーサル・オプションを管理するには、アクセス定義エディターを使用します。

以下のオプションを指定できます。

- 関係について選択される子表からの行の最大数。
- 表に対して一度に実行されるキー参照の最大数。

  キー参照の制限数を増やすことで、パフォーマンスが大幅に向上する場合があります。例えば、キーに 1 つの列がある場合にキー参照の制限数を 5 に指定すると、DBMS に対する 1 回の要求で 5 つのキーの値が検索されます。
- トラバーサル・パスは以下のように機能します。
  - 子から親に関係をトラバースして、各子行に対する親行を選択して関係の整合性を確保できるようにします。 (オプション 1)
  - 関係をトラバースして、子から親へのトラバースの結果選択された各親行に対する子行を追加で選択します。(オプション 2)

  開始表が子表である場合、またはアクセス定義によって参照される複数の親表を表が持つ場合、オプション 1 および 2 は関連しています。

  オプション 2 は、子から親へ関係をトラバースする場合のみ関連しています。例えば、プロセスが子から親にトラバースし (オプション 1)、親行が選択されている場合、オプション 2 を選択すると、プロセスはその親行に対して追加の子行を選択します。

  関係に対してオプション 2 を選択する場合は、抽出する子行の数に関する子の制限を考慮します。

アクセス定義内の関係トラバーサル・オプションを管理するには、以下のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**Access Definitions**」 ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**関係**」タブを選択し、「**トラバーサル・オプション**」タブを選択します。
3. トラバーサル・オプションを入力します。
4. アクセス定義を保存します。

表アクセス・オプションの選択

各関係の親表または子表にアクセスする方式を選択するには、Access Definition Editor を使用します。

以下のオプションを指定できます。

デフォルト
: Optim が最適な方法を決定します。キー参照は、 DBMS 索引を使用できる場合に使用され、スキャンは、索引を使用できない場合に使用されます。ただし、表の大部分にアクセスする場合、索引が存在する場合でもデフォルトでスキャンが使用されます。

Force scan
: 表内のすべての行を一度に読み取ります。

Force key lookup
: WHERE 節を使用して行を特定し、主キーまたは外部キーの値を検索します。

注: プロセス・レポートの統計情報でデフォルトの方法が効率が悪いことが示されている場合にのみ、デフォルトの方法をオーバーライドします。

表のアクセス・オプションを管理するには、以下のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**Access Definitions**」 ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**関係**」タブを選択し、「**表へのアクセス**」タブを選択します。
3. 表のアクセス・オプションを入力します。
4. アクセス定義を保存します。

アクセス定義での変数の管理

変数は、アクセス定義で指定されるユーザー定義のデフォルト値です。これらの置換変数を使用すると、SQL WHERE 節を作成できます。

変数を割り当てることで、アクセス定義を処理するたびに変数に値が与えられます。オプションとして、置換変数にデフォルト値を指定できます。変数は、アクセス定義とともに保存されます。

デフォルト値

変数を作成すると、変数の実行時に値が割り当てられていない場合に使用される、オプションのデフォルト値を入力できます。

デフォルト値は、列に対して適切なデータ型およびサイズにする必要があり、SQL 構文規則に準拠している必要があります。例えば、変数名が **ST** (州)、変数の区切り文字がコロン (**:**) であり、列が文字データを必要とするとします。

- 単一引用符で囲まれている変数を選択基準で使用する場合、値は単一引用符で囲まずに指定する必要があります。

選択基準	**値**
= ':ST'	CA

- 単一引用符で囲まれていない変数を選択基準で使用する場合、値は単一引用符で囲んで指定する必要があります。

選択基準	**値**
= :ST	'CA'

注: デフォルト値は実行時まで検証されません。値が列にとって不適切なデータ型やサイズである場合や、SQL 構文規則に準拠していない場合は、処理エラーが発生することがあります。

プロンプト・ストリング

実行時に変数値に対してプロンプトで表示されるテキストを入力する必要があります。プロセス要求ダイアログで表示するプロンプト・ストリングをそのとおりに入力します (最大 50 文字)。この処理は、プロセスを実行する前に表示されます。

変数の作成

アクセス定義内に変数を作成するには、「変数の追加」ウィンドウを使用します。

変数を作成するには、以下のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**Access Definitions**」 ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**変数**」タブを選択します。
3. 「**追加**」をクリックします。 「変数の追加」ウィンドウが開きます。
4. 変数の情報を入力します。変数名およびプロンプト・テキストを入力する必要があります。「**OK**」をクリックします。
5. アクセス定義を保存します。

ポイント・アンド・シュート・リストの管理

ポイント・アンド・シュート・リストを使用すると、サービスに含まれる開始表から特定の行を選択できます。

ポイント・アンド・シュート・リストはアクセス定義に含まれています。ポイント・アンド・シュートを使用して開始表から行を選択する場合、これらの行の主キーはポイント・アンド・シュート・ファイルに格納されます。サービス要求は主キーを使用して、最初に処理する行を識別します。

ポイント・アンド・シュート・リスト・ファイルの作成

データ・ストア別名内の表のポイント・アンド・シュート・ファイル・リストを作成するには、「新規ポイント・アンド・シュート・ファイル」ウィンドウを使用します。

ポイント・アンド・シュート・リスト・ファイルを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**データ・ストア別名**」ノードを展開し、ターゲット・データ・ストア別名を展開して、ポイント・アンド・シュート・ファイルを追加する表を表示します。
2. 表を展開し、「**ポイント・アンド・シュート・リスト**」を右クリックして「**新規ポイント・アンド・シュート**」をクリックします。 「新規ポイント・アンド・シュート・ファイル」ウィンドウが開きます。
3. ファイル名を入力して「**OK**」をクリックします。

「**ポイント・アンド・シュート・リスト**」にポイント・アンド・シュート・リスト・ファイルが表示されます。

行またはポイント・アンド・シュート・リストを選択するには、Point and Shoot Editor を使用する必要があります。

ポイント・アンド・シュート・リストの選択

アクセス定義のポイント・アンド・シュート・リストを選択するには、Access Definition Editor を使用します。

ポイント・アンド・シュート・リストを選択するには、開始表でポイント・アンド・シュート・ファイルを使用できなければなりません。

ポイント・アンド・シュート・リストを選択するには、以下のようにします。

1. リポジトリー・エクスプローラーでアクセス定義が含まれているフォルダーを展開し、「**Access Definitions**」 ノードを展開して、アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**ポイント・アンド・シュート**」 タブを選択します。
3. ポイント・アンド・シュート・リストを選択します。

4. アクセス定義を保存します。

Optim 関係での作業

Optim は、関係を使用して、関係する表から取得されるデータを決定し、データベースに定義される関係を利用できる場合にはそれに基づきます。ただし、Optim 関係を定義して、データベース内の関係を補足することができます。

Optim の関係を定義する場合、いくつかのデータベースの制限が緩和されます。例:

- 主キーと外部キーは必須ではありません。
- 対応する列は同一である必要はありませんが、互換性がある必要があります。
- 対応する列の少なくとも 1 つのペアが、列名によって指定される必要があります。ただし、式を使用して、2 番目の列の値を評価または定義できます。式には、ストリング・リテラル、数値定数、NULL、連結、およびサブストリングを含めることができます。

より柔軟な Optim の関係は、「拡張された」関係と呼ばれます。拡張された関係では、データベースの暗黙的、またはアプリケーションが管理する関係を複製でき、実稼働環境と同じ方法でリレーショナル・データのセットを操作できます。

また、Optim 関係は、以下の関係として Optim ディレクトリーに格納できます。

- 表の単一のペアに使用される**明示的な**関係。
- 同じベース名、列名、および属性を持つが、作成者 ID が異なる表の 1 つ以上のペアに使用される**汎用の**関係

汎用の関係は、複数の表集合で、作成者 ID のみが異なる場合に役立ちます。(例えば、テスト環境で、各プログラマーが同じ実動表の異なるコピーを使用しているとします。各表集合は、作成者 ID によって区別できます。)汎用の関係を使用して、すべての表集合に適用される関係の 1 つのセットを定義します。また、これらの表集合が追加されるとき、汎用の関係が自動的に適用されます。

制限事項

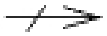
Optim の関係の作成でのルールはデータベースが定義する関係の作成よりも柔軟ですが、いくつかの制約事項があります。

- 関係の各表の少なくとも 1 つの列を参照している必要があります。
- 関係の表の最大 64 の列を参照できます。
- リテラルまたは定数を、リテラルまたは定数と突き合わせることはできません。
- ラージ・オブジェクト (LOB) 列または SQL 可変列は使用できません。
- 親表または子表のいずれかで指定されたすべての値の合計の長さは、3584 バイトを超えることができません。
- SQL Variant 列を使用して関係を作成することはできません。

マルチバイトまたは Unicode のデータベースの関係の定義では、以下の制約事項があります。

- サブストリング関数は使用できません。
- 文字データ (CHAR または NCHAR) とバイナリー (RAW) は連結できません。
- Oracle の文字セマンティクスが任意の CHAR 列に対して使用されている場合、関係のすべての CHAR 列は、文字セマンティクスまたは NCHAR データ型を持っている必要があります。

例:

親	サポート/非サポート	子	説明
CHAR		CHAR	サポートされています。セマンティクスが一致する必要があります。
NCHAR		NCHAR	サポートされています。セマンティクスは関連がありません。
CHAR		NCHAR	サポートなし
CHAR		VARCHAR	サポートされています。セマンティクスが一致する必要があります。
NCHAR		NVARCHAR	サポートされています。セマンティクスは関連がありません。
CHAR||NCHAR		NCHAR||CHAR	文字セマンティクスの場合はサポートされています。バイト・セマンティクスの場合はサポートされていません。
CHAR||NCHAR		NCHAR||NCHAR	文字セマンティクスの場合はサポートされています。バイト・セマンティクスの場合はサポートされていません。

関係の互換性ルール

Optim の関係を定義するとき、対応する値は互換性がある必要があります。

列タイプ	互換性がある対象
文字列	• 文字列 • 数値列 • ストリング・リテラル • 文字式
数値列	• 数値列 • 数値定数 • 文字列
バイナリー列	• バイナリー列 • 16 進数リテラル • 2 進式
ブール列	• ブール列 • ブール値定数 (True または False)
日時列	日時列
日付列	日付列

列タイプ	互換性がある対象
時刻列	時刻列
インターバル列	インターバル列

注:

- 処理中に、値は関連する行の選択に必要なデータ型に変換されます。デフォルトで、数値の文字データ型への変換の結果は、先行ゼロを使用して右寄せされます。特殊なレジストリー設定によって、デフォルトを先行スペースまたは末尾のスペースを使用した左寄せに変更できます。また、文字から数値のペア化では、数値列に 0 に等しい位取りが必要です。
- NULL が適格な列に対して NULL を使用できます。
- Unicode またはマルチバイトの列は同じ文字セットである必要があります。

データ型

以下のデータのクラスおよび関連付けられたデータ型がサポートされています。これらのデータ・クラスは、列の値を関係で使用するときのデータの互換性で重要になります。

文字クラス

DBMS	データ型
DB2	CHAR、VARCHAR、CLOB
Oracle	CHAR、VARCHAR2、LONG、CLOB、NCLOB、NCHAR、NVARCHAR
Sybase ASE	CHAR、VARCHAR、TXT
SQL Server	CHAR、VARCHAR、TXT
Informix	CHAR、VARCHAR、TXT

注: 1 バイト文字の列は、マルチバイトまたは Unicode 文字の列と互換性がありません。

数値クラス

DBMS	データ型
DB2	INTEGER、SMALLINT、DECIMAL、FLOAT、DOUBLE
Oracle	NUMBER、FLOAT
Sybase ASE	TINYINT、INT、SMALLINT、DECIMAL、FLOAT、REAL、MONEY、SMALL MONEY
SQL Server	TINYINT、INT、SMALLINT、DECIMAL、FLOAT、REAL、MONEY、SMALL MONEY
Informix	INTEGER、SMALLINT、DECIMAL、FLOAT、REAL、DOUBLE PRECISION、SMALLFLOAT、SERIAL、MONEY、NUMERIC

バイナリー・クラス

DBMS	データ型
DB2	CHAR (ビット・データの場合)、VARCHAR (ビット・データの場合)、BLOB
Oracle	RAW、LONG RAW
Sybase ASE	BINARY、VARBINARY、IMAGE
SQL Server	BINARY、VARBINARY、IMAGE

DBMS	データ型
Informix	BYTE

Boolean クラス

DBMS	データ型
Sybase ASE	BOOLEAN (TRUE または FALSE)

日付/時刻

DBMS	データ型
DB2	TIMESTAMP
Oracle	DATE、TIMESTAMP、TIMESTAMP WITH LOCAL TIME ZONE、TIMESTAMP WITH TIME ZONE
Sybase ASE	DATETIME、SMALL DATE TIME
SQL Server	DATETIME、SMALL DATE TIME
Informix	DATE、DATETIME

Date クラス

DBMS	データ型
DB2	DATE
Oracle	DATE
Informix	DATE

Time クラス

DBMS	データ型
DB2	TIME

Interval クラス

DBMS	データ型
Oracle	YEAR/MONTH INTERVAL、DAY/SECOND INTERVAL
Informix	YEAR/MONTH INTERVAL、DAY/TIME INTERVAL

Optim 関係の作成

「新規 Optim 関係」ウィザードを使用すると、新しい Optim 関係を作成できます。

Optim 関係を作成する前に、関係内の表のデータ・ストア別名が存在していなければなりません。

Optim 関係を作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Optim 関係**」を右クリックし、「**新規 Optim 関係**」をクリックします。 「新規 Optim 関係」ウィザードが開きます。
2. ウィザードのステップを完了します。

   関係内の親表および子表を選択する必要があります。

新しい Optim 関係が Relationship Editor に表示されます。

関係内の親列および子列を識別する列式を定義するには、リレーションシップ・エディターを使用する必要があります。

Optim 関係の編集

Optim 関係で列の親と子を識別する列式を定義するには、Relationship Editor を使用します。このエディターを使用して、総称関係を作成し、列を編集することができます。

列式の作成

Optim 関係における親列と子列を識別する列式を作成するには、「列式の追加」ウィンドウを使用します。

Optim 関係で列式を作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Optim Relationships**」ノードを展開し、編集する関係をダブルクリックします。 Relationship Editor が開きます。
2. 「**列式の追加...**」をクリックします。 「列式の追加」ウィンドウが開きます。
3. 「**親列の選択**」をクリックします。 「親列の選択」ウィンドウが開きます。
4. 親列を選択して「**OK**」をクリックすると、「列式の追加」ウィンドウに戻ります。
5. 「**子列の選択**」をクリックします。 「子列の選択」ウィンドウが開きます。
6. 子列を選択して「**OK**」をクリックすると、「列式の追加」ウィンドウに戻ります。
7. 「**OK**」をクリックしてエディターに戻ります。 エディターに親列と子列がリストされます。
8. 関係を保存します。

関係における列の編集

Optim 関係の列を編集するには、「親列の選択」および「子列の選択」ウィンドウを使用します。

Optim 関係の列を編集するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Optim Relationships**」ノードを展開し、編集する関係をダブルクリックします。 Relationship Editor が開きます。
2. 編集する列が含まれている行を選択します。 親列を編集するには、以下のようにします。
   a. 「**親列の選択**」をクリックします。 「親列の選択」ウィンドウが開きます。
   b. 親列を選択して「**OK**」をクリックするとエディターに戻ります。

   子列を編集するには、以下のようにします。
   a. 「**子列の選択**」をクリックします。 「子列の選択」ウィンドウが開きます。
   b. 子列を選択して「**OK**」をクリックすると、エディターに戻ります。

   選択した項目がエディターにリストされます。
3. 関係を保存します。

関係における列の順序の変更

Optim 関係で列の順序を変更するには、Relationship Editor を使用します。

Optim 関係の列の順序を変更するには、次のようにします。

1. リポジトリー・エクスプローラーで「**Optim Relationships**」ノードを展開し、編集する関係をダブルクリックします。 Relationship Editor が開きます。

2. 再配列する列が含まれている行を選択します。
3. 順序を変更するには、「**行を上に移動**」 または 「**行を下に移動**」 をクリックします。
4. 関係を保存します。

汎用関係の作成

汎用 Optim 関係を作成するには、Relationship Editor を使用します。

データベースによっては、作成者 ID を除いて同一である表のセットが含まれている場合があります。各表に対して関係を明示的に定義するのではなく、作成者ID に関わらず、ベース名が同じであるすべての表に対して、汎用関係を定義できます。汎用関係では基本表を変更できます。

汎用 Optim 関係を作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Optim Relationships**」ノードを展開し、編集する関係をダブルクリックします。 Relationship Editor が開きます。
2. 「**汎用**」をクリックします。
3. 関係を保存します。

Optim 主キーでの作業

主キーとは、表の各行を一意的に定義する値が含まれている列のことです。データベース表はOptim がデータベース表のデータを挿入、更新、リストア、または削除するための主キーを持っている必要があります。Optim は、データベースに定義されている主キーを使用します。ただし、Optim 主キーを定義して、データベース内の主キーを補足することができます。

主キーは以下の場合に必要です。

- 処理中に複数回アクセスされる表。例えばアクセス定義で参照される複数の親表を持つ子表など。
- 開始表に対してポイント・アンド・シュート機能を有効にする場合。

注: 主キーが定義されておらず、特定のタスクの実行に必要な場合、エラー・メッセージが表示されます。

Optim 主キーのタイプ

2 種類の主キーを定義できます。

- **明示的**な主キーは 1 つの表に適用されます。
- **汎用的な**主キーは、同じベース名、列名、および属性を持ち、作成者 ID が異なる表に適用されます。

汎用の主キーと明示的な主キーとの間で、機能または外観に違いはありません。ただし、表が両方のタイプのキーを持つ場合、明示的な主キーが使用されます。

命名規則

主キーの完全修飾名は、定義されているデータベース表の完全修飾名と同じです。この名前は *alias.creatorid.tablename* という構成になっています。

alias 表が存在するデータベースを特定する別名 (1 から 12 文字)。

creatorid
表に割り当てられている作成者 ID (1 から 64 文字)。

tablename
基本表の名前 (1 から 64 文字)。

注:

- 主キーに対する列の長さの合計は、3584 バイトに制限されています。

主キーの作成

表に対する新しい主キーを作成するには、「新規主キー」ウィザードを使用します。

主キーを作成する前に、表のデータ・ストア別名が存在していなければなりません。

主キーを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Optim 主キー**」を右クリックし、「**新規主キー**」をクリックします。 「新規主キー」ウィザードが開きます。
2. ウィザードのステップを完了します。

   表を選択する必要があります。

Primary Key Editor に新しい主キーが表示されます。

表の主キー列を設定するには、Primary Key Editor を使用する必要があります。

主キーの編集

表の主キーを編集するには、Primary Key Editor を使用します。キー列を選択し、汎用主キーを定義できます。

主キー列の選択

表の主キー列を選択するには、Primary Key Editor を使用します。キー列の順序を設定することもできます。

表の主キー列を選択するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Optim Primary Keys**」ノードを展開し、編集する関係をダブルクリックします。 Primary Key Editor が開きます。
2. 「**使用可能な列**」リストで、キー列を選択します。
3. **>>** をクリックして、列を「**キー列**」リストに移動させます。

   キー列の順序を設定するには、以下のようにします。

   a. 「**キー列**」リストで、移動させる列を選択します。
   b. 「**上へ移動**」をクリックして列を上位に移動させるか、「**下へ移動**」をクリックして列を下位に移動させます。
4. 主キーを保存します。

汎用主キーの作成

汎用 Optim 主キーを作成するには、主キー・エディターを使用します。

データベースによっては、作成者 ID を除いて同一である表のセットが含まれている場合があります。各表に対して主キーを明示的に定義するのではなく、作成者 ID に関わらず、ベース名が同じであるすべての表に対して、汎用主キーを定義します。

汎用 Optim 主キーを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで「**Optim Primary Keys**」ノードを展開し、編集する関係をダブルクリックします。 Primary Key Editor が開きます。
2. 「**汎用**」をクリックします。
3. 主キーを保存します。

表マップでの作業

表マップは、互換性のあるデータのソース表と宛先表を関連させるための仕様を定義します。名前が異なる表をマップし、表の名前を変更し、処理から表を除外し、または列マップを含めることで、データに対するコントロールを向上させることができます。

表マップによって以下のことができます。

- 変換、挿入、またはロード・サービスでデータの配置を指示する。
- 変換、挿入、またはロード・サービスから 1 つ以上の表を除外する。
- 列マップを含める。

表マップを使用する処理に応じて、2 つの表のセットがソース表と宛先表として参照されます。

- **ソース表**は、変換、挿入、またはロード処理で使用されるデータが抽出された表です。
- **ターゲット表**は、データの変換、挿入、またはロード先になる表です。

注: 一致した表でも、作成者 ID または名前が異なる場合があります。

表マップは Optim リポジトリーに格納されるか、データ管理サービスに組み込まれます。

表マップ内の表のペアに対して、任意で列マップを使用することもできます。列マップは、表のペア内の列を特定して突き合わせます。列マップは、列名または属性が一致しない場合、またはデータ変換が必要な場合に使用する必要があります。

命名規則

表マップの完全修飾名は、*identifier.name* という形式になります。

identifier
: 表マップに割り当てられている識別子 (1 から 8 文字)。

name
: 表マップに割り当てられている名前 (1 から 12 文字)。

論理的な命名規則のセットを使用して、それぞれの使用を区別し、定義を編成すると、アクセスが容易になります。

表マップの作成

新しい表マップを作成するには、「新規表マップ」ウィザードを使用します。

表マップを作成する前に、ソース表のファイルのデータ・ストア別名が存在していなければなりません。

表マップを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで表マップが含まれているフォルダーを展開し、「**表マップ**」を右クリックして、「**新規表マップ**」をクリックします。 「新規表マップ」ウィザードが開きます。
2. ウィザードのステップを完了します。

ソース表のファイルのデータ・ストア別名を選択する必要があります。

Table Map Editor に新しい表マップが表示されます。

表マップを保存する前に、エディターを使用してターゲット・データ・ストアおよびスキーマを定義する必要があります。

表マップの編集

ターゲット・データを編集し、列マップを追加するには、Table Map Editor を使用します。

表マップ内のデフォルトのターゲット・データの定義

表マップ内でデフォルトのターゲット・データ・ストアおよびスキーマを定義するには、Table Map Editor を使用します。

デフォルトの表マップのターゲット・データを定義する前に、ターゲット・データのデータ・ストア別名が存在していなければなりません。

表マップ内でデフォルトのターゲット・データを定義するには、以下のようにします。

1. リポジトリー・エクスプローラーで表マップが含まれているフォルダーを展開し、「**Table Maps**」 ノードを展開し、表マップをダブルクリックします。 Table Map Editor が開きます。
2. 「**表マップ**」タブを選択します。
3. 「**データ・ストア別名とスキーマのマップ**」領域で、ソース・データ・ストアが含まれている行を選択します。
4. ターゲット・データ・ストア別名を定義するには、「**ターゲット・データ・ストア別名**」セルをクリックし、リストからターゲット・データ・ストア別名を選択します。
5. ターゲット・スキーマを定義するには、「**ターゲット・スキーマ**」セルをクリックして、リストからターゲット・スキーマを選択します。
6. 表マップを保存します。

表マップ内のターゲット・データの編集

表マップ内のソース表のターゲット・データ・ストア、スキーマ、および表を編集するには、Table Map Editor を使用します。

表マップのターゲット・データを編集する前に、ターゲット・データについて、デフォルトのターゲット・データ・ストア別名が存在していなければなりません。

表マップ内のターゲット・データを編集するには、以下のようにします。

1. リポジトリー・エクスプローラーで表マップが含まれているフォルダーを展開し、「**Table Maps**」 ノードを展開し、表マップをダブルクリックします。 Table Map Editor が開きます。
2. 「**表マップ**」タブを選択します。
3. 「**表マップ**」領域で、ソース表が含まれる行を選択します。
4. ターゲット・データ・ストア別名を定義するには、「**ターゲット・データ・ストア**」セルをクリックし、ターゲット・データ・ストア別名の名前を入力します。
5. ターゲット・スキーマを定義するには、「**ターゲット・スキーマ**」セルをクリックして、リストからターゲット・スキーマを選択します。
6. ターゲット表を定義するには、「**ターゲット表**」セルをクリックして、リストからターゲット表を選択します。

7. 表マップを保存します。

表マップへの列マップの追加

マップされた表のペアに列マップを追加するには、Table Map Editor を使用します。

表マップに列マップを追加するには、次のようにします。

1. リポジトリー・エクスプローラーで表マップが含まれているフォルダーを展開し、「**Table Maps**」 ノードを展開し、表マップをダブルクリックします。 Table Map Editor が開きます。
2. 「**表マップ**」タブを選択します。
3. 「**表マップ**」領域で、列マップの表が含まれる行を選択します。
4. 「**列マップの追加**」をクリックします。 「新規列マップ」ウィンドウが開き、選択した表が含まれている列マップのリストが表示されます。
5. 特定の列マップを選択するか、「**新しい列マップの作成**」を選択して、選択した表に基づく列マップを作成します。「**OK**」をクリックします。
   a. 新しい列マップを作成する場合は、Column Map Editor が開き、選択した表から列が追加されます。
   b. 新しい列マップを編集して保存します。
6. 表マップを保存します。

列マップでの作業

列マップにより、データ管理サービスでの処理について列を一致させるまたは除外するために必要な仕様が得られます。変換、挿入、およびロード・サービスは、1 つ以上の列マップを参照する表マップを参照する必要があります。列マップを使用すると、関数または列マップ・プロシージャーによってデータ変換を定義することができます。

列マップは、列名または属性が一致しない場合、データ変換が必要な場合、または 1 つ以上の列を処理から除外する場合に使用する必要があります。変換、挿入、またはロード・サービスで参照される列マップでは、データの変更、日付の経年処理、または通貨の変換を行うことができます。関数または列マップ・プロシージャーによってデータ変換を定義することができます。

新しい列マップを作成する場合には、マップする列のソースとしてファイル・データ・ストア別名を選択する必要があります。同様に、ターゲット・データのデータ・ストア別名を指定する必要があります。

Optim リポジトリーに格納される列マップは、再利用、または他のユーザーとの共有のために使用できます。ローカルの列マップは、データ管理サービスの一環として保管され、それ以外のサービスには使用できません。関連付けられている表マップがサービスのローカルであり、表マップと列マップの両方とも、特定のサービスでのみ使用できます。

命名規則

列マップの完全修飾名は、*identifier.name* という 2 つの部分で構成されています。

identifier
　列マップに割り当てられている識別子 (1 から 8 文字)。

name　列マップに割り当てられている名前 (1 から 12 文字)。

列マップを作成する場合には、命名規則の論理セットを使用して定義を特定して編成すると、アクセスが容易になります。

列マップの互換性ルール

以下のデータのクラスおよび関連付けられたデータ型がサポートされています。これらのデータ・クラスは、列の値を関係および列マップで指定するときのデータの互換性で重要になります。

文字

DBMS	データ型
DB2	CHAR、VARCHAR、CLOB
Oracle	CHAR、VARCHAR2、LONG、CLOB、NCLOB、NCHAR、NVARCHAR
Sybase ASE	CHAR、VARCHAR、TXT
SQL Server	CHAR、VARCHAR、TXT
Informix	CHAR、VARCHAR、TXT

注: 1 バイト文字の列は、マルチバイトまたは Unicode 文字の列と互換性がありません。

数値

DBMS	データ型
DB2	INTEGER、SMALLINT、DECIMAL、FLOAT、DOUBLE
Oracle	NUMBER、FLOAT
Sybase ASE	TINYINT、INT、SMALLINT、DECIMAL、FLOAT、REAL、MONEY、SMALL MONEY
SQL Server	TINYINT、INT、SMALLINT、DECIMAL、FLOAT、REAL、MONEY、SMALL MONEY
Informix	INTEGER、SMALLINT、DECIMAL、FLOAT、REAL、DOUBLE PRECISION、SMALLFLOAT、SERIAL、MONEY、NUMERIC

バイナリー

DBMS	データ型
DB2	CHAR (ビット・データの場合)、VARCHAR (ビット・データの場合)、BLOB
Oracle	RAW、LONG RAW
Sybase ASE	BINARY、VARBINARY、IMAGE
SQL Server	BINARY、VARBINARY、IMAGE
Informix	BYTE

ブール値

DBMS	データ型
Sybase ASE	BOOLEAN (TRUE または FALSE)

日付/時刻

DBMS	データ型
DB2	TIMESTAMP
Oracle	DATE、TIMESTAMP、TIMESTAMP WITH LOCAL TIME ZONE、TIMESTAMP WITH TIME ZONE
Sybase ASE	DATETIME、SMALL DATE TIME
SQL Server	DATETIME、SMALL DATE TIME

DBMS	データ型
Informix	DATE、DATETIME

日付

DBMS	データ型
DB2	DATE
Oracle	DATE
Informix	DATE

時刻

DBMS	データ型
DB2	TIME

インターバル

DBMS	データ型
Oracle	YEAR/MONTH INTERVAL、DAY/SECOND INTERVAL
Informix	YEAR/MONTH INTERVAL、DAY/TIME INTERVAL

列マップの作成

新しい列マップを作成するには、「新規列マップ」ウィザードを使用します。

列マップを作成する前に、ソース・データとターゲット・データのファイル・データ・ストア別名が存在していなければなりません。

列マップを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで列マップを保存するフォルダーを展開し、「**列マップ**」を右クリックして「**新規列マップ**」をクリックします。 「新規列マップ」ウィザードが開きます。
2. ウィザードのステップを完了します。

   ソース・データのファイル・データ・ストア別名と表を選択する必要があります。また、ターゲット・データのデータ・ストア別名と表も選択する必要があります。

Column Map Editor に新しい列マップが表示されます。

列マップの編集

列マップを編集してデータ・マスキング関数を適用するには、Column Map Editor を使用します。

データ・マスキング関数の適用

列に関数を適用して編集するには、Column Map Editor を使用します。

データ・マスキング関数を適用するには、次のようにします。

1. リポジトリー・エクスプローラーで、列マップが含まれているフォルダーを展開し、「**列マップ**」ノードを展開して、列マップをダブルクリックします。 Column Map Editor が開きます。
2. ポリシーの列を選択します。

3. 「**関数の適用**」をクリックします。 「関数の適用」ウィンドウが開きます。
4. 適用する関数を選択します。「**OK**」をクリックします。 関連付けられている列に関数名が表示され、Column Map Editor で関数エディターが開きます。
5. 関数に応じて、次のいずれかのステップを行ってください。

   「関数式」タブを選択して関数式を編集します。

   オプション・タブを選択し、関数のオプションを選択します。
6. 列マップを保存します。

ソース列のマッピング

ソース列をターゲットにマップするには、Column Map Editor を使用します。同じ名前および互換性のあるデータ型を持つソース列およびターゲット列は、自動的にマップされます。

ソース列をマップするには、以下のようにします。

1. リポジトリー・エクスプローラーで、列マップが含まれているフォルダーを展開し、「**列マップ**」ノードを展開して、列マップをダブルクリックします。 Column Map Editor が開きます。
2. ソース列をクリックして、リストから列名を選択します。
3. 列マップを保存します。

列マップ・プロシージャーでの作業

列マップ・プロシージャーは、サービス実行時に列のデータをマスクしたり変換したりするために使われるプロシージャーです。列マップ・プロシージャーという名前が示すように、列マップにこれを追加する必要があります。 Lua スクリプト言語を使用して、列マップ・プロシージャーを作成することができます。

列マップ・プロシージャーの作成:

列マップ・プロシージャーを作成するには、Lua スクリプト・エディターを使用します。

列マップ・プロシージャーを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで、列マップが含まれているフォルダーを展開し、「**列マップ**」ノードを展開して、列マップをダブルクリックします。 Column Map Editor が開きます。
2. 列マップ・プロシージャーの列を選択します。
3. 「**プロシージャーの追加**」をクリックします。 Lua スクリプト・エディターが開きます。
4. プロシージャーを作成して保存します。 プロシージャーに関連付けられている列に、**プロシージャー**が表示されます。
5. 列マップを保存します。

列マップ・プロシージャーの編集:

列マップ・プロシージャーを編集するには、Lua スクリプト・エディターを使用します。

列マップ・プロシージャーを編集するには、次のようにします。

1. リポジトリー・エクスプローラーで、列マップが含まれているフォルダーを展開し、「**列マップ**」ノードを展開して、列マップをダブルクリックします。 Column Map Editor が開きます。
2. 列マップ・プロシージャーに関連付けられている列を選択します。
3. 「**プロシージャーの編集**」をクリックします。 Lua スクリプト・エディターが開きます。
4. プロシージャーを編集して保存します。

5. 列マップを保存します。

列マップ・プロシージャーのパラメーター値の入力:

列マップ・プロシージャーのパラメーター値を入力するには、Column Map Editor を使用します。

列マップ・プロシージャーのパラメーター値を入力するには、以下のようにします。

1. リポジトリー・エクスプローラーで、列マップが含まれているフォルダーを展開し、「**列マップ**」ノードを展開して、列マップをダブルクリックします。 Column Map Editor が開きます。
2. プロシージャーが含まれる列を選択します。 Column Map Editor で、「**プロシージャー**」エディターが開きます。
3. 「**プロシージャーのパラメーター**」フィールドで、パラメーターの値をコンマで区切って入力します。例えば `abc, def` のようにします。
4. 列マップを保存します。

第 4 章 データ管理サービスの設計

データを抽出、変換、ロード、または挿入するには、データ管理サービスを使用します。サービスではアクセス定義を使用して、抽出するデータを定義します。またサービスでは、表マップと列マップを使用して、ソース・データおよびターゲット・データをマップします。サービスが処理するエンティティーにデータ・プライバシー・ポリシーを適用することにより、データをマスクすることができます。 Optim Directory からデータ管理サービスに処理要求をトランスフォームすることもできます。埋め込みモードでサービスをテストするには、Optim Manager を使用します。

抽出サービスでの作業

抽出サービスを使用して、1 つ以上の表から関連する行のセットをコピーして、ファイル・データ・ストアに行を保存します。

抽出サービスでは、ソース表およびファイル・データ・ストアからデータとオブジェクト定義を抽出するために必要なパラメーターのセットが指定されます。抽出された情報はファイル・データ・ストアに格納されます。

抽出サービスでは、表および列の定義が常に抽出されます。これらの定義は、必要に応じて宛先表の作成に使用されます。また、主キー、関係、索引などのオブジェクト定義を抽出することもできます。

命名規則

抽出サービスの完全修飾名は、*identifier.name* で構成されています。

identifier
: 抽出サービス名の接頭部になる識別子 (1 から 8 文字)。

name 抽出サービスに割り当てられる名前 (1 から 12 文字)。

抽出サービスを作成する場合には、命名規則の論理セットを使用してそれぞれの使用を識別し、定義を編成することで、アクセスが容易になります。

抽出サービスの作成

抽出サービスを作成するには、「新規サービス」ウィザードを使用します。

抽出サービスではアクセス定義が必要になります。サービスの作成時には、アクセス定義を選択または作成することができます。

抽出サービスを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーで抽出サービスが含まれているフォルダーを展開し、「**サービス**」を右クリックして「**新規サービス**」をクリックします。 「新規サービス」ウィザードが開きます。
2. ウィザードのステップを完了します。

   アクセス定義を選択または作成する必要があります。ターゲット・ファイル・データ・ストアの名前を入力する必要もあります。

Extract Service Editor に新しい抽出サービスが表示されます。

抽出サービスの編集

抽出するデータ・オブジェクト、型変換、ファイル圧縮、変数のオーバーライドなど、抽出サービスの処理オプションを指定するには、Extract Service Editor を使用します。

抽出処理オプションの指定

抽出サービスの処理オプションを指定するには、Extract Service Editor の「**サービス・プロパティー**」 タブを使用します。

以下の抽出サービス処理オプションを指定できます。

- データベース接続の数を管理します。データベース接続の数を増やすと、複数のスレッドで同時に行を抽出できるようになるので、大量のデータを処理する場合のパフォーマンスが向上します。
- 抽出される行の数を制限します。
- 添付ファイルを許可します。
- 抽出サービス・レポートに統計情報を含めます。

抽出サービスの処理オプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーで抽出サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集する抽出サービスをダブルクリックします。 Extract Service Editor が開きます。
2. 「**サービス・プロパティー**」タブを選択します。
3. サービス処理オプションを編集します。
4. 抽出サービスを保存します。

オブジェクトおよびグループ・オプションの指定

抽出するデータ・オブジェクトを指定するには、Extract Service Editor の「**データおよびオブジェクト**」タブを使用します。

以下のオプションを指定できます。

- サービスで抽出する対象をデータのみ、オブジェクトのみ、または両方に指定します。
- 抽出するオブジェクトを選択します。

抽出するデータ・オブジェクトを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーで抽出サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集する抽出サービスをダブルクリックします。 Extract Service Editor が開きます。
2. 「**データおよびオブジェクト**」タブを選択します。
3. サービス処理オプションを編集します。
4. 抽出サービスを保存します。

変換オプションの指定

Extract Service Editor の「**変換**」タブを使用して、変換オプションを指定します。サービス内で抽出されたデータを変換することができます

以下のオプションを指定できます。

- サービスによって抽出データが変換するかどうかを指定します。
- 変換できずに廃棄される行の最大数を指定します。この最大数を超えると、サービスが停止します。

変換オプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーで抽出サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集する抽出サービスをダブルクリックします。 Extract Service Editor が開きます。
2. 「**変換**」タブを選択します。
3. サービス処理オプションを編集します。
4. 抽出サービスを保存します。

ファイル圧縮オプションの指定

ファイル・データ・ストアまたは抽出された表の圧縮オプションを指定するには、Extract Service Editor の「**ファイル圧縮オプション**」タブを使用します。

以下のオプションを指定できます。

- ファイルまたは特定の表が圧縮されているかどうかを示します。
- 各表について、圧縮しきい値を使用するか、可能な限り圧縮するかを選択できます。圧縮しきい値は、表の圧縮によって得られる最小の縮小率を示します。表のしきい値を設定するには、1 から 99 の範囲で値を入力します。

ファイル・データ・ストアでは以下の圧縮方式を使用できます。

インライン圧縮

データが抽出された時点で、ファイル・データ・ストアに書き込まれる前に圧縮されます。インライン圧縮ではポスト圧縮に比べて入出力が少なくなりますが、抽出サービス中にデータベース・リソースが使用されます。

ポスト圧縮に比べて、インライン圧縮では、抽出処理で必要とされるストレージ・リソースが少なくなります。

ポスト圧縮

抽出されたデータが、圧縮されていないファイル・データ・ストアに書き込まれます。次のステップでは、Optim では圧縮されていないファイル・データ・ストアが読み取られ、圧縮されたファイル・データ・ストアが書き込まれます。ポスト圧縮では、インライン圧縮に比べてデータベース接続が早く閉じられるという利点があります。ただしポスト圧縮では、圧縮されていないファイル・データ・ストアを閉じ、読み取り、さらに新しいバージョンの圧縮を作成する必要があるため、合計経過時間が増加します。

データベース・リソースが必要とされる時間が短縮されることから、 データベース・リソースの競合に関する懸念があるサイトでは、ポスト圧縮が有益である場合があります。ただしポスト圧縮では、経過時間が増加し、抽出サービスを処理するために必要なストレージ要件が増大します。圧縮操作ではより多くのストレージが必要になりますが、増加された一時ストレージは、圧縮が完了すると解放されます。

ファイル圧縮オプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーで抽出サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集する抽出サービスをダブルクリックします。 Extract Service Editor が開きます。
2. 「**ファイル圧縮オプション**」タブを選択します。
3. サービス処理オプションを編集します。
4. 抽出サービスを保存します。

変数オプションの指定

変数のデフォルト値をオーバーライドするには、Extract Service Editor の「**変数**」タブを使用します。

変数オプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーで抽出サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集する抽出サービスをダブルクリックします。 Extract Service Editor が開きます。
2. 「**変数**」タブを選択します。
3. デフォルト値をオーバーライドする値を入力します。
4. 抽出サービスを保存します。

抽出サービスのアクセス定義の変更

抽出サービスに関連付けられたアクセス定義を変更するには、「アクセス定義の変更」ウィザードを使用します。

抽出サービスに関連付けられたアクセス定義を変更するには、以下のようにします。

1. リポジトリー・エクスプローラーで抽出サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集する抽出サービスをダブルクリックします。 Extract Service Editor が開きます。
2. 「**変更**」をクリックします。 「アクセス定義の変更」ウィザードが開きます。
3. ウィザードのステップを完了します。

   アクセス定義を選択するか、「ローカル・アクセス定義の作成」を選択します。
4. 抽出サービスを保存します。

変換サービスでの作業

ファイル・データ・ストア内のデータをトランスフォームするには、変換サービスを使用します。データを変換してデータ・プライバシーを保証したり、アプリケーションのテスト要件に合わせてデータを体系的に変換したりすることができます。

表マップを指定して、ソース・ファイル・データ・ストア内の表とターゲット・ファイル・データ・ストア内の表を突き合わせるか、変換サービスから表を除外します。

表マップ内の列マップを使用して、変換するデータおよび変換方法を指定します。

命名規則

注: 変換サービスの完全修飾名は、*identifier.name* で構成されています。

identifier
: 変換サービス名の接頭部になる識別子 (1 から 8 文字)。

name
: 変換サービスに割り当てられる名前 (1 から 12 文字)。

変換サービスを作成する場合には、命名規則の論理セットを使用してそれぞれの使用を識別し、編成することで、アクセスが容易になります。

変換サービスの作成

変換サービスを作成するには、「新規サービス」ウィザードを使用します。

変換サービスでは表マップが必要になります。サービスの作成時には、表マップを選択または作成することができます。

変換サービスを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーでサービスが含まれているフォルダーを展開し、「**サービス**」を右クリックして「**新規サービス**」をクリックします。 「新規サービス」ウィザードが開きます。
2. ウィザードのステップを完了します。

   表マップを選択または作成する必要があります。ターゲット・ファイルのデータ・ストアの名前を入力するか、ソース・ファイルのデータ・ストアを上書きすることができます。

Convert Service Editor に新しい変換サービスが表示されます。

変換サービスの編集

変換サービスの処理オプションを指定するには、Convert Service Editor を使用します。

変換処理オプションの指定

変換サービスの処理オプションを指定するには、Convert Service Editor の「**処理オプション**」タブを使用します。

以下の変換サービス処理オプションを指定できます。

- 廃棄する行の制限を設定する。
- ターゲット・ファイルを圧縮する。
- 添付ファイルを許可する。

変換サービスの処理オプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーで、変換サービスが含まれているフォルダーを展開し、「**サービス**」ノードを展開し、編集する変換サービスをダブルクリックします。 Convert Service Editor が開きます。
2. 「**処理オプション**」タブを選択します。
3. サービス処理オプションを編集します。
4. 変換サービスを保存します。

変換サービスの表マップの変更

変換サービスに関連付けられた表マップを変更するには、「表マップの変更」ウィザードを使用します。

変換サービスに関連付けられた表マップを変更するには、以下のようにします。

1. リポジトリー・エクスプローラーで、変換サービスが含まれているフォルダーを展開し、「**サービス**」ノードを展開し、編集する変換サービスをダブルクリックします。 Convert Service Editor が開きます。
2. 「**変更**」をクリックします。 「表マップの変更」ウィザードが開きます。
3. ウィザードのステップを完了します。

   表マップを選択するか、「ローカル表マップの作成」を選択します。
4. 変換サービスを保存します。

挿入サービスでの作業

挿入サービスを使用して、ファイル・データ・ストアに保管されているデータを宛先データベースに挿入します。

表マップおよび列マップを使用して、ソースから宛先にデータをマップします。ファイル・データ・ストア内の表の宛先を指定するには、表マップを使用する必要があります。1 つ以上の宛先表について列マップを参照することもできます。列マップによって、各列のソース・データを指定し、データを挿入する前に任意でトランスフォームすることができます。

ファイル・データ・ストア内の表がターゲットに存在しない場合、Create ユーティリティーを使用して表を作成します。

命名規則

挿入サービスの完全修飾名は、*identifier.name* で構成されています。

identifier
挿入サービス名の接頭部になる識別子 (1 から 8 文字)。

name 挿入サービスに割り当てられる名前 (1 から 12 文字)。

挿入サービスを作成する場合には、命名規則の論理セットを使用してそれぞれの使用を識別し、編成することで、アクセスが容易になります。

挿入サービスの作成

挿入サービスを作成するには、「新規サービス」ウィザードを使用します。

挿入サービスでは表マップが必要になります。サービスの作成時には、表マップを選択または作成することができます。

挿入サービスを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーでサービスが含まれているフォルダーを展開し、「**サービス**」を右クリックして「**新規サービス**」をクリックします。 「新規サービス」ウィザードが開きます。
2. ウィザードのステップを完了します。

   表マップを選択または作成する必要があります。

Insert Service Editor に新しい挿入サービスが表示されます。

挿入サービスの編集

挿入サービスの処理オプションを指定するには、Insert Service Editor を使用します。行の挿入および削除、トリガーや制約の処理のためのオプションを指定できます。

挿入処理オプションの指定

挿入サービスの処理オプションを指定するには、Insert Service Editor の「**処理オプション**」タブを使用します。

以下の処理オプションを指定することができます。

- 実行される操作のタイプを選択します。表をロックし、コミット頻度と廃棄行制限を設定するパラメーターを指定します。

- すべてまたは指定した表から行を削除します。削除は、テスト中にデータをリストアするために有益です。何らかの理由で行を削除できない場合は、前回のコミットまでに削除されたすべての行がリストアされ、挿入処理が停止されます。
- データベース・トリガーを無効にします。
- データベース制約を無効にします。

行の処理では、以下のオプションを使用できます。

挿入　表に新しい行を挿入します。

- ソース・データ内の行の主キーが宛先表内の行の主キーに一致しない場合に、行が挿入されます。
- ソース・データ内の行の主キーが宛先表内の行の主キーに一致する場合には、行がバイパスされ、廃棄としてマークされます。

混合　「表ごとに指定」ウィンドウで選択した各表に従って、更新、挿入、または更新/挿入を行います。「表ごとに指定」ウィンドウを使用するには、「**表ごとに指定**」をクリックして、各表の処理オプションを選択します。

- 「**混合**」を選択して、「表ごとに指定」ウィンドウで選択項目を指定しない場合、またはすべての表に対して同じ選択項目を設定した場合は、処理オプションが変更され、すべての表に対して使用される処理が示されます。

注: 「**行削除オプション**」領域で「**すべての行を削除**」が選択されている場合は、「**混合**」を選択することはできません。

更新　表内の行を更新します。表には主キーが必要です。

- ソース・データ内の行の主キーが宛先表内の行の主キーに一致する 場合に、行が更新されます。
- ソース・データ内の行の主キーが宛先表内の行の主キーに一致しない場合には、その行は失敗として報告されます。

注: 「**行削除オプション**」領域で「**すべての行を削除**」が選択されている場合は、「**更新**」を選択することはできません。

更新/挿入

表で行を更新および挿入します。表には主キーが必要です。

- ソース・データ内の行の主キーが宛先表内の行の主キーに一致しない場合に、行が挿入されます。
- ソース・データ内の行の主キーが宛先表内の行の主キーに一致する 場合に、行が更新されます。

注: 「**行削除オプション**」領域で「**すべての行を削除**」が選択されている場合は、「**更新/挿入**」を選択することはできません。

挿入処理オプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーで挿入サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集する挿入サービスをダブルクリックします。 Insert Service Editor が開きます。
2. 「**処理オプション**」タブを選択します。
3. サービス処理オプションを編集します。
4. 挿入サービスを保存します。

挿入サービスの表マップの変更

挿入サービスに関連付けられた表マップを変更するには、「表マップの変更」ウィザードを使用します。

挿入サービスに関連付けられた表マップを変更するには、以下のようにします。

1. リポジトリー・エクスプローラーで挿入サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集する挿入サービスをダブルクリックします。 Insert Service Editor が開きます。
2. 「**変更**」をクリックします。 「表マップの変更」ウィザードが開きます。
3. ウィザードのステップを完了します。

   表マップを選択するか、「ローカル表マップの作成」を選択します。
4. 挿入サービスを保存します。

ロード・サービスでの作業

ロード・サービスを使用して、ファイル・データ・ストアの内容を特定の DBMS ローダーに適した形式に変換します。さらに、対応するデータベース・ロード・ユーティリティーを開始します (指定した場合)。

ロード・サービスによって、ファイル・データ・ストア内の表ごとに適切な形式のデータ・ファイルと、データベース・ローダーを開始するために必要な構文が含まれる SQL ファイルか BAT ファイル (バッチ実行) が生成されます。SQL ファイルか BAT ファイルのどちらが生成されるかは、DBMS によって決まります。

ロード・サービスには、DBMS ローダー用にデータを準備するために使用するパラメーターと、ロードの処理に必要な指示が含まれています。データをロードする宛先をマップするには、ロード・サービス内で表マップを指定します。データをロードする前にトランスフォームするには、ロード・サービスでオプションの列マップを使用します。

ロードするデータは、ファイル・データ・ストアに含まれている必要があります。

ロードと挿入

Optim では、ロード・サービスまたは挿入サービスを使用して、データベースにデータを移動することができます。どちらの方法を使用するかを判断する際は、以下の点を考慮してください。

- データのボリュームとデータベース・ロード・ユーティリティーを使用する速度によっては、挿入サービスの利点が相殺される場合があります。
- データには、すべてのデータを正常に挿入するための挿入サービスの機能を超える、参照整合性 (RI) サイクルが含まれている可能性があります。
- データベース・ロード・ユーティリティーでは、データベースの排他制御が必要とされ、ロード・サービス中のユーザー・アクセスが防止されます。挿入サービスの実行中には、他のユーザーもデータベースを使用できます。
- データベース・ロード・ユーティリティーでは、新しいデータが挿入されるか、既存のデータが置換されます。挿入サービスでは、更新/挿入処理を 1 つのステップで実行することができます。

出力ファイル名

ロード・サービスでは、以下のタイプのファイルが生成され、データベース・ユーティリティーのロード・プロセスがサポートされます。

データ・ファイル

データ・ファイルには、使用している DBMS に適した形式で作成された、ロードするデータが含まれています。Optim では、ファイル・データ・ストア内の各表に対して、データ・ファイルが生成されます。データ・ファイルにはファイル・データ・ストアと同じ名前が付けられますが、連続する番号が付けられたファイル名拡張子が付けられます。例えば、3 つの表が含まれた *demo.xf* という名前のファイル・データ・ストアの場合は、3 つのデータ・ファイル *demo.001*、*demo.002*、および *demo.003* が生成されます。

メッセージ・ファイル

メッセージ・ファイルには、ロード・サービス中にデータベース・ロード・ユーティリティーによって生成される情報が含まれています。一般的に、ロード・サービス全体に対して 1 つのメッセージ・ファイルが生成されます。メッセージ・ファイルには、ファイル・データ・ストアと同じ名前が付けられますが、拡張子 *.msg* が付けられます。例えば、ファイル・データ・ストアの名前が *demo.xf* である場合、メッセージ・ファイルの名前は *demo.msg* になります。

SQL ファイル

DB2 については、ローダーを手動で実行するローダー構文が含まれた各宛先表に対して、1 つのステートメントを持つ 1 つの SQL ファイルが生成されます。SQL ファイルには、ファイル・データ・ストアと同じ名前が付けられますが、拡張子 *.sql* が付けられます。

BAT ファイル

Oracle、Sybase ASE、SQL Server、および Informix については、各表のローダーを手動で実行する構文が含まれた BAT ファイルが生成されます。BAT ファイルは、表マップで指定されたデータ・ストア別名ごとに生成されます。各 BAT ファイルは、対応する変換されたファイル・データ・ストアと同じディレクトリーにあります。ローダーを手動で実行する場合は、BAT ファイルを (例えば Notepad で) 編集して、8 つの疑問符のストリングが特定のパスワード情報で置換されるようにする必要があります (Informix を除く)。

さらに、各データ・ファイルに対して定様式ファイルが生成されます。定様式ファイルには対応するデータ・ファイルと同じ名前が付けられますが、ファイル名の拡張子が異なります。ロードする表の数が 500 より少ない場合、定様式ファイル名の拡張子の数字は、データ・ファイル名の拡張子の数字に 500 を加えた数字になります。(例えば、*demo.001*、*demo.002*、*demo.003* という名前の 3 つのデータ・ファイルがある場合、対応する定様式ファイルの名前はそれぞれ *demo.501*、*demo.502*、*demo.503* になります。) 表の数が 500 を超える場合には、より複雑なファイル拡張子生成アルゴリズムが使用されます。

注: ファイル・サーバーで 8 文字を超える長いファイル名が許可されていない場合は、ファイル・データ・ストアに長い名前があると、ロード・サービスは失敗します。最良の解決方法は、ファイル・データ・ストアでは長いファイル名を使用しないことです。必要に応じて、ロード・サービスで使用する前にファイルをコピーして名前を変更することができます。

命名規則

ロード・サービスの完全修飾名は、*identifier.name* で構成されています。

identifier
: サービス名の接頭部になる識別子 (1 から 8 文字)。

name
: サービスに割り当てられる名前 (1 から 12 文字)。

ロード・サービスを作成する場合には、命名規則の論理セットを使用してそれぞれの使用を識別し、編成することで、アクセスが容易になります。

ロード・サービスの作成

「新規サービス」ウィザードを使用して、ロード・サービスを作成します。

ロード・サービスでは表マップが必要になります。サービスの作成時には、表マップを選択または作成することができます。

ロード・サービスを作成するには、次のようにします。

1. リポジトリー・エクスプローラーで抽出サービスが含まれているフォルダーを展開し、「**サービス**」を右クリックして「**新規サービス**」をクリックします。 「新規サービス」ウィザードが開きます。
2. ウィザードのステップを完了します。

   表マップを選択または作成する必要があります。

Load Service Editor に新しいロード・サービスが表示されます。

ロード・サービスの編集

ロード処理オプションを指定するには、Load Service Editor を使用します。ロード処理に関するオプションや、各ターゲット・データ・ストアに関連付けられている DBMS ローダーに関するオプションを指定できます。

ロード処理オプションの指定

ロード・サービスの処理オプションを指定するには、Load Service Editor の「**処理オプション**」タブを使用します。

以下のオプションを指定できます。

- 複数のデータ・ストア別名が使用されている場合は、複数の DBMS ローダーを実行する方法を選択します。パラレル (別の DBMS ローダーを同時に実行する) またはシーケンス (別の DBMS ローダーを順次実行する) を選択できます。
- エラーが発生した場合に DBMS ローダーを停止します。複数の DBMS ローダーを順次実行している場合は、エラーによって DBMS の処理が停止し、後続のすべての DBMS ローダーも停止されます。
- データベース・トリガーを無効にします。
- データベース制約を無効にします。

ロード・サービスの処理オプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーでロード・サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集するロード・サービスをダブルクリックします。 Load Service Editor が開きます。
2. 「**処理オプション**」タブを選択します。
3. サービス処理オプションを編集します。
4. ロード・サービスを保存します。

DBMS ローダー・オプションの指定

各 DBMS ローダーのオプションを指定するには、Load Service Editor の「**ロード・オプション**」タブを使用します。

以下のオプションを指定できます。

- 処理モード。

- 各ターゲット・データ・ストア別名に固有のオプション。

選択したターゲット・ファイル・データ・ストアの DBMS に応じて、以下の処理モードを使用できます。

挿入 ソース・ファイル・データ・ストアから、空のターゲット表に行が挿入されます。ターゲット表にデータが含まれている場合は、ローダーからエラーが返されます。

置換 ターゲット表内にある既存のすべての行がクリアされ、ソース・ファイル・データ・ストアの行によって置換されます。(「**置換**」ではロギングが実行されないため、「**切り捨て**」に比べてリソースを大幅に消費する可能性があります。)

追加 ソース・ファイル・データ・ストアから、ターゲット表に行が挿入されます。主キーの値が一致すると、重複する行は廃棄されるか、例外表に挿入されます (指定されている場合)。

切り捨て
「**切り捨て**」は「**置換**」と同様ですが、削除される行がデータベースに記録されません。また切り捨てでは RI 制約を無効にする必要があります。

DBMS ローダー・オプションを指定するには、以下のようにします。

1. リポジトリー・エクスプローラーでロード・サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集するロード・サービスをダブルクリックします。 Load Service Editor が開きます。
2. 「**ロード・オプション**」タブが開きます。
3. ターゲット・データ・ストア別名を選択します。
4. サービス処理オプションを編集します。
5. ロード・サービスを保存します。

ロード・サービスの表マップの変更

ロード・サービスに関連付けられた表マップを変更するには、「表マップの変更」ウィザードを使用します。

ロード・サービスに関連付けられた表マップを変更するには、以下のようにします。

1. リポジトリー・エクスプローラーでロード・サービスが含まれているフォルダーを展開し、「**Services**」ノードを展開し、編集するロード・サービスをダブルクリックします。 Load Service Editor が開きます。
2. 「**変更**」をクリックします。 「表マップの変更」ウィザードが開きます。
3. ウィザードのステップを完了します。

   表マップを選択するか、「ローカル表マップの作成」を選択します。
4. ロード・サービスを保存します。

データ管理サービスのテスト

Optim Manager を埋め込みモードで使用して、データ管理サービスをテストします。

データ管理サービスをテストするには、次のようにします。

1. リポジトリー・エクスプローラーで抽出サービスが含まれているフォルダーを展開し、**Services** ノードを展開します。
2. サービスを右クリックして「**サービスの実行**」をクリックします。 Optim Manager が開き、「サービスの実行」ウィンドウが表示されます。

3. 「**実行**」をクリックします。 サービスの進行をモニターするには、「**サービスのモニター**」 タブを使用します。

第 5 章 データのマスキング

国民 ID 番号、クレジット・カード番号、日付、数値、個人情報などをマスクすることができます。データ・マスキング関数を入力するか、LUA スクリプトを使用して列マップ・プロシージャーを作成するには、列マップを使用します。データをトランスフォームするには、変換サービスを使用します。

データ・マスキング関数の適用

列に関数を適用して編集するには、Column Map Editor を使用します。

データ・マスキング関数を適用するには、次のようにします。

1. リポジトリー・エクスプローラーで、列マップが含まれているフォルダーを展開し、「**列マップ**」ノードを展開して、列マップをダブルクリックします。 Column Map Editor が開きます。
2. ポリシーの列を選択します。
3. 「**関数の適用**」をクリックします。 「関数の適用」ウィンドウが開きます。
4. 適用する関数を選択します。「**OK**」をクリックします。 関連付けられている列に関数名が表示され、Column Map Editor で関数エディターが開きます。
5. 関数に応じて、次のいずれかのステップを行ってください。

   「関数式」タブを選択して関数式を編集します。

   オプション・タブを選択し、関数のオプションを選択します。
6. 列マップを保存します。

データ・マスキング関数

データ・マスキング関数では、さまざまな方法で機密データの変換またはマスキングを行うことができます。

Lookup 関数

Lookup 関数を使用して、参照表から宛先表に追加する値を選択します。Lookup 関数および Hash Lookup 関数を使用して、ソース値に基づき値を選択します。代わりに Random Lookup 関数を使用すると、ソース値に関係なく参照表から値を選択できます。

Lookup 関数

Lookup 関数は、ソース列の値に従い、参照表から宛先列の値を取得します。 Lookup 関数には、単一の列と複数の列の 2 つの形式があります。

単一列の形式では、1 つの宛先列に 1 つの値が挿入されます。複数の列の形式は、複数の参照表の列からの値を対応する宛先列に挿入します。

参照表の値によって置き換えられる任意のソース列に対して複数の列の Lookup 関数を入力できますが、列マップを編集して、同時に置換されるその他のソース列の名前を削除する必要があります。

ignore パラメーターを使用すると、指定したソース列の行に指定した値 (NULL、SPACES (CHAR 列の場合)、または長さゼロの VARCHAR) が含まれる場合に、参照表を無視してソース値を使用することができます。

preserve パラメーターを使用すると、指定したソース列の行に指定した値 (NULL、SPACES (CHAR 列の場合)、または長さゼロの VARCHAR) が含まれる場合に、参照表を無視してソース値を使用することができます。*preserve* はソース列に値が含まれていないときに参照表を無視する場合にも使用できます。

一致項目が参照表で検出されない場合、変換エラーが報告されます。

構文:

```
LOOKUP ( [sourcesearchcol, | SRCSEARCH=(sourcecol1,...,sourcecoln)]
           [dest=(col1, coln) , ]
           lktablename ( {LookupTableSearchcol
           | LKPSEARCH=(LookupTableSearchCol1,...,LookupTableSearchColn},
           {value | values=(col1, coln) }
 [,cache | ,nocache ] )
           [,ignore=(colname ( spaces, null, zero_len ), )
           | PRESERVE=( [ NOT_FOUND, ] colname (spaces, null, zero_len),... ) ] )
```

sourcesearchcol
: 単一列の検索では、検索値が含まれるソース表の列の名前 (オプション)。指定しない場合は、宛先列の名前が使用されます。

SRCSEARCH=
: 複数列の検索では、検索値が含まれるソース表の列の名前。列名は、括弧で囲み、コンマで区切る必要があります。

dest= 参照表の値が挿入される宛先表の列の名前。(複数の列の参照に必要です。)

col1, coln
: 宛先表の列の名前。列名の順序は、***values=*** パラメーターの参照表の列に対応している必要があります。

lktablename
: 参照表の名前。参照表の名前を、*dbalias.creatorid.tablename*、*creatorid.tablename*、または *tablename* と指定できます。表名を完全修飾しない場合は、宛先表の修飾子が使用されます。

LookupTableSearchcol
: 単一列の参照で、ソース列からの検索値に突き合わせるための値が含まれる参照表の列の名前。

value 宛先に挿入される変換後の検索値を含む、参照表内の列名。(単一の列の参照に必要です。)

values=
: 宛先で挿入される値が含まれる参照表の列の名前。(複数の列の参照に必要です。)

col1, coln
: 参照表の列の名前。列名の順序は、*dest=* パラメーターの宛先表の列に対応している必要があります。

cache | nocache
: 検出された参照値の表をメモリーに維持するには、*cache* (デフォルト) を指定し、検出された値を廃棄するには、*nocache* を指定します。*cache* を使用すると、値を複数回取得する場合はより高速になりますが、メモリーが余分に必要です。

ignore=
: 列に決まった値 (null、spaces、zero、または長さがゼロの varchar) を持つ行がある場合に参照値の代わりに宛先で挿入される値を持つソース列のリスト。

col　ソース列の名前。

For single column lookup, enter one column name only.

複数の列の参照では、列名の順序は、*dest=* パラメーターの宛先表の列に対応している必要があります。列の数は *dest=* パラメーターの列と等しい必要があり、少なくとも 1 つの列に値が含まれている必要があります。特定の列の値を指定しないようにする場合は、値を入力しないでください。例えば *coln()* のようにします。

null　ソース列の行に NULL 値がある場合に、参照表を無視します。

SPACES
ソース列の行に SPACES 値がある場合に、参照表を無視します。CHAR 列専用です。

ZERO_LEN
ソース列の行に長さがゼロの VARCHAR 値がある場合に、参照表を無視します。

preserve=
列に決まった値 (null、spaces、zero、または長さがゼロの varchar) を持つ行がある場合に参照値の代わりに宛先で挿入される値を持つソース列のリスト。

NOT_FOUND
ソース列の行に一致する値がない場合に、参照表を無視します。

注:

preserve= と *ignore=* は、同時に使用することはできません。*ignore=* は将来のリリースでは非推奨になります。

col、*null*、*spaces*、および *zero_len* オペランドは、*preserve=* または *ignore=* と合わせて使用する場合には同じ効果が得られます。

単一列の例

Lookup 関数を使用して、参照表のソース値を別の表の対応する値に変換します。

例えば、ソース列 STATE に、州の省略形 (**NJ** など) が含まれ、宛先列に、完全な州名 (この例では **New Jersey**) が含まれるとします。 STATE_LOOKUP という名前の参照表には、州の省略形またはコードに対する列 (CODE)、および対応する名前に対する列 (NAME) があります。

STATE_LOOKUP 表を使用して宛先列の値を取得するには、次のように指定します。

```
LOOKUP(STATE,STATE_LOOKUP(CODE,NAME))
```

Lookup 関数は、ソース表の STATE 列の値 (**NJ**) に一致する値を STATE_LOOKUP 表の CODE 列で検索します。一致が見つかった場合、関数は対応する NAME 列の値 (**New Jersey**) を宛先列に挿入します。

複数列の例

Lookup 関数を使用して、ソース列の値に基づき、参照表の行の列からの値を宛先表の行の列に挿入します。

例えば、社会保障番号が入ったソース列 (SOC_SEC) に基づき、宛先列 (FIRST_NAME および LAST_NAME) の値を参照表からの名前および姓で置き換えることができます。NAME_LOOKUP という名前の表には、ソース表の社会保障番号を持つ列 (SSN) と、宛先で対応する氏名をマスク処理するための列 (FIRST_MASK および LAST_MASK) が含まれています。

社会保障番号に基づき宛先表の名前を置き換えるには、以下のように指定します。

```
LOOKUP(SOC_SEC,DEST=(FIRST_NAME,LAST_NAME),
NAME_LOOKUP(SSN,VALUES=(FIRST_MASK, LAST_MASK)))
```

Lookup 関数は、ソース表の SOC_SEC 列の値に一致する値を NAME_LOOKUP 表の SSN 列で検索します。一致が見つかった場合、関数は対応する値を参照表の FIRST_MASK 列と LAST_MASK 列から対応する宛先列に挿入します。

無視の例

以下のステートメントを使用して、単一の列の例を拡張します。ここでは、参照表の値ではなくソース値 *NULL* および *SPACES* を使用します。

```
LOOKUP(STATE,STATE_LOOKUP(CODE,NAME),
IGNORE=(STATE(NULL,SPACES)))
```

NoCache の例

以下のステートメントを使用して、単一の列の例を拡張します。ここでは、特定された参照値の表をメモリーに維持しません。

```
LOOKUP(STATE,STATE_LOOKUP(CODE,NAME),NOCACHE)
```

Hash Lookup 関数

Hash Lookup 関数は、ソース列から派生するハッシュされた値に従い、参照表から宛先列の値を取得します。 Hash Lookup 関数を使用すると、任意の環境で同じソースおよび参照表を使用する場合に、データを一貫してマスクできます。

ハッシュ処理されるソース列は、参照表の値によって置換される列である必要はありません。

Hash Lookup 関数は、大/小文字の区別があります。例えば、ソース値 John および JOHN は、異なる値にハッシュされます。 TRIM パラメーターを使用して、ハッシュされる前にソース値を大文字に変換できます。

Hash Lookup 関数には、単一の列と複数の列の 2 つの形式があります。単一列の形式では、1 つの宛先列に 1 つの値が挿入されます。複数列の形式では、1 つのソース列から得られる単一のハッシュ値に基づいて、参照表の複数の列から対応する複数の宛先列に値が挿入されます。

参照表の値によって置き換えられる任意のソース列に対して複数の列の Hash Lookup 関数を入力できますが、列マップを編集して、同時に置換されるその他のソース列の名前を削除する必要があります。

参照表には、途切れることなく連続している番号の値を含むキー列と、置換値を含むその他の列を含める必要があります。 キー列は数値データ型である必要があります。 通常、参照表には順に番号が付けられています。 関数はソース列をハッシュして、参照表のキー列で 1 から最大値までの連続番号を派生させます。ソース表からハッシュされた値は、参照表の連続番号に突き合わせられ、対応する参照表の行からの値は、宛先で挿入されます。

ハッシュされた値を派生させるために使用されるソース列に特定の値 (NULL、spaces (CHAR 列の場合)、長さがゼロの VARCHAR) が含まれている場合、値はハッシュされず、以下の予約された値が参照表のキーとして使用されます。

ソース値	参照表へのキー
NULL	-1
スペース (CHAR または VARCHAR)	-2
ゼロ長 VARCHAR	-3

参照表には、これらの数値のそれぞれについて 1 行が含まれている必要があり、これによりこれらのソース値のそれぞれに対して 1 つの参照値を挿入できます。これらのソース値のいずれかが検出されたときに、対応する数値が参照表内に存在しない場合は、変換エラーが報告されます。

ignore パラメーターを使用すると、指定したソース列の行に指定した値 (NULL、SPACES (CHAR 列の場合)、または長さゼロの VARCHAR) が含まれる場合に、参照表を無視してソース値を使用することができます。

preserve パラメーターを使用すると、指定したソース列の行に指定した値 (NULL、SPACES (CHAR 列の場合)、または長さゼロの VARCHAR) が含まれる場合に、参照表を無視してソース値を使用することができます。*preserve* はソース列に値が含まれていないときに参照表を無視する場合にも使用できます。

trim パラメーターを使用すると、ハッシュされる前にソース値から切り捨てる文字を指定することができます。例えば、ソース値からコンマを切り捨てることを選択した場合、値 Smith、John、および Smith John はそれぞれ同じ値にハッシュされます。また、このパラメーターを使用して、ハッシュされる前にソース値を大文字に変換できます。

ソース値が大文字に変換される場合、トリム文字も大文字に変換されます。

seed パラメーターを使用して、ハッシュ・アルゴリズムで実行される計算を変更することもできます。ソース列と *seed* 値からハッシュされた値が参照表の順序番号とマッチングされ、宛先列の置換値が取得されます。

構文:

```
HASH_LOOKUP( [sourcecol,] [trim=([char1char2 ] [\u]),]
 dest=(col1, coln), lktablename (search,
  { value | values=(col1, coln) } ) [ ,cache | ,nocache ]
 [,ignore=(col (spaces, null, zero_len ), )
| PRESERVE=( [ NOT_FOUND, ] colname (spaces, null, zero_len), ) ] )[,seed=n])
```

sourcecol

ハッシュ値の派生元となる、ソース表の列の名前 (オプション)。指定しない場合は、宛先列の名前が使用されます。

trim= ハッシュ処理する前にソース値から切り取る文字のリストと、ハッシュ処理する前にソース値を大文字に変換するためのオプション。 文字のトリム後の結果のソース値が NULL または全桁スペースの場合、ソース値はハッシュされず、適切な予約値 (-1 または -2) が割り当てられます。

char1char2...

ハッシュされる前にソース値からトリムされる文字。このリストは大文字小文字が区別されます。スペースまたはコンマを文字として指定できます。文字の最初のオカレンスの後、リストの追加のオカレンスは無視されます。

円記号「¥」または右括弧「)」を指定するには、円記号のエスケープ文字をその文字の前に付ける必要があります。例えば右括弧を指定する場合には、*trim=(\))* と入力します。

エスケープ文字は、円記号、右括弧とともに使用するか、または大文字の標識の一部としてのみ使用できます。

\u　ソース値がハッシュされる前に大文字に変換されることを示します。トリムされる文字も、大文字に変換されます。

dest=　参照表の値が挿入される宛先表の列の名前。(複数の列の参照に必要です。)

col1,coln
宛先表の列の名前。列名の順序は、values= パラメーターの参照表の列に対応している必要があります。

lktablename
参照表の名前。参照表の名前を、**dbalias.creatorid.tablename**、**creatorid.tablename**、または **tablename** と指定できます。表名を完全修飾しない場合は、宛先表の修飾子が使用されます。

search　ソース列からのハッシュ値に突き合わせるための連続値が含まれる参照表の列の名前。

value　宛先に挿入される変換後の検索値を含む、参照表内の列名。(単一の列の参照に必要です。)

values=
宛先で挿入される値が含まれる参照表の列の名前。(複数の列の参照に必要です。)

col1,coln
参照表の列の名前。列名の順序は、*dest=* パラメーターの宛先表の列に対応している必要があります。

cache | nocache
検出された参照値の表をメモリーに維持するには、*cache* (デフォルト) を指定し、検出された値を廃棄するには、*nocache* を指定します。*cache* を使用すると、値を複数回取得する場合はより高速になりますが、メモリーが余分に必要です。

ignore=
列に決まった値 (NULL、SPACES、ZERO、または長さがゼロの VARCHAR) を持つ行がある場合に参照値の代わりに宛先で挿入される値のソース列のリスト。

col　ソース列の名前。

単一の列の参照では、列名を 1 つのみ入力します。

複数の列の参照では、列名の順序は、*dest=* パラメーターの宛先表の列に対応している必要があります。列の数は *dest=* パラメーターの列と等しい必要があり、少なくとも 1 つの列に値が含まれている必要があります。特定の列の値を指定しないようにする場合は、値を入力しないでください。例えば *coln()* のようにします。

NULL　ソース列の行に NULL 値がある場合に、参照表を無視します。

SPACES
ソース列の行に SPACES 値がある場合に、参照表を無視します。CHAR 列専用です。

ZERO_LEN
ソース列の行に長さがゼロの VARCHAR 値がある場合に、参照表を無視します。

preserve=
列に決まった値 (NOT_FOUND、null、spaces、または長さがゼロの varchar) を持つ行がある場合に参照値の代わりに宛先で挿入される値を持つソース列のリスト。

NOT_FOUND
ソース列の行に一致する値がない場合に、参照表を無視します。

注:

preserve= と *ignore=* は、同時に使用することはできません。*ignore=* は将来のリリースでは非推奨になります。

col、*null*、*spaces*、および *zero_len* オペランドは、*preserve=* または *ignore=* と合わせて使用する場合には同じ効果が得られます。

seed= ハッシュ・アルゴリズムの計算を変えるには、*seed=* を使用します。1 から 2,000,000,000 までの値を使用できます。 値 0 を使用すると、*seed=* パラメーターは無視されます。

単一列の例

Hash Lookup 関数を使用して、ソース列からハッシュされた値に基づき、参照表の列からの値を宛先表の列に挿入します。

例えば、ソース列 FIRST_NAME に名前が含まれ、宛先列に参照表からの置換する名前が含まれているとします。参照表 NAME_LOOKUP には名前の列 (FIRST) と、連続値の列 (SEQ) が含まれます。

NAME_LOOKUP 表を使用して宛先列の値を取得するには、次のように指定します。

```
HASH_LOOKUP(FIRST_NAME,NAME_LOOKUP(SEQ, FIRST))
```

Hash Lookup 関数は、ソース列からのハッシュ値を NAME_LOOKUP 表の SEQ 列の値と突き合わせます。一致が見つかった場合、関数は対応する FIRST 列の値を宛先列に挿入します。

複数列の例

Hash Lookup 関数を使用して、ソース列からハッシュされた値に基づき、参照表の行の列からの値を宛先表の行の列に挿入します。

例えば、名前を含むソース列 (FIRST_NAME) からハッシュ処理で得られた値に基づいて、宛先列 (FIRST および LAST) の値を参照表の名前と姓で置換することができます。NAME_LOOKUP という名前の表には、連続した値を持つ列 (SEQ) と、宛先の値をマスク処理するための列 (FIRST_MASK および LAST_MASK) が含まれています。

ソース列からハッシュされた値に基づき宛先表の名前を置き換えるには、以下のように指定します。

```
HASH_LOOKUP(FIRST_NAME,DEST=(FIRST,LAST), NAME_LOOKUP(SEQ,VALUES=(FIRST_MASK, LAST_MASK)))
```

Hash Lookup 関数は、ソースの FIRST_NAME 列からのハッシュ値を NAME_LOOKUP 表の SEQ 列の値と突き合わせます。一致が見つかった場合、関数は対応する値を参照表の FIRST_MASK 列と LAST_MASK 列から対応する宛先列に挿入します。

無視の例

以下のステートメントを使用して、単一の列の例を拡張します。ここでは、参照表の値ではなくソース値 NULL および SPACES を使用します。

```
HASH_LOOKUP(FIRST_NAME,NAME_LOOKUP(SEQ, FIRST),IGNORE=(FIRST_NAME(NULL,SPACES)))
```

NoCache の例

以下のステートメントを使用して、単一の列の例を拡張します。ここでは、特定された参照値の表をメモリーに維持しません。

```
HASH_LOOKUP(FIRST_NAME,NAME_LOOKUP(SEQ, FIRST),NOCACHE)
```

トリムの例

以下のステートメントを使用して、単一の列の例を拡張します。ここでは、ハッシュされる前にソース値からスペースとコンマをトリムし、ソース値を大文字に変換します。

```
HASH_LOOKUP(FIRST_NAME, TRIM=( ,\u),NAME_LOOKUP(SEQ,FIRST))
```

Random Lookup 関数

Random Lookup 関数は、指定された参照表から値をランダムに選択し、宛先列に挿入します。関数は 1 と指定した限度の値、または 1 と参照表の行の数との間で乱数を生成し、表への添え字として使用します。添え字に対応する行からの列の値は、宛先列に挿入されます。

Random Lookup 関数には、単一の列と複数の列の 2 つの形式があります。単一列の形式では、1 つの宛先列に 1 つの値が挿入されます。複数の列の形式は、複数の参照表の列からの値を対応する宛先列に挿入します。

参照表の値によって置き換えられる任意のソース列に対して複数の列の Random Lookup 関数を入力できますが、列マップを編集して、同時に置換されるその他のソース列の名前を削除する必要があります。

ignore パラメーターを使用すると、指定したソース列の行に指定した値 (NULL、SPACES (CHAR 列の場合)、または長さゼロの VARCHAR) が含まれる場合に、参照表を無視してソース値を使用することができます。

preserve パラメーターを使用すると、指定したソース列の行に指定した値 (NULL、SPACES (CHAR 列の場合)、または長さゼロの VARCHAR) が含まれる場合に、参照表を無視してソース値を使用することができます。

構文:

```
RAND_LOOKUP(lktablename, { columnname | dest=(col1,coln) ,values=(col1,coln) }
 [,limit ] [,ignore=(col(spaces, null, zero_len), ) | PRESERVE=( colname (spaces, null, zero_len), ) ] )
```

lktablename
: 参照表の名前。参照表の名前を、***dbalias.creatorid.tablename***、***creatorid.tablename***、または ***tablename*** と指定できます。表名が完全修飾されていない場合は、宛先表の修飾子が使用されます。

columnname
: 宛先での挿入のためにランダムに選択される値を含む参照表の列の名前。(単一の列の参照に必要です。)

dest=
: 参照表の値が挿入される宛先表の列の名前。(複数の列の参照に必要です。)

  col1,coln
  : 宛先表の列の名前。列名の順序は、*values=* パラメーターの参照表の列に対応している必要があります。

values=
: 宛先で挿入される値が含まれる参照表の列の名前。(複数の列の参照に必要です。)

  col1,coln
  : 参照表の列の名前。列名の順序は、*dest=* パラメーターの宛先表の列に対応している必要があります。

limit 列の値の選択に使用される参照表からの行の数の制限 (オプション)。2,000,000,000 以下の整数値を指定してください。制限値を指定しない場合は、すべての行が使用されます。

注: 列の値の表が、メモリーに生成されます。この表のサイズはシステム・リソースによって制限される場合があります。

ignore=

列に決まった値 (NULL、SPACES、または長さがゼロの VARCHAR) を持つ行がある場合に参照値の代わりに宛先で挿入される値を持つソース列のリスト。

col ソース列の名前。

単一の列の参照では、列名を 1 つのみ入力します。

複数の列の参照では、列名の順序は、*dest=* パラメーターの宛先表の列に対応している必要があります。列の数は *dest=* パラメーターの列と等しい必要があり、少なくとも 1 つの列に値が含まれている必要があります。特定の列の値を指定しないようにする場合は、値を入力しないでください。例えば *coln()* のようにします。

null ソース列の行に NULL 値がある場合に、参照表を無視します。

spaces ソース列の行に SPACES 値がある場合に、参照表を無視します。CHAR 列専用です。

zero_len

ソース列の行に長さがゼロの VARCHAR 値がある場合に、参照表を無視します。

preserve=

列に決まった値 (NOT_FOUND、null、spaces、または長さがゼロの varchar) を持つ行がある場合に参照値の代わりに宛先で挿入される値を持つソース列のリスト。

NOT_FOUND

ソース列の行に一致する値がない場合に、参照表を無視します。

注:

preserve= と *ignore=* は、同時に使用することはできません。*ignore=* は将来のリリースでは非推奨になります。

col、*null*、*spaces*、および *zero_len* オペランドは、*preserve=* または *ignore=* と合わせて使用する場合には同じ効果が得られます。

単一列の例

STATE_LOOKUP という表の最初の 50 行の STATE 列からランダムに値を選択して宛先列に挿入するには、以下のように指定します。

```
RAND_LOOKUP(STATE_LOOKUP,STATE,50)
```

複数列の例

STATE_LOOKUP という名前の表のランダムな行の CITY、STATE、および ZIPCODE 列から値を選択して、対応する宛先列に挿入するには、以下のように指定します。

```
RAND_LOOKUP(STATE_LOOKUP,
DEST=(CITY,STATE,ZIPCODE),
VALUES=(CITY,STATE,ZIP))
```

無視の例

以下のステートメントを使用して、単一の列の例を拡張します。ここでは、参照表の値ではなくソース値 NULL および SPACES を使用します。

```
RAND_LOOKUP(STATE_LOOKUP,STATE,50, IGNORE=(STATES(NULL,SPACES)))
```

Shuffle 関数

Shuffle 関数は、ソース値を列からの別の値で置き換え、その値が宛先列に挿入されます。ソース行と置換値を含む行は必ず異なりますが、データによっては、ソース値と置換値が同一である場合があります。

ソース値に一致しない値が見つかるまで関数が置換値を取得し直す (「再試行」) 回数を指定することができます。または、置換値とソース値の一致を許可することができます。 それぞれの Shuffle 関数は、列マップで使用されるその他の Shuffle 関数とは独立して機能します。

Shuffle 関数には、単一の列と複数の列の 2 つの形式があります。単一の列の形式は、置換値を単一の宛先列に挿入します。複数の列の形式は、行の複数の列からの置換値を対応する宛先列に挿入します。列は、列マップ内の複数の Shuffle 関数に含めることはできません。再試行フィーチャーが複数の列のシャッフルで使用される場合、ソース行の列の値が対応する置換行の列の値に一致するとき、関数は別の置換行を再フェッチします。 (複数の列の形式は、Propagate 関数では使用できません。)

複数列の Shuffle 関数を作成するには、シャッフルされた値によって置き換えられるソース列に対して関数を入力し、列マップを編集して、同時に置換される値を持つその他のソース列の名前を削除します。

ignore パラメーターは、ソース行または置換行のいずれかに、指定された値 (NULL、SPACES (CHAR 列の場合)、または長さがゼロの VARCHAR) が含まれる場合に、関数がソース行の置換または置換行を使用しないようにします。再試行が許可されていない場合、*ignore* パラメーターは置換行に適用されません。

構文:

```
SHUFFLE [ ( dest=(col1,coln) ) ] |
[ ( dest=(col1,coln) , retry[=number] ) ] |
[ ( dest=(col1,coln) [ , retry[=number] ] , ignore=( col1 ( [spaces] | [spaces,null]
| [spaces,null,zero_len] | [null] | [null,zero_len] | [zero_len] ) , coln (....) ) ) ] |
[ ( retry[=number] ) ] |
[ ( retry[=number] , ignore=( col ( [spaces] | [spaces,null] | [spaces,null,zero_len]
| [null] | [null,zero_len] | [zero_len] ) ) ) ] |
[ ( ignore=( col ( [spaces] | [spaces,null] | [spaces,null,zero_len]
| [null] | [null,zero_len] | [zero_len] ) ) ) ]
```

ここで、それぞれ以下のとおりです。

dest= 置換値が挿入される宛先表の列の名前。(複数の列のシャッフルに必要です。)

col1, *coln*, **...**
宛先表の列の名前。

retry ソース行と一致しない値を検索するために置換値を再フェッチする回数。 置換値をソースと一致させる場合は、ゼロを入力します。

注: 列に重複値が多数ある場合に再試行値を大きくすると、処理時間が長くなります。そのような列では、再試行値としてゼロを使用するのが望ましい場合もあります。

=number
0 から 1000 の範囲で値を入力します。置換値がソースと一致することを許可する場合は、0 を入力します。

ignore=

ソース値または置換値のいずれかが、指定された値 (NULL、SPACES (CHAR 列の場合)、または長さがゼロの VARCHAR) である場合に、関数がソース値を置換しない、または置換値を使用しない列のリスト。置換値が無視される場合、関数は別の置換値を再フェッチします。再試行が許可されていない場合、*ignore* パラメーターは置換値に適用されません。

col ソース列の名前。

単一の列のシャッフルでは、列名を 1 つのみ入力します。

複数の列のシャッフルでは、列名の順序は、*dest=* パラメーターの宛先表の列に対応している必要があります。列の数は *dest=* パラメーターの列と等しい必要があり、少なくとも 1 つの列に値が含まれている必要があります。特定の列の値を指定しないようにする場合は、値を入力しないでください。例えば *coln()* のようにします。

null ソース値または置換値のいずれかが NULL 値である場合には、ソース値を置換したり置換値を使用したりしないでください。

spaces ソース値または置換値のいずれかが SPACES 値である場合には、ソース値を置換したり置換値を使用したりしないでください。CHAR 列専用です。

zero_len
ソース値または置換値のいずれかが長さがゼロの VARCHAR 値である場合には、ソース値を置換したり置換値を使用したりしないでください。

単一列のデフォルトの例

以下の例では、シャッフルされた値を単一の列に挿入しています。

```
SHUFFLE
```

単一列の再試行の例

以下の例では、シャッフルされた値を単一の列に挿入して、ソースに一致しない置換値を最大 12 回再フェッチします。

```
SHUFFLE(RETRY=12)
```

複数列の例

以下の例では、シャッフルされた値を STATE 列および ZIP 列に挿入して、ソースに一致しない置換値を最大 12 回再フェッチします。

```
SHUFFLE(DEST=(STATE,ZIP),RETRY=12)
```

無視の例

以下の例では、シャッフルされた値を STATE 列および ZIP 列に挿入して、ソースに一致しない置換値を最大 12 回再フェッチします。また、この例では、ソース行または置換行に NULL 値または SPACES 値が含まれる場合に、STATE 列についてはソース値を置換しない、または置換値を使用しないが、ZIP 列については、ソース行または置換行を無視しません。

```
SHUFFLE(DEST=(STATE,ZIP),RETRY=12,
IGNORE=(STATE(NULL,SPACES),ZIP()))
```

TRANS SSN 関数

TRANS SSN 関数を使用して、有効な固有の米国社会保障番号 (SSN) を生成します。デフォルトでは、TRANS SSN は、ソースの SSN に基づいて、一貫性のある方法で変更された宛先 SSN をアルゴリズムにより生成します。また、TRANS SSN では、ソース・データが SSN 値を持たない場合、あるいはソースの SSN を一貫性のある方法で変換する必要がない場合は、ランダムな SSN を生成することもできます。

SSN は 3 つのサブフィールドから成ります。 最初の 3 桁 (地域) は、その SSN が発行された州によって一般的に定められた地域を表します。 次の 2 桁 (グループ) は、地域番号に対応するグループ番号を定義したものです。 最後の 4 桁 (通し番号) は、順次シリアル番号です。 処理のタイプ (デフォルトまたはランダム) に関係なく、TRANS SSN は地域番号に適合したグループ番号を持つ SSN を生成します。

デフォルトの処理方法では、ソースの地域番号、およびソース SSN に基づき変更されたグループ番号とシリアル番号を含む SSN が生成されます。

ランダムな処理方法では、ソースの地域番号を含めることができる SSN が生成され、宛先地域番号に対して社会保障庁によって発行された最新のグループ番号が使用されます。シリアル番号は 0001 で始まり、地域番号に対して追加の SSN が生成されるごとに 1 ずつ増分します。シリアル番号が 9999 を超えると、シリアル番号は 0001 にリセットされ、その地域番号に対して発行された最新の番号の前のグループ番号が使用されます。

TRANS SSN の構文は、以下のとおりです。

TRANS SSN [('[=*flags*] [*sourcecol* **[preserve=invalid]')]**

flags 大/小文字を区別しない 1 つ以上の処理オプション・フラグを指定できます。

- *n* ソース値に基づかないランダムな SSN を生成します。
- *m* 773 から 899 までの値を含み、無効な地域番号を除外した、すべての SSN 地域値の最大グループを使用します。
- *r* 入力された SSN と同じ州に対応するランダム地域番号を持つ SSN を生成します。
- *v* ソースのグループ番号を社会保障局で使用されている番号と比較して、グループ番号を検証します。
- - 宛先 SSN には、フィールドを区切るダッシュが含まれている必要があります (例: 123-45-6789)。少なくとも 11 文字の長さの文字型の宛先列を必要とします。

sourcecol
ソース列の名前。ソース列名が指定されていない場合、宛先列名が使用されます。ソース列名が指定されておらず、宛先列名がソース表の列名と一致しない場合、処理中にエラーが発生します。

preserve=invalid
ソース列に無効な SSN が含まれている場合は、それを生成された値で置換しないでください。ソース列の値は宛先列で使用されます。

許可されているデータ型

以下のソース・データ型および宛先データ型が許可されています。

CHAR 列のデータ長は 9 から 256 文字である必要があります。

DECIMAL
列の精度は 9 から 20 で、位取りは 0 にする必要があります。

INTEGER
制限なし。

VARCHAR
列のデータ長は 9 から 254 文字である必要があります。

ソース列または宛先列がこれらの制限に準拠しない場合、処理中にエラーが発生します。

宛先処理のルール

宛先の SSN 値には、宛先のデータ型または値に応じて以下のルールが適用されます。

CHAR ソース値が 0、スペース、または長さゼロの VARCHAR の場合は、宛先値はスペースに設定されます。

ソース値が 11 文字以上で埋め込みのダッシュ (-) を含んでいるか、'-' フラグが指定されている場合は、宛先列の長さが 11 文字以上であれば、宛先値にダッシュが含まれます。

DECIMAL, INTEGER
ソース値が 0、スペース、または長さゼロの VARCHAR の場合は、宛先値は 0 になります。

VARCHAR
ソース値が 0、スペース、または長さゼロの VARCHAR の場合は、宛先の長さは 0 になります。

ソース値が 11 文字以上で埋め込みのダッシュ (-) を含んでいるか、'-' フラグが指定されている場合は、宛先列の長さが 11 文字以上であれば、宛先値にダッシュが含まれます。

NULL ソース値が NULL である場合は、宛先値は NULL になります。

スキップされる行

以下の条件の場合に、ソース行がスキップされ、宛先に書き込まれないことがあります。

- ソース値が NULL で、宛先列で NULL 値が許可されていない場合。
- ソース列が CHAR または VARCHAR で、ソース値が 9 文字未満の場合、ソース値に非数値の文字 (3 つのサブフィールド間のダッシュは除く) が含まれている場合、またはソース値が大きすぎる場合。
- ソースの地域番号が社会保障局で使用されていない場合。
- ソースのグループ番号が、社会保障局によってその地域番号では使用されていない場合 ('v'フラグが指定されている場合のみ)。
- ソースのシリアル番号が 0000 である場合、または SSN が社会保障局で発行されていない予約値 (例えば、078-05-1120) である場合。
- ソース値を TRANS SSN がサポートする形式に変換できない場合。

エラー・メッセージ

以下のエラー・メッセージが発行される場合があります。

SSN01 *Parm on Col ccccc ("ppp") is invalid*

解説 列に無効な処理オプション・フラグが設定されている TRANS 関数が含まれています。

ユーザー・アクション
指定されている列の TRANS 関数が有効な処理オプション・フラグ (n、r、v、-) を使用していることを確認してください。

SSN02 *Col ccccc not on source*

解説 *sourcecol* パラメーターとして入力された列、または宛先列名 (*sourcecol* パラメーターが省略されている場合) がソース表で見つかりませんでした。

ユーザー・アクション
ソース表をチェックして、不一致または欠落した列を解決してください。

SSN03 *Source Col ccccc-aaa invalid*

解説 示されている属性が無効なため、ソース列の形式がサポートされていません。

ユーザー・アクション
ソース列をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。

SSN04 *Dest Col ccccc-aaa invalid*

解説 示されている属性が無効なため、宛先列の形式がサポートされていません。

ユーザー・アクション
宛先列をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。

SSN05 *Get col ccccc data-rc=nnn*

解説 ソース列から値を取得している間に、予期しない内部エラーが発生しました。

ユーザー・アクション
ソース列および宛先列の値をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。問題が解決しない場合は、 IBM ソフトウェア・サポートに連絡してください。

SSN08 *Put col ccccc data-rc=nnn*

解説 宛先列で値を設定している間に、予期しない内部エラーが発生しました。

ユーザー・アクション
ソース列および宛先列の値をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。問題が解決しない場合は、 IBM ソフトウェア・サポートに連絡してください。

他のエラーが発生した場合は、技術サポートに連絡してください。

例 1

以下の例では、宛先列に一致するソース列名を使用して、ソース値に基づかないランダムな SSN を生成します。

```
TRANS SSN ('=n')
```

例 2

以下の例では、宛先列とは異なるソース列名 (NATIONAL_ID) を使用して、デフォルトの処理方法を使用し、ダッシュが含まれる SSN を生成します。

```
TRANS SSN ('=- NATIONAL_ID')
```

TRANS CCN 関数

TRANS CCN 関数を使用して、有効でユニークなクレジット・カード番号 (CCN) を生成します。デフォルトでは、TRANS CCN は、ソースの CCN に基づいて、一貫性のある方法で変更された CCN をアルゴリズムにより生成します。また、TRANS CCN では、ソース・データが CCN 値を持たない場合、あるいはソースの CCN を一貫性のある方法で変換する必要がない場合は、ランダムな CCN を生成することもできます。

ISO 7812 で定義された CCN の構成では、まず 6 桁の発行者 ID があり、その後に可変長のアカウント番号が続き、最後にチェック・ディジットとして 1 桁の数字があります。 チェック・ディジットは CCN の正確性を検査するためのもので、発行者 ID とアカウント番号を Luhn アルゴリズムに通すことによって生成されます。 CCN の最大長は 19 桁です。

デフォルトの処理方法では、ソース CCN からの発行者 ID の最初の 4 桁を含み、発行者 ID 番号の残りの 2 桁およびアカウント番号をソース CCN に基づいて変更することによって、CCN を生成します。また、有効なチェック・ディジットも割り当てられます。

ランダムな処理方法の場合、生成される CCN にはソースの発行者 ID 番号の最初の 4 桁か、American Express、Discover、MasterCard、または VISA に割り当てられた発行者 ID 番号が含まれる可能性があります。また、有効なチェック・ディジットも割り当てられます。ソース発行者の ID 番号の最初の 4 桁が含まれている場合、それらの桁に基づく最初のアカウント番号は 1 で始まり、それらの桁を使用する追加の CCN ごとに、アカウント番号は 1 ずつ増分します。

TRANS CCN の構文は、以下のとおりです。

```
TRANS CCN [( '[=flag] [sourcecol] [preserve=invalid]' )]
```

flag ランダムな CCN を生成するためのオプション・フラグを指定します。

- *n* American Express、Discover、MasterCard、または VISA に割り当てられた発行者 ID 番号を持つ、ソース値に基づかないランダムな CCN を生成します。
- *r* ソース発行者の ID 番号の最初の 4 桁を含むランダムな CCN を生成します。

sourcecol

ソース列の名前。ソース列名が指定されない場合には、宛先列名が使用されます。

ソース列名が指定されておらず、宛先列名がソース表の列名と一致しない場合、処理中にエラーが発生します。

preserve=invalid

ソース列に無効な CCN が含まれている場合は、それを生成された値で置換しないでください。ソース列の値は宛先列で使用されます。

許可されているデータ型

以下のソース・データ型および宛先データ型が許可されています。

CHAR 列の長さは 13 から 256 文字にする必要があります。

VARCHAR

列の長さは 13 から 254 文字にする必要があります。

DECIMAL

列の精度は 13 から 254 で、位取りは 0 にする必要があります。

ソース列または宛先列がこれらの制限に準拠しない場合、エラー・メッセージが表示されます。

宛先処理のルール

宛先のデータ型または値に従い、以下のルールが宛先 CCN 値に適用されます。

CHAR ソース値がスペース、または長さゼロの VARCHAR の場合は、宛先値はスペースに設定されます。

VARCHAR
ソース値がスペース、または長さゼロの VARCHAR の場合は、宛先の長さは 0 になります。

DECIMAL
ソース値が 0 の場合、宛先値は 0 です。

NULL ソース値が NULL である場合は、宛先値は NULL になります。

スキップされる行

以下の条件の場合に、ソース行がスキップされ、宛先に書き込まれないことがあります。

- ソース値が NULL で、宛先列で NULL 値が許可されていない場合。
- ソース値が 13 文字未満の場合、ソース値に非数値の文字が含まれている場合、ソース値が大きすぎる場合、またはソース値のチェック・ディジットが誤っている場合。
- ソース値の長さが、クレジット・カードの発行者に対して無効である場合。
- ソース値を TRANS CCN がサポートする形式に変換できない場合。

エラー・メッセージ

以下のエラー・メッセージが発行される場合があります。

CCN01

Parm on Col ccccc ("ppp") is invalid

解説 示されている列に無効な処理オプション・フラグが設定されている TRANS 関数が含まれています。

ユーザー・アクション
指定されている列の TRANS 関数が有効な処理オプション・フラグ (n、r、6) を使用していることを確認してください。

CCN02

Col ccccc not on source

解説 *sourcecol* パラメーターとして入力された列、または宛先列名 (*sourcecol* パラメーターが省略されている場合) がソース表で見つかりませんでした。

ユーザー・アクション
ソース表をチェックして、不一致または欠落した列を解決してください。

CCN03

Source Col ccccc-aaa invalid

解説 示されている属性が無効なため、ソース列の形式がサポートされていません。

ユーザー・アクション
ソース列をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。

CCN04

Dest Col ccccc-aaa invalid

解説　示されている属性が無効なため、宛先列の形式がサポートされていません。

ユーザー・アクション
宛先列をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。

CCN05

Get col ccccc data-rc=nnn

解説　ソース列から値を取得している間に、予期しない内部エラーが発生しました。

ユーザー・アクション
ソース列および宛先列の値をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。問題が解決しない場合は、 IBM ソフトウェア・サポートに連絡してください。

CCN08

Put col ccccc data-rc=nnn

解説　宛先列で値を設定している間に、予期しない内部エラーが発生しました。

ユーザー・アクション
ソース列および宛先列の値をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。問題が解決しない場合は、 IBM ソフトウェア・サポートに連絡してください。

他のエラーが発生した場合は、技術サポートに連絡してください。

例 1

以下の例では、宛先列と異なるソース列名 (CREDITCARD) を使用して、ソース値に基づかないランダムな CCN を生成します。

```
TRANS CCN ('=n CREDITCARD')
```

例 2

以下の例では、宛先列と異なるソース列名 (CREDITCARD) を使用して、デフォルトの処理方法を使用し、CCN を生成します。

```
TRANS CCN ('CREDITCARD')
```

TRANS EML 関数

E メール・アドレスを生成するには、TRANS EML 関数を使用します。E メール・アドレスは、ユーザー名とその後に続くドメイン・ネームを「@」で区切った 2 つの部分から構成されます。例えば、user@domain.com のようになります。

TRANS EML は、宛先データまたは連続番号に連結されたリテラルのいずれかに基づいて、ユーザー名が含まれる E メール・アドレスを生成します。ドメイン名は、ソース・データの E メール・アドレスまたはリテラルに基づいて生成されるか、大規模な E メール・サービス・プロバイダーのリストからランダムに選択されます。E メール・アドレスは、大文字または小文字に変換することもできます。

TRANS EML は、(通常はユーザーの名前を含む) 1 つまたは 2 つの宛先表の列の値に基づき、ユーザー名を生成できます。処理オプションを使用して、最初の列の値の最初の文字 (名前の頭文字など) のみを使用したり、ピリオドまたは下線を使用して 2 つの列の値を区切ったりすることができます。

ユーザー名が 1 つの宛先列の値またはリテラルに基づいている場合、名前は連続番号と連結されます。ユーザー名が宛先表の 2 つの列の値に基づいており、区切りのピリオドや下線が使用されていない場合、値は連結されます。ユーザー名のパラメーターが指定されていない場合、名前はリテラル“email”に連続番号を連結した形式になります。ユーザー名の連続番号は、1 で始まり 1 ずつ増加する接尾部です。

TRANS EML の構文は、以下のとおりです。

```
TRANS EML [( '[=flags] , [{sourcecol | "domain" | , }
[{name1col[name2col] | "userpfx"}] ] [preserve=invalid]' )]
```

flags 大/小文字を区別しない 1 つ以上の処理オプション・フラグを指定できます。

n 大規模 E メール・サービス・プロバイダーのリストからランダム・ドメイン・ネームを生成します。

. *name1col* と *name2col* の値をピリオドで区切ります。

_ *name1col* と *name2col* の値を下線で区切ります。

i *name1col* の値の先頭文字のみを使用します。

l E メール・アドレスを小文字に変換します。

u E メール・アドレスを大文字に変換します。

sourcecol

ドメイン・ネームの指定に使用される E メール・アドレスを含むソース列名。

'n' フラグも *domain* パラメーターも定義されていない場合は、ソース列のドメイン名が使用されます。(*sourcecol* が定義されていない場合は、宛先列名に基づいてソース列名が決定されます。)

ソース列名が指定されておらず、宛先列名がソース表の列名と一致しない場合、処理中にエラーが発生します。

domain ドメイン・ネームを形成する最大 31 文字のリテラル。

, *sourcecol* パラメーターまたは *domain* パラメーターのいずれも定義されておらず、ドメイン・ネームに対してリテラルまたは列名のいずれかを定義する場合、コンマが必要です。

name1col

ユーザー名の最初 (または唯一) の部分を形成するために使用される値を含む宛先表の列名。

name2col

ユーザー名の 2 番目の部分を形成するために使用される値を含む宛先表の列名。

userpfx ユーザー名を形成するための連続番号と連結した最大 31 文字のリテラル。

preserve=invalid

ソース列に無効な E メール・アドレスが含まれている場合は、それを生成された値で置換しないでください。ソース列の値は宛先列で使用されます。

許可されているデータ型

以下のソース・データ型および宛先データ型が許可されています。

CHAR 列の長さは 3 から 4096 文字である必要があります。

VARCHAR

列の長さは 3 から 4094 文字である必要があります。

ソース列または宛先列がこれらの制限に準拠しない場合、エラー・メッセージが発行されます。

宛先処理のルール

宛先のデータ型または値に従い、以下のルールが宛先の E メールの値に適用されます。

CHAR ソース値がスペース、または長さゼロの VARCHAR の場合は、宛先値はスペースに設定されます。

VARCHAR
ソース値がスペース、または長さゼロの VARCHAR の場合は、宛先の長さは 0 になります。

NULL ソース値が NULL である場合は、宛先値は NULL になります。

スキップされる行

以下の条件の場合に、ソース行がスキップされ、宛先に書き込まれないことがあります。

- ソース値が NULL で、宛先列で NULL 値が許可されていない場合。
- ソース値が 3 文字未満の長さの VARCHAR である場合。
- ソースの E メールの値に「@」が含まれていない場合。
- ソース値を TRANS EML がサポートする形式に変換できない場合。

エラー・メッセージ

以下のエラー・メッセージが発行される場合があります。

EML01

Parm on Col ccccc ("ppp") is invalid

解説 示されている列に無効な処理オプション・フラグが設定されている TRANS 関数が含まれています。

ユーザー・アクション
指定されている列の TRANS 関数が有効な処理オプション・フラグ (n, . , _ , i, l, u) を使用していることを確認してください。

EML02

Col ccccc not on source

解説 *sourcecol* パラメーターとして入力された列、または宛先列名 (*sourcecol* パラメーターが省略されている場合) がソース表で見つかりませんでした。

ユーザー・アクション
ソース表をチェックして、不一致または欠落した列を解決してください。

EML03

Source Col ccccc-aaa invalid

解説 示されている属性が無効なため、ソース列の形式がサポートされていません。

ユーザー・アクション
ソース列をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。

EML04

Dest Col ccccc-aaa invalid

解説 示されている属性が無効なため、宛先列の形式がサポートされていません。

ユーザー・アクション

宛先列をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。

EML05

Get col ccccc data-rc=nnn

解説 ソース列から値を取得している間に、予期しない内部エラーが発生しました。

ユーザー・アクション

ソース列および宛先列の値をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。問題が解決しない場合は、 IBM ソフトウェア・サポートに連絡してください。

EML08

Put col ccccc data-rc=nnn

解説 宛先列で値を設定している間に、予期しない内部エラーが発生しました。

ユーザー・アクション

ソース列および宛先列の値をチェックし、型、長さ、精度、および位取りの値が適切であることを確認してください。問題が解決しない場合は、 IBM ソフトウェア・サポートに連絡してください。

EML09

Domain literal sssss too long

解説 ドメイン・ネームのリテラル (domain) として指定されたストリングが、最大制限の 31 文字を超えています。

ユーザー・アクション

31 文字以下で構成されるドメイン・ネームを指定してください。

EML10

User literal sssss too long

解説 ユーザー名のリテラル (*userpfx*) として指定されたストリングが、最大制限の 31 文字を超えています。

ユーザー・アクション

31 文字以下で構成されるユーザー名を指定してください。

EML11

Name1 Col ccccc not on dest

解説 示される TRANS 関数を実行するには、*name1col* 列が宛先表で指定されている必要があります。

ユーザー・アクション

指定された *name1col* 列が、TRANS 関数で示される *name1col* 列と一致していることを確認してください。

name1col の列名が宛先表で検出されませんでした。

EML12

Name1 Col ccccc-aaa invalid

解説 示される TRANS 関数を実行するには、指定されている *name1col* 列が有効な形式である必要があります。

ユーザー・アクション

name1col 列のアベイラビリティー、型、および長さが適切であることを確認してください。

EML13

Name2 Col ccccc not on dest

解説 示される TRANS 関数を実行するには、*name2col* 列が宛先表で指定されている必要があります。

ユーザー・アクション

指定された *name2col* 列が、TRANS 関数で示される *name2col* 列と一致していることを確認してください。

name1col の列名が宛先表で検出されませんでした。

EML14

Name2 Col ccccc-aaa invalid

解説 示される TRANS 関数を実行するには、指定されている *name2col* 列が有効な形式である必要があります。

ユーザー・アクション

name2col 列のアベイラビリティー、型、および長さが適切であることを確認してください。

他のエラーが発生した場合は、技術サポートに連絡してください。

例 1

以下の例では、リテラル (optim.com) を使用してドメイン・ネームを形成し、2 つの宛先表の列 (NAME_FIRST および NAME_LAST) を使用して、下線が含まれるユーザー名を形成します。

```
TRANS EML ('=_ "optim.com" NAME_FIRST NAME_LAST')
```

例 2

以下の例では、ソース列からのドメイン・ネームおよびリテラル (OptimUser) を使用して、連続番号が接尾部に使用されるユーザー名を形成します。

```
TRANS EML (', "OptimUser" ')
```

TRANS COL 関数

TRANS COL 関数では、固有の形式がないデータ、または一般的でない形式のデータをマスクできます。TRANS COL では、宛先のソース・データの形式と文字タイプが維持されます。

ソース・データが大文字である場合、TRANS COL では宛先で英大文字が生成されます。この関数では英数字がマスクされますが、ソース・データ内のその他の文字は変更されずに宛先にコピーされます。TRANS COL を使用して、CHAR、VARCHAR、および非浮動数値データ型をマスクできます。固有の値、同じソースのオカレンスごとに異なる値、またソースとは異なる長さの値を生成できます。

TRANS COL の構文は、以下のとおりです。

```
TRANS COL ( '{ unique | hash }[ source=colname ]
[ copy=( (start,len [, "lit" ] )...) ]
```

[seed= {*"lit"* | var (*variable*)| RANDOM}]
[length=*n* | max] [preserve=([null] [spaces] [zero_len])]
[TRIM=(*char1 [charn,...]* [\u] [\r])] [num]')

unique 固有の宛先値を生成します。宛先値の長さは、ソース値の長さと同じになります。

hash ソース値のハッシュによって宛先値を生成します。ハッシュを使用すると、異なるソース値によって、処理が実行されるたびに同じ宛先値が生成される場合があります。

注:
固有のパラメーターまたはハッシュ・パラメーターを使用した場合には、 同じソース値に対して、同じ宛先値を取得することができます。ハッシュをシード・パラメーターと併せて使用すると、処理を実行するたびに異なる宛先値が生成されます。

source=**colname**
宛先列名とソース列名が異なる場合にソース列の名前を指定するには、このパラメーターを使用します。指定した値は大文字に変換されます。値が大文字に変換されないようにするには、値を二重引用符で囲みます。

copy= 1 つ以上のサブストリングのペアが、マスクされずに宛先にコピーされます。リテラル・ストリングを指定すると、指定された位置のソース文字が置換されます。*copy=* パラメーターは、文字データ型の列に対してのみ有効です。

seed= マスキング・アルゴリズムの動作を変更するために使用する値。リテラル・ストリング、環境変数に対する参照、または RANDOM を指定します。

"lit" リテラル・ストリングを指定するには、文字列を二重引用符で囲みます。

var **(variable)**
環境変数を括弧で囲んで指定します。変数の名前と値には二重引用符を含めることはできません。

RANDOM
現在のシステム日時からランダム・シード値を生成します。

length={n | max }
ソース値とは長さが異なる宛先値を生成します。列の幅いっぱいの宛先値を生成するには、*length=max* を使用します。ソース値よりも短い長さを指定すると、宛先に書き込まれたときにソース値の切り捨てが行われます。*n* に対して選択する値は、宛先列に対して定義された長さを超えることはできません。*length=* パラメーターは、*hash=* とともに指定した場合のみ有効です。

preserve=
宛先で置換されないようにする 1 つ以上のソース値をリストします。

null ソース列に NULL 値がある場合は、宛先で値を置換しません。

spaces ソース列にスペース値がある場合は、宛先で値を置換しません。CHAR 列専用です。

zero_len
ソース列に長さがゼロの VARCHAR 値がある場合は、宛先で値を置換しません。

TRIM=(char1 [charn,...])
指定されたソース列の文字がマスクされず、宛先に書き込まれません。例えば *TRIM=(x,y,z,1,2,3)* と指定すると、x、y、z、1、2、3 のいずれかの文字がソース列の任意の場所に現れた場合にそれらがマスクされず、宛先に書き込まれません。

[\u] マスキングの前に文字を大文字に変換するには、このパラメーターを使用します。文字が

大文字表記でない場合は変更されません。例えば *TRIM=(x,y \u)* を指定すると、ソース列の任意の場所に現れる文字 x および y がマスクされず、ソース列のその他の文字がマスクされる前に大文字に変換されます。

[\r] 末尾のスペースを削除するにはこのオペランドを使用します。例えば *TRIM=(x,y \u \r)* を指定すると、ソース列の任意の場所に現れる文字 x および y がマスクされず、マスクされる前にソース列のその他の文字が大文字に変換され、末尾のスペースが削除されます。

num このパラメーターを使用すると、文字データ型列の整数が、数値データ型列の整数と同じように変換されます。num パラメーターは、文字データ型の列内の数値に対してのみ有効です。この方法で使用した場合には、異なるデータ型での外部キーの整合性が維持されます。このパラメーターを使用する場合には、*copy=* も *length=* も指定しないでください。

例:

- ソース値 **CDE-7834** に対して TRANS COL UNIQUE を指定すると、宛先値 **ZWQ-4598** が生成されます。
- ソース値 **CDE-7834-2008** に対して TRANS COL UNIQUE SEED=RANDOM を指定すると、宛先値 **SWX-3162-8451** が生成されます。
- ソース値 **Smith, John** に対して TRANS COL UNIQUE を指定すると、宛先値 **Fnxwq, Lrzp** が生成されます。
- ソース値 **SMITH JOHN** に対して TRANS COL UNIQUE を指定すると、宛先値 **FNXWQ LRZP** が生成されます。
- ソース値 **CDE-7834-2008** に対して TRANS COL UNIQUE (COPY=((1,3)(10,4))) を指定すると、宛先値 **CDE-4032-2008** が生成されます。
- ソース値 **CDE-7834** に対して TRANS COL HASH (LENGTH=13) を指定すると、宛先値 **ZWQ-4598RN7A0** が生成されます。
- ソース値 **XYZ 477 6835** に対して TRANS COL UNIQUE PRESERVE=(spaces) を指定すると、宛先値 **LMN 623 0972** が生成されます。
- ソース値 **InfoSphere** に対して TRANS COL HASH TRIM=(e \u) を指定すると、宛先値 **RBIWACRL** が生成されます。

TRANS NID

TRANS NID 関数を使用して、国民 ID 番号をマスクします。

国民 ID 番号は、ソース値の一部を保持する反復可能な方式によって、またはソース値の一部を保持しないランダムな方式によってマスクすることができます。

出力値で使用される区切り文字のタイプ (ダッシュ、ピリオド、スペース、または区切り文字なし) を指定することもできます。

TRANS 関数は、以下の構文を使用します。

```
[TRANS] NID ('SWI=country_code, [FMT=(output_format)], [MTD={mask|random}],
        [SRC=column_name], [VAL={Y|N}], [PRE=INV]')
```

SWITCH **または** *SWI*

マスクする国民 ID のタイプを示す 2 文字の値。このパラメーターは必須です。 許可されているスイッチの値は 1 つだけです。

以下の 2 文字の値が有効です。

カナダ: Canadian Social Insurance Number (SIN)
CA

フランス: French National Institute for Statistics and Economic Studies Number (INSEE)
FR

イタリア: Italian Fiscal Code Number (CF)
IT

スペイン: Spanish Fiscal Identification Number (NIF)
ES

英国: United Kingdom National Insurance Number (NINO)
UK

米国: United States Social Security Number (SSN)
US

FMT **または** *FORMAT*

出力形式、およびマスクするソース値の部分を決定します。このパラメーターの構文は、マスクされている国民 ID によって決定されます。

- Canadian Social Insurance Number (SIN)
- French National Institute for Statistics and Economic Studies Number (INSEE)
- Italian Fiscal Code Number (CF)
- Spanish Fiscal Identification Number (NIF)/Foreign Identification Number (NIE)
- United Kingdom National Insurance Number (NINO)
- United States Social Security Number (SSN)

このパラメーターが省略され *MTD=mask* である場合は、ソース形式が使用されてデフォルトのフィールドがマスクされます。

このパラメーターが省略され *MTD=random* である場合は、出力値に区切り文字が含まれません。

宛先列が小さいために区切り文字を使用する出力形式を含めることができない場合、区切り文字は含まれません。

MTD **または** *METHOD*

マスキング方式 (反復可能またはランダム) を決定します。このパラメーターを省略すると、反復可能なマスキングが実行されます (MTD=mask)。

mask 反復可能な方法でソース値がマスクされます。出力値はソース値に基づいています。これがデフォルトです。

random
出力値はランダム・マスキング・アルゴリズムによって生成されます。出力値はソース値に基づくものではありません。

注:

- *MTD=random* は、パラメーター *VAL=Y* および *PRE=INV* との互換性はありません。
- *MTD=random* および *FMT* パラメーターは、ソース値の一部を出力値にコピーし、出力値にソース値を含めないことを指定します。ただし、*FMT* パラメーターで指定されている区切り文字は出力値に含められます。

SRC **または** *SOURCE*
ソース値が含まれる列の名前。このパラメーターは、ソース列と宛先列の名前が一致しない場合のみ使用します。

VAL **または** *VALIDATE*
ソース値に対して各国特有の妥当性検査が実行されているかどうかを判定します。このパラメーターを省略すると、妥当性検査が実行されません (*VAL=N*)。

Y ソース値を検証します。

注: このオプションは、*MTD=random* の場合は使用できません。

N ソース値を検証しません。これがデフォルトです。

PRE **または** *PRESERVE*
宛先列に無効なソース値がコピーされていないかどうかを判定します。このパラメーターを省略すると、無効なソース値が宛先列にコピーされず、それらの値が含まれる行がスキップされます。このパラメーターには、INV または INVALID のいずれかのオプションだけが含まれます。

INV **または** *INVALID*
無効なソース値を宛先列にコピーします。

注: このオプションは、*MTD=random* の場合は使用できません。

実例

以下の構文では、反復可能な方式によって United States Social Security Numbers (SSN) がマスクされます。この関数によって、ソース SSN の最初の 3 桁がコピーされ、出力値にダッシュ区切り文字が含められます。出力値に対して妥当性検査が実行されます。

```
NID('SWITCH=US, FMT=(US=3C-2X-4X), MTD=MASK, VAL=Y')
```

以下の構文では、ランダム方式によって French National Institute for Statistics and Economic Studies numbers (INSEE) がマスクされます。この関数ではデフォルトの出力形式が使用され、区切り文字は含まれません。

```
NID('SWITCH=FR, MTD=RANDOM')
```

以下の構文によって、Spanish Fiscal Identification Numbers (NIF) が反復可能な方式によってマスクされます。この関数では無効なソース値が維持され、デフォルトの出力形式が使用されます。

```
NID('SWITCH=ES, MTD=MASK, PRE=INV')
```

カナダ社会保険番号のマスキング

TRANS NID 関数を使用して、カナダ社会保険番号 (SIN) のマスキングを行うことができます。

SIN は、1 桁の地域コード番号と 8 桁のシリアル番号で構成された 9 桁の番号です。最初の 3 桁はヘッダーと呼ばれます。シリアル番号の最後の 1 桁はチェック・ディジットです。

TRANS NID 関数によって、マスクされた先行する 8 桁の出力値に基づいて計算されるチェック・ディジットを持つ、マスクされた SIN が生成されます。

出力形式 *(FMT=)*

SIN では以下の出力形式を使用できます。

C は、コピーされる値を示します。*X* は、マスクされる値を示します。たとえば、*3C4X* は、最初の 3 文字がコピーされて次の 4 文字がマスクされることを示します。

マスクされるフィールド	区切り文字を持たない形式	ダッシュ区切り文字を持つ形式	スペース区切り文字を持つ形式	ピリオド区切り文字を持つ形式
ヘッダー桁を持たないシリアル番号 (*MTD=mask* デフォルト)	CA=3C6X	CA=3C-3X-3X	CA=3C 3X 3X	CA=3C.3X.3X
シリアル番号とヘッダー桁	CA=9X	CA=3X-3X-3X	CA=3X 3X 3X	CA=3X.3X.3X

データ型

以下のデータ型は、ソース列および宛先列についてのみ許可されています。各データ型に対する制限事項が記されています。

文字 (CHAR) 型と各国語文字 (NCHAR) 型
列の長さは 9 文字以上でなければなりません。

10 進数 (DECIMAL) 型
列の精度は 9 から 20 で、位取りは 0 にする必要があります。

整数 (INTEGER) 型
制限はありません。

可変長文字 (VARCHAR) 型と各国語可変長文字 (NVARCHAR) 型
列の長さは 9 文字以上でなければなりません。

妥当性検査 *(VAL=Y)*

(VAL=Y) パラメーターを使用している場合は、以下のいずれかに該当するとソース行がスキップされます。

- 1 桁目が 8 である。
- 連続する 3 つのゼロが、1-3、4-6、または 7-9 の位置にある。
- チェック・ディジットが無効である。

特殊処理

処理中には、以下のチェックが行われます。

- ソース値が 0 (INTEGER または DECIMAL データ型)、スペース、あるいは長さゼロの VARCHAR または NVARCHAR であるか
  - 宛先列が INTEGER または DECIMAL データ型で、出力値がゼロ
  - 宛先列が CHAR または NCHAR データ型で、出力値がスペース
  - 宛先列が VARCHAR または NVARCHAR データ型で、宛先の長さが 0
- 宛先列の長さが 11 文字以上であり、出力値に区切り文字が指定されている場合には、宛先列が CHAR、NCHAR、VARCHAR、または NVARCHAR データ型であれば区切り文字が含まれる
- ソース値が NULL である場合に出力値が NULL になる

スキップされる行

以下の場合には、ソース行がスキップされ、宛先表に書き込まれません。

NULL 値
ソース値が NULL であるが、宛先列で NULL 値が許可されていない

無効な SIN

- ソース列が CHAR、NCHAR、VARCHAR、または NVARCHAR であり、ソース値が 9 文字未満 (区切り文字を含まない)
- ソース列が CHAR、NCHAR、VARCHAR、または NVARCHAR であるが、ソース値が 9 文字を超えている (区切り文字を含まない)
- ソース値に非数値が含まれている

フランスの National Institute for Statistics and Economic Studies Number のマスク

TRANS NID 関数を使用して、French National Institute for Statistics and Economic Studies numbers (INSEE) のマスキングを行うことができます。

INSEE 番号は、*SYYMMDDCCCOOOKK* という形式の 15 桁の番号です。

S 性別および市民権情報

YY 生年の下 2 桁

MM 誕生月

DD 所属部門

CCC 出生コミューン

OOO オーダー番号

KK Ctrl キーまたはチェック・ディジット

TRANS NID 関数では、以下のルールに従って、マスキングされた INSEE が生成されます。

- 部門フィールドがマスキングされている場合は、互換性のある値によってコミューン・フィールドもマスキングされます。
- オーダー・フィールドは常にマスキングされます。
- チェック・ディジット・フィールドは、先行するマスキングされた 13 桁の出力に基づいて計算されます。

出力形式 *(FMT=)*

INSEE では以下の出力形式を使用できます。

すべての形式で、オーダーとチェック・ディジットがマスキングされます。部門フィールドがマスキングされている場合は、互換性のある値によってコミューン・フィールドもマスキングされます。

C は、コピーされる値を示します。*X* は、マスクされる値を示します。たとえば、*3C4X* は、最初の 3 文字がコピーされて次の 4 文字がマスクされることを示します。

(オーダーとチェック・ディジットに加えて) マスキングされるフィールド	区切り文字を持たない形式	ダッシュ区切り文字を持つ形式	スペース区切り文字を持つ形式
性別、年、月、コミューン (*MTD=mask* デフォルト)	FR=5X2C8X	FR=5X2C6X-2X	FR=5X2C6X 2X
性別	FR=1X9C5X	FR=1X9C3X-2X	FR=1X9C3X 2X
性別、年	FR=3X7C5X	FR=3X7C3X-2X	FR=3X7C3X 2X
性別、月	FR=1X2C2X5C5X	FR=1X2C2X5C3X-2X	FR=1X2C2X5C3X 2X
性別、コミューン	FR=1X6C8X	FR=1X6C6X-2X	FR=1X6C6X 2X
性別、部門	FR=1X4C8X	FR=1X4C6X-2X	FR=1X4C6X 2X
性別、年、月	FR=5X5C5X	FR=5X5C3X-2X	FR=5X5C3X 2X
性別、年、コミューン	FR=3X4C8X	FR=3X4C6X-2X	FR=3X4C6X 2X
性別、年、部門、コミューン	FR=3X2C10X	FR=3X2C8X-2X	FR=3X2C8X 2X
性別、月、コミューン	FR=1X2C2X2C8X	FR=1X2C2X2C6X-2X	FR=1X2C2X2C6X 2X
性別、月、部門、コミューン	FR=1X2C12X	FR=1X2C10X-2X	FR=1X2C10X 2X
性別、年、月、部門、コミューン	FR=15X	FR=13X-2X 13X	FR=13X 2X
年	FR=1C2X7C5X	FR=1C2X7C3X-2X	FR=1C2X7C3X 2X
年、月	FR=1C4X5C5X	FR=1C4X5C3X-2X	FR=1C4X5C3X 2X
年、コミューン	FR=1C2X4C8X	FR=1C2X4C6X-2X	FR=1C2X4C6X 2X
年、部門	FR=1C2X2C10X	FR=1C2X2C8X-2X	FR=1C2X2C8X 2X
年、月、コミューン	FR=1C4X2C8X	FR=1C4X2C6X-2X	FR=1C4X2C6X 2X
年、月、部門	FR=1C14X	FR=1C12X-2X	FR=1C12X 2X
月	FR=3C2X5C5X	FR=3C2X5C3X-2X	FR=3C2X5C3X 2X
月、コミューン	FR=3C2X2C8X	FR=3C2X2C6X-2X	FR=3C2X2C6X 2X
月、部門	FR=3C12X	FR=3C10X-2X	FR=3C10X 2X
コミューン	FR=7C8X	FR=7C6X-2X	FR=7C6X 2X
部門	FR=5C10X	FR=5C8X-2X	FR=5C8X 2X

データ型

以下のデータ型は、ソース列および宛先列についてのみ許可されています。各データ型に対する制限事項が記されています。

文字 (CHAR) 型と各国語文字 (NCHAR) 型
列の長さは 15 文字以上でなければなりません。

可変長文字 (VARCHAR) 型と各国語可変長文字 (NVARCHAR) 型
列の長さは 15 文字以上でなければなりません。

妥当性検査 *(VAL=Y)*

(VAL=Y) パラメーターを使用している場合は、以下のいずれかに該当するとソース行がスキップされます。

- ソース・コミューン・フィールドの値が無効である
- ソース・チェック・ディジット・フィールドが無効である

特殊処理

処理中には、以下のチェックが行われます。

- ソース値がスペース、あるいは長さゼロの VARCHAR または NVARCHAR であるか
  - 宛先列が CHAR または NCHAR データ型で、出力値がスペース
  - 宛先列が VARCHAR または NVARCHAR データ型で、宛先の長さが 0
- ソース値が NULL である場合に出力値が NULL になる
- 宛先列の長さが 16 文字以上であり、出力値に区切り文字が指定されている場合に、区切り文字が含まれる

スキップされる行

ソース値に以下のエラーが含まれている場合には、ソース行がスキップされ、宛先表に書き込まれません。

NULL 値
ソース値が NULL であるが、宛先列で NULL 値が許可されていない

無効な INSEE

- 入力された INSEE 値のサイズが有効な値よりも大きいか小さい
- 入力された INSEE 値に無効な区切り文字または方向が正しくない区切り文字が含まれている
- 性別フィールドの値が 1、2、7、または 8 ではない

イタリアの Fiscal Code Number のマスキング

TRANS NID 関数を使用して、イタリアの Fiscal Codes (CF) のマスキングを行うことができます。

CF は、*FFF-NNN-YYMDD-RRRRC* の形式を持つ 16 文字の英数字です。

FFF エンコードされた姓

NNN エンコードされた名前

YY 誕生年

M 誕生月

DD 誕生日

RRRR 地域コード

C 制御文字。

TRANS NID 関数では、以下のルールに従って、マスキングされた CF が生成されます。

- 名前または姓のフィールド内の子音は子音としてマスクされ、母音は母音としてマスクされます。母音に X が続く場合は、出力値にコピーする必要があります。
- 制御文字フィールドは、マスクされた先行する 15 桁の出力値に基づいて計算されます。

出力形式 *(FMT=)*

CF では以下の出力形式を使用できます。

C は、コピーされる値を示します。*X* は、マスクされる値を示します。たとえば、*3C4X* は、最初の 3 文字がコピーされて次の 4 文字がマスクされることを示します。

マスクされるフィールド	区切り文字を持たない形式	ダッシュ区切り文字を持つ形式	スペース区切り文字を持つ形式
生年月日、地域 (*MTD=mask* デフォルト)	IT=6C10X	IT=3C-3C-5X-5X	IT=3C 3C 5X 5X
姓、名、地域	IT=6X5C5X	IT=3X-3X-5C-5X	IT=3X 3X 5C 5X
姓、名、生年月日	IT=11X4C1X	IT=3X-3X-5X-4C1X	IT=3X 3X 5X 4C1X
姓、名	IT=6X9C1X	IT=3X-3X-5C-4C1X	IT=3X 3X 5C 4C1X
生年月日	IT=6C5X4C1X	IT=3C-3C-5X-4C1X	IT=3C 3C 5X 4C1X
地域	IT=11C5X	IT=3C-3C-5C-5X	IT=3C 3C 5C 5X
姓、名、生年月日、地域	IT=16X	IT=3X-3X-5X-5X	IT=3X 3X 5X 5X

データ型

以下のデータ型は、ソース列および宛先列についてのみ許可されています。各データ型に対する制限事項が記されています。

文字 (CHAR) 型と各国語文字 (NCHAR) 型
列の長さは 16 文字以上でなければなりません。

可変長文字 (VARCHAR) 型と各国語可変長文字 (NVARCHAR) 型
列の長さは 16 文字以上でなければなりません。

妥当性検査 *(VAL=Y)*

VAL=Y パラメーターを使用している場合は、無効な制御文字があるとソース行がスキップされます。

特殊処理

処理中には、以下のチェックが行われます。

- ソース値がスペース、あるいは長さゼロの VARCHAR または NVARCHAR であるか
  - 宛先列が CHAR または NCHAR データ型で、出力値がスペース
  - 宛先列が VARCHAR または NVARCHAR データ型で、宛先の長さが 0
- ソース値が NULL である場合に出力値が NULL になる
- 宛先列の長さが 19 文字以上であり、出力値に区切り文字が指定されている場合に、区切り文字が含まれる

スキップされる行

以下の場合には、ソース行がスキップされ、宛先表に書き込まれません。

NULL 値
ソース値が NULL であるが、宛先列で NULL 値が許可されていない

CF が無効
ソース列に無効なソース値があるか、ソース値が大きすぎる

ソース値が 16 文字未満である

スペインの Fiscal Identification Number と Foreign Identification Number のマスキング

TRANS NID 関数を使用して、スペインの Fiscal Identification Numbers (NIF) と Foreign Identification Numbers (NIE) のマスキングを行うことができます。

NIF は、*NNNNNNN-A* という形式の 8 文字の値です。最初の 7 文字はシリアル番号で、最後の文字は英字の接尾部です。接尾部はチェック・ディジットです。

海外のスペイン国民は Foreign Identification Number (NIE) を使用します。これは NIF と同じ形式の 9 文字の値ですが、先頭に X が付きます。NIE の形式は *X-NNNNNNN-A* です。

TRANS NID 関数によって、マスクされた先行する 7 桁の出力値に基づいて計算されるチェック・ディジットを持つ、マスクされた NIF または NIE が生成されます。

出力形式 *(FMT=)*

NIF および NIE では以下の出力形式を使用できます。

いずれの形式についても、すべての文字がマスクされます。NIF および NIE 番号では同じフォーマット・オプションが使用されます。NIE ソース値については、出力値に必ず X 接頭部が付けられます。

マスクされるフィールド	区切り文字を持たない形式	ダッシュ区切り文字を持つ形式	スペース区切り文字を持つ形式
シリアル、接尾部 (*MTD=mask* デフォルト)	ES=8X	ES=7X-1X	ES=7X 1X

データ型

以下のデータ型は、ソース列および宛先列についてのみ許可されています。各データ型に対する制限事項が記されています。

文字 (CHAR) 型と各国語文字 (NCHAR) 型
列の長さは 8 文字以上でなければなりません。

可変長文字 (VARCHAR) 型と各国語可変長文字 (NVARCHAR) 型
列の長さは 8 文字以上でなければなりません。

妥当性検査 *(VAL=Y)*

VAL=Y パラメーターを使用している場合は、無効な接尾部があるとソース行がスキップされます。

特殊処理

処理中には、以下のチェックが行われます。

- ソース値がスペース、あるいは長さゼロの VARCHAR または NVARCHAR であるか
  - 宛先列が CHAR または NCHAR データ型で、出力値がスペース
  - 宛先列が VARCHAR または NVARCHAR データ型で、宛先の長さが 0
- ソース値が NULL である場合に出力値が NULL になる
- 宛先列の長さが 11 文字以上であり、出力値に区切り文字が指定されている場合に、区切り文字が含まれる

スキップされる行

以下の場合には、ソース行がスキップされ、宛先表に書き込まれません。

長さが無効

入力値に区切り文字がなく、8 文字未満 (NIF) または 9 文字未満 (NIE) である

入力値に区切り文字が含まれ、9 文字未満 (NIF) または 11 文字未満 (NIE) である

無効なパターン

ソース値の長さが有効であるが、NIF または NIE のパターンに一致しない

区切り文字

NIE ソース値の位置 2 および 10 に異なる区切り文字がある

ソース値に無効な区切り文字が含まれている

NULL 値

ソース値が NULL であるが、宛先列で NULL 値が許可されていない

英国の National Insurance Number のマスキング

TRANS NID 関数を使用して、英国の National Insurance Numbers (NINO) のマスキングを行うことができます。

NINO は、2 文字 (接頭部)、6 桁 (番号)、および 1 つのオプション文字 (接尾部) の 3 つの部分で構成されています。

出力形式 *(FMT=)*

NINO では以下の出力形式を使用できます。

NINO は、区切り文字なしで、あるいは、区切り文字によって 3 つまたは 5 つの部分からなるフォーマットでマスキングされます。

C は、コピーされる値を示します。*X* は、マスクされる値を示します。たとえば、*3C4X* は、最初の 3 文字がコピーされて次の 4 文字がマスクされることを示します。

区切り文字なしで NINO を作成するには、以下のパラメーターを使用します。

マスクされるフィールド	区切り文字を持たない形式
接頭部、番号	UK=8X1C
番号 (*MTD=mask* デフォルト)	UK=2C6X1C
接頭部、番号、接尾部 (*MTD=random* のみ)	UK=9X

3 つまたは 5 つの部分を持つ形式で NINO を作成するには、以下のパラメーターを使用します。

マスクされるフィールド	ダッシュ区切り文字を持つ形式	スペース区切り文字を持つ形式	ピリオド区切り文字を持つ形式
接頭部、番号 (3 つの部分)	UK=2X-6X-1C	UK=2X 6X 1C	UK=2X.6X.1C
接頭部、番号 (5 つの部分)	UK=2X-2X-2X-2X-1C	UK=2X 2X 2X 2X 1C	UK=2X.2X.2X.2X.1C
番号 (3 つの部分)	UK=2C-6X-1C	UK=2C 6X 1C	UK=2C.6X.1C
番号 (5 つの部分)	UK=2C-2X-2X-2X-1C	UK=2C 2X 2X 2X 1C	UK=2C.2X.2X.2X.1C

マスクされるフィールド	ダッシュ区切り文字を持つ形式	スペース区切り文字を持つ形式	ピリオド区切り文字を持つ形式
接頭部、番号、接尾部 (3 つの部分) (*MTD=random* のみ)	UK=2X-6X-1X	UK=2X 6X 1X	UK=2X.6X.1X
接頭部、番号、接尾部 (5 つの部分) (*MTD=random* のみ)	UK=2X-2X-2X-2X-1X	UK=2X 2X 2X 2X 1X	UK=2X.2X.2X.2X.1X

データ型

以下のデータ型は、ソース列および宛先列についてのみ許可されています。各データ型に対する制限事項が記されています。

文字 (CHAR) 型と各国語文字 (NCHAR) 型
列の長さは 9 文字以上でなければなりません。

可変長文字 (VARCHAR) 型と各国語可変長文字 (NVARCHAR) 型
列の長さは 9 文字以上でなければなりません。

妥当性検査 *(VAL=Y)*

VAL=Y パラメーターは NINO マスキングに対して無効であり、無視されます。

特殊処理

処理中には、以下のチェックが行われます。

- ソース値がスペース、あるいは長さゼロの VARCHAR または NVARCHAR であるか
  - 宛先列が CHAR または NCHAR データ型で、出力値がスペース
  - 宛先列が VARCHAR または NVARCHAR データ型で、宛先の長さが 0
- ソース値が NULL である場合に出力値が NULL になる
- 宛先列に区切り文字を含めることができない場合に、出力値に区切り文字が含まれない

スキップされる行

以下の場合には、ソース行がスキップされ、宛先表に書き込まれません。

NULL 値
ソース値が NULL であるが、宛先列で NULL 値が許可されていない

NINO が無効

- ソース値のサイズが無効な NINO のサイズより小さいか大きい
- ソース値に含まれている区切り文字の位置が誤っている
- ソース値に無効な区切り文字が含まれている
- ソース値に無効な接頭部が含まれている
- ソース値に A、B、C、または D 以外の接尾部が含まれている
- ソース値に 000001 ～ 999999 の範囲にない番号フィールドが含まれている

米国社会保障番号のマスキング

TRANS NID 関数を使用して、米国社会保障番号 (SSN) のマスキングを行うことができます。

SSN は、*AAAGGSSSS* 形式の 3 つのサブフィールドで構成されています。

AAA 地域番号。通常この地域は、SSN が発行された州によって決定されます。

GG グループ番号。グループ番号は、地域番号に基づいて割り当てられます。

SSSS シリアル番号。

TRANS NID 関数では、以下のルールに従って、マスキングされた SSN が生成されます。

- 地域番号に適合するグループ番号が生成されます。グループ番号は、地域の社会保障局によって使用される最新のグループになります。
- シリアル番号は 0001 で始まり、地域番号に対して追加の SSN が生成されるごとに 1 ずつ増分します。シリアル番号が 9999 を超えると、シリアル番号は 0001 にリセットされ、その地域番号に対して発行された最新の番号の前のグループ番号が使用されます。
- *MTD=mask* の場合、出力値にはソース地域番号と同じ州に対応する地域番号が含まれます。

出力形式 *(FMT=)*

SSN では以下の出力形式を使用できます。

C は、コピーされる値を示します。*X* は、マスクされる値を示します。たとえば、*3C4X* は、最初の 3 文字がコピーされて次の 4 文字がマスクされることを示します。

マスクされるフィールド	区切り文字を持たない形式	ダッシュ区切り文字を持つ形式	スペース区切り文字を持つ形式	ピリオド区切り文字を持つ形式
グループ、シリアル番号 (*MTD=mask* デフォルト)	US=3C6X	US=3C-2X-4X	US=3C 2X 4X	US=3C.2X.4X
地域、グループ、シリアル番号	US=9X	US=3X-2X-4X	US=3X 2X 4X	US=3X.2X.4X

データ型

以下のデータ型は、ソース列および宛先列についてのみ許可されています。各データ型に対する制限事項が記されています。

文字 (CHAR) 型と各国語文字 (NCHAR) 型
列の長さは 9 文字以上でなければなりません。

DECIMAL
列の精度は 9 から 20 で、位取りは 0 にする必要があります。

INTEGER
制限なし。

可変長文字 (VARCHAR) 型と各国語可変長文字 (NVARCHAR) 型
列の長さは 9 文字以上でなければなりません。

妥当性検査 *(VAL=Y)*

(VAL=Y) パラメーターを使用している場合は、以下のいずれかに該当するとソース行がスキップされます。

- ソースの地域番号が最大値を超えている場合。
- ソースの地域番号が社会保障局で使用されていない場合。
- ソースのグループ番号が、ソースの地域番号で使用されていない。

特殊処理

処理中には、以下のチェックが行われます。

- ソース値が 0 (INTEGER または DECIMAL データ型)、スペース、あるいは長さゼロの VARCHAR または NVARCHAR であるか
  - 宛先列が INTEGER または DECIMAL データ型で、出力値がゼロ
  - 宛先列が CHAR または NCHAR データ型で、出力値がスペース
  - 宛先列が VARCHAR または NVARCHAR データ型で、宛先の長さが 0
- 宛先列の長さが 11 文字以上であり、出力値に区切り文字が指定されている場合に、区切り文字が含まれる
- ソース値が NULL である場合に出力値が NULL になる

スキップされる行

以下の場合には、ソース行がスキップされ、宛先表に書き込まれません。

NULL 値
ソース値が NULL であるが、宛先列で NULL 値が許可されていない

SSN が無効

- ソース列が CHAR、NCHAR、VARCHAR、または NVARCHAR であるが、ソース値が 9 文字を超えているか 9 文字未満である (区切り文字を含まない)
- ソース値に非数値が含まれている
- ソース値が 0 であるか、078-05-1120 や 457-55-5462 などの予約値である

Substring 関数

Substring 関数は、指定された列の内容のサブストリングを返します。

```
SUBSTR(columnname, start, [length])
```

columnname
文字列またはバイナリー列の名前。

start ストリングの先頭文字の位置。

length 使用する文字数。

- ロケールでコンマが小数点として使用されている場合、数値パラメーターを区切る各コンマの後にスペースを入れる必要があります (例えば、*start* と *length* の間のコンマの後)。
- *start* および *length* は、1 以上の整数です。
- *start* と *length* を足したデータ長が、合計データ長に 1 を加えた値を超えることはできません。

- *column-name* および *start* の値は必須です。1 つの整数しか指定しない場合、*start* の値として使用されます。サブストリングは *start* で始まり、列の値の残りが含まれます。

例

PHONE_NUMBER 列が CHAR(10) として定義されている場合、Substring 関数を使用して市外局番をマップできます。宛先列の電話番号 (市外局番) の最初の 3 桁のサブストリングを取得するには、以下のように指定します。

SUBSTR(PHONE_NUMBER, 1, 3)

Random 関数

Random 関数は、最小値および最大値で示される範囲内でランダムに選択される数値を返します。

RAND(*low, high*)

low ランダムの取りうる最小値。

high ランダムの取りうる最大値。

- Random 関数は文字データまたは数値データとともに使用します。
- ロケールでコンマが小数点として使用されている場合、コンマの後にスペースを入れる必要があります。
- *low* および *high* は、**-2,147,483,648** から **2,147,483,647** の範囲内の整数です。
- *low* および *high* は、宛先列のデータ型および長さによってさらに制限されます。
- *low* は、*high* より小さくする必要があります。
- Random 関数を連結式で使用する場合、可変長ストリングが返されます。

例

Random 関数を使用すると、テスト・データベースの売り上げデータをマスクまたは変更できます。YTD_SALES 列が DECIMAL(7,2) と定義されているとします。小数点の左側の桁の最大数は 5 です。この列が取りうる範囲は、-99999 から 99999 です。1000 (最小値) から 89999 (最大値) の範囲内でテスト・データを作成するには、以下のように指定します。

RAND(1000, 89999)

この例では、指定した範囲である 1000.00 から 89999.99 までのランダムな売り上げの値が関数によって返されます。

Sequential 関数

Sequential 関数は、順次増分される数値を返します。構文:

SEQ(*start, step*)

start 開始値。

step 増分する値。

- Sequential 関数は文字データまたは数値データとともに使用します。
- ロケールでコンマが小数点として使用されている場合、コンマの後にスペースを入れる必要があります。
- *start* および *step* は、**-2,147,483,648** から **2,147,483,647** の範囲内の整数です。

- *start* および *step* は、宛先列のデータ型および長さによってさらに制限されます。
- 計算された値が宛先列の長さを超えた場合、関数は自動的に *start* の値にリセットされます。
- Sequential 関数を連結式で使用する場合、可変長ストリングが返されます。

例 1

Sequential 関数を使用すると、テスト・データベースの顧客データを変更できます。CUST_ID 列が CHAR(5) と定義されているとします。1 から開始して 50 ずつ増分するには、以下のように指定します。

```
SEQ(1, 50)
```

この例では、関数は「00001」から開始して 50 ずつ増分し、「00051」や「00101」などを生成する CUST_ID の値を返します。結果が「99951」を超えると、関数は *start* の値の 1 にリセットされます。

例 2

列マップで Sequential 関数を使用すると、テスト・データベースの売り上げデータをマスクできます。YTD_SALES 列が DECIMAL(7,2) と定義されているとします。 1000 から開始して 100 ずつ増分するには、以下のように指定します。

```
SEQ(1000, 100)
```

この例では、関数は 1000 から開始して 100 ずつ増分し、1100 や 1200 などを生成する YTD_SALES の値を返します。結果が 99999 を超えると、関数は *start* の値の 1000 にリセットします。

例 3

SALESMAN_ID 列が CHAR(6) と定義されているとします。「NJ」で始まり、50 から開始して 10 ずつ増分する数値が後に続く値を挿入するには、関数を連結式で以下のように使用します。

```
'NJ'||SEQ(50, 10)
```

この例では、関数は「NJ50」から開始して 10 ずつ増分し、「NJ60」や「NJ70」などを生成する SALESMAN_ID の値を返します。結果が「NJ9990」を超えると、関数は *start* の値にリセットします。

Identity 関数または Serial 関数

Identity 関数および Serial 関数は、宛先列のために連続値 (整数) を提供するよう DBMS に指示します。これらの関数の構文規則は、以下のとおりです。

```
IDENTITY( )
```

```
SERIAL ( )
```

- Identity 関数は、DB2、Sybase ASE、および SQL Server データベースの Identity 列のために使用します。
- Serial 関数は、Informix データベースの Serial 列のために使用します。
- どちらの関数も挿入処理 (更新/挿入) およびロード処理において有効ですが、変換処理では無効です。
- 挿入プロセス (更新/挿入) で行が更新される場合、Identity 関数または Serial 関数のターゲットとなる宛先列は、元の値を保持します。さらに、宛先列が主キーに含まれる場合、行の更新時に列の値は変更されません。

- 挿入処理のために Identity 関数または Serial 関数を Propagate 関数と共に使用できます。ただし、ロード・プロセスでは Identity 列や Serial 列を伝搬できません。

Oracle シーケンス関数

Oracle シーケンス関数は、Oracle シーケンスを使用して宛先列に値を割り当てます。

schema.seqname.NEXTVAL [INCL_UPD]

schema Oracle シーケンスの名前の修飾子。

seqname
順次値を割り当てる Oracle シーケンスの名前。

NEXTVAL
次の Oracle の値を宛先列に挿入するキーワード。

INCL_UPD
挿入処理中に行が更新された場合に、列に割り当てられているシーケンス値を更新するオプションのキーワード。指定されていない (デフォルト) 場合、列の値は行の更新時には変更されません。

- Oracle シーケンス関数を使用すると、行に対して Oracle データベースに挿入されるユニークな連続値を割り当てることができます。
- Oracle シーケンス関数は、挿入またはロード処理のための列マップで使用する場合には有効ですが、変換処理では無効です。
- 行が挿入 (更新/挿入) 処理で更新され、宛先列が主キーの一部である場合には、宛先表の行が更新されても列の値は変更されません。更新/挿入を実行するときに Oracle シーケンスを使用するには、関数に *INCL_UPD* を含めます。
- ロード処理では、Oracle シーケンス関数を使用して各宛先行に新しい値が割り当てられます。ロードによって DBMS が呼び出され、これらの値が取得されます。Oracle Loader を実行しないよう選択した場合は、これらのシーケンス値は使用されません。

例 1

Oracle シーケンスの名前が **schema.numeven** である場合に、シーケンス値を割り当てて顧客番号を増分するには、以下のように指定します。

```
schema.numeven.NEXTVAL
```

例 2

最初の例を拡張し、既存のシーケンス値を更新するには、以下のように指定します。

```
schema.numeven.NEXTVAL(INCL_UPD)
```

主キーまたは外部キーの値を伝搬する関数

Propagate 関数は、主キーまたは外部キーの列に値を割り当て、その値をすべての関連する表に伝搬します。

構文:

```
PROP( { value [, columnname| ] EXIT exitname |
PROC { LOCAL | identifier.name } } )
```

value 列に割り当てる値。有効な列マップのソース値を指定します (例えば、列名、ストリング・リテラル、式、または関数)。値は列に対して適切である必要があります。

columnname
関数の対象である値が含まれるソース列の名前。結果として生成される値は、マップされた表の宛先列、および参加している関連する表の適切な宛先列に挿入されます。

名前とデータ型の両方で宛先列に一致するソース列がない場合にのみ、列名が必要になります。指定しない場合は、宛先列の名前が使用されます。

exitname
列マップの終了名。

identifier.name
列マップのプロシージャー名。

- ロケールでコンマが小数点として使用されている場合、数値パラメーターを区切る各コンマの後に、スペースを入れる必要があります。
- Propagate 関数は、挿入 (更新または更新/挿入以外)、ロード、または変換処理における列マップで有効です。
- Propagate 関数を使用する場合、少なくとも 1 つの関連する表がプロセスに含まれている必要があります。同じプロセスに対して、Propagate を複数回使用できます。
- Propagate 関数を主キー列または対応する外部キー列のいずれかに対して使用できますが、両方には使用できません。
- 複数の列が関係を定義する場合、それらの列の 1 つ以上に対して Propagate 関数を使用できます。ただし、Optim の拡張関係では、列対列の関係にのみ Propagate 関数を指定できます。
- 挿入処理に対して、Identity 関数または Serial関数を Propagate 関数内で使用できます。ただし、ロード処理では Identity 関数を伝搬できません。
- Propagate 関数で指定されたパラメーターは、実行時まで検証されません。競合がある場合、処理は実行されません。
- 挿入は伝搬サイクルを生成することがあります。ただし、ロードおよび変換処理では伝搬サイクルが生成されない場合があります。サイクルはプロセスが実行時に検証されるときに検出されます。ロード要求または変換要求によって伝搬サイクルが生成される場合、処理は実行されません。
- Optim は、ソース値および対応する宛先列に割り当てられている値を記憶します。このため、ソースが式である場合に宛先列に伝搬できます。評価された式がソース値に一致する場合、Optim は対応する宛先の値を割り当てます。評価された式がソース値に一致しない場合、変換エラーが発生します。

挿入、ロード、または変換の各プロセスを実行する前に、列マップを確認して、Propagate 関数がプロセスで使用されている方法を検証できます。

例 1

乱数を生成して、デフォルト宛先列に割り当て、関連する表の宛先列にその数値を伝搬できます。 10000 から 99999 の間で値を生成して、マップされた宛先列に挿入し、関連する表の宛先列に伝搬するには、以下のように指定します。

```
PROP(RAND(10000, 99999))
```

例 2

ソース列と宛先列の名前が一致しない場合に、例 1 と同じ関数を実行できます。Propagate 関数にソース列の名前 (CUST_NUMBER) を含めるには、以下のように指定します。

```
PROP(RAND(10000, 99999), CUST_NUMBER)
```

例 3

Oracle シーケンスを使用して、宛先列の値を生成し、関連する表の宛先列にその値を伝搬できます。**schema.numeven** という名前の Oracle シーケンスを伝搬するには、以下のように指定します。

```
PROP(schema.numeven.NEXTVAL)
```

連結式

連結では、連結演算子 (CONCAT、||、または +) を使用して、列の値を結合、または列の値を別の値と結合できます。連結式には、文字値またはバイナリー値を含めることができますが、両方を含めることはできません。

文字値 連結された文字値には、文字列、ストリング・リテラル、文字列の値のサブストリング、Sequential 関数、または Random 関数を指定できます。

バイナリー値
連結された文字値には、バイナリー列、16 進数リテラル、バイナリー列のサブストリング、Sequential 関数、または Random 関数を指定できます。

- 連結式には、長さがゼロのストリング・リテラル (' ')、特殊レジスター、または Age 関数を含めることはできません。

例

CUSTOMERS 表では、アドレスを ADDRESS1 と ADDRESS2 の 2 つの列に格納するとします。SHIP_TO 表ではアドレスを ADDRESS という 1 つの列に格納します。連結式を使用して、1 つの表内の 2 つの列からのアドレス情報を別の表の 1 つの列に結合できます。

アドレスを結合するには、以下のいずれかを指定します。

CUSTOMERS 表	**SHIP_TO 表**
ADDRESS1 || ADDRESS2	ADDRESS
ADDRESS1 CONCAT ADDRESS2	ADDRESS
ADDRESS1 + ADDRESS2	ADDRESS

数値表現

数値表現を使用すると、対応するソース列および宛先列のデータ型に互換性がある場合は、常にソース列に値を指定できます。

数値表現は以下の構文で構成されています。

```
operand1 operator operand2
```

各オペランドは、数値列または数値定数である必要があります。演算子は加算 (+)、減算 (-)、除算 (/)、または乗算 (*) を行うかどうかを指定します。

例 1

DECIMAL(5,2) と定義されている列 UNIT_PRICE の値を 10 % ずつ増分するには、以下のように指定します。

```
1.1 * UNIT_PRICE
```

例 2

整数列の ON_HAND_INVENTORY の値を半分に除算するには、以下のように指定します。

```
ON_HAND_INVENTORY / 2.
```

リテラルおよび VALUE 関数

ストリングや 16 進数、NULL などの値、特殊レジスターなどのリテラルを指定するには、リテラルおよび VALUE 関数を使用します。

関数	説明
列名	明示的な列名 (列名は大/小文字を区別しません)。
NULL	NULL。宛先列は NULL 可能である必要があります。
数値定数	数値定数。定数の値は、データ型、精度、および位取りで定義されているとおりに、宛先列に適合する必要があります。
ブール値定数	ブール値定数 (TRUE または FALSE)。
特殊レジスター	特殊レジスターは、以下のとおりです。 • CURRENT DATE • CURRENT_DATE • CURRENT TIME • CURRENT_TIME • CURRENT TIMESTAMP • CURRENT_TIMESTAMP • CURRENT SQLID • CURRENT_SQLID • CURDATE( ) • CURTIME( ) • GETDATE( ) • GETTIME( ) • SYSDATE( ) • NOW( ) • WORKSTATION_ID • USER
ストリング・リテラル	単一引用符で囲まれたストリング・リテラル。宛先列には、文字データが含まれている必要があります。 例: 'CA' または '90210'。
16 進数リテラル	16 進数リテラル。 例: X‘1234567890ABCDEF' または 0X1234567890ABCDEF

関数	説明
日時リテラル	単一引用符で囲まれた日時リテラル。日付と時刻をスペースで区切ります。 10 進数の小数部を使用して日時の形式を設定するには、時刻の後にピリオドを置き、そのあとに小数部を続けます。日付形式は、コンピューターのコントロール・パネルの「**地域オプション**」の設定によって決定されます。**注:** Oracle Timestamp with Time Zone 列の場合、時間帯の接尾部を末尾に指定する必要があります。

Age 関数

Age 関数を使用して、ソース列の値の経時処理を行います。ソース列には、文字、数値、日付、またはタイム・スタンプ・データを含めることができます。 CHAR 列または VARCHAR 列の最大長は 256 バイトです。

Age 関数の形式は、以下のとおりです。

AGE(*parameters*)

- Age 関数を定義して、1 つ以上の経時処理パラメーターを含めます。
- Age 関数の複数のパラメーターを区切るには、コンマまたはスペースを使用します。
- パラメーターは任意の順序で指定できます。

パラメーター	形式	有効値
列名: ソース列の名前が宛先列と異なる場合に、ソース列の名前を指定します。	*SC=column-name* *SRCCOL=column-name*	列名
デフォルト: プロセス要求で指定された日付調整値に基づき、日付を経時処理します。	*DEF*	プロセス要求で指定された日付調整値を使用します。
なし: 値を経時処理しません。	*NONE*	プロセス要求での指定にかかわりなく、値は経時処理されません。
増分: 増分経時処理は、既知の時間単位に基づきます。 Optim は、単一の単位 (例: 20 年) または複数の単位 (例: 2 年、3 カ月、2 日) による日付の経時処理をサポートします。	+ または -] *n*Y [+ または -] *n*M [+ または -] *n*W [+ または -] *n*D (正符号 [+] はオプションです。)	*n*Y -2500 から +1581 まで *n*M -30000 から +30000 まで *n*W -30000 から +30000 まで *n*D -99999 から +99999 まで
特定の年: 適切な形式で、特定の 4 桁の年に基づき日付を経時処理します。	*nnnn*Y	1582 から 3999
複数/ルール: ビジネス・ルールを適用する回数に基づき日付を経時処理します。複数/ルールを使って Age 関数を定義する場合には、RULE パラメーターも含める必要があります。	*nnnnn*R	1 から 30000

セマンティック経時処理

セマンティック経時処理は、休日、週末などに発生する日付を管理するために定義されたルールのセットに基づいています。セマンティック経時処理を使用すると、有効な営業日に日付が発生するよう調整できます。

カレンダー –

ルールが適用される特別な日付を定義するカレンダーの名前。 CALENDAR を使用する場合は、RULE も指定する必要があります。

CA=*calendar-name*

CALENDAR=*calendar-name*

ルール –

特別な日付の調整を定義するルールの名前。 DEF が指定される場合、プロセス要求で指定されているデフォルトのルールが使用されます。

RU=*rule-name*

RULE=*rule-name*

RU=DEF

RULE=DEF

世紀ピボット –

2 桁の年の世紀を決定します。 00 から 99 までの値を入力してください。

PI=*nn*

PIVOT=*nn*

- **AGE(RU=DEF)** を指定した場合、プロセス要求で指定された RULE が使われます。その他の Age 関数パラメーターの値を指定する必要があります。
- RULE を使用し、CALENDAR を指定しない場合、Age 関数はプロセス要求で指定されたデフォルトのカレンダーを使用します。
- Age 関数の中の必要な場所に CALENDAR、RULE、および PIVOT が含まれない場合、プロセス要求で指定されたデフォルト値が適用されます。
- 2 桁の年に関して適切な世紀を指定するには、Age 関数に PIVOT を含める必要があります。
- PIVOT 値を指定した場合、PIVOT 値に等しい (またはそれより大きい) 2 桁の年は、すべて 20 世紀 (19xx) と見なされます。 PIVOT 値より小さい 2 桁の年は、すべて 21 世紀 (20xx) とみなされます。デフォルトの PIVOT は **65** です。

日付形式

ソース日付形式および宛先日付形式は、単一の有効な日付形式を含んでいる必要があり、宛先列の長さ以下である必要があります。形式ストリングを単一引用符で区切る必要があります。

ソース日付形式 –

文字の列および数値の列を経時処理するためにソース列の形式ストリングを適用します。

SF=*'format-string'*

SRCFMT=*'format-string'*

ソース列が文字または数値である場合、SRCFMT またはソース出口ルーチン (SRCEXIT) を使って列の内容を記述する必要があります。これらのパラメーターは互いに排他的です。

宛先日付形式 –

文字の列および数値の列を経時処理するために宛先列の形式ストリングを適用します。

DF='format-string'

DSTFMT=*'format-string'*

宛先列が文字または数値である場合、DSTFMT または宛先出口ルーチン (DSTEXIT) を指定できます。宛先の形式を指定しない場合、日付の経時処理関数はデフォルトで SRCFMT を使用します。 Age 関数の宛先列をバイナリーにすることはできません。

以下の文字ストリングを使用して、日付形式のコンポーネントを指定します。

年	月	日	時刻	秒の小数部
YYYY	MONTH	DDD	HH	FFFFFF
CCYY	MMM	DD	MI	FFFFF
YY	MM	D	SS	FFFF
	M			FFF
				FF
				F

- 形式ストリングで疑問符 (**?**) を指定した場合、Age 関数は文字値をそのままマップします。 (日付形式の中にスラッシュ、ダッシュ、ピリオドなどを含める場合に疑問符を使用してください。)
- 形式ストリングでアスタリスク (*****) を指定した場合、Age 関数はソース列の残りのすべての文字を宛先列にマップします。 (列の値が追加文字に連結された日付である場合に、アスタリスクを使用してください。)

注: Calendar ユーティリティーを使用すると、デフォルトの区切り文字およびデフォルトの出力年を定義できます。これらのデフォルトは、ソース形式および宛先形式で区切り文字または特定の年が必要な場合に適用されます。

例 1

日付列を 2 年、6 カ月、40 週、および 15 日で経時処理した後、ルールを適用するには、Age 関数の形式を以下のように指定します。

```
AGE(+2Y,+6M,+40W,+15D,RU=NEXTPAYDAY)
```

例 2

日付列の年の部分のみを 2020 年に経時処理し、ルールを適用するには、Age 関数の形式を以下のように指定します。

```
AGE(2020Y,RU=NEXTWORKDAY)
```

例 3

複数/ルールを使って日付列を経時処理し、PSAPRULE というカレンダーを使用して NEXTSTRTQTR というルールの 5 回の出現ごとに増分するには、Age 関数の形式を以下のように指定します。

```
AGE(CA=PSAPRULE,RU=NEXTSTRTQTR,5R)
```

例 4

以下のパラメーターによって、文字または数値の列のデータを経時処理するには、

- 指定されたソース列。
- 最初の 2 文字を年の最後の 2 桁として使用し、残りの 3 桁をユリウス暦のカレンダーの日として使用するソース形式。
- ソースが 2 桁の年で形式設定されているために、適切な世紀を決定する世紀ピボット。この例での世紀ピボットは 42 です。 42 以上の 2 桁の年はすべて、20 世紀 (19xx) と見なされます。 42 未満の 2 桁の年はすべて、21 世紀 (20xx) と見なされます。

- 日付を 5 年で経時処理します。

Age 関数の形式は、以下のとおりです。

```
AGE(5Y,SC=ORDER_DATE,SF='YYDDD',PI=42)
```

Currency 関数

Currency 関数を使用して、ソース列の通貨値を 1 つの通貨から別の通貨に変換します。ソース列は数値として定義される必要がありますが、浮動小数点にすることはできません。以下の 2 つの変換方法を使用できます。

直接変換

通貨定義で定義されている値に基づき、変換パラメーターを提供します。 Currency 関数を使用して、列の金額値を変換する (前の値を置き換える) か、または異なるソース列と宛先列を定義することにより、元の値と変換後の値を保持します。ソースおよび宛先の通貨タイプを明示的に定義することも、参照列を特定して通貨タイプを示すこともできます。

最初の計算設定では、ソース通貨から宛先通貨に対する変換レートを使用します。 2 番目の計算変換設定では、宛先通貨からソースに対する変換レートを使用します。

三角変換

列の値をソース通貨からユーロドルに変換した後、ユーロドルを宛先通貨に変換します。両方のレート (1 つはユーロドルとソース通貨との間のレート、もう 1 つはユーロドルと宛先通貨との間のレート) を換算表で提供する必要があります。宣言式は TRIANG または TR です。

Currency 関数の形式は、以下のとおりです。

```
CURRENCY( {ST=code | SS=(column-name,Types Table number)}
 {DT=code | DS=(column-name,Types Table number)}
 [SC=column-name] [TR] [CU=Currency Definition name]
 [TD=transaction-date-column-name] [DF='format']
 [NS=scale] )
```

Currency 関数には、ソース通貨タイプ (ST) またはソース仕様 (SS) と、宛先通貨タイプ (DT) または宛先仕様 (DS) の組み合わせが少なくとも含まれている必要があります。その他のすべてのパラメーターはオプションです。

以下の 2 つの方法のいずれかに従って、ソース通貨タイプおよび宛先通貨タイプを指定できます。

1. ST/DT キーワードを使用すると、3 文字の ISO 4217 通貨コードを使った通貨の明示的な指定が可能です。
2. SS/DS キーワードを使用すると、通貨タイプの間接的な指定が可能です (行の中の指定された列の値がキーとして使われます)。指定された通貨定義タイプ表での定義に従い、キーは通貨タイプに相関します。

トランザクション日付 (TD) を指定する場合、トランザクション日付列で DATE 形式を使用しないならば、日付形式 (DF) も指定する必要があります。指定されたトランザクション日付が、通貨定義レート表で指定された日付範囲の外にある場合、最も近い日付範囲が変換計算に使用されます。

注: 必要なデータ (例えば通貨タイプ、レート) が欠落している場合、実行時に変換エラーが発生します。

以下の表では、Currency 関数パラメーターの有効な形式と可能な値について説明します。パラメーターは任意の順序で指定できます。Currency 関数の中でパラメーターを区切るには、コンマまたはスペースを使用してください。

パラメーター	形式
ソース列	SC=*column-name* *SRCCOL=column name*
ソース通貨タイプ	ST=*code* *SRCTYP=code* *ここで、code は ISO 4217 通貨コード*
宛先通貨タイプ	DT=*code* DSTTYP=*code* *ここで、code は ISO 4217 通貨コード*
ソース仕様	SS=*column name, Types Table number* *SRCSPC=column name, Types Table number* *ここで、column name, Types Table number はソース通貨タイプの指定に使われる Types 表の列と番号 (通貨定義で定義される)*
宛先仕様	DS=*column name, Types Table number* *DSTSPC=column name, Types Table number* *ここで、column name, Types Table number は宛先通貨タイプの指定に使われる Types 表の列と番号 (通貨定義で定義される)*
三角変換 (ユーロ・ドルを経由した変換を強制)	TR TRIANG
通貨定義	CU=*Currency Definition name* *CURTBL=Currency Definition name* *ここで Currency Definition name は、適切な変換パラメーターを含む通貨定義。*
トランザクション日付	TD=*column name* TRNDAT=*column name* *ここで、column name は変換日付を識別するためのトランザクション日付列の名前。*
日付形式	DF='*format*' *DATFMT='format'、ここで format はトランザクション日付列の形式 (日付タイプでない場合)。*
数値の位取り	NS=*scale* *NUMSCL=scale* *ここで scale は、位取りが未定義の Oracle 数値宛先列に適用される位取りです。*

例 1

フィンランド・マルカからユーロ・ドルに変換するには、Currency 関数の形式を以下のように指定します。

```
CURRENCY(ST=FIM DT=EUR)
```

元の値を保持する必要がある場合は、Currency 関数を使用して、宛先表の別の列に値を提供します。

例 2

フィンランド・マルカからユーロ・ドルに変換し、新しい列を作成して、ITEM_COST というラベルが付いた列に元のソース値 (フィンランド・マルカ単位) を保持するには、Currency 関数の形式を以下のように指定します。

```
CURRENCY(ST=FIM DT=EUR SC=ITEM_COST)
```

自動生成される E メール名

「自動生成される E メール名」機能は、連番が連結されたリテラルに基づくユーザー名を使って E メール・アドレスを生成します。連番は 1 で始まり 1 ずつ増える接尾部です。この機能は、指定したソース列にある E メール・アドレスのドメイン・ネームを使用します。

フォーマット設定された E メール名

「フォーマット設定された E メール名」機能は、1 つまたは 2 つの属性から得られる値に基づくユーザー名を使って E メール・アドレスを生成します。このポリシーは、指定されたソース列にある E メール・アドレスのドメイン・ネームを使用します。

乱数関数

乱数関数では、下限値および上限値で指定される範囲内でランダムに選択された数値を生成します。

乱数関数を使用して、文字データまたは数値データを置き換えることができます。 下限値および上限値は、-2,147,483,648 から 2,147,483,647 までの範囲内の整数である必要があります。 下限値は上限値より小さい必要があります。

連番関数

連番関数では、順次増分される数値を生成します。

連番関数を使用して、文字データまたは数値データを置き換えることができます。 開始値、および番号の増分値を入力する必要があります。 開始値および増分値は、-2,147,483,648 から 2,147,483,647 までの範囲内の整数である必要があります。

生成される値は、宛先列のデータ・タイプおよび長さで制限されます。 生成された値が宛先列の長さを超える場合は、この関数により自動的に開始値にリセットされます。

出口ルーチンの使用

データの変換、作成、挿入、ロード、またはリストアを行うサービスを作成する際には、1 つ以上の列マップを含む表マップを指定して、宛先列に適切な値を派生させることができます。

列マップで列値を指定するには、いくつかの方法があります。 1 つの方法は、出口ルーチンをソースとして指定することです。そのままでは宛先列に定義できないような値を、この出口で設定します。別の方法として、出口ルーチンを使って行を処理から除外することができます。以下に示す、3 つのタイプの出口ルーチンを使用できます。

標準出口

標準出口ルーチンは、列マップで宛先列の値を派生させるために呼び出されます。列マップのスコープを超えるデータ変換を実行する必要がある場合には、この種類の出口ルーチンが役立ちます。例えば出口を使用

すると、複雑なアルゴリズムに従って選択した行の従業員部門番号を変更する操作、または処理対象となる特定の行を選択して他のすべての行を廃棄する操作が可能です。

標準出口では、ソース LOB 列のサブストリング・セグメントを取得することができます。宛先 LOB 列で新しい LOB 値を挿入するために、出口でファイルを作成し、列マップ処理プログラムにファイル名を戻すことができます。

ソース・フォーマット出口

ソース・フォーマット出口は、そのままでは列マップでサポートされない Age 関数内のソース列をフォーマット設定するために呼び出されます。この出口ルーチンは、文字の列または整数の列のソース日付を検査し、Age 関数への入力として使用可能な日付形式に変換します。

宛先フォーマット出口

宛先フォーマット出口は、そのままでは列マップでサポートされない Age 関数内の宛先列をフォーマット設定するために呼び出されます。この出口ルーチンは、4 つの異なる宛先形式のいずれか 1 つに日付を変換します。その形式は、宛先列のデータ型によって決まります。

列マップの中の出口

列マップで出口ルーチンを使用するには、適切なソース列の中で以下のいずれか 1 つを指定する必要があります。

標準出口
EXIT *dllname[(parm[[,]parm]...)]*

ソース・フォーマット出口
AGE(SRCEXIT=*dllname*)

宛先フォーマット出口
AGE(DSTEXIT=*dllname*)

プロセスでは、処理されるデータ行ごとに列マップ出口ルーチンを一度ずつ呼び出し、最後の行が処理された後に終了呼び出しを渡します。標準出口ルーチンで指定されたオプション・パラメーターは出口に渡されますが、それはストリング (単一引用符で囲む) または数値リテラル (8 以下) でなければなりません。

出口ルーチンの作成

どのプログラミング言語でも出口ルーチンを作成できます。ただし、サブルーチンの呼び出しは、C プログラミング言語で使われる規約に従う必要があります。

ヘッダー・ファイル

出口ルーチンで使われるパラメーター、構造体、および戻りコードを定義するために、出口ルーチンは 2 つの C プログラム・ヘッダー・ファイルをインクルードする必要があります。つまり PSTEXIT.H と、メタデータ (表名、列名など) の文字フォーマットに応じて PSTCMXIT.H または PSTCMWXT.H のいずれかです。

PSTEXIT.H
Optim 定義のデータ型に対応するデータ型、戻りコード、および構造体を指定します。

PSTCMXIT.H
1 バイト (例えば ASCII) 形式のメタデータ用です。

列マップ・コールバック・ルーチンのプロトタイプを提供し、列マップ出口パラメーターの *defines* および *structure definitions* を指定します。

PSTCMWXT.H

UTF-16 (WCHAR) 形式のメタデータ用です。

列マップ・コールバック・ルーチンのプロトタイプを提供し、列マップ出口パラメーターの *defines* および *structure definitions* を指定します。

これらのヘッダー・ファイルは、IBM InfoSphere Optim アプリケーション・ファイルと同じディレクトリー内にあります。

注: パラメーターに 1 バイト形式と UTF-16 形式がある場合、この章では UTF-16 形式を括弧に入れて示します。

DLL の使用

それぞれの出口ルーチンを、別々の DLL としてコンパイルおよびリンクする必要があります。 Optim は、実行時にそれぞれの DLL を動的にロードします。 1 つの DLL の中には、特定の種類の列マップ出口の出現を 1 つだけ含めることができます。ただし、それぞれの種類の列マップ出口ルーチンを 1 つずつ、同じ DLL に含めることができます。

列マップで DLL の名前を使用する必要があります。プラットフォームに定義されている DLL 名を指定してください (つまり、総称名を使用しないでください)。以下のような、出口ルーチンを実装する実際の関数を指定する DLL を作成し、それらの関数をエクスポートしてください。

標準出口

PSTColMapExit
(PSTColMapWExit)

ソース・フォーマット出口

PSTColMapAgeSrcExit
(PSTColMapAgeSrcWExit)

宛先フォーマット出口

PSTColMapAgeDstExit
(PSTColMapAgeDstWExit)

要件

それぞれの列マップ出口ルーチンは、以下の要件を満たす必要があります。

- 出口は、宛先列のために適切な値を生成する必要があり、他のどのストレージ域をも変更してはなりません。
- Optim は主キーを使って内部作業域を構築します。主キーに含まれる列に出口ルーチンが値を割り当てる場合、同じソース行に関してそのルーチンが複数回呼び出されるならば、出口ルーチンで同じ出力値が毎回必ず生成されるようにしてください。
- 1 つの出口が頻繁に呼び出される可能性があるため、不必要なオーバーヘッドを避けてください。例えば、出口は最初の呼び出しで初期化処理を行い、後続の呼び出しのために作業域に情報を保存することができます。

標準出口ルーチン

標準出口を作成する際には、出口関数 PSTColMapExit (PSTColMapWExit) および以下のパラメーターを指定します。

1 バイト

```
PSTColMapExit
   (PST_STRUCT_CM_EXIT_PARM * pInputParms,
    PST_STRUCT_CM_EXIT_COL_LIST * pSrcColList,
     PST_STRUCT_CM_EXIT_COL_LIST * pDstColList)
```

UTF-16

```
PSTColMapWExit
   (PST_STRUCT_CM_WEXIT_PARM * pInputParms,
    PST_STRUCT_CM_WEXIT_COL_LIST * pSrcColList,
     PST_STRUCT_CM_WEXIT_COL_LIST * pDstColList)
```

パラメーター

プロセスが標準出口ルーチンを呼び出すとき、プロセスはそれぞれの呼び出しで以下のパラメーターを渡します。

pInputParms

PST_STRUCT_CM_EXIT_PARM (PST_STRUCT_CM_WEXIT_PARM) へのポインター。この構造体には、ソース表と宛先表、現在の呼び出しの性質、出口で指定されるオプション・パラメーターの数などの情報に加えて、コールバック関数、作業域、オプション・パラメーターへのポインターが含まれています。

この構造体の最初のフィールドは **FuncCode** フィールドであり、PST_CM_FUNC_TRANSFORM (PST_CMW_FUNC_TRANSFORM) または PST_CM_FUNC_TERMINATE (PST_CMW_FUNC_TERMINATE) のいずれかによって識別されます。

NumParms フィールドには、列マップ出口で指定されるオプション・パラメーターの数が含まれます (0 から 8 まで)。

pParm フィールドには、列マップ出口で指定される各オプション・パラメーターへのポインターから成る配列が含まれます。

pScrColList

PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST) へのポインター。この構造体はソース列について記述します。

pDstColList

PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST) へのポインター。この構造体は宛先列について記述します。

コールバック関数

Optim プロセスが標準出口ルーチンを呼び出すたびに、プロセスは以下のコールバック関数のアドレスを渡します。

pPSTGetColValue( )

現行のデータ行のすべてのソース列およびほとんどの宛先列のデータを取り出します。通常、出口ルーチンはこの関数を 1 回だけ呼び出してソース列のデータを取り出します。しかし、出口ルーチンはこの関数を複数回呼び出して、さまざまな列のデータを取り出すことができます。

pPSTPutColValue( )
現在のデータ行の宛先列のデータを指定します。出口ルーチンは宛先列の値を決定し、その値を Optim に返します。行が拒否される場合、またはプロセスが異常終了される場合を除いて、この関数を呼び出す必要があります。

処理

標準出口ルーチンの標準的な処理は、次のような手順に要約されます。

1. Optim から呼び出されるごとに、出口ルーチンは 1 回目の呼び出しであるかどうかを確認します。 1 回目の呼び出しでは、出口はすべての初期化タスクおよび通常の処理 (ステップ 2) を実行します。 2 回目以降の呼び出しでは、通常の処理 (ステップ 2) だけが実行されます。
2. Optim は列データを標準出口ルーチンに渡しません。しかし、出口ルーチンは **pPSTGetColValue()** コールバック関数を呼び出してソース列のデータを取得することができます。宛先列の値を決定するうえで、これらの値が必要とされます。
3. 出口ルーチンが宛先値を生成した後、出口は **pPSTPutColValue( )** コールバック関数を呼び出して宛先列に値を格納するか、データ行のスキップや異常終了をプロセスに指示する適切な戻りコードを渡します。
4. 最後のデータ行が処理された後、Optim は終了呼び出しを出口ルーチンに渡します。これは FuncCode フィールドの値 PST_CM_FUNC_TERMINATE (PST_CMW_FUNC_TERMINATE) によって示されます。この呼び出しにより、出口ルーチンは動的に割り振られたすべてのストレージを解放するよう促されます。最終的なタスクが完了すると、出口ルーチンは戻りコードを Optim に渡します。

戻りコード

以下の戻りコードが標準出口ルーチンに適用されます。

```
PST_CM_EXIT_SUCCESS
(PST_CMW_EXIT_SUCCESS)
```

宛先列に値が割り当てられたか、正常に変換が行われました。

```
PST_CM_EXIT_REJECT_ROW
(PST_CMW_EXIT_REJECT_ROW)
```

宛先列は、値を割り当てることも、変換することもできません。その行は廃棄されます。

```
PST_CM_EXIT_ABORT_PROCESS
(PST_CMW_EXIT_ABORT_PROCESS)
```

致命的エラー。処理を終了します。エラー・メッセージを返すには、作業域にメッセージを配置し、使用されない領域をブランクまたは NULL に設定してください。

ソース・フォーマット出口

Optim 用のソース・フォーマット出口ルーチンを作成する際には、出口関数 PSTColMapAgeSrcExit (PSTColMapAgeSrcWExit) および以下のパラメーターを指定します。

1 バイト

```
PSTColMapAgeSrcExit
   (PST_STRUCT_CM_AGE_SRCFMT_PARM * pInputParms,
   PST_STRUCT_CM_EXIT_COL_LIST * pSrcColList,
   PST_STRUCT_CM_EXIT_COL_LIST * pDstColList)
```

UTF-16

```
PSTColMapAgeSrcWExit
   (PST_STRUCT_CM_AGE_SRCFMT_WPARM *
pInputParms,
   PST_STRUCT_CM_WEXIT_COL_LIST * pSrcColList,
   PST_STRUCT_CM_WEXIT_COL_LIST * pDstColList)
```

パラメーター

Optim プロセスがソース・フォーマット出口ルーチンを呼び出すとき、プロセスは以下のパラメーターを渡します。

pInputParms

PST_STRUCT_CM_AGE_SRCFMT_PARM (PST_STRUCT_CM_AGE_SRCFMT_WPARM) へのポインター。この構造体には、ソース表と宛先表の情報、現在の呼び出しの性質に関する情報、およびコールバック関数と作業域へのポインターが含まれています。

この構造体の最初のフィールドは **FuncCode** フィールドであり、PST_CM_SRCFMT_TRANSFORM (PST_CMW_SRCFMT_TRANSFORM) または PST_CM_SRCFMT_TERMINATE (PST_CMW_SRCFMT_TERMINATE) のいずれかによって識別されます。

pSrcColList

PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST) へのポインター。この構造体はソース列について記述します。

pDstColList

PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST) へのポインター。この構造体は宛先列について記述します。

コールバック関数

Optim プロセスがソース・フォーマット出口ルーチンを呼び出すたびに、プロセスは以下のコールバック関数のアドレスを渡します。

pPSTGetColValue( )

現行のデータ行のすべてのソース列およびほとんどの宛先列のデータを取り出します。通常は、最初のパラメーターでソース列のデータが提供されるため、出口ルーチンはこの関数を呼び出す必要がありません。しかし、異なる列のデータを取り出すために、出口ルーチンでこの関数を呼び出すことができます。

処理

ソース・フォーマット出口ルーチンの標準的な処理は、次のような手順に要約されます。

1. Optim から呼び出されるごとに、出口ルーチンは 1 回目の呼び出しであるかどうかを確認します。 1 回目の呼び出しでは、出口はすべての初期化タスクおよび通常の処理 (ステップ 2) を実行します。 2 回目以降の呼び出しでは、通常の処理 (ステップ 2) だけが実行されます。
2. 出口は、列マップで定義された Age 関数による指定に従い、ソース列の値を受け取ります。 Optim は、出口ルーチンを呼び出す前に Age 関数を適用せず、ヘッダー・ファイルの **ValueType** フィールドで示されるいずれか 1 つの **InputValue** 共用体フィールドに未加工値を格納します。
3. 宛先列の値を計算するために出口が他の列を調べる必要がある場合、出口は **pPSTGetColValue( )** コールバック関数を呼び出してそれらの列の値を取得する必要があります。
4. 宛先値が生成された後、出口ルーチンは値をフォーマット設定して、**OutputTimeStamp** フィールドまたは **OutputSybDateTime** フィールドのいずれかにそれを配置する必要があります。 Optim はこの値を検

証して、Age 関数を適用します。出口は、データが保存されているフィールドを示したり、データ行のスキップや異常終了をプロセスに指示したりする適切な戻りコードを渡す必要があります。

5. 最後のデータ行が処理された後、Optim は終了呼び出しを出口ルーチンに渡します。これは **FuncCode** フィールドの値 PST_CM_SRCFMT_TERMINATE (PST_CMW_SRCFMT_TERMINATE) によって識別されます。この呼び出しにより、出口ルーチンは動的に割り振られたすべてのストレージを解放するよう促されます。最終的なタスクが完了すると、出口ルーチンは戻りコードを Optim に渡します。

異常終了モード

以下に示すような、いくつかの方法に従い、ソース・フォーマット出口ルーチンで処理を異常終了させることができます。

- スキップされる日付または無効な日付を持つ行を処理する。プロセス要求でこれらのオプションのいずれかを選択した場合、ソース列と宛先列の属性が同じであれば、ソース列は変更されずに宛先列にコピーされます。どちらのオプションも選択しない場合、行は拒否されます。
- 行を拒否する。出口ルーチンでの指定に基づき、スキップされる日付や無効な日付に関する処理オプションとは無関係に行を拒否します。
- 出口ルーチンでの指定に基づき、プロセス全体を異常終了させる。

出口ルーチンは、どの日付形式またはどの異常終了モードを使用するかを示す戻りコードを渡します。

プロセスの間、出口ルーチンは入力行のすべての列、および宛先行のいくつかの列を調査することができます。ただし、出口ルーチンを含み、列マップの中で現在の宛先列より後に定義されている宛先列を、出口ルーチンで調査することはできません。その他のすべての宛先列は使用可能です。

戻りコード

以下の戻りコードがソース・フォーマット出口に適用されます。

```
PST_CM_SRCFMT_USE_TIMESTAMP
(PST_CMW_SRCFMT_USE_TIMESTAMP)
```

出口に渡された最初のパラメーターの **OutputTimeStamp** フィールドの値が宛先列に割り当てられます。

```
PST_CM_SRCFMT_USE_SYB_DATETIME
(PST_CMW_SRCFMT_USE_SYB_DATETIME)
```

出口に渡された最初のパラメーターの **OutputSybDateTime** フィールドの値が宛先列に割り当てられます。

```
PST_CM_SRCFMT_SKIP
(PST_CMW_SRCFMT_SKIP)
```

経時処理は適用されません。「**スキップされる日付を持つ行を処理する**」オプションを選択しない場合、行は拒否されます。オプションを選択した場合、ソースと宛先の間に互換性があれば、データはソースにコピーされます。互換性がない場合、行は拒否されます。

```
PST_CM_SRCFMT_COL_INVALID
(PST_CMW_SRCFMT_COL_INVALID)
```

経時処理は適用されません。「無効な日付を持つ行を処理する」オプションを選択しない場合、行は拒否されます。オプションを選択した場合、ソースと宛先の間に互換性があれば、データはソースにコピーされます。互換性がない場合、行は拒否されます。

```
PST_CM_SRCFMT_REJECT_ROW
(PST_CMW_SRCFMT_REJECT_ROW)
```

ソース列に値を割り当てることができません。その行は拒否 (廃棄) されます。

```
PST_CM_SRCFMT_ABORT_PROCESS
(PST_CMW_SRCFMT_ABORT_PROCESS)
```

致命的エラー。終了します。エラー・メッセージを返すには、作業域にメッセージを配置し、使用されない領域をブランクまたは NULL に設定してください。

宛先フォーマット出口

Optim 用の宛先フォーマット出口ルーチンを作成する際には、出口関数 PSTColMapAgeDstExit (PSTColMapAgeDstWExit) および以下のパラメーターを指定します。

1 バイト

```
PSTColMapAgeDstExit
   (PST_STRUCT_CM_AGE_DSTFMT_PARM * pInputParms,
   PST_STRUCT_CM_EXIT_COL_LIST * pSrcColList,
   PST_STRUCT_CM_EXIT_COL_LIST * pDstColList)
```

UTF-16

```
PSTColMapAgeDstWExit
   (PST_STRUCT_CM_AGE_DSTFMT_WPARM * pInputParms,
   PST_STRUCT_CM_WEXIT_COL_LIST * pSrcColList,
   PST_STRUCT_CM_WEXIT_COL_LIST * pDstColList)
```

パラメーター

Optim プロセスが宛先フォーマット出口ルーチンを呼び出すとき、プロセスは以下のパラメーターを渡します。

pInputParms

PST_STRUCT_CM_AGE_DSTFMT_PARM (PST_STRUCT_CM_AGE_DSTFMT_WPARM) へのポインター。この構造体には、ソース表と宛先表の情報、現在の呼び出しの性質に関する情報、およびコールバック関数と作業域へのポインターが含まれています。

この構造体の最初のフィールドは次のもので識別される **FuncCode** フィールドです。

PST_CM_DSTFMT_TO_CHAR (PST_CMW_DSTFMT_TO_WCHAR)、
PST_CM_DSTFMT_TO_INTEGER (PST_CMW_DSTFMT_TO_INTEGER)、
PST_CM_DSTFMT_TO_TIMESTAMP (PST_CMW_DSTFMT_TO_TIMESTAMP)、
PST_CM_DSTFMT_TO_SYB_DATETIME (PST_CMW_DSTFMT_TO_SYB_DATETIME)、または
PST_CM_DSTFMT_TERMINATE (PST_CMW_DSTFMT_TERMINATE)

pSrcColList

PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST) へのポインター。この構造体はソース列について記述します。

pDstColList

PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST) へのポインター。この構造体は宛先列について記述します。

コールバック関数

Optim プロセスが宛先フォーマット出口ルーチンを呼び出すたびに、プロセスは以下のコールバック関数のアドレスを渡します。

pPSTGetColValue( )

現行のデータ行のすべてのソース列およびほとんどの宛先列のデータを取り出します。通常は、経

時処理されたソース列のデータが最初のパラメーターで提供されるため、出口ルーチンはこの関数を呼び出す必要がありません。しかし、異なる列のデータを取り出すために、出口ルーチンでこの関数を呼び出すことができます。

形式

入力日付は PST_C_TIMESTAMP フォーマットおよび PST_C_SYB_DATETIME フォーマットです。出口は宛先列のデータ型に基づいて、その日付を以下のいずれか 1 つのフォーマットに変換します。

PST_C_CHAR_SZ
CHAR および VARCHAR 宛先列。

PST_C_INTEGER_CHAR_SZ
NUMERIC 宛先列。

PST_C_TIMESTAMP
DB2 および Oracle DATE/TIME 列。

PST_C_SYB_DATETIME
Sybase ASE DATETIME 列。

処理

宛先フォーマット出口ルーチンの標準的な処理は、次のような手順に要約されます。

1. Optim から呼び出されるごとに、出口ルーチンは 1 回目の呼び出しであるかどうかを確認します。 1 回目の呼び出しでは、出口はすべての初期化タスクおよび通常の処理 (ステップ 2) を実行します。 2 回目以降の呼び出しでは、通常の処理 (ステップ 2) だけが実行されます。
2. 出口は、列マップで定義された Age 関数による指定に従い、ソース列の値を受け取ります。 Optim は列マップ出口を呼び出す前に Age 関数を適用し、経時処理された値をヘッダー・ファイルの **InputTimeStamp** および **InputSybaDateTime** フィールドに格納します。
3. 宛先列の値を計算するために出口が他の列を調べる必要がある場合、出口は **pPSTGetColValue( )** コールバック関数を呼び出してそれらの列の値を取得する必要があります。
4. 宛先値が生成された後、出口ルーチンは値をフォーマット設定し、**OutputValue** 共用体のいずれか 1 つのフィールドにそれを配置する必要があります。 **FuncCode** フィールドは、**OutputValue** 共用体の中で値を配置する必要のあるフィールドを示します。出口は、データが保存されているフィールドを示したり、行のスキップや異常終了をプロセスに指示したりする適切なコードを返す必要があります。
5. 最後のデータ行が処理された後、Optim は終了呼び出しを出口ルーチンに渡します。これは **FuncCode** フィールドの値 PST_CM_DSTFMT_TERMINATE (PST_CMW_DSTFMT_TERMINATE) によって示されます。この呼び出しにより、出口ルーチンは動的に割り振られたすべてのストレージを解放するよう促されます。最終的なタスクが完了すると、出口ルーチンは戻りコードを Optim に渡します。

異常終了モード

以下に示すような、いくつかの方法に従い、宛先フォーマット出口ルーチンで処理を異常終了させることができます。

- スキップされる日付または無効な日付を持つ行を処理する。プロセス要求でこれらのオプションのいずれかを選択した場合、ソース列と宛先列の属性が同じであれば、ソース列は変更されずに宛先列にコピーされます。どちらのオプションも選択しない場合、行は拒否されます。
- 行を拒否する。出口ルーチンでの指定に基づき、スキップされる日付や無効な日付に関する処理オプションとは無関係に行を拒否します。
- 出口ルーチンでの指定に基づき、プロセス全体を異常終了させる。

出口ルーチンは、変換が成功したかどうか、またはどの異常終了モードを使用するかを示す戻りコードを渡します。

プロセスの間、出口ルーチンは入力行のすべての列、および宛先行のいくつかの列を調査することができます。ただし、出口ルーチンを含み、列マップの中で現在の宛先列より後に定義されている宛先列を、出口ルーチンで調査することはできません。その他のすべての宛先列は使用可能です。

戻りコード

以下の戻りコードが宛先フォーマット出口に適用されます。

```
PST_CM_DSTFMT_SUCCESS
(PST_CMW_DSTFMT_SUCCESS)
```

出口に渡された最初のパラメーターの **FuncCode** フィールドにおける指定に従い、宛先列に値が割り当てられます。

```
PST_CM_DSTFMT_SKIP
(PST_CMW_DSTFMT_SKIP)
```

「**スキップされる日付を持つ行を処理する**」オプションを選択しない場合、行は拒否されます。オプションを選択した場合、ソースとターゲットの間に互換性があれば、ソース・データはターゲットにコピーされます。互換性がない場合、行は拒否されます。

```
PST_CM_DSTFMT_COL_INVALID
(PST_CMW_DSTFMT_COL_INVALID)
```

「**無効な日付を持つ行を処理する**」オプションを選択しない場合、行は拒否されます。オプションを選択した場合、ソースとターゲットの間に互換性があれば、データはソースにコピーされます。互換性がない場合、行は拒否されます。

```
PST_CM_DSTFMT_REJECT_ROW
(PST_CMW_DSTFMT_REJECT_ROW)
```

宛先列に値を割り当てることができません。その行は拒否 (廃棄) されます。

```
PST_CM_DSTFMT_ABORT_PROCESS
(PST_CMW_DSTFMT_ABORT_PROCESS)
```

致命的エラー。終了します。エラー・メッセージを返すには、作業域にメッセージを配置し、使用されない領域をブランクまたは NULL に設定してください。

Lua スクリプトを使用した列マップ・プロシージャーの作成

列マップ・プロシージャーは、サービス実行時に列のデータをマスクしたり変換したりするために使われるプロシージャーです。列マップ・プロシージャーという名前が示すように、列マップにこれを追加する必要があります。 Lua スクリプト言語を使用して、列マップ・プロシージャーを作成することができます。

Lua プログラミング言語の詳細情報については、Lua の Web サイト http://www.lua.org を参照してください。

列マップ・プロシージャー用の Lua 関数

列マップ・プロシージャーでは、標準的なほとんどの Lua 関数がサポートされます。また、Optim に固有の関数も列マップ・プロシージャーでサポートされます。

ユーザー定義関数の標準的な関数名

以下の関数名を Lua 列マップ・プロシージャーで使用します。これらの関数はそれぞれ、示されている時点で自動的に呼び出されます。また、他の関数名を持つ関数を作成して、標準関数からこれらの関数を呼び出すこともできます。

名前	説明	必須かどうか
cm_load()	この関数は、すべての表が処理される前に呼び出されます。	いいえ
cm_unload()	この関数は、すべての表が処理された後に呼び出されます。	いいえ
cm_starttable()	この関数は、それぞれの表の処理の始めに呼び出されます。	いいえ
cm_endtable()	この関数は、それぞれの表の処理の終わりに呼び出されます。	いいえ
cm_transform()	この関数は、処理されるそれぞれの行に関して呼び出されます。	はい

それぞれの列マップ・プロシージャーは cm_transform() 関数を含んでいる必要があります。cm_transform() 関数が存在しない場合、コードは複合式として評価されます。

グローバル関数

以下の関数は、すべての列マップ・プロシージャー実行コンテキストで使用可能です。

名前	説明
numparms()	列マップ・プロシージャーに渡されるパラメーターの数を取得します。
parms.get(*n*)	インデックス *n* におけるパラメーターの値を取得します。
print()	処理レポートにメッセージを出力します。
rejectrow()	行をスキップして、次の行に移動します。

データ・ストア関数

以下の関数を使用して、ソースおよびターゲットのデータ・ストアについての情報を取得します。

名前	説明
source.getdbalias()	ソース・データ・ストアの DB 別名を取得します。
source.getcreatorid()	ソース・データ・ストアの作成者 ID を取得します。
target.getdbalias()	ターゲット・データ・ストアの DB 別名を取得します。
target.getcreatorid()	ターゲット・データ・ストアの作成者 ID を取得します。

表関数

以下の関数を使用して、ソース表およびターゲット表についての情報を取得します。

名前	説明
source.gettablename()	ソース表の名前を取得します。
target.gettablename()	ターゲット表の名前を取得します。

列関数

以下の関数を使用することで、ソース列とターゲット列についての情報を取得し、列データを変換し、結果をターゲット列に書き込んで終了します。

名前	説明
source.column.getvalue()	ソース列から非数値を取得します。 データ型 PST_SQL_BINARY の列から値を取得するために source.column.getvalue() 関数を使用すると、実行時エラーが生成されます。
source.column.getasdouble()	ソース列から数値を取得します (列マップ・プロシージャーでは数値データを倍精度形式で処理します)。 この関数は、データ型 PST_SQL_DECIMAL、PST_SQL_DOUBLE、PST_SQL_FLOAT、PST_SQL_INFX_DEC_FLOAT、および PST_SQL_ORA_NUMBER の列に対して使用してください。
source.column.getlength()	ソース列のストリングの長さを取得します。
source.column.getname()	ソース列の名前を取得します。
source.column.gettype()	ソース列のデータ型を取得します。
target.column.setvalue()	ターゲット列の値を設定します。 データ型 PST_SQL_BINARY の列の値を設定するために target.column.setvalue() 関数を使用した場合、実行時エラーが生成されます。
target.column.getlength()	ターゲット列のストリングの長さを取得します。
target.column.getname()	ターゲット列の名前を取得します。
target.column.gettype()	ターゲット列のデータ型を取得します。
optimmask()	Optim Data Privacy Providers (ODPP) プロバイダーを呼び出します。
userexit()	ユーザー出口を呼び出します。

サポートされない関数

以下の種類の関数は列マップ・プロシージャーではサポートされません。

- Lua io ライブラリーの中に組み込まれる入出力機能
- string.dump()

列マップ・プロシージャーの制約事項

列マップ・プロシージャーを作成するときには、以下の制約に注意してください。

倍精度形式で処理される数値データ

列マップ・プロシージャーでは、数値データを倍精度形式で処理します。このため、source.column.getasdouble() 関数を使ってソース列から数値データを取得する必要があります。

エンコード

列マップ・プロシージャーの内部処理では UTF-16 エンコード方式を使用します。

PST_SQL_BINARY データ型がサポートされない

データ型 PST_SQL_BINARY の列から値を取得するために source.column.getvalue() 関数を使用すると、実行時エラーが生成されます。また、データ型 PST_SQL_BINARY の列の値を設定するために target.column.setvalue() 関数を使用した場合にも、実行時エラーが生成されます。

サポートされない関数

以下の種類の関数は列マップ・プロシージャーではサポートされません。

- Lua io ライブラリーの中に組み込まれる入出力機能
- string.dump()

列マップ・プロシージャーの例: 汎用プロシージャー

列マップ・プロシージャーの例では、標準的な関数 cm_load、cm_unload、cm_starttable、cm_endtable、および cm_transform を含む列マップ・プロシージャーの構造を示します。

```
-------------------------------------------------------------------------------
--
--   IBM Optim sample column map procedure
--
--   Name:          OptimSample
--
--   Revision:      1.0
--
--   Description:   Demonstrates all capabilities of Optim/Lua column map procedures
--
--   Input:         Zero or more parameters, which will simply be echoed back to the
--                  Optim process report
--
--
--   Output:        Section in Optim process report showing information from this
--                  column map procedure.  The column itself is left unchanged.
--
--
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
--  cm_load function - Called before any tables are processed
-------------------------------------------------------------------------------
function cm_load()

  print("  *** Start of Process ***")

  colinfoshown = false

  -- Display parameters passed from Column Map
```

```
  print("  Argument Count: " .. string.format("%d", numparms()))
    for i = 1, numparms(), 1 do
      print("    Argument " .. string.format("%d", i-1) .. " " .. parms.get(i-1))
  end

end

-------------------------------------------------------------------------------
--  cm_unload function - Called after all tables are processed
-------------------------------------------------------------------------------
function cm_unload()

  print("  *** End of Process ***")

end

-------------------------------------------------------------------------------
--  cm_starttable function - Called at the start of processing for each table
-------------------------------------------------------------------------------
function cm_starttable()

  print("  \nStart of processing table")
  fullname = source.getdbalias() .. "." .. source.getcreatorid() .. "." .. source.gettablename()
  print("      Source Table: " .. fullname)
  fullname = target.getdbalias() .. "." .. target.getcreatorid() .. "." .. target.gettablename()
  print("      Target Table: " .. fullname)

end

-------------------------------------------------------------------------------
--  cm_endtable function - Called at the end of processing for each table
-------------------------------------------------------------------------------
function cm_endtable()

  print("  \nEnd of processing table")

end

-------------------------------------------------------------------------------
--  cm_transform function - Called for each row processed
-------------------------------------------------------------------------------
function cm_transform()


  if (not colinfoshown) then
     colinfoshown = true
     print("  Processing column " .. source.column.getname())
     print("     Type: " ..  source.column.gettype())
     print("     Length: " .. source.column.getlength())
  end

  -- This statement gets the value in the column for which cm_transform was called
  -- Optionally, the name of another column can be specified, for example:
  -- source.column.getvalue("COL1") will return the value in column COL1
  oldvalue = source.column.getvalue()

  -- This code sets that target column to the same value as the source column
  -- Logic to change the value would be placed here.
  -- If you wish to NOT insert this row into the target table, call the rejectrow() function
    newvalue = oldvalue
    target.column.setvalue(newvalue)
end
```

列マップ・プロシージャーの例: 切り替えルックアップ

この列マップ・プロシージャーの例では、ルックアップ表のデータを使用して mask_parms 関数で列をマスクする方法を示します。

```
--------------------------------------------------------------------------------
--
--   IBM Optim sample column map procedure
--
--   Name:          OptimSwitchedLookup
--
--   Revision:      1.0
--
--   Description:   Masks a column by using table lookup.  The lookup table to
--                  be used is determined by the value of another column.
--
--   Input:         Parameter 1 (Required):
--                  A string that indicates the type of lookup to use:
--                  HASH, RANDOM, or LOOKUP
--
--                  Parameter 2 .... n-1 (Required)
--                  An expression that indicates the lookup table to use. The format is
--                  COND(column-name=value) DATASOURCE(datasource_parameters)
--                  This parameter can be repeated multiple times.  If a row does not
--                  satisty any of the COND parameters than it will not be inserted
--                  into the target table.
--
--                  Parameter n (Optional):
--                  A string containing additional parameters to be
--                  copied into the optimmask invocation.
--                  This is in addition to the datasource_parameters value in
--                  the COND clause of Parameter 2 and the mask_parms_constant field
--                  that is declared at the start of this script.
--
--
--
--   Output:       - The masked column data as set by the target.column.setvalue function
--                 - Text directed to the Optim report by the print function
--
--   Return Codes: 0 - Successful execution
--                 1 - Reject row (Use in cm_transform; row will not be inserted to
--                     destination table)
--                 2 - Abort process
--
--                 Two helper functions are specified to specify conditions other than
--                 return code 0.
--                 There is no need to code a return statement when using these
--                 functions.
--                 error(string) - This call causes the Optim process to abort
--                                 and string is shown in the Optim report
--                                 as an error message.
--                 rejectrow()   - This call causes Optim to reject the row
--                                 currently being processed. The row is
--                                 not inserted into the destination table.
--
--
--
--------------------------------------------------------------------------------

function cm_transform()

    -- Change this field to contain parameters that should
    -- be placed into all optimmask calls
    local mask_parms_constant = 'CACHE=Y,WHENNF=PRE'

    --
    -- Validate number of parameters:
```

```
--
nparm = numparms()
if (nparm  < 2) then
   process_error("Call to column map procedure OptimSwitchedLookup must have 2 or more parameters")
end

--
-- Process Parameter 1 (lookup type)
parm = parms.get(0)
if (string.lower(parm) == "hash") then
  provalue = "HASH_LOOKUP"
elseif (string.lower(parm) == "random") then
  provalue = "RANDOM_LOOKUP"
elseif (string.lower(parm) == "lookup") then
  provalue = "LOOKUP"
else
  process_error("Invalid parameter.  Expected HASH, RANDOM or LOOKUP.  Found " .. parm)
end


--
-- Process COND/DATASOURCE parameters
--
gotcond = false
for parmptr = 1, nparm-1 do
  parm = parms.get(parmptr)
  if (string.lower(string.sub(parm, 1, 5)) == "cond(") then
    gotcond = true
    datasource_parameters = process_cond()
    if (datasource_parameters > "") then
      break;
    end
  end
end
if (not gotcond) then
  process_error("No COND parameter found")
end

-- No COND matched this row, so reject the row
if (datasource_parameters <= "") then
  rejectrow()
end

--
-- Process optional additional optimmask parameter
--
lastparm = parms.get(nparm-1)
if (string.lower(string.sub(lastparm, 1, 5)) == "cond(") then
   optimmask_additional_parms = ""
else
   optimmask_additional_parms = lastparm
end

--
-- Construct call to optimmask, make the call,
-- and place new value into target column
--
mask_parms = "PRO=" .. provalue .. "," .. mask_parms_constant

-- This use of environment variables to store userid and password
-- for the system table is for simple illustrative purposes only.
-- For greater security, store this information in environment variables
-- in an encrypted format.
userid = os.getenv("optimmaskuserid")
if (userid) then
  mask_parms = mask_parms .. ",USER=" .. userid
end
```

```
    password = os.getenv("optimmaskpswd")
    if (password) then
      mask_parms = mask_parms .. ",PASS=" .. password
    end

    mask_parms = mask_parms .. "," .. optimmask_additional_parms

    oldvalue = source.column.getvalue()
        newvalue = optimmask(oldvalue, mask_parms)
        target.column.setvalue(newvalue)
end



function process_cond()

  strptr = 6     -- Point to first character after "COND("

  -- Get the column name
  equalsign = string.find(parm, "=", strptr, true)
  if (not equalsign) then
    process_error("Syntax error around character " .. strptr .. " in expression: " .. parm)
  end
  colname = string.sub(parm, strptr, equalsign-1)

  -- Get the column value
  strptr = equalsign + 1
  closeparen = string.find(parm, ")", strptr, true)
  if (not closeparen) then
    process_error("COND expression is missing closing parenthesis in expression: " .. parm)
  end
  colvalue = string.sub(parm, strptr, closeparen-1)

  --Debug - Print scan results:
  --print ("Found colname=" .. colname .. " in parm: " .. parm)
  --print ("Found colvalue=" .. colvalue .. " in parm: " .. parm)

  -- If COND(column-name=value) condition not met for this row, then
  -- do no further processing on this parameter.  Return "" to indicate no hit.
  if (source.column.getvalue(colname) ~= colvalue) then
    return ""
  end

  -- Got a match on COND, so get the DATASOURCE value
  strptr = closeparen + 1
  datasourceValuePtr = string.find(parm, "DATASOURCE(", strptr, true)
  if (not datasourceValuePtr) then
    process_error("DATASOURCE clause not found in expression: " .. parm)
  end
  -- (Note: This scanning is very simple; it assumes that there
  -- is no errant text between the COND and DATASOURCE clauses)
  closeparen = string.find(parm, ")", datasourceValuePtr, true)
  if (not closeparen) then
    process_error("COND expression is missing closing parenthesis in expression: " .. parm)
  end
  return string.sub(parm, datasourceValuePtr+11, closeparen-1)

end
function process_error(msg)
  errprefix = "Error in column map procedure for column " .. source.column.getname()
  errprefix = errprefix .. " in table " .. source.getdbalias() .. "."
     .. source.getcreatorid() .. "." .. source.gettablename()
  error(errprefix .. ":\n   " .. msg)
end
```

列マップ・プロシージャーの例: 国民識別番号のマスキング

この列マップ・プロシージャーの例では、国民識別番号を含む列を mask_parms 関数によってマスクする方法を示します。

```
--------------------------------------------------------------------------------
--
--   IBM Optim sample column map procedure
--
--   Name:          OptimNID
--
--   Revision:      1.0
--
--   Description:   Masks a national ID column based on the value of another
--                  column or a script argument with the country code.
--
--   Input:         Parameter 1 (Required):
--                  A string country code (eg. US for United States)
--                  --OR--
--                  COL(column-name) where column-name is the name of a column
--                  that contains the country code
--
--                  Parameter 2 (Optional):
--                  A string containing additional parameters to be copied into the
--                  optimmask invocation
--                  This is in addition to the mask_parms_constant field that is
--                  delcared at the start of this script.
--
--
--
--   Output:       - The masked column data as set by the target.column.setvalue function
--                 - Text directed to the Optim report by the display function
--
--   Return Codes: 0 - Successful execution
--                 1 - Reject row (Use in cm_transform;  row will not be inserted to
--                     destination table)
--                 2 - Abort process
--
--                 Two helper functions are specified to specify conditions other than
--                 return code 0.
--                 There is no need to code a return statement when using these
--                 functions.
--                 error(string) - This call will cause the Optim process to abort
--                                 and string will be shown in the Optim report
--                                 as an error message.
--                 rejectrow()   - This call will cause Optim to reject the row
--                                 currently being processed The row will
--                                 not be inserted into the destination table.-
--
--
--------------------------------------------------------------------------------

function cm_transform()

    -- Change this field to contain parameters that should
    -- be placed into all optimmask calls
    local mask_parms_constant = 'MTD=REPEATABLE'

    --
    -- Obtain parameters:
    --
    nparm = numparms()
    if nparm  ~= 1 and nparm ~= 2 then
       msg = "Call to column map procedure for column "
                   msg = msg ..  source.column.getname()
       msg = msg .. " must have 1 or 2 parameters"
       error(msg)
```

```
    end
    parm1 = parms.get(0)
    if (nparm == 2) then
       parm2 = parms.get(1)
    end

    --
    -- Get country code into swivalue field based on parameters
    --
    if string.lower(string.sub(parm1, 1, 4)) == 'col(' then
       closeparen = string.find(parm1, ')', 5, true)
       if (not closeparen) then
           error("No closing parenthesis found in expression: " .. parm1)
       end
       colname = string.sub(parm1, 5, closeparen-1)
       swivalue = source.column.getvalue(colname)
    else
       swivalue = parm1
    end

    --
    -- Construct call to optimmask, make the call,
               -- and place new value into target column
    --
    mask_parms = 'PRO=NID, SWI=' .. swivalue .. ', ' .. mask_parms_constant
    if (parm2) then
       mask_parms = mask_parms .. ', ' .. parm2
    end
    oldvalue = source.column.getvalue()
        newvalue = optimmask(oldvalue, mask_parms)
        target.column.setvalue(newvalue)
end
```

第 6 章 データの管理

データの編集、表示、および比較を行うには、Optim Designer ユーティリティーを使用します。表を作成することもできます。

データの参照

ファイル・データ・ストアの内容を、データベースに復元せずに参照するには、Browse ユーティリティーを使用します。

特定のトピックや関数に関する詳細情報を入手するには、Browse ユーティリティーのヘルプ機能を使用します。ヘルプ・ウィンドウを開くには、「**ヘルプ**」 > 「**コンテンツ**」を選択し、項目を右クリックして、「**内容の表示**」を選択するか、または F1 を押します。

データを参照するには、次のようにします。

1. Optim Designer から、「**ツール**」 > 「**参照**」をクリックします。 「参照」ウィンドウが開きます。
2. 「**ファイル**」 > 「**開く**」をクリックして、「開く」ウィンドウを開き、サービスによって作成されたデータ・ファイルを選択します。このファイルは、Configuration ユーティリティーで指定されたデータ・ディレクトリーに保管されています。
3. 「**開く**」をクリックします。 「参照」ウィンドウに、データ・ファイル内の表がリストされます。
4. 「**ツール**」 > 「**選択したオブジェクトをすべて作成**」をクリックします。 「データの参照」ウィンドウで、表が開きます。

データの編集

データベース表内のデータ・セットを、関係を変えずに参照したり編集したりするには、Table Editor を使用します。このエディターは、任意の数の表や関係で構成されている任意の複雑なデータ・モデルを処理し、その際にデータ・セット間の関係が変わらないことが保証されます。

特定のトピックや関数に関する詳細情報を入手するには、Table Editor のヘルプ機能を使用します。ヘルプ・ウィンドウを開くには、「**ヘルプ**」 > 「**コンテンツ**」を選択し、項目を右クリックして、 「**内容の表示**」を選択するか、または F1 を押します。

データを編集するには、次のようにします。

1. Optim Designer から、「**ツール**」 > 「**編集**」をクリックします。 「オプションの編集」ウィンドウが開きます。
2. 「**表名**」または「**指定されたアクセス定義**」に入力し、「**初期表示**」または「**モード**」オプションを選択して、「**OK**」をクリックします。 Table Editor ウィンドウが開きます。
3. 1 つ以上の表を同時に編集します。

   ポインターを別の行に移動するか、メニュー・オプションを使用して、データベースに変更をコミットします。コミットの各インスタンスは、取り消しレベルとしてカウントされます。取り消しレベルは、データベースにコミットされる、行への変更として定義されます。

データをデータベースにコミットしようとしてエラー状態になった場合、データはコミットされませんが、この試行は引き続き取り消しレベルとしてカウントされます。このエディターを使用すると、変更したデータを特定のコミット・ポイントに復元できます。

フェッチ・セットとは、このエディターがデータベース内の 1 つの表から読み取る行のセットのことです。

- すべての表について、変更を取り消し、現在のフェッチ・セット内に取得されている元のバージョンに行を復元するには、「**ツール**」 > 「**取り消し**」を選択します。
- 行に関する変更を取り消すには、その行を右クリックし、「**取り消し**」をクリックします。
- 現在のフェッチ・セット内でコミットされている行の、連続している各バージョンと、元のバージョンを表示するには、その行を右クリックし、「**取り消し...**」をクリックします。

データの比較

ソース表のセット間でデータを比較するには、Compare ユーティリティーを使用します。 Optim リポジトリーに保管される比較要求を定義したり、比較の結果を比較ファイル内に保存したりできます。

比較要求の定義

比較対象のデータ・ソースや処理オプションを定義するには、Compare Request Editor を使用します。

特定のトピックや関数に関する詳細情報を入手するには、Compare ユーティリティーのヘルプ機能を使用します。ヘルプ・ウィンドウを開くには、「**ヘルプ**」 > 「**コンテンツ**」を選択し、項目を右クリックして、 「**内容の表示**」を選択するか、または F1 を押します。

比較要求を定義するには、次のようにします。

1. リポジトリー・エクスプローラーで「**比較**」を右クリックし、「**新規比較**」をクリックします。 Compare Request Editor が開きます。
2. 「**一般**」タブで、以下のステップを実行します。
   a. 「**比較ファイル**」フィールドに、比較ファイルの名前を入力します。
   b. 「**結果を即時に表示**」を選択して、比較プロセスの完了後にそのプロセスの結果を表示します。
   c. 「**比較**」モードを選択します。

1 つの表

ソース・ファイルとソース・ファイル
あるソース・ファイル内の 1 つの表のデータと、別のソース・ファイル内のデータを比較します。

ソース・ファイルとデータベース表
ソース・ファイル内の 1 つの表のデータと、データベース内の 1 つの表のデータを比較します。

データベース表とデータベース表
あるデータベース内の 1 つの表のデータと、別のデータベース内の 1 つの表のデータを比較します。

複数の表

ソース・ファイルとソース・ファイル
あるソース・ファイル内の複数の表のデータと、別のソース・ファイル内のデータを比較します。

ソース・ファイルとアクセス定義
ソース・ファイル内の複数の表のデータと、アクセス定義で指定されているデータを比較します。

ソース・ファイルとすべてのデータベース表
ソース・ファイル内の複数の表のデータと、データベース内のデータを比較します。

注: この選択項目は、データベース・アプリケーションをテストする場合に非常に便利です。ソース・ファイルは「変更前」イメージを表し、比較相手のデータベース表は「変更後」イメージを表します。

アクセス定義とアクセス定義
あるアクセス定義で指定されているデータと、別のアクセス定義で指定されているデータを比較します。

アクセス定義とすべてのデータベース表
アクセス定義で指定されているデータと、データベース内のデータを比較します。

3. 「**ソース**」タブで、比較モードに基づいて比較するデータ・ソースを選択します。
4. 「**ファイル**」 > 「**実行**」をクリックして、比較プロセスを開始します。
5. 「**ファイル**」 > 「**保存**」をクリックして、比較要求を保存します。要求名を IDENTIFIER.NAME という形式で入力します。

このプロセスは、Configuration ユーティリティーで指定したデータ・ディレクトリー内に比較ファイルを作成します。 Browse ユーティリティーを使用して、比較の結果を表示します。

比較要求の編集

比較要求を編集するには、Compare Request Editor を使用します。

特定のトピックや関数に関する詳細情報を入手するには、Compare ユーティリティーのヘルプ機能を使用します。ヘルプ・ウィンドウを開くには、「**ヘルプ**」 > 「**コンテンツ**」を選択し、項目を右クリックして、 「**内容の表示**」を選択するか、または F1 を押します。

比較要求を編集するには、次のようにします。

1. リポジトリー・エクスプローラーで「**比較**」ノードを展開して、編集する要求を右クリックし、「**編集**」をクリックします。 Compare Request Editor が開きます。
2. 要求を編集します。
3. 「**ファイル**」 > 「**保存**」をクリックして、比較要求を保存します。

比較プロセスの実行

比較プロセスを実行して比較ファイルを生成し、Browse ユーティリティーを使用してこのファイルを表示できます。

比較要求を実行するには、次のようにします。

以下のいずれかの方法を使用して、要求を実行します。
リポジトリー・エクスプローラーから比較プロセスを実行するには、比較要求を右クリックし、「**実行**」をクリックします。
Compare Request Editor から比較プロセスを実行するには、「**ファイル**」 > 「**実行**」をクリックします。

このプロセスは、Configuration ユーティリティーで指定したデータ・ディレクトリー内に比較ファイルを作成します。Browse ユーティリティーを使用して、比較の結果を表示します。

表の作成

ソース・データ・ファイル内の定義に基づいてオブジェクトを作成するには、Create ユーティリティーを使用します。

特定のトピックや関数に関する詳細情報を入手するには、Create ユーティリティーのヘルプ機能を使用します。ヘルプ・ウィンドウを開くには、「**ヘルプ**」 > 「**コンテンツ**」を選択し、項目を右クリックして、「**内容の表示**」を選択するか、または F1 を押します。

表を作成するには、以下のようにします。

1. Optim Designer から、「**ツール**」 > 「**作成**」をクリックします。 「オプションの作成」ウィンドウが開きます。
2. サービスによって作成されたソース・データ・ファイルを選択します。このファイルは、Configuration ユーティリティーで指定されたデータ・ディレクトリーに保管されています。
3. 「**表マップ・オプション**」から 1 つ選択します。

   ローカル表マップを選択すると、Local - Table Map Editor が開きます。以下のステップを完了します。

   a. 「**修飾子**」フィールドで、ターゲット・データ・ストア別名と作成者 ID を `data_store_alias.creator_id` の形式で入力しなければなりません。
   b. 宛先の表名を変更します。
   c. 「**ファイル**」 > 「**更新して戻る**」をクリックします。

   名前が付けられている表マップを選択する場合は、以下のステップを完了します。

   a. 「**表マップ名**」フィールドで、表マップ名を入力するか選択します。
4. 「**OK**」をクリックします。 「作成」ウィンドウが開きます。
5. 作成するオブジェクトの編集や選択を行います。
6. 作成プロセスを開始するには、「**ツール**」 > 「**選択したオブジェクトをすべて作成**」をクリックします。

第 7 章 Optim Designer を使用したデータの管理

Optim Designer には、Optim データ・プライバシーおよびテスト・データ管理のソリューションとして単一の設計インターフェースが用意されています。

Optim Designer を使用したデータの抽出

このチュートリアルでは、Optim Designer を使用して、Optim 抽出サービスを作成する方法について学習します。このチュートリアルでは、アクセス定義と抽出サービスを定義します。

このチュートリアルを完了すると、抽出サービスを実行できるようになります。

学習目標

演習を完了すると、以下のタスクの実行方法を習得できます。

- リポジトリー・エクスプローラーのフォルダーを作成する
- アクセス定義を作成して、抽出するデータを定義する
- アクセス定義を編集して、選択基準を定義する
- 抽出サービスを作成する
- 抽出サービスをテストする

所要時間

このモジュールを完了するには、約 60 分かかります。

前提条件

このチュートリアルには、Optim リポジトリーへの接続と、Optim サンプル・データを含むデータ・ストア別名が組み込まれている、Optim Designer 環境が必要です。

このチュートリアルは Optim Designer 環境で実行できます。

リポジトリー・エクスプローラーのフォルダーの作成

この演習では、リポジトリー・エクスプローラーのフォルダーを作成します。リポジトリー・エクスプローラーのフォルダーには、サービス、アクセス定義、列マップ、表マップが含まれます。フォルダーを使用して、これらのオブジェクトを編成します。

データ・アクセス計画および選択ポリシーを作成するには、次のようにします。

1. リポジトリー・エクスプローラー・ビューを右クリックし、「**新規**」 > 「**フォルダー**」をクリックします。 「新規フォルダー」ウィンドウが開きます。
2. 「**名前**」に `Tutorial` と入力し、「**OK**」をクリックします。

これで、Tutorial という名前のリポジトリー・エクスプローラーのフォルダーが作成されました。

アクセス定義の作成

この演習では、アクセス定義を作成します。処理するデータに対して、表、関係トラバーサル、および選択基準を指定するには、アクセス定義を使用します。

アクセス定義を作成する前に、開始表が含まれるデータベースのデータ・ストア別名が存在していなければなりません。

アクセス定義を作成するには、以下のようにします。

1. リポジトリー・エクスプローラーでチュートリアルのフォルダーを展開し、「**アクセス定義**」を右クリックして「**新規アクセス定義**」をクリックします。 「新規アクセス定義」ウィザードが開きます。
2. 「アクセス定義名の入力」ページで、SAMPLE.AD と入力します。「**次へ**」をクリックします。
3. 「データ・ストア別名の選択」ページで、Optim サンプル・データが含まれるデータ・ストア別名を選択します。「**次へ**」をクリックします。
4. 「開始表の選択」ページで、以下のステップを実行します。
   a. 「**開始表**」フィールドで、検索パターン *SCHEMA*.OPTIM_CUSTOMERS を入力します。 *SCHEMA* は Optim サンプル・データが含まれるスキーマです。例えば SAMPLE.OPTIM_CUSTOMERS のようにします。
   b. 「**表の検索**」をクリックします。 パターンと一致する表が表リストに表示されます。
   c. OPTIM_CUSTOMERS 表を選択します。
   d. 「**次へ**」をクリックします。
5. 「表選択メソッドの指定」ページで、「**関連表の検索**」をクリックします。「**次へ**」をクリックします。
6. 「表のパターンの入力と関連表の選択」ページで、以下のステップを実行します。
   a. 「**表の検索**」をクリックします。 OPTIM_CUSTOMERS に関連した表が表リストに表示されます。
   b. 「**すべて選択**」をクリックします。
   c. 「**完了**」をクリックします。

Access Definition Editor に新しいアクセス定義が表示されます。

選択基準の定義

この演習では、アクセス定義内で選択基準を定義します。 SQL WHERE 節を定義してデフォルト値で置換変数を使用することで、選択基準を使用して特定の関連するデータ集合にフォーカスすることができます。

選択基準は、SQL 構文に準拠していること、および関係演算子または論理演算子を含むことが必要です。論理演算子および構文は、DBMS によって異なります。詳しくは、該当する DBMS 資料を参照してください。

表の必要なデータ集合を選択するには、論理演算子 AND および OR の組み合わせが必要な場合があります。

選択基準を定義するには、次のようにします。

1. リポジトリー・エクスプローラーでチュートリアルのフォルダーを展開し、「**アクセス定義**」ノードを展開して、SAMPLE.AD アクセス定義をダブルクリックします。 Access Definition Editor が開きます。
2. 「**表**」 タブを選択します。
3. 表リストから OPTIM_CUSTOMERS 表を選択します。

4. 「**選択基準の編集**」をクリックします。 「選択基準の編集」ウィンドウが開き、表の WHERE 節が表示されます。
5. 基準 COUNTRY_CODE='US' を入力してください。

   「**構文チェック**」をクリックして、構文エラーおよび確認エラーを特定します。
6. 「**OK**」をクリックして Access Definition Editor に戻ります。

   構文が無効である場合は、エラーを確認するプロンプトが開きます。エラーが含まれている基準は保存できません。
7. メインメニューで、「**ファイル**」 > 「**保存**」をクリックして、アクセス定義を保存します。

これで、OPTIM_CUSTOMERS 表から、COUNTRY_CODE 列の値が 'US' である行のみを選択する選択基準が定義されました。

抽出サービスの作成

この演習では、アクセス定義で指定されているデータに基づく抽出サービスを作成します。抽出サービスを使用して、1 つ以上の表から関連する行のセットをコピーして、ファイル・データ・ストアに行を保存します。

抽出サービスを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーでチュートリアルのフォルダーを展開し、「**サービス**」を右クリックして「**新規サービス**」をクリックします。 「新規サービス」ウィザードが開きます。
2. 「サービス名の入力とサービスの選択」ページで、以下の手順を実行します。
   a. 「**名前**」フィールドで、SAMPLE.EXTRACT と入力します。
   b. サービス・タイプのリストで、「**抽出**」を選択します。
   c. 「**次へ**」をクリックします。
3. 「アクセス定義の選択」ページで、SAMPLE.AD を選択します。「**次へ**」をクリックします。
4. 「ターゲット・ファイル・データ・ストア名の入力」ページで、「**ターゲット・ファイル・データ・ストア名**」 フィールドに、 SAMPLE_EXT と入力します。
5. 「**完了**」をクリックします。

Extract Service Editor に新しい抽出サービスが表示されます。このサービスを使用して、変換、挿入、またはロード・サービス用のファイル・データ・ストアを作成できます。このファイル・データ・ストアを使用して、列マップや表マップを作成することもできます。

Optim Designer を使用したデータのマスク

このチュートリアルでは、Optim Designer を使用して、ファイル・データ・ストア内のデータを変換する変換サービスを作成する方法について学習します。

このチュートリアルでは、 123 ページの『Optim Designer を使用したデータの抽出』のチュートリアルで作成した SAMPLE_EXT ファイル・データ・ストアが必要です。

このチュートリアルを完了すると、プライバシー関数を適用し、変換サービスを実行できるようになります。

学習目標

演習を完了すると、以下のタスクの実行方法を習得できます。

- リポジトリー・エクスプローラーのフォルダーを作成する
- アクセス定義を作成して、抽出するデータを定義する
- アクセス定義を編集して、選択基準を定義する
- 抽出サービスを作成する
- 抽出サービスをテストする

所要時間

このモジュールを完了するには、約 60 分かかります。

前提条件

このチュートリアルには、Optim リポジトリーへの接続と、Optim サンプル・データを含むデータ・ストア別名が組み込まれている、Optim Designer 環境が必要です。

このチュートリアルは Optim Designer 環境で実行できます。

表マップの作成

この演習では、表マップを作成します。表マップは、互換性のあるデータのソース表と宛先表を関連させるための仕様を定義します。名前が異なる表をマップし、表の名前を変更し、処理から表を除外し、または列マップを含めることで、データに対するコントロールを向上させることができます。

表マップを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーでチュートリアルのフォルダーを展開し、「**表マップ**」を右クリックして「**新規表マップ**」をクリックします。 「新規表マップ」ウィザードが開きます。
2. 「表マップ名の入力」ページで、「**名前**」フィールドに SAMPLE.TMAP と入力します。 「**次へ**」をクリックします。
3. 「データ・ストア別名の選択」ページで、SAMPLE.EXT を選択します。
4. 「**完了**」をクリックします。

Table Map Editor に新しい表マップが表示されます。

表マップを保存する前に、エディターを使用してターゲット・データ・ストアおよびスキーマを定義する必要があります。

表マップの編集

この演習では、Table Map Editor を使用して表マップ内のソース表のターゲット・データ・ストアおよびスキーマを定義します。

表マップ内のターゲット・データを編集するには、以下のようにします。

1. リポジトリー・エクスプローラーでチュートリアルのフォルダーを展開し、「**表マップ**」ノードを展開して、SAMPLE.TMAP 表マップをダブルクリックします。 Table Map Editor が開きます。
2. 「**表マップ**」タブを選択します。
3. 「**データ・ストア別名とスキーマのマップ**」領域で、ソース・データ・ストアが含まれている行を選択します。

4. 「**ターゲット・データ・ストア別名**」セルをクリックし、Optim サンプル・データが含まれるデータ・ストア別名を選択します。
5. 「**ターゲット・スキーマ**」セルをクリックし、Optim サンプル・データが含まれるスキーマを選択します。
6. メインメニューで、「**ファイル**」 > 「**保存**」をクリックして、表マップを保存します。

表マップは、ソース・データをマスクし、同じスキーマとデータ・ストア別名を維持するために、同じソース表とターゲット表を使用します。

列マップの作成

この演習では、列マップを作成します。列マップにより、データ管理サービスでの処理について列を一致させるまたは除外するために必要な仕様が得られます。変換、挿入、およびロード・サービスは、1 つ以上の列マップを参照する表マップを参照する必要があります。列マップを使用すると、関数または列マップ・プロシージャーによってデータ変換を定義することができます。

列マップを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーでチュートリアルのフォルダーを展開し、「**列マップ**」を右クリックして「**新規列マップ**」をクリックします。 「新規列マップ」ウィザードが開きます。
2. 「列マップ名の入力」ページで、「名前」フィールドに SAMPLE.CMAP と入力します。「**次へ**」をクリックします。
3. 「ソース・ファイル・データ・ストア別名の選択」ページで、SAMPLE_EXT を選択します。「**次へ**」をクリックします。
4. 「ソース表の選択」ページで、以下のステップを実行します。
   a. 「**表の検索パターン**」フィールドで、検索パターン *DATA_STORE_ALIAS.SCHEMA*.OPTIM_CUSTOMERS を入力します。 *DATA_STORE_ALIAS.SCHEMA* は Optim サンプル・データが含まれるデータ・ストア別名およびスキーマです。例えば OPTIM.SAMPLE.OPTIM_CUSTOMERS のようにします。
   b. 「**表の検索**」をクリックします。 パターンと一致する表が表リストに表示されます。
   c. OPTIM_CUSTOMERS 表を選択します。
   d. 「**次へ**」をクリックします。
5. 「ターゲット・データ・ストア別名の選択」ページで、Optim サンプル・データが含まれるデータ・ストア別名を選択します。「**次へ**」をクリックします。
6. 「ターゲット表の選択」ページで、以下のステップを実行します。
   a. 「**表の検索パターン**」フィールドで、検索パターン *SCHEMA*.OPTIM_CUSTOMERS を入力します。 *SCHEMA* は Optim サンプル・データが含まれるスキーマです。例えば SAMPLE.OPTIM_CUSTOMERS のようにします。
   b. 「**表の検索**」をクリックします。 パターンと一致する表が表リストに表示されます。
   c. OPTIM_CUSTOMERS 表を選択します。
7. 「**完了**」をクリックします。

Column Map Editor に新しい列マップが表示されます。

データ・マスキング関数の適用

この演習では、データ・マスキング関数を列マップ内の列に適用します。国民 ID 番号、クレジット・カード番号、日付、数値、個人情報などをマスクすることができます。

データ・マスキング関数を適用するには、次のようにします。

1. リポジトリー・エクスプローラーでチュートリアルのフォルダーを展開し、「**列マップ**」ノードを展開して、SAMPLE.CMAP 列マップをダブルクリックします。 Column Map Editor が開きます。
2. PHONE_NUMBER 列を選択します。
3. 「**関数の適用**」をクリックします。 「関数の適用」ウィンドウが開きます。
4. 「**Shuffle 関数**」を選択します。「**OK**」をクリックします。 「**ソース列**」列に関数名が表示され、Column Map Editor で関数エディターが開きます。
5. 関数エディターの 「**列マップ式**」フィールドで、 `SHUFFLE(RETRY=12)` と入力します。
6. メインメニューで、「**ファイル**」 > 「**保存**」をクリックして、列マップを保存します。

これで、Shuffle 関数が PHONE_NUMBER 列に適用されました。この関数は、この列の値を、同じ列の他の値に置き換えて、データをマスクします。この関数は、置換値がソース値と一致しない場合は最大 12 回検索します。

表マップへの列マップの追加

この演習では、表マップに列マップを追加します。表マップは、変換サービスに必要です。関連付けられている列マップを使用すると、変換サービスがその列マップで定義されているデータ・マスキング関数を実行できるようになります。

表マップに列マップを追加するには、次のようにします。

1. リポジトリー・エクスプローラーでチュートリアルのフォルダーを展開し、「**表マップ**」ノードを展開して、SAMPLE.TMAP 表マップをダブルクリックします。 Table Map Editor が開きます。
2. 「**表マップ**」タブを選択します。
3. 「**表マップ**」領域で、OPTIM_CUSTOMERS 表を選択します。
4. 「**列マップの追加**」をクリックします。 「新規列マップ」ウィンドウが開き、選択した表が含まれている列マップのリストが表示されます。
5. SAMPLE.CMAP 列マップを選択します。「**OK**」をクリックします。
6. メインメニューで、「**ファイル**」 > 「**保存**」をクリックして、表マップを保存します。

これで、SAMPLE.CMAP 列マップ (および関連付けられたデータ・マスキング関数) が SAMPLE.TMAP 表マップに追加されました。変換サービスがデータを変換できるようになります。

変換サービスの作成

この演習では、データをマスクする変換サービスを作成します。

変換サービスを作成するには、以下のようにします。

1. リポジトリー・エクスプローラーでチュートリアルのフォルダーを展開し、「**サービス**」を右クリックして「**新規サービス**」をクリックします。 「新規サービス」ウィザードが開きます。
2. 「サービス名の入力とサービスの選択」ページで、以下の手順を実行します。
   a. 「**名前**」フィールドで、`SAMPLE.CONVERT` と入力します。
   b. サービス・タイプのリストで、「**変換**」を選択します。
   c. 「**次へ**」をクリックします。
3. 「表マップの選択」ページで、SAMPLE.TMAP 表マップを選択します。「**次へ**」をクリックします。

4. 「ターゲット・プロパティーの入力」ページで、「**ターゲット・ファイル・データ・ストア名**」フィールドに、SAMPLE_CONV と入力します。
5. 「**完了**」をクリックします。

Convert Service Editor に新しい変換サービスが表示されます。このサービスを使用して、OPTIM_CUSTOMERS 表の PHONE_NUMBER 列をマスクできます。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。 本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。 これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。 本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。 それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様自身の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。 そのため、他の操作環境で得られた結果は、異なる可能性があります。 一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。 さらに、一部の測定値が、推定値である可能性があります。 実際の結果は、異なる可能性があります。 お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。 卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。 記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。 より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

以下は、International Business Machines Corporation の米国およびその他の国における商標です。

IBM
IBM ロゴ
DB2
AIX
Informix
InfoSphere
Optim

Adobe、Acrobat、PostScript、およびすべての Adobe 関連の商標は、Adobe Systems Incorporated の米国およびその他の国における商標または登録商標です。

Apache Derby は、The Apache Software Foundation の商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java™ およびすべての Java 関連の商標は、Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ハ行]

[ヤ行]

[ラ行]

A

B

C

D

E

H

I

L

O

P

R

S

T

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21