Magic eDeveloper V10
タスク基本構造
（ヘッダ明細入力）

Magic Software Japan K.K.

Enabling Business with Superior Technology

初版　2008 年 7 月 7 日

マジックソフトウェア・ジャパン株式会社

目次

1 はじめに

本書の目的

本書は、ヘッダ・明細の階層的データ構造を扱うプログラムを、異なった手法を使って実装したサンプルの解説です。

データベースを使った業務アプリケーションには、1:N（いわゆるヘッダ・明細型）の階層構造を持ったデータを扱うものが非常に多くあります。受注、発注、入庫、出庫、経費、その他の伝票類はほとんどがこのデータ構造を持っています。

Magic では、このようなデータ構造を便利に扱う機能が豊富に用意されているので、業務アプリケーションを開発・保守するのが非常に容易になっていますが、Magic を使ってヘッダ・明細型のプログラムを作成する方法はひとつだけではなく、多くのバリエーションが可能です。そのため、Magic の理解を深めようとすると、さまざまある実装方法について、その違いと長所短所をきちんと理解し整理しておく必要があります。

そこで、本書は次のことを狙いとしました。

- ヘッダ・明細型の入力という、同一のことを実現するための異なる実装方法を比較検討することにより、それぞれの長所短所を把握する。
- 実際に作ろうとするアプリケーションでの実装方式を選択する指針とする。
- プログラム パターンの標準化のための参考資料とする。
- Magic でできる広がりの可能性への理解を深める。

本書の構成

本書は次のような構成になっています。

最初に、第 2 章「サンプルアプリケーションの設定」、第 3 章「ペットショップデモ受注入力画面の概要」、第 4 章「命名規則」で、本書で使うサンプルアプリケーションの設定や機能概要、命名規則の説明をします。

第 5 章「実装方法のいろいろ」では、本書で説明するさまざまな実装方式を、アルゴリズムの違い、トランザクション設定の違い、タスクタイプの違いにより分類します。

それぞれの実装方式の詳細については、第 6 章「ヘッダ・明細型プログラムの基本形」から第 11 章「リッチクライアント」の各章で説明しています。

その中で、第 6 章「ヘッダ・明細型プログラムの基本形」では、必要最小限の機能を満たす「基本形」について解説します。これは、物理トランザクションを使ったオンラインプログラムで、チュートリアル Getting Started で作成したものに少し追加をしたものです。昔ながらのペットショップデモでの受注入力プログラムと基本的に同じロジックを使っています。この章では、プログラム中で使われている Magic の持つ個々の基本機能についての説明もしています。

第 7 章「ONL/物理/バッチ更新 」から第 11 章「リッチクライアント」では、この基本形を移植していく形で機能追加を行い、実装方法を変えた応用形を紹介します。

第 12 章「実装方法の選択」では、数ある実装方法の中から、自分のアプリケーションの要求仕様にあった方式を選択していくための指針を説明しています。

最後に、第 13 章「リッチクライアントとオンラインとの違い」では、付録として、リッチクライアントタスクとオンラインタスクの違い（主に制限事項）についてまとめました。

前提知識

本書の読者は、Magic eDeveloper V10 の基本的な操作・設定等についてすでによく知っていることを前提にしています。また、SQL データベースを使った Magic システム開発についても理解していることを前提にしています。
これらの前提知識については、以下の書籍が参考になります。いずれも、弊社ホームページの Magic スキルアップセンター http://www.magicsoftware.co.jp/training/introduction/introduction.html よりダウンロードすることができます。

書籍名	内容
Getting Started V10	Magic eDeveloper V10 を始めて利用される方を対象に、スタンドアロンのオンラインアプリケーションをステップバイステップで作りながら Magic の基本を学んでいきます。Magic の初歩から、タスクの動作、フォームの設計、データソースの定義、イベント指向エンジン、1 対 1 リレーション、1 対多リレーション、バッチタスク、帳票印刷、メニュー作成までを学びます。
Magic eDeveloper V10 チュートリアル SQL 編	SQL データベースを使って Magic アプリケーションを作成するための基本事項を勉強します。SQL データベースとしては SQL Server 2005 を使い、インストール、Magic のデータベースの設定、データソースリポジトリの扱い、Pervasive からの移行、ロックとトランザクション、一時ファイルを使ったプログラミング手法などについて学びます。

第 10 章「オンライン・遅延トランザクション」では、遅延トランザクションが出てきますので、遅延トランザクションについての理解も必要です。遅延トランザクションについては、次の書籍を参考にしてください。

書籍名	内容
Magic eDeveloper V10 遅延トランザクション	本書は、Magic eDeveloper V10 の独自のデータ管理機能である「遅延トランザクション」の基礎を学ぶことを目的としています。遅延トランザクションの基本概念、排他制御機能、トランザクションのネスト、プログラミング上の考慮点などについて説明します。

第 11 章「リッチクライアント」では、V10.1SP4b の新機能であるリッチクライアントによる実装方法を解説していますので、リッチクライアントについての理解が必要です。Magic Studio V10 製品に添付の次の書籍を参考にしてください。

書籍名	内容
インタラクティブなリッチクライアントの開発と実行	インタラクティブな Web アプリケーションの開発と実行のためのリッチクライアント技術に関する概要を説明したものです。

また、本書の全般にわたって、Magic スキルアップセンターにある、次のサンプルアプリケーションも参考になります。

書籍名	内容
Magic eDeveloper V10 コーディングサンプル	より本格的なアプリケーションに近い Magic アプリケーションのコーディングサンプルです。Getting Started V10、Magic eDeveloper V10 チュートリアル SQL 編を終了し、より上級の Magic 開発者となることを目指している方を対象にしています。
Magic eDeveloper V10 コーディングサンプル（Ver2）	Magic eDeveloper V10 の持つ機能を生かした「Magic らしい」アプリケーションのサンプルをシリーズで紹介するものです。「Magic eDeveloper V10 コーディングサンプル1、受注入力デモ MS-SQL マルチユーザ対応版」を改良し、コンポーネント、モデル、イベント指向プログラミングを徹底的に活用しています。

2 サンプルアプリケーションの設定

本書の内容をより具体的に理解していただくために、本書にはサンプルアプリケーションが提供されています。本章では、サンプルアプリケーションを、Magic Studio で実際に使えるようにするための手順を説明します。

サンプルアプリケーションを利用するには、以下のものが必要ですので、用意しておいてください。

- Magic Studio V10 (Ver. 10.1SP4b 以降) 製品版、あるいは体験版
- Microsoft SQL Server 2005 および Microsoft Management Studio
- Java RTE (1.5 以降) (第 11 章「リッチクライアント」のサンプルを実行する場合に必要)

以下に、それぞれについて説明します。

2.1 Magic StudioV10

サンプルアプリケーションは、Magic Studio Ver. 10.1SP4b で作成されていますので、サンプルを実行させるには 10.1SP4b あるいはそれ以降の Magic Studio 製品が必要です。

2.1.1 Magic Studio V10 について

Magic Studio V10 については、下記の弊社ホームページ「Magic eDeveloper V10 の製品概要」(http://www.magicsoftware.co.jp/training/introduction/introduction.html)をご参照ください。

Magic Studio V10 がサポートしているオペレーティングシステムは、以下のものがあります。

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista

より詳細なシステム要件については、弊社ホームページ「Magic eDeveloper V10 動作環境、サポート DBMS、OS 一覧」(http://www.magicsoftware.co.jp/products/mgenv/dbms10.html) に最新情報が掲載されていますので、参照してください。

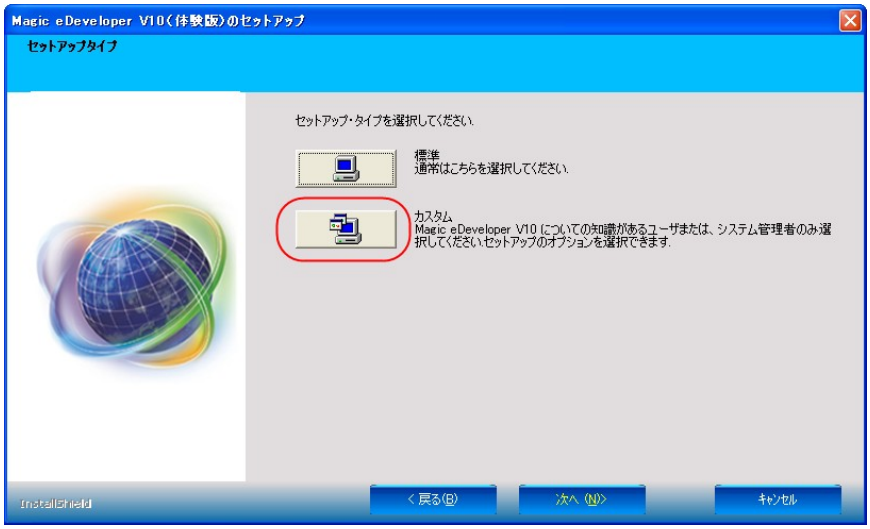
Magic Studio 製品をお持ちでない場合には、Magic Studio V10 体験版（無償、日付制限 60 日）も提供されていますので、そちらをご利用ください。体験版は、上記ホームページ「Magic eDeveloper V10 の製品概要」から申し込むことができます。

本書の説明や図では、Magic Studio V10 体験版を使っています。

2.1.2　インストール時の注意事項

Magic をインストールする際には、「カスタム」インストールで、MS SQL Server ゲートウェイ を選択してください。本書のサンプルでは MS SQL Server を使うので、MS SQL Server 用のゲートウェイ がインストールされている必要があります。

インストーラの 5 番目の画面で、セットアップタイプとして、「標準」か「カスタム」かを聞いてきますので、「カスタム」のボタンを押してください。

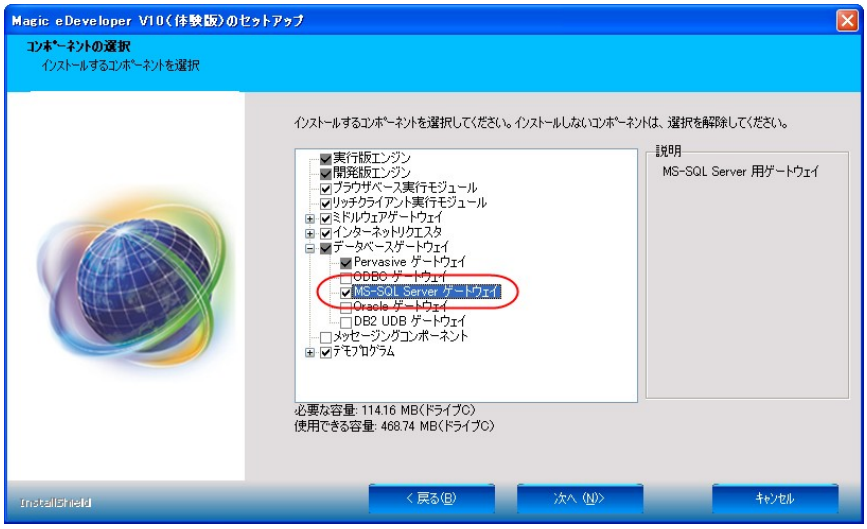


その 2 画面先で、「コンポーネントの選択」画面が出ますので、ここで

データベースゲートウェイ

⇒ MS-SQL Server ｹﾞｰﾄｳｪｲ

を選択してください。



その他については、デフォルトのままの設定で構いません。

2.2 Microsoft SQL Server 2005

2.2.1 Microsoft SQL Server 2005

サンプルアプリケーションでは、DBMS として、Microsoft SQL Server 2005 を使っています。読者の PC 環境で SQL Server 2005 を利用できない場合には、Magic Studio 製品のボーナス CD に Express Edition がバンドルされていますので、インストールしてご利用ください。

Magic Studio 製品のボーナス CD をお持ちでない場合には、Microsoft 社のホームページより、Express Edition をダウンロードできます。
(http://www.microsoft.com/japan/msdn/sqlserver/)

- 本書では、Microsoft SQL Server 2005 Express Edition を、デフォルトの設定のままインストールして利用しています。この場合、設定は次のようになります。

主な設定値	以下の説明での設定値
インスタンス名	SQLEXPRESS
認証モード	Windows 認証モード
リモート接続	ローカル接続のみを許可
有効なプロトコル	共有メモリのみ

- これ以外の設定にしたい場合には、インストールの途中「登録情報」画面で「詳細構成オプションを非表示にする」のチェックをはずして、詳細パラメータ入力ができるようにしてください。
- また、接続に関する設定は、インストール後、SQL Server 2005 セキュリティ構成 ユーティリティで変更できます。

2.2.2 Microsoft Management Studio

Microsoft SQL Server 2005 には管理ツールとして Microsoft Management Studio があります。データベースの操作をする場合に必要となりますので、SQL Server と共にインストールしてください。

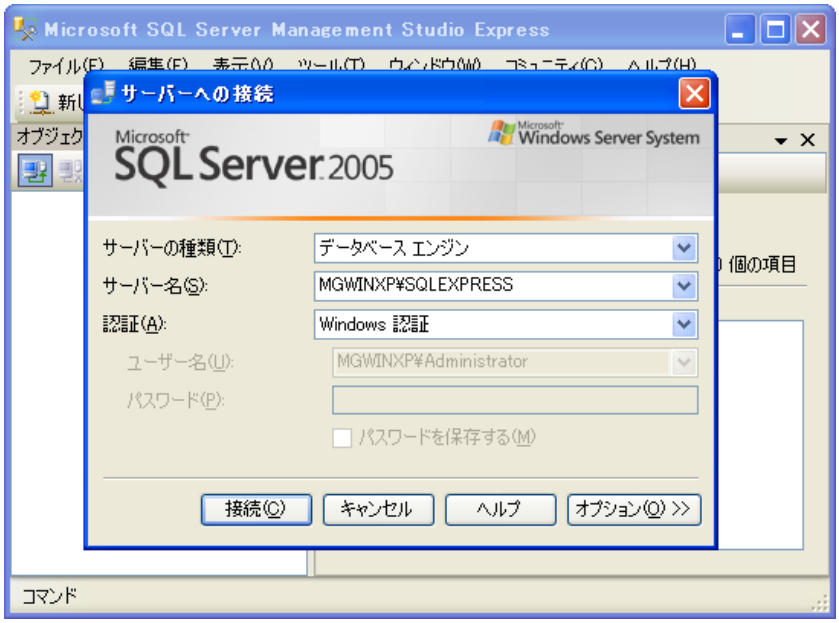
Management Studio も、Magic Studio 製品のボーナス CD に Express Edition がバンドルされています。また、上記の Microsoft のホームページからも無償で Express Edition をダウンロードすることができます。

2.2.3 データベースの作成

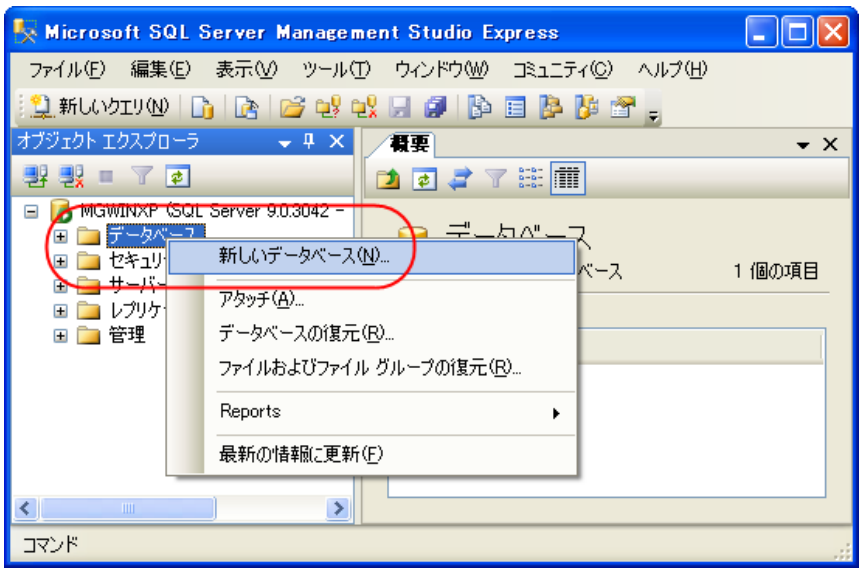
SQL Server 2005 および Management Studio をインストールしたら、Management Studio を使って、サンプルデータベース用のデータベースを作成しておいてください。ここでは、MAGIC という名前のデータベースを作成します。

MAGIC というデータベースを作成するには・・・

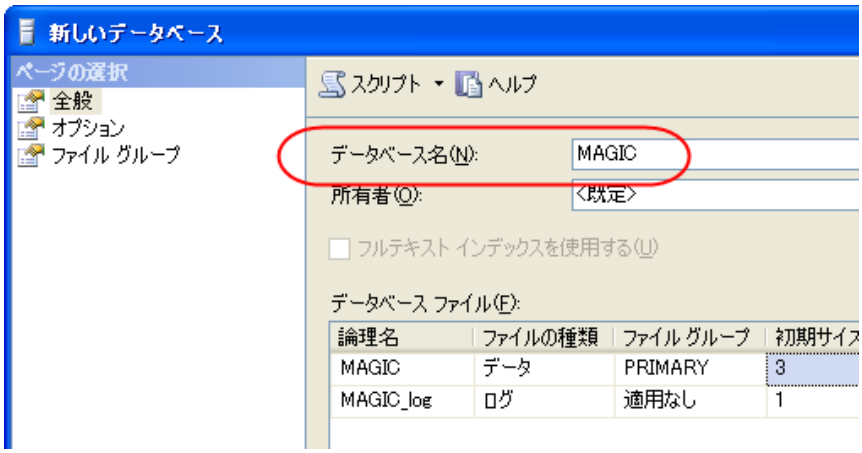
1. Microsoft SQL Server Management Studio (Express) を起動し、サーバに接続します。



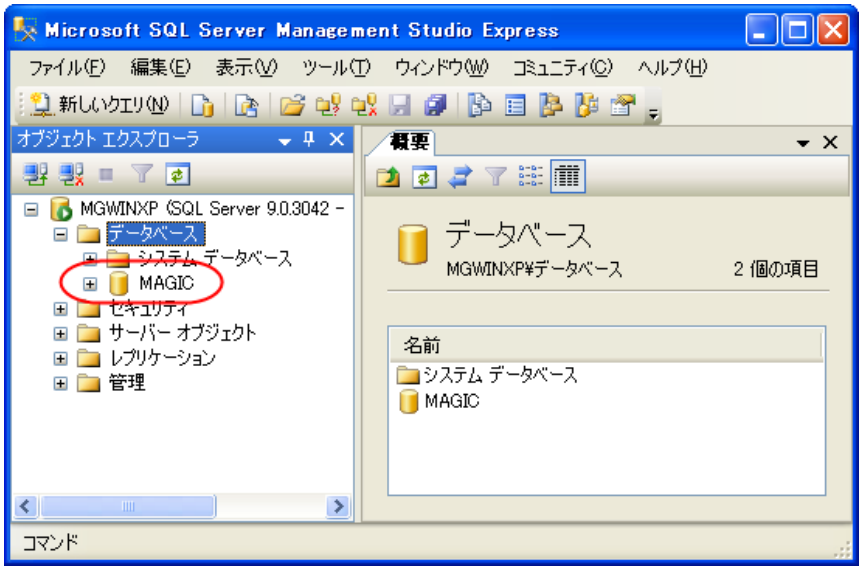
2. 「データベース」ノードのポップアップメニューで「新しいデータベース」を選択してください。



3. 「データベース名」として、「MAGIC」として、OK ボタンを押してください。



4. 「MAGIC」というデータベースが作成されたことを確認してください。



2.3 Java RTE

サンプルのうち、リッチクライアントプログラム(第 11 章「リッチクライアント」参照)を実行するためには、Java RTE がインストールされている必要があります。

Magic Studio の製品 CD からインストールした場合、インストーラが、PC 環境に Java がインストールされているかをチェックします。もし Java がインストールされていないようであれば、Java RTE をインストールするかを聞いてきます。Yes で答えれば、 製品 CD に同梱されている Java RTE (1.5 Update4)を同時にインストールします。

もし Sun Microsystems から提供されている、最新の Java RTE をインストールしたい場合には、ここで No と答えて、Magic のインストールを続けます。Magic のインストールが終わったら、Sun Microsystems の Java のホームページ (http://java.sun.com/)から、最新の Java RTE をダウンロードして、インストールしてください。

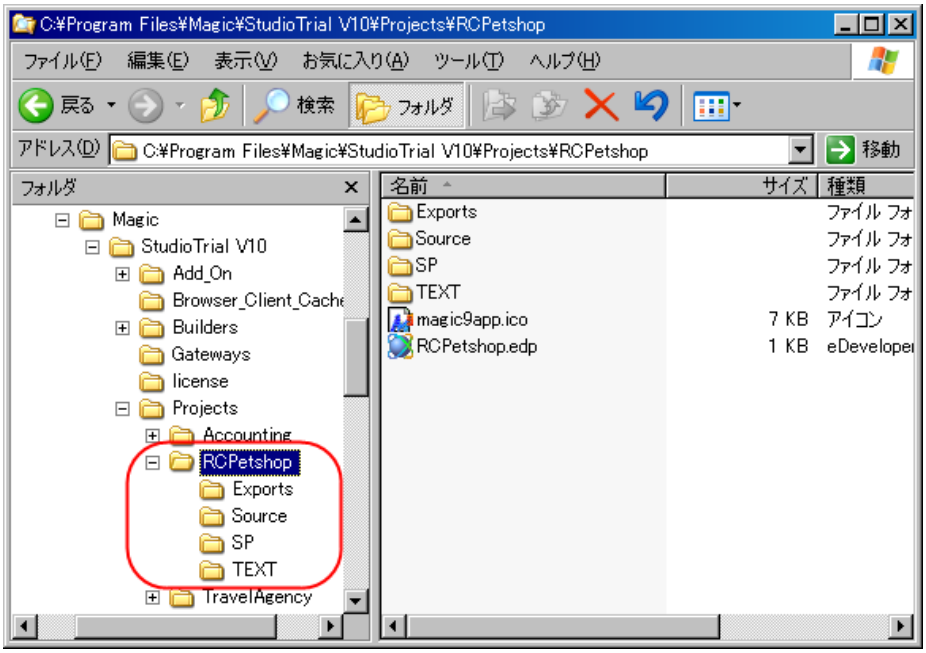
Magic 体験版を利用する場合には、体験版には Java が同梱されていないのでインストールされません。別途 Sun Microsystems からダウンロードしてインストールする必要があります。

2.4 ZIP ファイルの展開

必要なソフトウェアをインストールしたら、サンプルアプリケーションを利用するため、Magic Studio の環境設定を行います。

まずは、サンプルアプリケーションは、ZIP 形式で提供されているので、Magic をインストールしたフォルダの下にある Projects サブフォルダの下に解凍してください。

下図は、体験版をデフォルトのディレクトリにインストールした場合に、サンプルを解凍したときのイメージです。



2.5 プロジェクトの再構築（オプション）

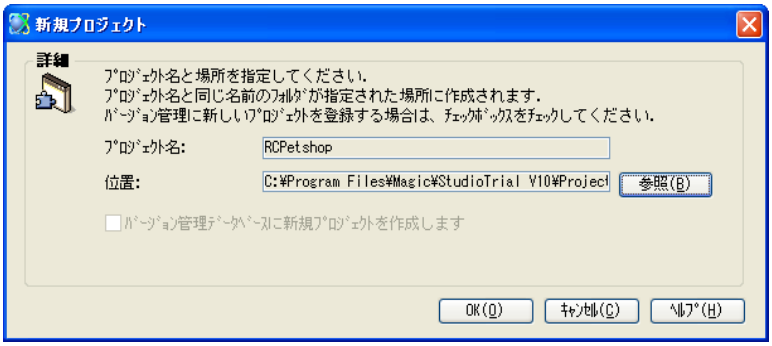
サンプルの ZIP ファイルを展開すると、SP4 で作成したプロジェクトファイルがすでにそこに展開されていますので、そのまま、RCPetshop.edp ファイルをダブルクリックして、Magic Studio でプロジェクトを開くことができます。

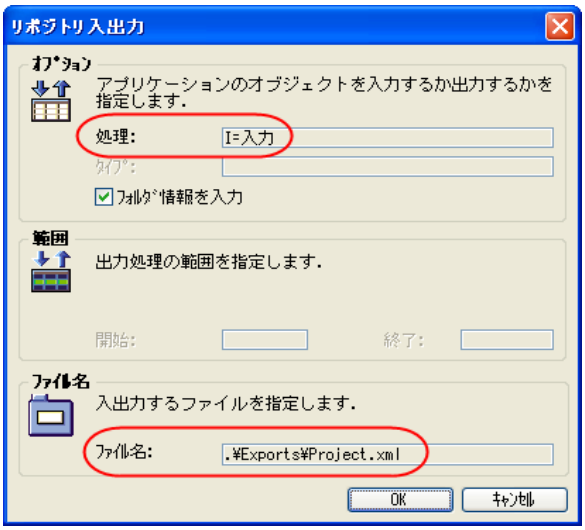
プロジェクトファイルを再構築（リポジトリ入力）することは必要ではありませんが、プロジェクトを初期状態に戻したい場合や、Magic Studio の互換性がない場合には、プロジェクトの再構築を行います。

以下の手順で、プロジェクトを再構築してください。

プロジェクトを再構築するには・・・

1. RCPetshop.edp ファイル、および Source ディレクトリ以下のファイルを削除します。
2. Magic Studio を起動し、メニュー「ファイル⇒ 新規作成(N)」で新規プロジェクトダイアログを開きます。
3. プロジェクト名として、「RCPetshop」と指定して、OK ボタンを押します。
4. メニュー「ファイル ⇒ リポジトリ入出力」を選び、「処理」は「I=入力」、「ファイル名」は「.¥Exports¥Project.xml」と指定します。
5. OK ボタンを押すと、インポートします。





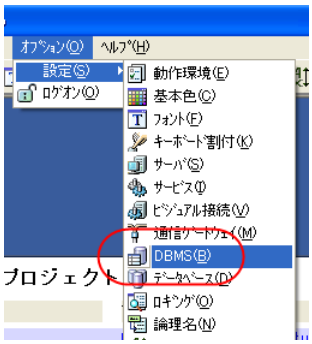
2.6 DBMS テーブルの設定

Magic から MS-SQL Server を扱うために、Magic で **DBMS テーブル**と、**データベーステーブル**の二つの設定を行う必要があります。DBMS テーブルは、DBMS の種類（Pervasive、MS-SQL Server、Oracle など）毎に、共通の設定を行います。一方、データベーステーブルは、個々のデータベースのための設定を行います。

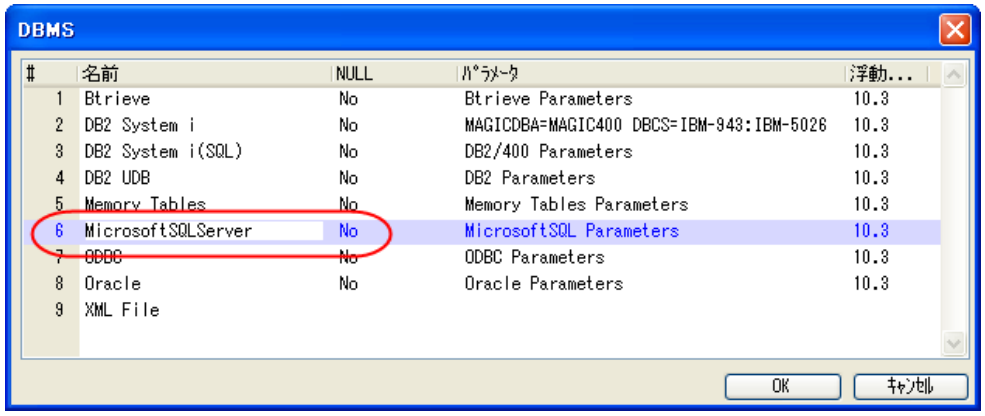
ここではまず、DBMS テーブルの設定を行います。データベーステーブルの設定は、次節で説明します。

DBMS テーブルを設定するには・・・

1. プロジェクトが開いていたら、プロジェクトを閉じます。
2. メニュー「オプション(O) ⇒ 設定（S) ⇒ DBMS(B)」を選んで、DBMS テーブルを開きます。

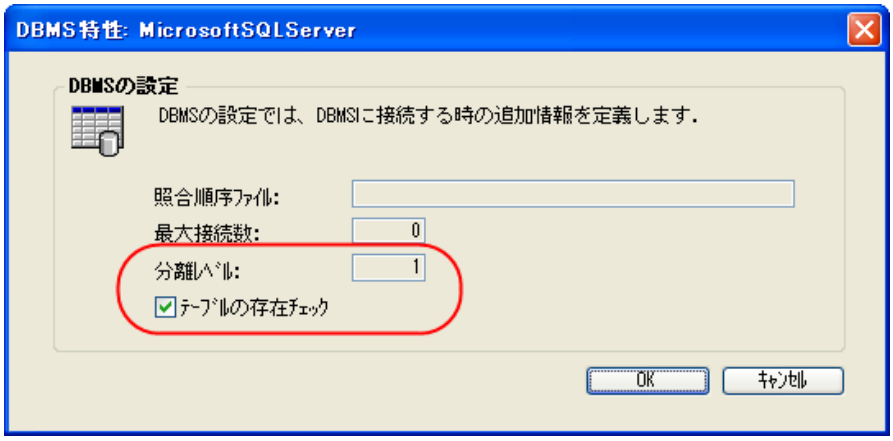


3. MicrosoftSQLServer の行にカーソルを置きます。



4. ポップアップメニューから「特性(P)」を選びます（あるいは、Alt+Enter キーを押します）。DBMS 特性ダイアログが開きます。
5. 「分離レベル」は「1」にしてください。これは、トランザクションの分離レベルを設定するもので、MS-SQL Server の場合、「1」は「READ COMMITTED」を意味します。デフォルトは「0」で、これは「READ UNCOMMITTED」ですが、この設定の場合は排他制御が非常に弱く、ダーティリードなども起こるので、適当ではありません。
6. 「テーブルの存在チェック」はオンにしてください。

最終的には、右図のような設定になります。



DBMS テーブルについての詳しい情報は、リファレンスヘルプの項目

- Magic リファレンス ⇒ 設定 ⇒ DBMS

あるいは次のキーワードを参照してください。

- 分離レベル
- テーブルの存在チェック

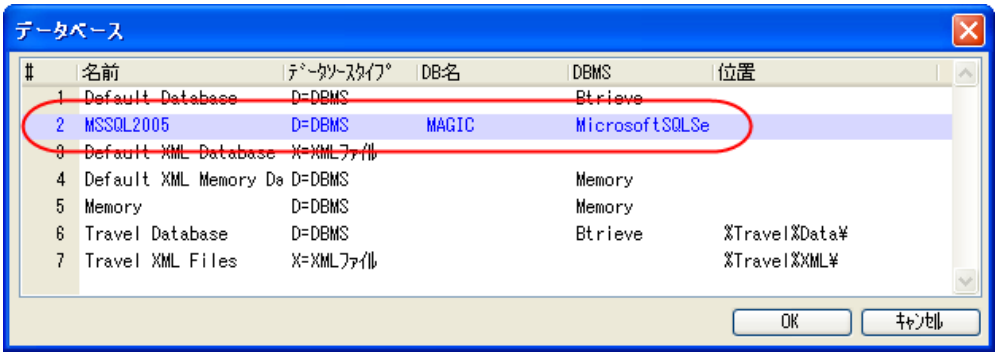
2.7 データベーステーブルの設定

次には、サンプルプロジェクトで使うデータベースの設定をします。

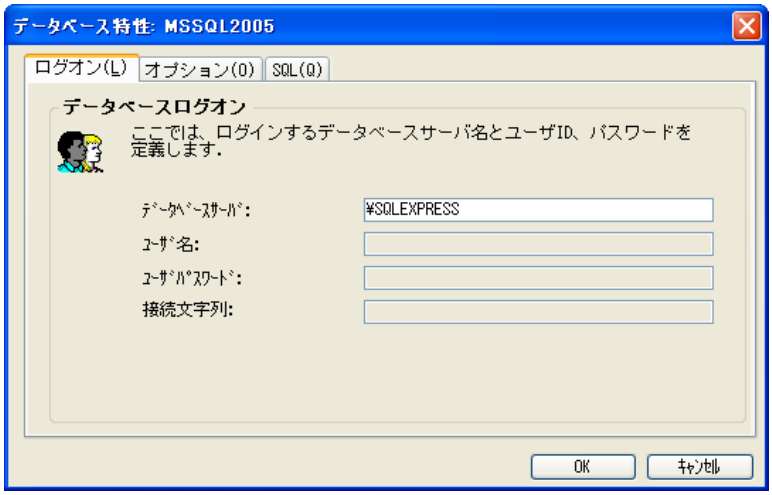
データベースを設定するには・・・

1. メニュー「オプション ⇒ 設定 ⇒ データベース」を選んで、データベーステーブルを開きます。
2. データベーステーブルで、下記のような **MSSQL2005** という名前のデータベースを新規作成します。

設定	値
名前	MSSQL2005
データソースタイプ	D=DBMS
DB 名	SQL Server に作成したデータベースの名前。(MAGIC など)
DBMS	MicrosoftSQLServer
位置	(空白)



3. ポップアップメニュー「特性(R)」を選ぶか、あるいは Alt+Enter キーを押して、データベース特性を開きます。
4. 「ログオン(L)」タブを開いて、ログインパラメータを設定します。



ローカル PC にデフォルトの設定（インスタンス名が「SQLEXPRESS」、認証が Windows 認証）で MS-SQL Server 2005 Express Edition をインストールした場合には、次の設定で接続できます。

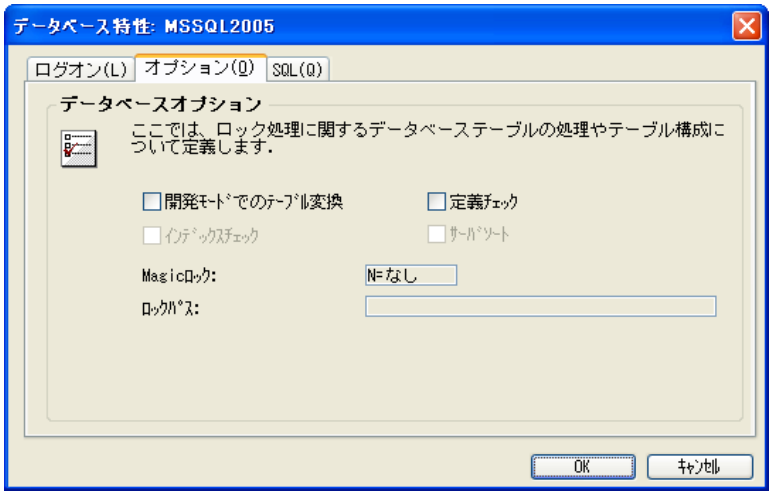
- データベースサーバ：￥SQLEXPRESS　（先頭の「￥」マークに注意）
- ユーザ名、ユーザパスワード、接続文字列：（空のまま）

MS-SQL Server の構成が異なる場合には、異なった設定をする必要があります。MS-SQL Server の場合には一般に、

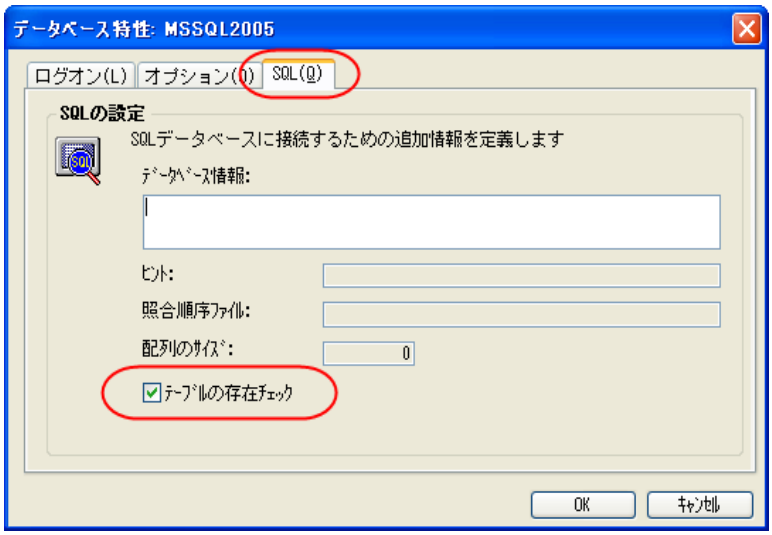
- データベースサーバ：（DBMS ホスト名）￥（インスタンス名）
- ユーザ名、ユーザパスワード：MS-SQL Server に設定したユーザ名とパスワード
- 接続文字列：（空のまま）

です。

5. 「オプション(O)」タブを開きます。
   「開発モードでのテーブル変換」、「定義チェック」などはチェックをはずし、「Magic ロック」は「N=なし」にしてください。



6. 「SQL(Q)」タブを開きます。
7. 「テーブルの存在チェック」フラグをオンにします。これは、後にサンプルデータを作成する際に必要となります。

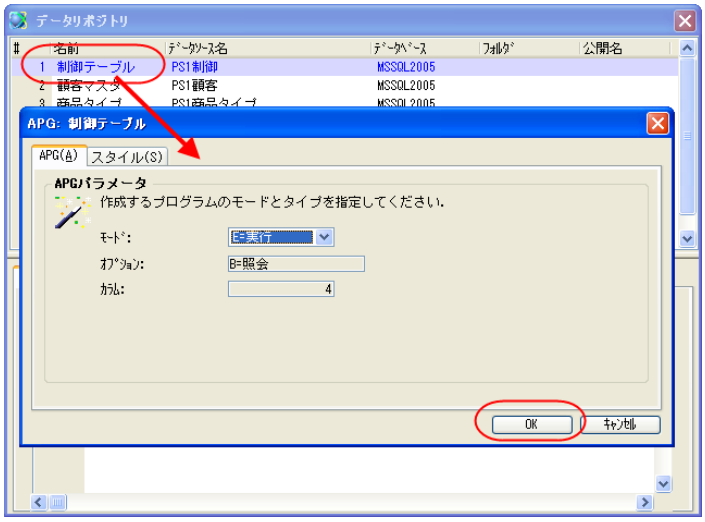


2.8 データベース設定の確認

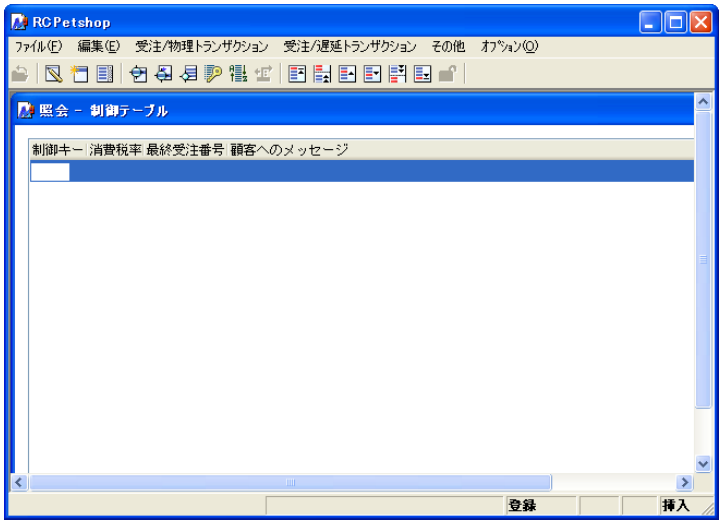
ここで、データベース関係の設定が正しく行われているかを確認します。

データベースの設定を確認するには・・・

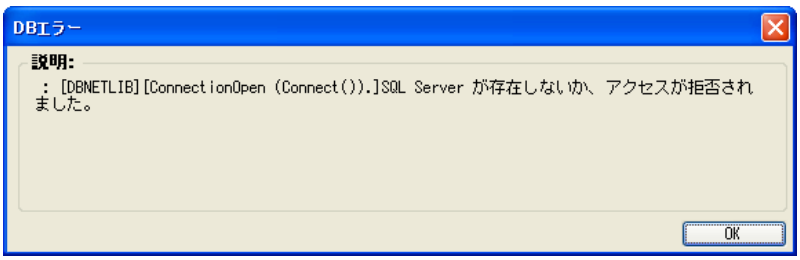
1. Magic を起動し、プロジェクトを開きます。
2. データリポジトリを開きます。
3. 先頭のテーブル「制御テーブル」にカーソルを置いて、Ctrl+G で APG を起動します。



4. データがまだないので、空ですが、登録モードでテーブルが開かれれば OK です。



データベーステーブルの設定が間違っていると、ここでエラーが出ます。データベーステーブルの設定を確認しなおしてください。

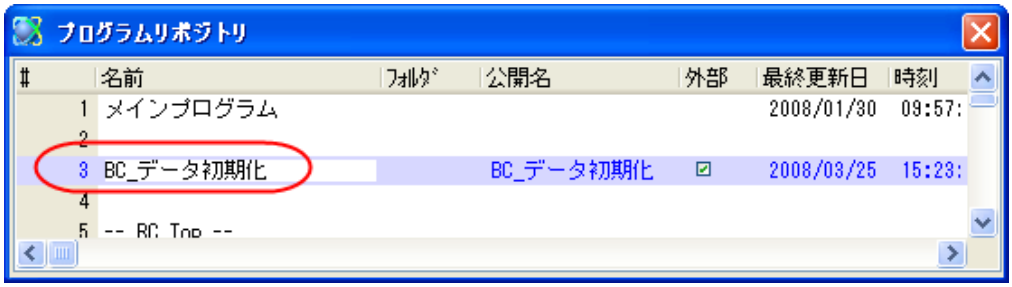


2.9 テーブルとサンプルデータの作成

データベースへの接続が確認できたら、アプリケーションで使うテーブルとサンプルデータを作成します。
テーブルとサンプルデータは、プログラム 3 番「BC_データ初期化」を実行することにより自動的に作成されます。

サンプルデータを DBMS に作成するには・・・

1. プロジェクトを開き、プログラムリポジトリを開きます。
2. プログラム 3 番「BC_データ初期化」にカーソルを合わせ、F7 で実行します。データ量は少ないので、すぐに終了するはずです。



以上で、必要なデータの初期化ができました。

アプリケーションをいろいろと操作して、データを修正した後で、初期状態にリセットしたい場合にも、この手順で行うことができます。

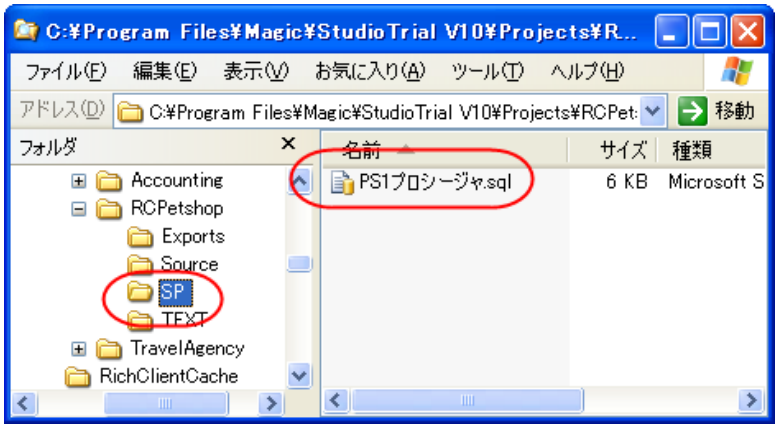
2.10 ストアドプロシージャの作成

サンプルアプリケーションのプログラムには、SQL Server のストアドプロシージャを利用するものがあります：

第 9 章　　ONL/物理/DSQL
第 10.4 節　ONL/遅延/DSQL
第 11.5 節　RC/DSQL

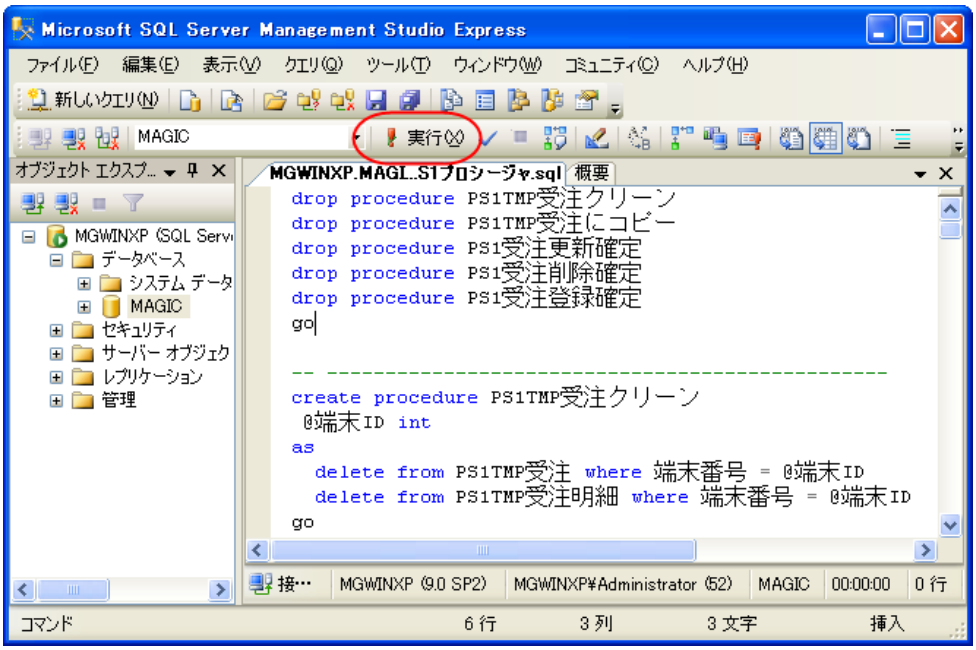
このプログラムを正しく実行させるために、ここでストアドプロシージャを作成しておきます。

ストアドプロシージャの定義は、プロジェクトのディレクトリの下の「SP」サブディレクトリに「PS1 プロシージャ.sql」という名前で格納されています（右図）。

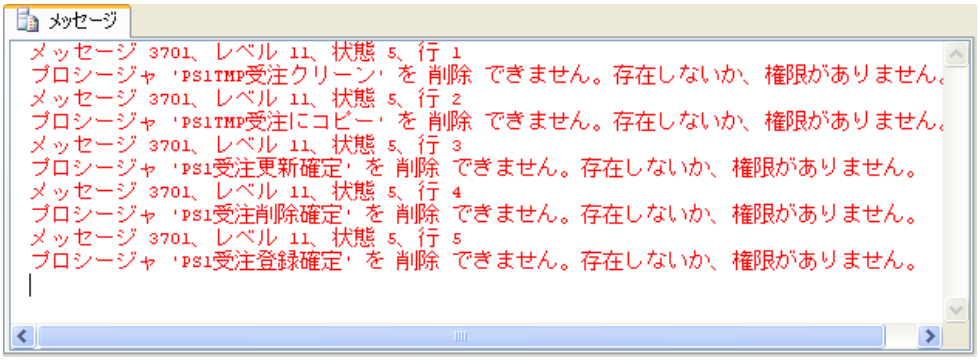


ストアドプロシージャを DBMS に作成するには・・・

1. SQL Server Management Studio を起動し、データベースに接続します。
2. 「オブジェクトエクスプローラ」でサンプルで利用しているデータベース（例えば「MAGIC」）にカーソルを置きます。
3. メニュー「ファイル ⇒ 開く ⇒ ファイル」で、上記ファイル「PS1 プロシージャ.sql」を指定します。ファイルの内容が表示されます。データベースが「MAGIC」であることを確認してください。
4. メニュー「クエリ ⇒ 実行」を選んで、実行します。



実行すると、次のようなエラーが出ますが、無視してかまいません。



これで、DBMS にストアドプロシージャが作成されました。

ここで作成されるストアドプロシージャは、前節「2.9 テーブルとサンプルデータの作成」で作成されるテーブルを参照しています。従って、本節の手順は、必ず、前節の手順を終えてから実行してください。

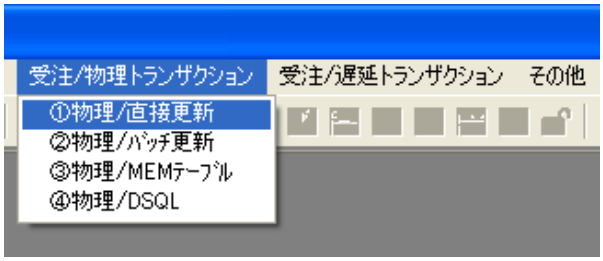
2.11 実行してみる

以上で、サンプルプログラムを実行する準備がすべて整いました。
ここではアプリケーションを実行してみて、動作を確認します。

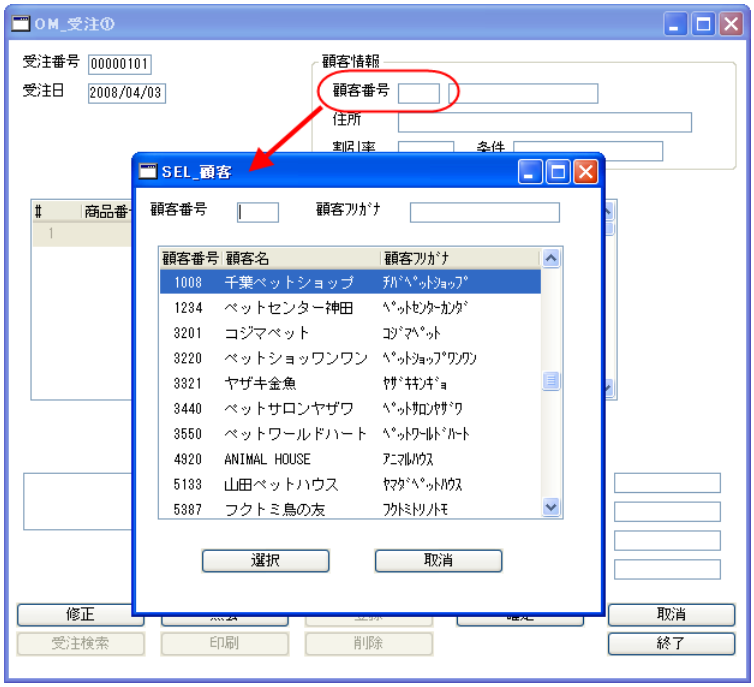
1. プロジェクトを開きます。
2. メニュー「デバッグ(D) ⇒ プロジェクトの実行(J)」を選びます。

   実行画面が現れます。

3. 実行画面のメニュー「受注/物理トランザクション ⇒ 直接更新」を選んでください。



4. 受注入力画面が登録モードで始まりますので、適当に入力して、動作を確認してください。



他のメニューも試して見てください。

正常に動作しているようであれば、OK です。

3 ペットショップデモ受注入力画面の概要

サンプルアプリケーションは、おなじみのペットショップデモをもとにしたもので、非常に簡単な受注入力プログラムです。

プログラムとしては、次のような種類のものがあります。

プログラム種類	プログラム番号
受注入力（12 種類の異なる実装方法あり）、およびその補助的バッチプログラム	34 ～ 106
選択プログラム（顧客選択、商品選択、受注選択）	12 ～ 21
印刷プログラム	23
ストアドプロシージャ呼出用のバッチ SQL タスク	25 ～ 32
データ初期化プログラム	3
端末番号管理	8 ～ 10
リッチクライアントの初期画面用プログラム	6

本書では、受注入力プログラムのいろいろな実装方法について説明・比較するのが目的なので、本章以下では、受注入力プログラムのみを解説することにして、そのほかのプログラムについては説明を省略します。

本章では、受注入力プログラムがもつ機能とデータベース設計について簡単に説明しておきます。

3.1 受注入力プログラムの仕様

3.1.1 画面構成

サンプルアプリケーションに収められている受注入力プログラムは、実装方法は異なるものの、いずれも下図のような画面構成となっています。

- 受注レコードは、スクリーン形式（1画面に1レコード）で表示されます。
- 受注明細レコードは、テーブル形式（位画面に複数レコード）で表示されます。
- 受注レコードと受注明細レコードは連動しています。

また、昔ながらのペットショップデモに比べ、いくつかの機能を持ったボタンが、画面下部に追加されています。

3.1.2 業務ルール

このプログラムは、次のような簡単な業務ルールを前提としています。

- 顧客からの注文を入力する「受注入力画面」である。
- 一人の顧客は、一回の注文で、複数の商品を注文できる。
- 注文の前に、顧客情報、商品情報は前もって登録されている。

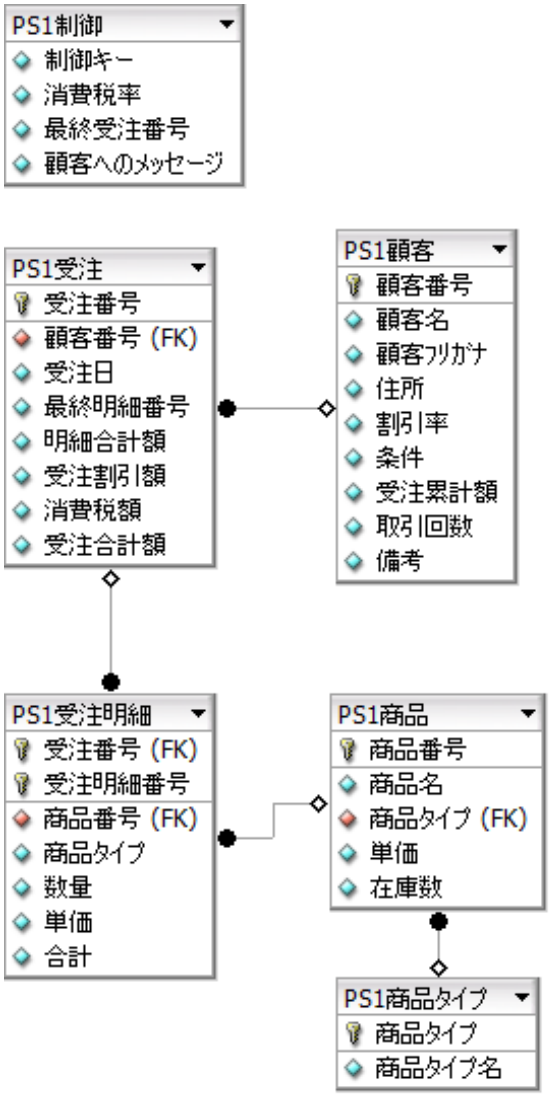
3.1.3 プログラムの操作

ユーザの利便のために、次のような操作上の仕様を実現します。

- 同一プログラムで、登録・照会・修正・削除・印刷ができる。
- 顧客、商品検索機能を持つ。
- 受注検索機能を持つ。
- 入力・修正途上のデータ全体の取り消しができる。

3.2 データベース 設計

このアプリケーションのデータベースを ER 図で表すと、下図のようになります。



- 受注番号は、「PS1 制御」 テーブルの「最終受注番号」カラムで管理されています。新しく受注番号を発番するには、この値に1を加えて作ります。
- 各顧客ごとに、受注累計額と取引回数が格納されます。また、各商品について、現在の在庫数が記録されます。この種の情報は、本来はマスタテーブルではなく、別テーブルに記録しておくべきものかもしれませんが、単純化してあります。

4 命名規則

プロジェクトの標準化においての基本は、命名規則を定めることです。
サンプルアプリケーションでは、次のような簡単な命名規則を使いました。

リポジトリ	種別	形式	備考	例
モデル	通常項目	（修飾子なし）		顧客番号
	プッシュボタン	PB_名前		PB_終了
	コンボボックス	CMB_名前		CMB_商品タイプ
	エディット	EDT_名前		EDT_カナ
	ラジオボタン	RB_名前		RB_明細/合計
	フォーム	FRM_名前		FRM_モーダル_標準
プログラム	通常	xy_名前	x：タスクタイプ ● O: オンライン ● B: バッチ ● R: リッチクライアント y: 主なタスクモード ● Q: 照会 ● M: 修正 ● C: 登録 ● D: 削除 ● T: テスト用	OM_受注①
	選択プログラム	SEL_名前 RSEL_名前	オンライン リッチクライアント	SEL_顧客 RSEL_顧客
データ項目	カラム	（修飾子なし）	（データリポジトリのカラム名と同じ）	顧客番号
	パラメータ	Px_名前	x：方向 ● I: 入力 ● O: 出力 ● B: 両方向	PB_顧客番号
	変数	Vx_名前	x: データタイプ ● S: 文字型 ● N: 数値型 ● L: 論理型 ● D: 日付型	VS_顧客ﾌﾘｶﾞﾅ
	ボタン変数	TB_名前	（モデルリポジトリの名前と同じ）	TB_終了

5 実装方法のいろいろ

第 3 章「ペットショップデモ受注入力画面の概要」では、本書で扱うサンプルの仕様について説明しましたが、これを Magic を使って実装する方法はいくつもの型が考えられます。
本書では、次章以下で、12 種類の実装方法を説明していきますが、これらの実装方法は、

- アルゴリズムによる分類
- トランザクションによる分類
- タスクタイプによる分類

の組み合わせによって、分類されます。以下にそれぞれについて説明していきます。

5.1 分類と組み合わせ

アルゴリズムによる分類

アルゴリズムによる分類は、一時テーブルを使うか使わないかによって、大きく二つに分けられます。

一時テーブルを使わない方式は、さらに、累計データの更新方法によって、細分類されます。累計データというのは、顧客マスタの累計取引額と取引回数、および商品マスタの在庫数などのデータで、レコード後処理のタイミングで更新されます（6.12「累計値の更新」参照）。データの更新を行うのに、第一の方式では、項目更新コマンドで直接行い、第二の方法では別のバッチタスクを呼び出して行います。

一時テーブルを使う方式のほうは、さらに、一時テーブルのコピーと書き戻しをいかにして行うかで細分類されます。第一の方法は Magic のバッチタスクで行う方法で、第二の方法は DBMS のストアドプロシージャを使う方式です。

以上をまとめると、次の表のようになります。

一時テーブルを・・・	方式	略号
使わない	累計データを項目更新コマンドで直接更新する方式。	直接更新
	累計データを別のバッチタスクを呼び出して更新する方式。	バッチ更新
使う	バッチタスクでコピー/書き戻しを行う方式。一時テーブルは、Memory GW に作ります。	MEM テーブル
	ストアドプロシージャでコピー/書き戻しを行う方式。一時テーブルは、DBMS 上に作ります。	DSQL

ここで「略号」というのは、以下の説明において、実装方法の区別をするために使います。

ここでは、アルゴリズムとして上記 4 種類のみをあげましたが、実際にはさらに細かなバリエーションが考えられます。詳細は省略しますが、例えば、次のようなものが考えられます。

- 一時テーブルを使う場合に、本書の例ではヘッダ・明細両方に一時テーブルを利用しましたが、明細テーブルのみに一時テーブルを使い、ヘッダテーブルには使わない、という方法もありえます。
- 受注番号を新規発番する場合に、本書では制御テーブルで最終受注番号を管理していましたが、MS-SQL Server の IDENTITY カラムや、Oracle のカウンターオブジェクトなど、RDBMS の機能を利用して発番させることもできます。
- 一時テーブルを使わない場合、累計データ（6.12「累計値の更新」参照）をリアルタイムに更新する必要がなければ、「バッチ更新」を行う必要はないかもしれません。

細かな変種を考えると組み合わせが非常に多くなってしまうので、本書では、典型的と思われる形だけを選択し、上の 4 種類を扱うようにしました

MS-SQL Server では、テーブル名の先頭に「#」、「##」をつけて、一時テーブルを作成する機能がありますが、本書で使う「一時テーブル」としては、この MS-SQL Server の一時テーブルの機能を利用していませんので、混同しないように注意してください。

本書での一時テーブルとしては、「MEM テーブル」の方法では Magic の Memory テーブルを利用していますし、「DSQL」の方法では、MS-SQL Server の通常のテーブルを利用しています。

トランザクション設定による分類

トランザクションの設定としては、物理トランザクションを使う方法と、Magic 独自の遅延トランザクションを使う方法とに分けられます。

トランザクション	略号
物理トランザクション	物理
遅延トランザクション	遅延

タスクタイプによる分類

タスクタイプとしては、オンラインとリッチクライアントとに分類されます。

タスクタイプ	略号
オンラインタスク	ONL
リッチクライアントタスク	RC

組み合わせ

以上の3通りの分類を組み合わせることにより、下表に示すような 12 種類のバリエーションが可能です。表中、括弧の中の数字は、そのタイプのプログラムの説明がされている章/節番号です。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

リッチクライアントの場合には、物理トランザクションが使えず、必ず遅延トランザクションを使うことになるので、「RC/物理/・・・」という組み合わせはなく、必ず「RC/遅延/・・・」という組み合わせとなります。このため、以下の説明では、RC の場合の「遅延」は省略します。

5.2 移植の順序

本書のサンプルは、「オンライン、物理トランザクション、直接更新」を基本形としました。これは第 3 章「ペットショップデモ受注入力画面の概要」で説明したような仕様の受注入力を Magic で実現する必要最小限のプログラムで、オリジナルのペットショップデモの受注入力と基本的に同じ構造とロジックとなっています。下の表中にも、「ONL/物理/直接更新」とは書かずに、「基本形」と書いています。

そのほかの方式は、この「基本形」を出発点として、順次移植する形で開発しました。移植の順序は次の表で、➡で示した通りです。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 ➡	ONL/遅延/直接更新 ➡	RC/直接更新
バッチ更新	ONL/物理/バッチ更新 ➡	ONL/遅延/バッチ更新 ➡	RC/バッチ更新
MEM テーブル	ONL/物理/MEM テーブル ➡	ONL/遅延/MEM テーブル ➡	RC/MEM テーブル
DSQL	ONL/物理/DSQL ➡	ONL/遅延/DSQL ➡	RC/DSQL

基本形はマルチユーザ環境に対応していません（6.26「複数ユーザ利用時の問題点」で詳説）ので、同時にデータベースにアクセスする人がいないスタンドアロン環境でしか利用できませんが、Magic の基本機能を数多く使っているので、その意味で「基本」となるものです。

本書では、第 6 章「ヘッダ・明細型プログラムの基本形」において、利用されている Magic の機能とか、プログラミングテクニックなどについて掘り下げて説明していきます。ここで説明される機能は、他の方式でも利用されています。

第 7 章「ONL/物理/バッチ更新 」から第 11 章「リッチクライアント」までの章では、基本形を出発点として、順次移植していき、その移植の際に修正したところ、留意するところなどを説明していきます。

5.3 図の凡例

プログラムの説明に入る前に、次章以下の説明で使う図について、意味を簡単に下記に説明します。

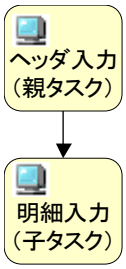
オンラインタスク
（黄色の四角）

バッチタスク
（水色の四角）

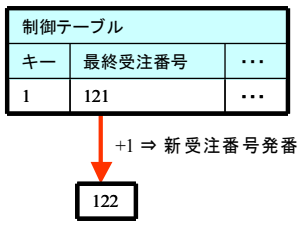
ストアドプロシージャ
（桃色の四角）

テーブルへの書き込み
（緑色の二重線）

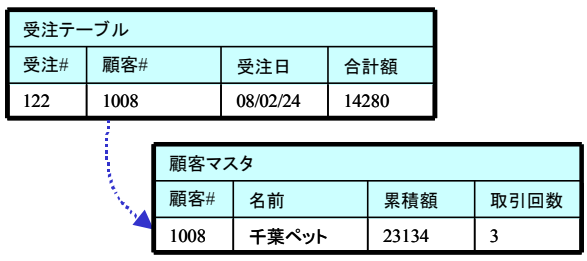
タスクの呼び出し
（黒色の実線）



データの変更
（赤色の実線）



他テーブルへの参照
（青色の点線）

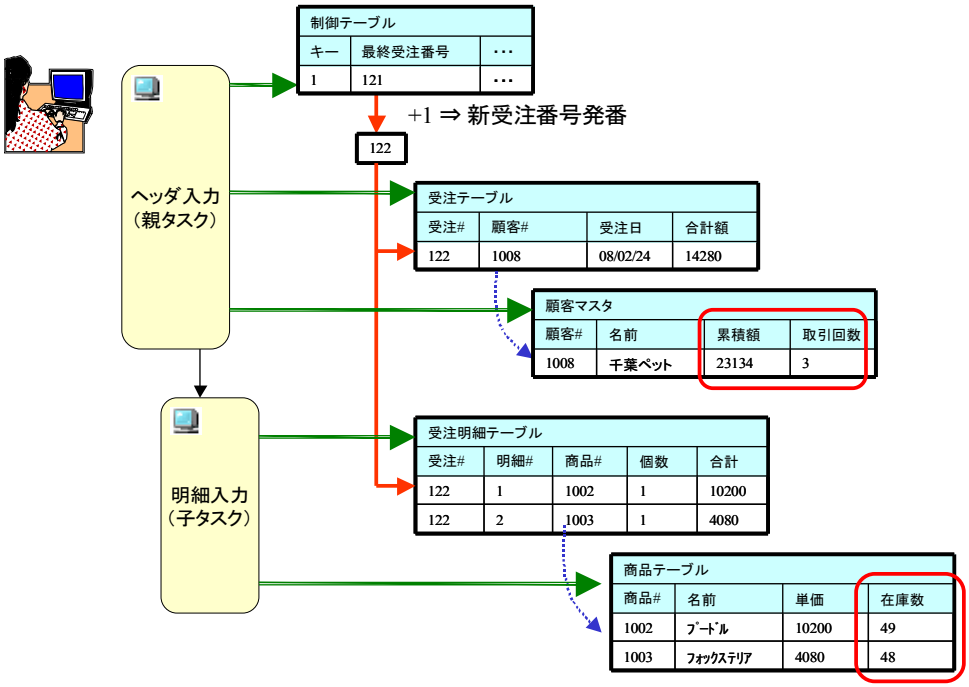


6 ヘッダ・明細型プログラムの基本形

本章で説明する「基本形」というのは、前述のような仕様の受注入力を実現する必要最小限のプログラムで、オリジナルのペットショップデモの受注入力と基本的に同じ構造とロジックとなっています。

このプログラムの概要は、下図のようなものです。

基本形



- 親子のオンラインタスクからなっています。
- 親タスクは受注テーブル（ヘッダ）を、サブタスクは受注明細テーブル（明細）をメインソースとしています。
- 親タスクでは、次のテーブルをリンクしています。
  - 制御テーブル（最終受注番号と消費税率を取得するため）
  - 顧客マスタ（顧客情報を取得するため）
- サブタスクでは、次のテーブルをリンクしています。
  - 商品マスタ（商品情報を取得するため）
- 親タスクのレコード後処理で、次のことを行っています。
  - 最終受注番号の更新（登録モード時のみ）
  - 顧客マスタの更新（受注累積額、取引回数の更新）
- サブタスクのレコード後処理で、次のことを行っています。
  - 受注レコードの明細合計額の更新
  - 商品マスタの更新（在庫数）

以下の説明では、親タスクがヘッダテーブル（受注テーブル）を担当していますので、「**ヘッダタスク**」と呼びます。一方、サブタスクは明細テーブルを担当していますので、「**明細タスク**」と呼びます。

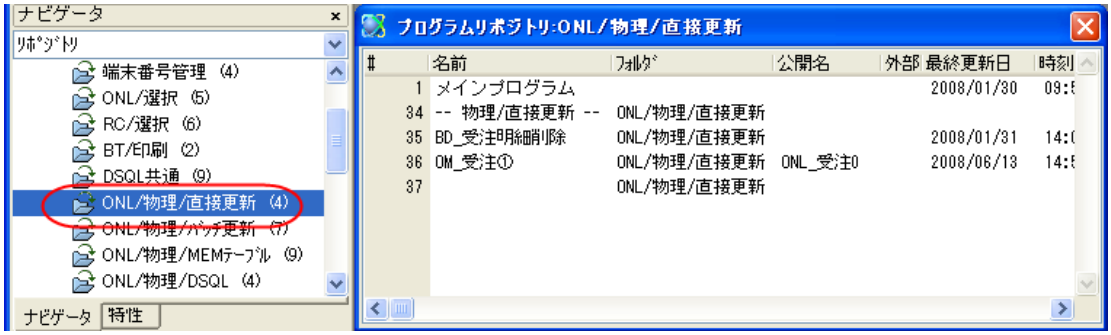
本章での説明は、Magic のごく基本的なことばかりですので、Magic でのプログラム作成に慣れている読者の方は、読み流してもらってかまいません。

この基本形はマルチユーザ環境に対応していませんので、同時にデータベースにアクセスする人がいないスタンドアロン環境でしか利用できません。その点で実用的ではないのですが、Magic の基本機能を数多く使っているので、その意味で「基本」となるものです。

6.1 プログラムの概要

6.1.1 プログラム

基本形のプログラムは、プログラムリポジトリのフォルダ「ONL/物理/直接更新」にあります。受注入力プログラムは、プログラム36番「OM_受注①」です。



バッチプログラム 35 番「BD_受注明細削除」は、受注データを削除する際に、明細レコードを削除するために利用します。

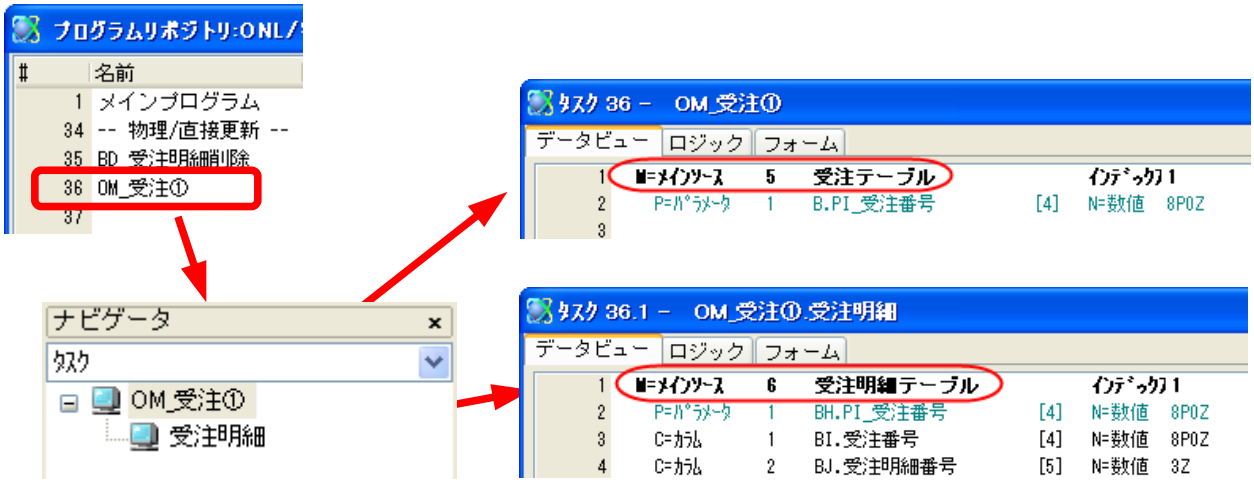
オリジナルのペットショップデモでは、受注金額が 0 円でないときには受注レコードを削除できないようになっていました。従って、手作業で明細レコードをすべて削除してから、受注レコードを削除する、という運用方法を想定していることになります。このように作ってある場合には、明細削除のバッチプログラムも不要になります。

6.1.2 プログラム構造

ヘッダ・明細型の階層的なデータ構造を扱うには、親子のオンラインタスクで扱います。

プログラム 36 番「OM_受注①」を開き、ナビゲータでタスク構造を見ると、下図のようにオンラインの親子タスク構造になっています。

親タスクのメインソースは、「受注テーブル」であり、サブタスクのメインソースは「受注明細テーブル」です。



6.2 明細タスクを呼び出すには

ヘッダタスクから明細タスクへの呼出は、V10 の新機能であるサブフォームを使っています。このため、明細タスクを呼び出すためのコールコマンドは使う必要がありません。

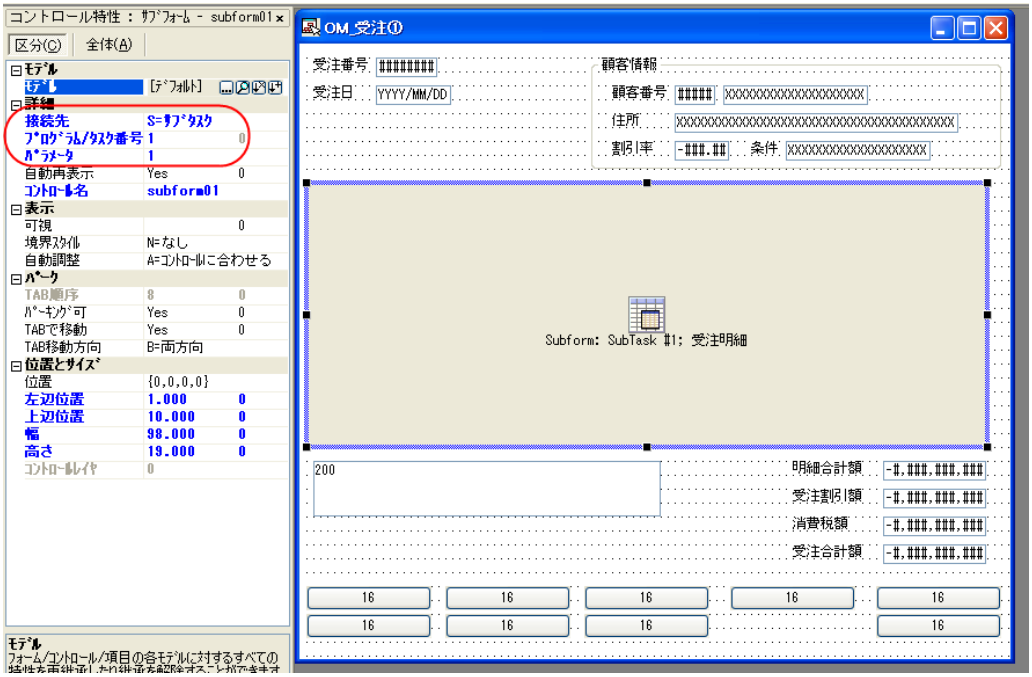
オリジナルのペットショップデモでは、サブフォームがまだサポートされていなかったので、「ファントムタスク」の手法を使っていました。V10 のオンラインタスクでもファントムタスクの機能はサポートされていますが、プログラムの簡単さや、リッチクライアントへの移行なども考慮して、サブフォームを使うことをお勧めします。

サブフォームについてのより詳しい情報は、「サブフォームコントロール」をキーワードとしてリファレンスヘルプを検索してください。

また、Studio 製品添付の「V10 新機能チュートリアル」の 7.4 章「サブフォーム」にも解説があります。

6.2.1 サブフォームの定義

ヘッダタスクのフォームエディタで、サブフォームコントロールを配置します（下図）。



サブフォーム特性として、右表の値を設定します。

このように設定しておくと、実行時には、ヘッダタスクのサブフォームコントロールの部分に、明細タスクの画面が埋め込まれた形で表示されます。

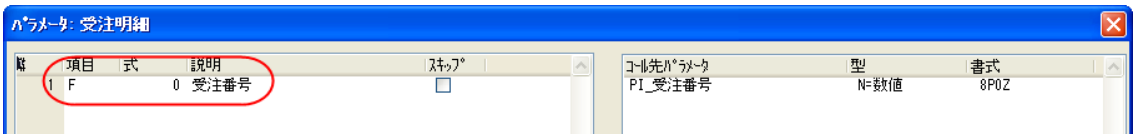
特性	値
接続先	S=サブタスク
プログラム/タスク番号	1
パラメータ	受注番号

6.2.2 パラメータ

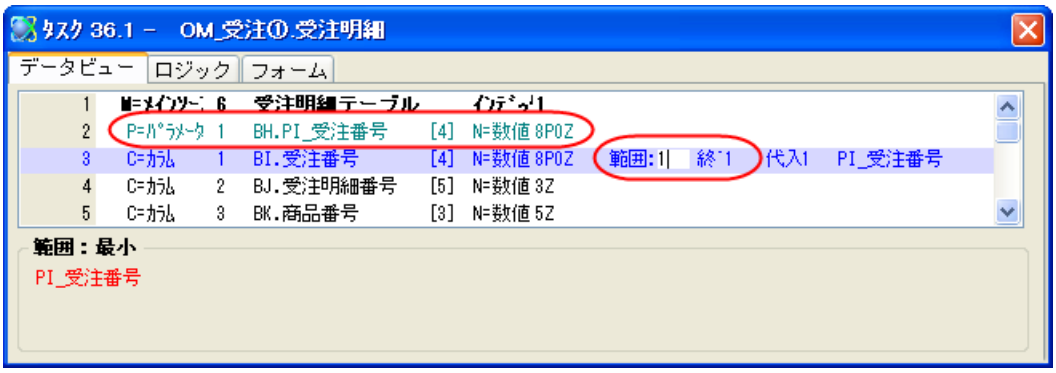
サブフォームを使った場合には、親子タスクの連動をするデータ項目を、パラメータで渡すことが必要です。こうすることにより、サブフォームの表示内容が、親タスクの表示内容に連動して、自動的に再表示されるようになります。

本章の受注明細プログラムの場合には、次のように設定します。

1. サブフォーム特性の「パラメータ」には、パラメータをひとつ作成し、F（受注番号）を指定します。



2. サブタスクでは、受注番号をパラメータとして受け取り、明細テーブルの範囲指定に使います。

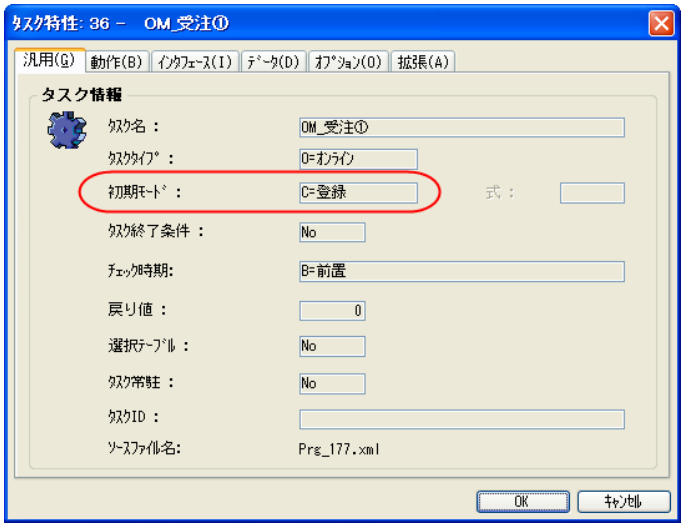


6.3 タスクモードの制御

6.3.1 登録モードで始めるには

ここでの受注入力プログラムの仕様では、開始直後の初期状態では、ユーザ入力ができる「登録モード」にします。

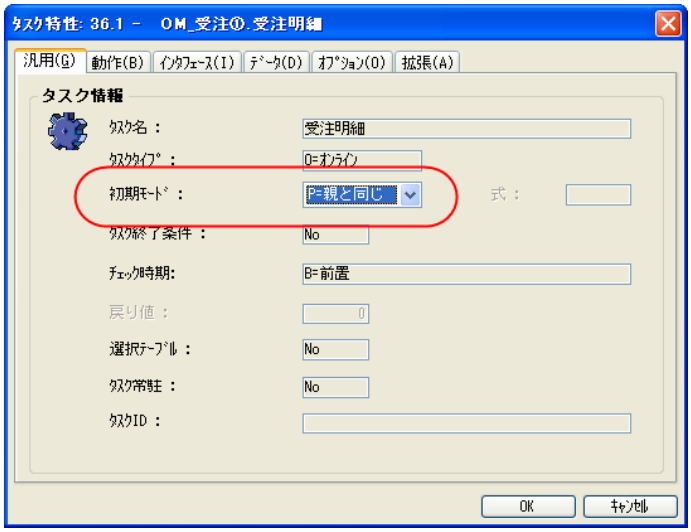
初期状態で登録モードにするには、ヘッダタスクのタスク特性で、「初期モード」を「C=登録」に設定しておきます（下図）。



6.3.2 明細タスクのタスクモードを制御するには

明細タスクのタスクモードは、ヘッダタスクのタスクモードと同じでなければなりません。

これを実現するには、明細タスクの初期モードとして「P=親と同じ」と指定します。



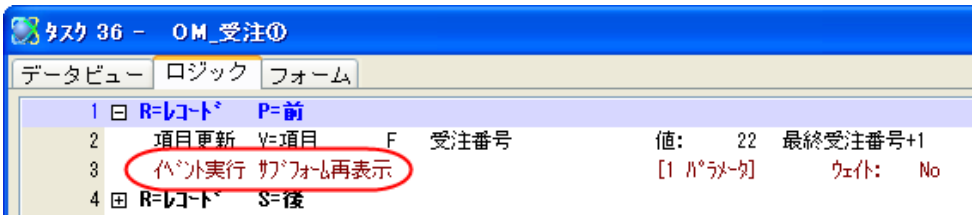
ファントムタスクの技法を使った場合には、タスク初期モード「P=親と同じ」を使うと、フォーカスが親子の間を移動した場合に、明細行の表示が消えてしまうということがありましたが、サブフォームを使った場合にはこういう現象は起こりません。

6.3.3 再表示の制御

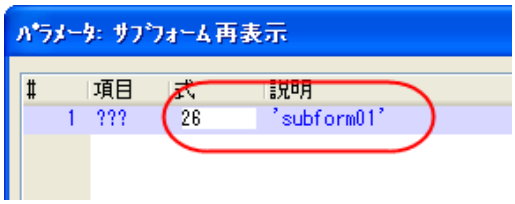
ただし、このままでは、ヘッダタスクのタスクモードが切り替わるときに、即座に明細タスクのタスクモードに反映されません。例えば、「修正」ボタンを押してから、「照会」ボタンを押すと、ヘッダタスクは照会モードになりますが、明細タスクは修正モードのままになってしまいます。

これに対応して、親子のタスクモードを同期させるには、適当なタイミングでサブフォームを再表示させる必要があります。

ヘッダタスクのタスクモードが切り替わった場合には、タスク前処理は走りませんが、レコード前処理が走りますので、ヘッダタスクのレコード前処理で「サブフォーム再表示」イベントを発行します。



このイベントへのパラメータとしては、サブフォームのコントロール名を指定します。



10.1SP4b では、オンラインタスクのサブフォームで、修正モードから登録モードに切り替わるときに、明細行のデータの古い内容が残ってしまう、という不具合があります。

この問題を回避するには、サブタスクのタスクモードを「E=式」とし、式で

IF (Stat(1,'Q'MODE), 'Q'MODE, 'M'MODE)

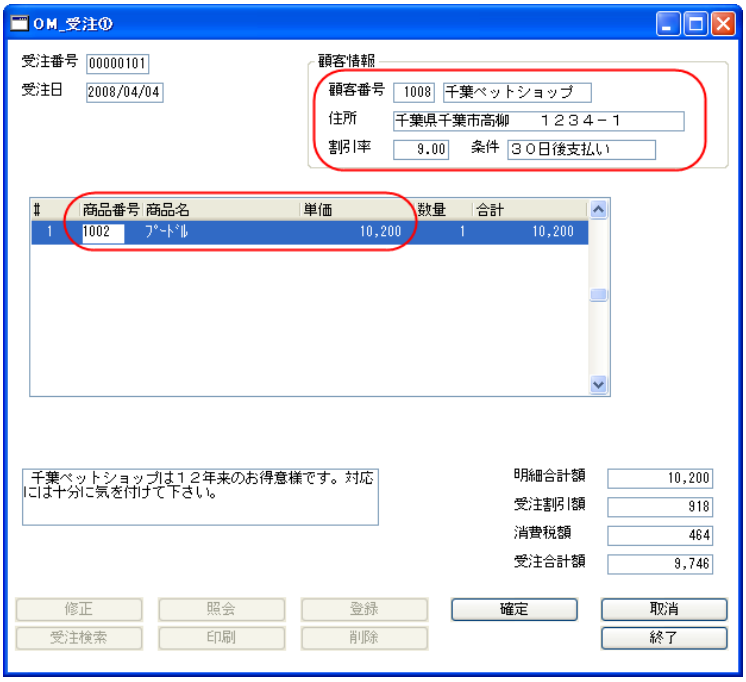
と指定します。

6.4 顧客・商品情報を取得するには

受注入力画面では、顧客番号を入力すると、その顧客に関する情報（名前、住所等）が、「顧客情報」に表示されます。この情報は、顧客マスタにあるので、顧客番号をキーとして、顧客マスタから情報を取得する必要があります。

また、顧客番号が変更された場合には、「顧客情報」の内容も、それに連動して更新されなければなりません。

このような動作をさせるためには、顧客番号をキーとして顧客マスタに**データリンク**（以下、単に**リンク**と書きます）を行います。



リンクは、タスクの「データビュー」画面で「L=照会リンク」コマンドを使って行います。

右図は、ヘッダタスクのデータビューを表示したものですが、顧客マスタ、および制御テーブルがリンクされています。

同様に、明細タスクのほうでは、商品番号を入力すると、その商品に関する情報（商品名、単価）が表示され、単価と数量とから自動的に合計を計算します。このために、商品番号をキーとして、商品マスタにリンクを行っています。（設定方法は同様なので、詳細は省略します）。

Magic のリンクでは、**自動再計算**を行います。すなわち、リンクとなるキー（顧客番号など）が変更された場合には、自動的に、リンク対象となるテーブル（顧客マスタなど）のレコードを再検索します。これにより、常に正しく連動したデータ値が表示されるようになります。

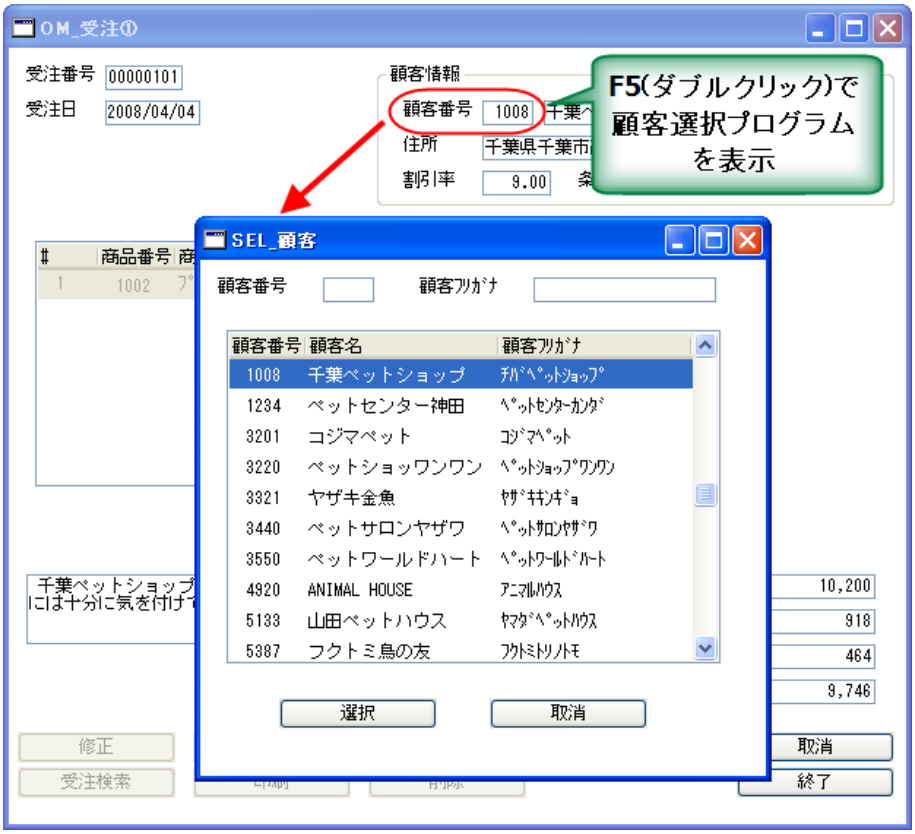
6.5 顧客一覧から顧客を選択するには

6.5.1 ズーム機能

顧客番号や商品番号の入力を容易にするために、**ズーム**ができると便利です。ズームというのは、一覧からキー値を選択することです。

例えば、「顧客番号」欄にカーソルがあるときに、F5 キー、あるいはダブルクリックをすることにより、顧客選択プログラムが表示されます。ここで、顧客を選択すれば、その顧客番号が「顧客番号」欄に設定されます。

このとき、「顧客番号」のような、ズームして選択を行う項目を「**ズーム項目**」と呼び、一覧表示をしてユーザに選択をさせるプログラムを「**選択プログラム**」と呼びます。



業務アプリケーションではこのような操作が非常に多いので、Magic ではズームを実現するために、次の二つの機能が提供されています。

- 選択プログラムの作成を容易にするため、タスク特性に「**選択テーブル**」パラメータがあります。
- ズーム項目から F5 キーあるいはダブルクリックにより選択プログラムを呼び出すことを容易にするため、項目の特性として、「**選択プログラム**」という特性があります。

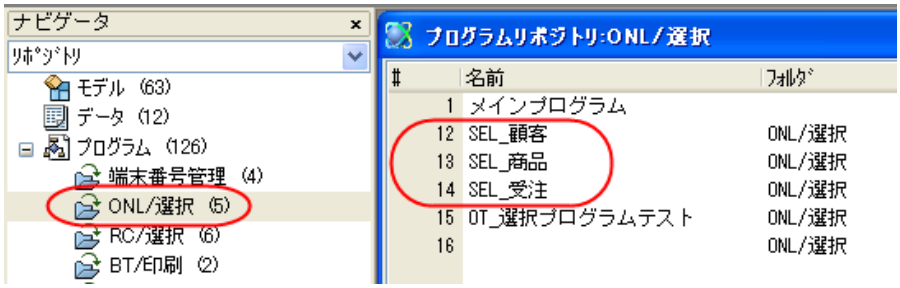
以下に、それぞれについて簡単に説明します。

6.5.2　タスク特性の「選択テーブル」パラメータ

一覧から選択するプログラムは、アプリケーションの多くのプログラムから利用されることがあるので、普通は独立したプログラムとして作成します。

本書のサンプルアプリケーションでは、次のものがあります。

- プログラム 12 番「SEL_顧客」
- プログラム 13 番「SEL_商品」
- プログラム 14 番「SEL_受注」

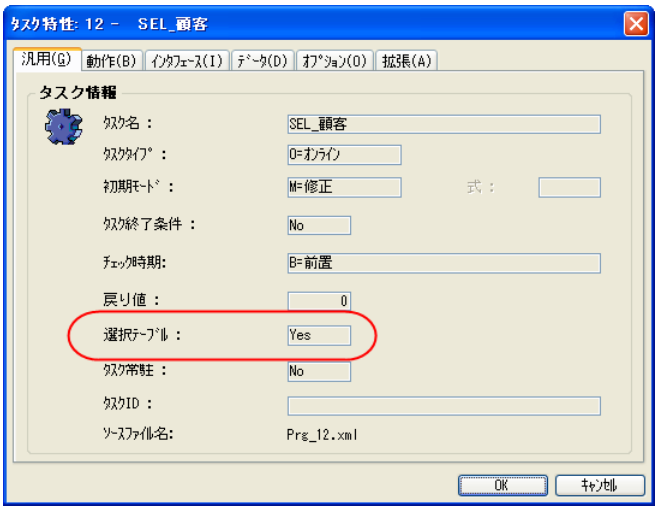


選択プログラムは、次のような仕様となります。ここでは、顧客選択プログラム「SEL_顧客」を例にしています。

パラメータ	● キー項目（顧客番号）　※ これは、入力、出力両方向です。
表示	● 顧客レコードの「顧客番号」と「顧客名」をテーブル形式で表示します。 ● 初期画面では、パラメータとして与えられた顧客番号でレコード位置づけをします。 ● 上下のスクロール、先頭レコード、最終レコードなどへの移動が可能です。
操作	● ユーザが Enter キーを押すか、ダブルクリックすると、そのときカーソルのあるレコードの顧客番号が、戻り値としてパラメータに設定されて、選択プログラムが終了します。 ● ユーザが キャンセルボタンを押すか、ESC キーを押すと、パラメータは変更されずに、そのまま選択プログラムが終了します。

このような一覧選択プログラムを実装する場合には、タスク特性で「選択テーブル」を Yes にしておくのが便利です（下図）。

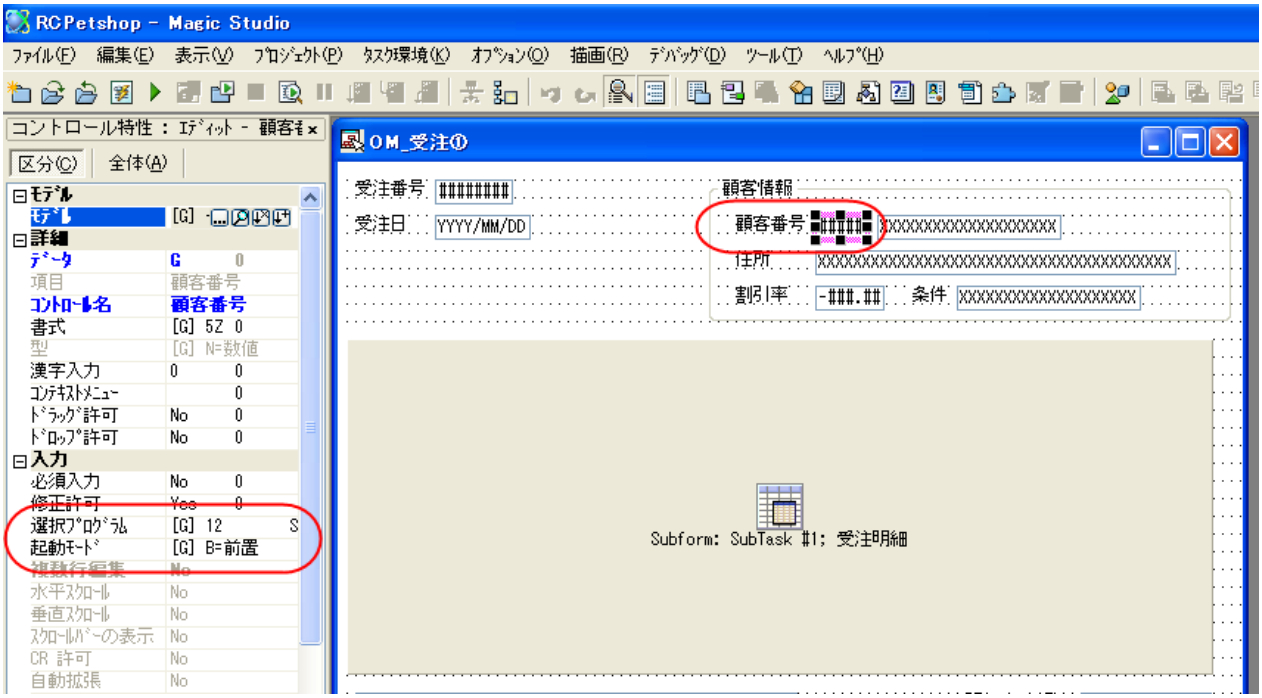
「選択テーブル」が Yes に設定されていれば、ユーザが Enter キーを押すか、あるいはダブルクリックをすると、Magic エンジンはレコード後処理を実行して、タスクを終了させます。この性質を使って、レコード後処理で、現在のキー値をパラメータに設定してやるようにすれば、一覧選択プログラムの作成が簡単になります。



6.5.3　項目の「選択プログラム」特性

ズーム項目を実現するには、その項目の「選択プログラム」特性を利用するのが便利です。「選択プログラム」特性を使えば、「コール」プログラムやイベントハンドラなどを使わなくとも、ズームを実現することができるようになります。

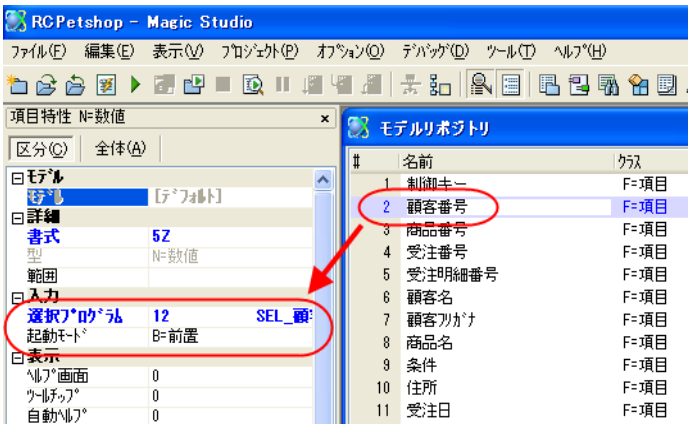
例えば、次の図は、受注入力プログラムの親タスクのフォームエディッタで、「顧客番号」項目の特性を表示させているところです。



ここでは、「選択プログラム」として、プログラム 12 番「SEL_顧客」が設定されています。このように設定されていると、実行時、カーソルがこの項目にあるときに、ユーザが F5 キーを押すか、あるいはダブルクリックをすると、「選択プログラム」に指定されている「SEL_顧客」プログラムが呼び出されます。このプログラムには、「顧客番号」項目がパラメータとして渡されます。

上記の「選択プログラム」特性は、もともとは、「顧客番号」モデルに定義されていて、そちらから継承しています。

モデルに選択プログラムを設定しておけば、そのモデルを参照して定義されているすべての項目において、「選択プログラム」特性が有効になり、アプリケーションの生産性がいっそう向上します。



6.6 入力値の検証

ユーザデータを入力する場合には、間違った値を入力していないか、確認するしくみをプログラムの側に入れておくことが多くあります。

今回の受注入力でも、次のようなデータ検査を行うことにします。

- 入力された顧客番号は、顧客マスタに登録されているか？
- 入力された商品番号は、商品マスタに登録されているか？
- 受注金額が0円以下になっていないか？

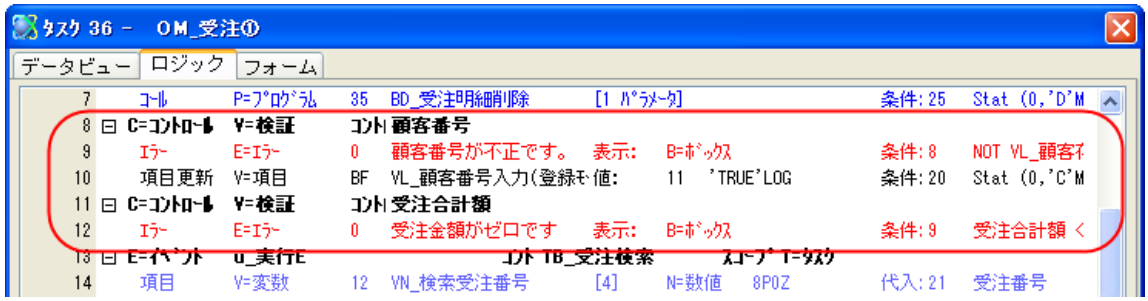
実際のアプリケーションでは、日付チェック、顧客情報のチェック、受注した商品情報や組み合わせに関するチェックなど、さまざまな確認条件があることと思いますが、ここでは、データ確認のしくみを理解することを目的とし、上記のチェックだけを行うようにします。

6.6.1 コントロール検証 ハンドラ

入力データの正当性を検証するのは、通常、検査したいデータ項目に対する**「コントロール検証」**ハンドラで行います。

コントロール検証ハンドラは、タスクの「ロジック」エディッタ中に定義されます。各コントロール検証ハンドラは、フォーム上の表示項目のコントロール名を参照して定義されます。

例えば、下図は、ヘッダタスクのロジックエディッタの一部です。



ここでは、次の二つのコントロールを参照して、コントロール検証ハンドラが定義されています。

- **顧客番号**: 入力された顧客番号が、顧客マスタに登録されているかをチェックします。もし登録されていない顧客番号が入力されたら、エラーコマンドでエラー情報を表示し、それ以上前に進めないようにします。
- **受注合計額**: これはフォームの一番最後の項目です。ここで、受注金額が0円以下になっていないかをチェックします。0円以下になっていたら、やはりエラーコマンドでエラー情報を表示し、それ以上前に進めないようにします。

6.6.2 コントロール検証ハンドラの実行タイミング

コントロール検証ハンドラは、参照しているコントロールをカーソルが通過する際に、実行されます。より正確には、次のような動作となります。

- 参照しているコントロールに、カーソルがパークしていたか否かは関係ありません。また、そのコントロールがパーク不可であってもよいし、あるいは無効なコントロール（グレーで表示されている）であったりしてもかまいません。
- カーソルは、次のような場合に「通過した」と判断されます。
  - そのコントロールにカーソルがある状態から抜け出して、別のコントロールに移動する場合。
  - コントロールの「タブ順序」に従って、カーソルが前のコントロールから後のコントロールに移動する

際（順方向）。
    - コントロールの「タブ順序」に従って、後ろにあるコントロールにカーソルがあった状態から、前のコントロールに移動する際（逆方向）。
- カーソルの移動のきっかけは関係ありません。例えば、次のような操作でカーソルが移動しますが、いずれの場合でもコントロール検証ハンドラは実行されます。
    - Tab キー（次項目）、Shift-Tab キー(前項目）などを押した場合
    - マウスカーソルで別項目をクリックした場合
    - 別レコードに移動する場合（ラインモードで[↑]/[↓]キーを押した場合、PgUp/PgDown、Ctrl-Home、Ctrl-End キーなどを押した場合）
    - ESC キーでタスクが終了する場合。（この場合は、現在カーソルがあるコントロールから、前方向にすべてのコントロールを通過していく、と解釈されます）

コントロール検証についてより詳しい情報は、以下のキーワードでリファレンスヘルプを検索してください。

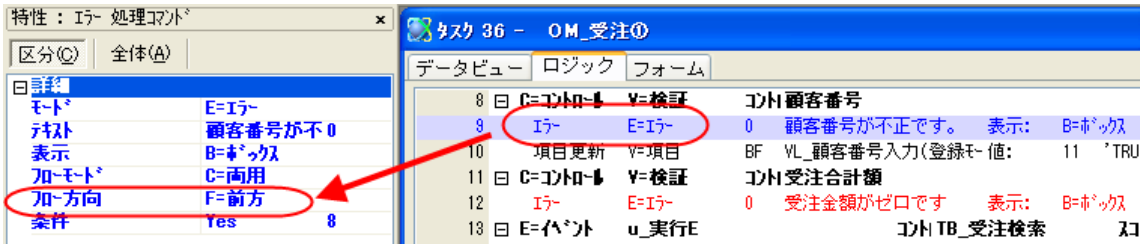
- コントロール検証

6.7 フローモード

顧客番号にコントロール検証ハンドラが設定されている場合、カーソルがいったん顧客番号に入ったら、正しい値を入力しない限り、そこから前にも後ろにも抜け出すことができません。これでは操作性があまり良くありません。

そこで、「前方に進むのは阻止されるが、後方に戻るのは許す」という仕様にしたいと思います。

これを実現するためには、カーソルの動く方向によって条件付けをすることのできると便利です。これは、「**フロー方向**」パラメータによって実現します。



- 「フロー方向」パラメータには、次のような選択肢があります。

フロー方向パラメータ値	意味
F=前方	カーソルが前方(順方向)に移動する場合にだけ実行する。
B=後方	カーソルが後方(逆方向)に移動する場合にだけ実行する。
C=両方向	カーソルがいずれに移動する場合にも実行する。

- 「フロー方向」パラメータは、オンラインタスクにおいて、各コマンドごとに設定することができます。

上図の例では、エラーコマンドは「フロー方向」=「F=前方」と設定されています。従って、このエラーコマンドは、カーソルが前方に進む場合にだけ実行されます。逆方向に進む場合には実行されません。これにより、間違った顧客番号のときには、前方へは進めないけれども、後ろ向きに(「日付」項目などに)戻っていくことはできます。

フローモードについてより詳しい情報は、以下のキーワードでリファレンスヘルプを検索してください。

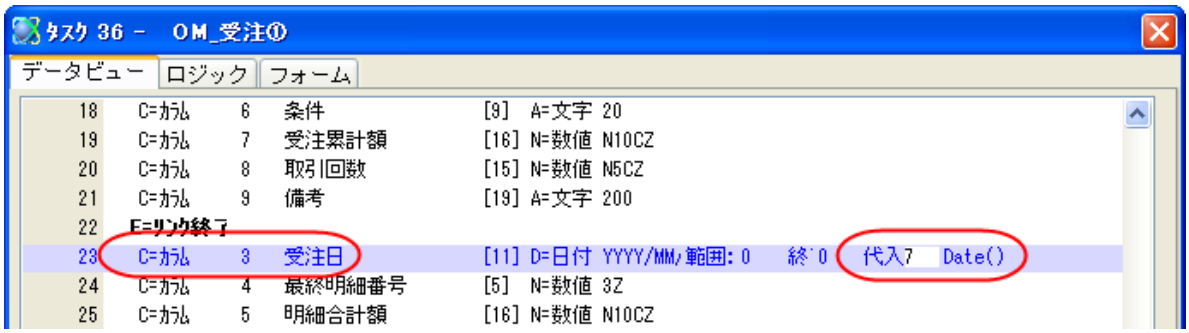
- フローモード

6.8 登録時の初期値設定

登録モードのときに、入力値の「デフォルト」を設定できると便利です。例えば、サンプルアプリケーションでは、「受注日」のデフォルト値は「今日」とする、という仕様になっています。

登録モードの時のデフォルト値は、データビューで、「代入」欄に式を使って指定することができます。

受注日のデフォルト値を「今日」にするには、データビューの「受注日」を定義している「C=カラム」行において、「代入」欄に、式で「Date()」を設定します。(下図)



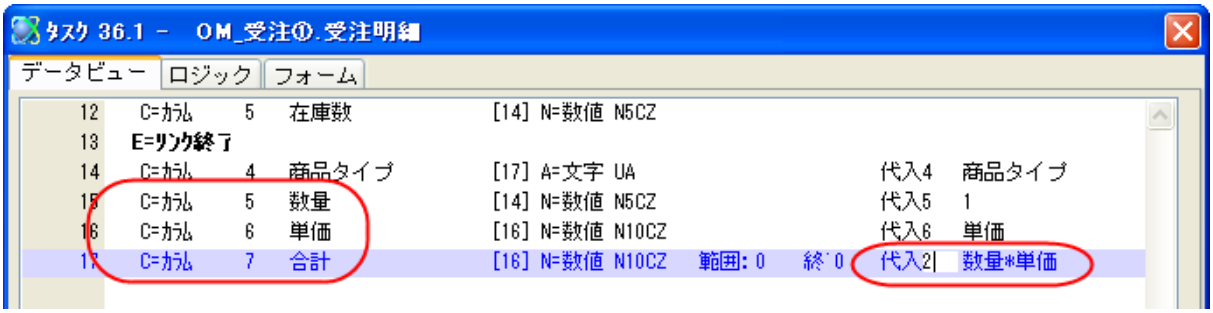
代入式についてのより詳しい情報は、以下のキーワードでリファレンスヘルプを検索してください。

- 代入式

6.9 依存関係のある値を更新する

業務アプリケーションでは、データ間に依存性のあるものが多数あります。あるデータが別のデータに依存する場合、値の変更が自動的に反映されると便利です。例えば、受注明細（サブタスク）においては、「合計」額は「数量」と「単価」の積ですので、数量の値をユーザが変更した場合には、合計の値も変更しなければなりません。Magic では、「代入」式を使って、依存性のあるデータの自動再計算を行わせることができます。

「合計」値の例では、データビューの、「合計」を定義している「C=カラム」行において、「代入」式に「数量 ＊ 単価」（実際には「BS*BT」というように、シンボルを使って定義します）を設定しておきます。（下図参照）



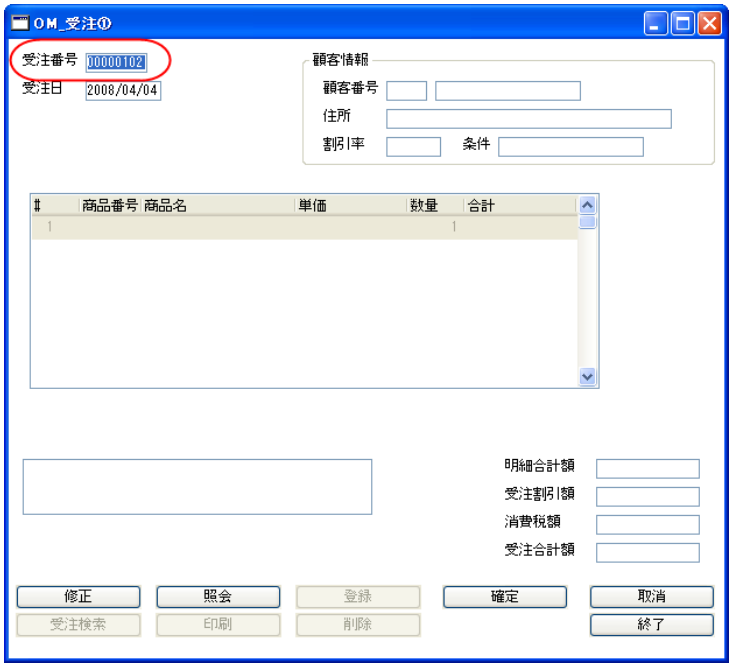
このように、「代入」に式が設定されていると、Magic エンジンは、この「合計」項目が、式中で参照されている「数量」、「単価」項目に依存している、と判断します。実行時には、「数量」あるいは「単価」のいずれかの値に変更があったら、Magic エンジンが自動的に「合計」の値を式に従って再計算します。

代入式における自動再計算についてのより詳しい情報は、以下のキーワードでリファレンスヘルプを検索してください。

- 代入式
- 自動再計算

6.10 連番（受注番号）の発行

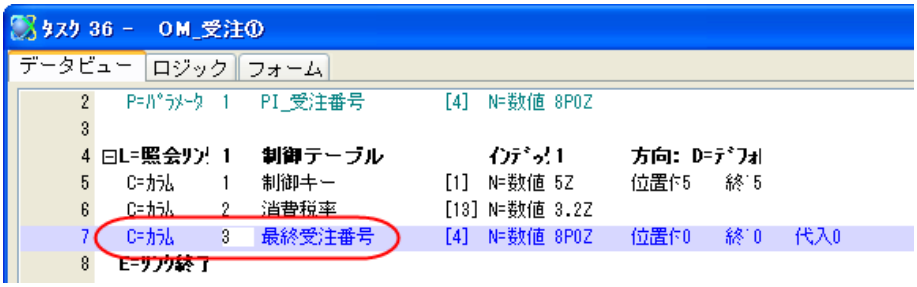
新規受注データを登録する際には、新しく受注番号を発番しなければなりません。



サンプルでは、「制御テーブル」に現在までの最終受注番号を記録しておき、新しく受注登録を行うごとに、＋1しながら、新しい番号を発番するようにしています。

実行時、どのタイミングでどのように新しい受注番号を発番するかにより、いくつかのバリエーションがありますが、基本形では、以下のような簡単なロジックによって発番しています。

まず、制御テーブルのデータを利用するために、ヘッダタスクのデータビューで、「制御テーブル」をリンクしています（下図）。この中で、「最終受注番号」（カラム 3 番）を選択しています。



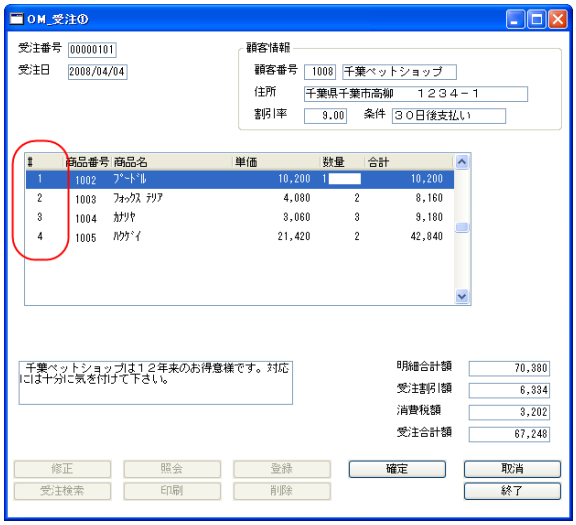
登録モードにおいて、レコード前処理で、「最終受注番号＋1」を新しい受注番号として設定しています。

ユーザが「確定」ボタンを押したら、レコード後処理が実行されます。レコード後処理では、現在の受注番号を、制御テーブルの最終受注番号に書き込んでいます。

「確定」ボタンに関する設定については、「入力の確定」（6.16 節、63 ページ）で説明します。

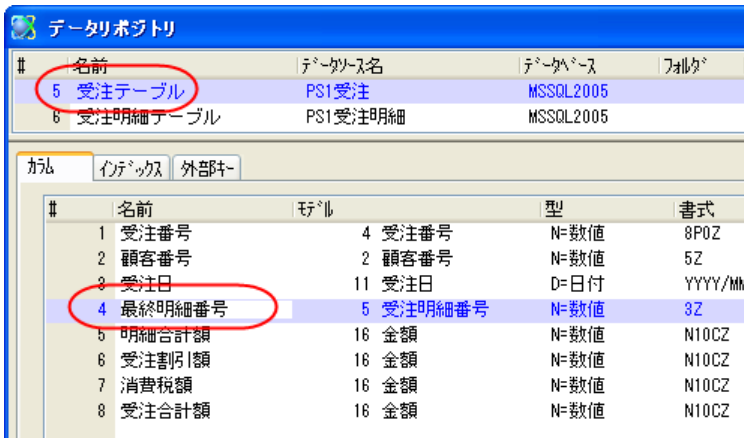
6.11 連番(受注明細番号)の発行

次には、同じく連番の発行についてですが、受注明細レコードの連番の発行方法について説明します。



各受注データにおいて、受注明細には1から始まる行番号を振ります（右図）。

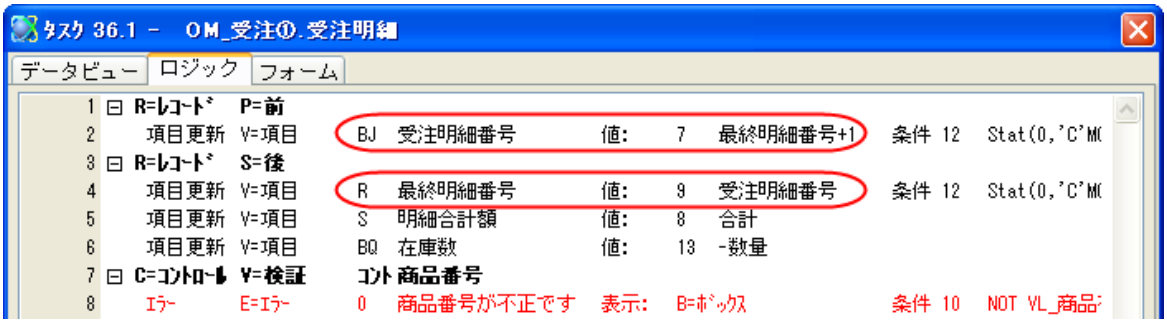
受注明細番号を発番するのも、前節で説明したように、「最終明細番号」を記録しておいて、レコードが作成されるたびに ＋1 して発番するようにしています。



ただし、明細番号は、各受注レコードごとに1から始まるので、最終明細番号は、システム全体に共通な「制御テーブル」ではなく、各受注レコードに記録しておきます(右図)。

これを実現するため、明細タスクでは登録モードにおいて、レコード前処理で、「最終明細番号＋1」を新しい明細番号として設定しています。

ユーザが明細行を１行入力し終わったら、レコード後処理が実行されます。レコード後処理では、現在の明細番号を、ヘッダタスクの受注レコードの最終明細番号に書き込んでいます。

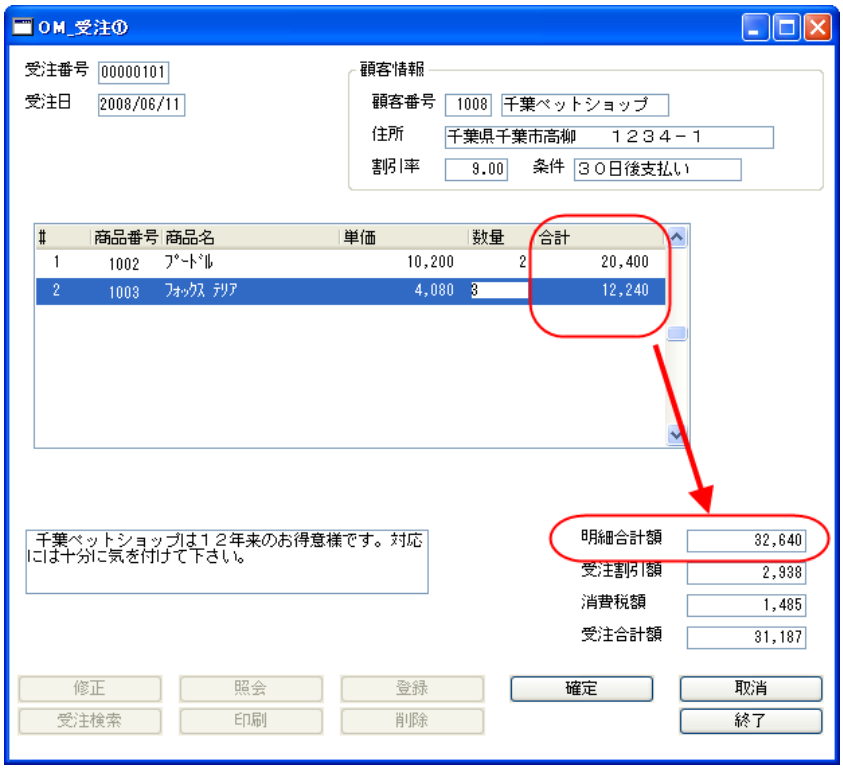


実際のアプリケーションでは、明細行が削除されたり、途中で挿入されたりする可能性があることを考慮して、明細番号のリナンバリングを行うロジックが必要になるでしょうが、サンプルアプリケーションでは省略しています。

6.12 累計値の更新

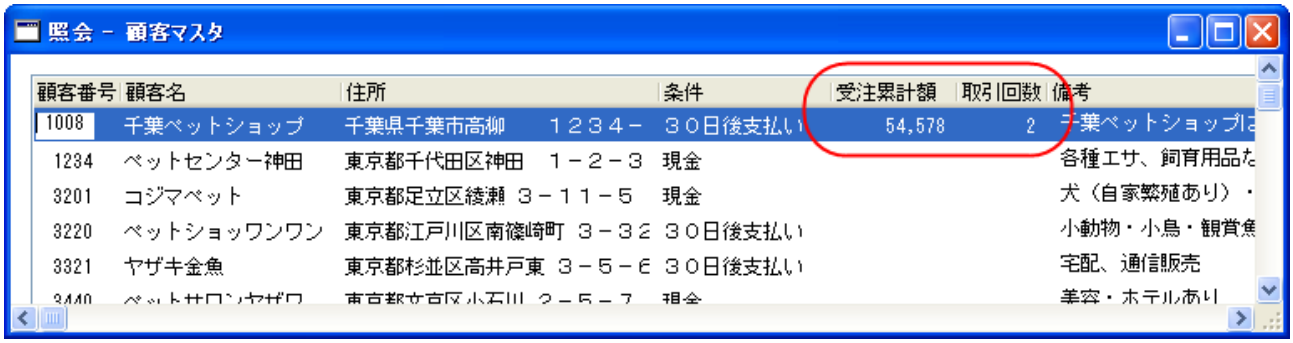
業務アプリケーションでは、金額や商品個数、取引回数などの累計値を計算して保存しておくことがよくあります。

例えば、受注入力プログラムでは、「明細合計額」欄の数字は、明細行の「合計」額の合計と常に等しくなければなりません。



また、別の例として、顧客マスタを見てみると、各顧客について、次のような累計データがあります。

- 「受注累計額」は、各顧客に関する受注データレコードの「受注合計額」の合計になっていなければならない。
- 「取引回数」は、各顧客に関する受注データのレコード件数でなければならない。



Magic では、累計値の計算のために便利な機能として、「加算」モードの項目更新コマンド（以下、「加算更新」と呼びます）が用意されていますので、ここでその利用方法について説明します。

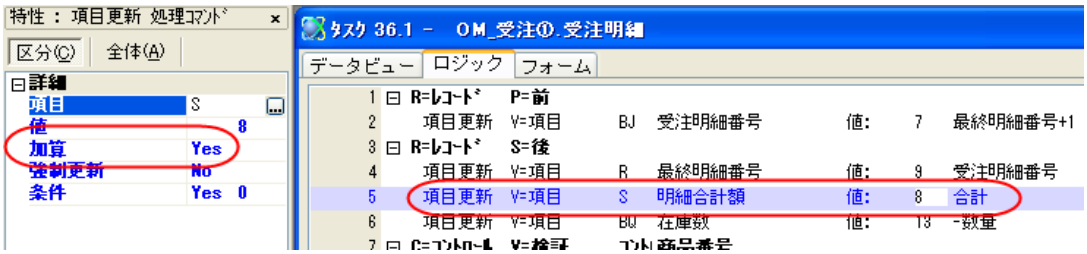
6.12.1 受注レコードの明細合計額

各注文において、明細の合計値を正しく計算しておく必要があります。

明細の合計は、全明細行について、「合計」金額を加算して求めますので、その意味でデータ依存性のある値となりますが、先に 6.9 「依存関係のある値を更新する」で説明した「合計 ＝ 数量 ＊ 単価」というような単純な式で表すことができません。従って、「代入」式によっては表現することのできないデータ依存性となります。

このような、「対象となるレコードの中の特定のカラムの合計」を計算するような場合には、「加算」モードの項目更新コマンドを利用します。

今の例では、明細レコードを扱うサブタスクのレコード後処理で、「明細合計額」（親タスクで定義されている、受注レコードのカラム）に対して項目更新コマンドが実行されます（下図、第 5 行目）。



6.12.2 加算更新の実行ルール

項目更新コマンドは、次のようなルールで実行されます。

修正モードの場合：「加算」モードの項目更新では、単純な代入ではなく、「元の値」と「現在の値」の差分が加算される形で代入が行われます。

例えば、ある明細行で、1000 円の商品を 2 個注文していたとします。ユーザが注文個数を 3 個に増やしたとすると、合計金額は、2000 円から 3000 円に増えます。この差額（＋1000 円）が、「明細合計額」に加算されるようになります。逆に、2 個だったものが 1 個に減ると、差額（-1000 円）が、「明細合計額」に加算（つまり、1000 円の減算）が行われます。

登録モードの場合：登録モードの場合には、初期値は0であったと解釈されます。

例えば、新しく明細行を作成し、5000 円の商品を 2 個注文したとします。この場合には、初期値は0円とみなされるので、差額（＋5000 円）が、明細合計額に加算されます。

削除モードの場合：逆に、レコードが削除される場合には、修正後の値が0になると解釈されます。

例えば、1000 円の商品を 2 個注文している明細行を削除すると、修正後の値は0円とみなされるので、差額（-2000 円）が「明細合計額」に加算（すなわち 2000 円の減算）されます。

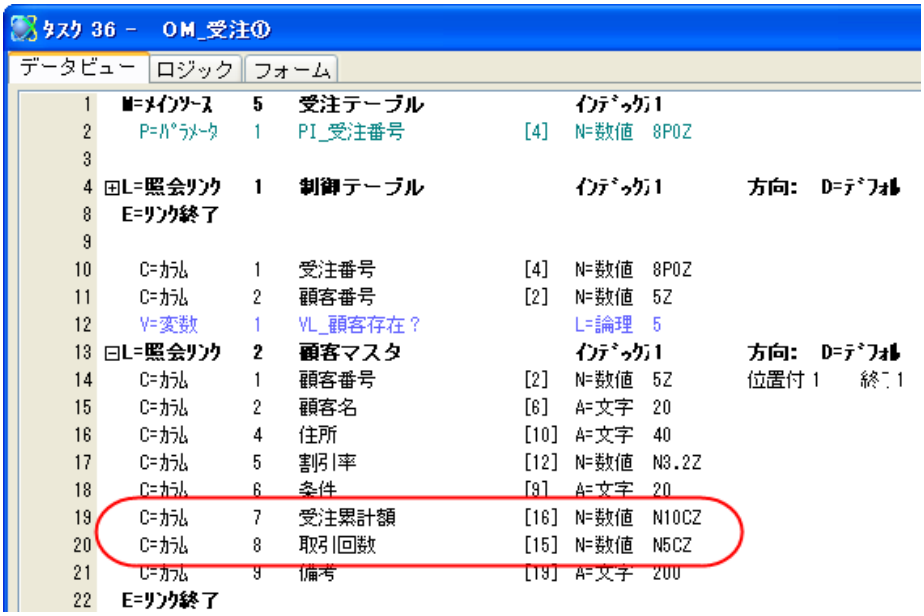
このようにして、加算モードの項目更新を使うことにより、常に合計額を正しく維持することができるようになります。

6.12.3 顧客マスタの受注累計額と取引回数

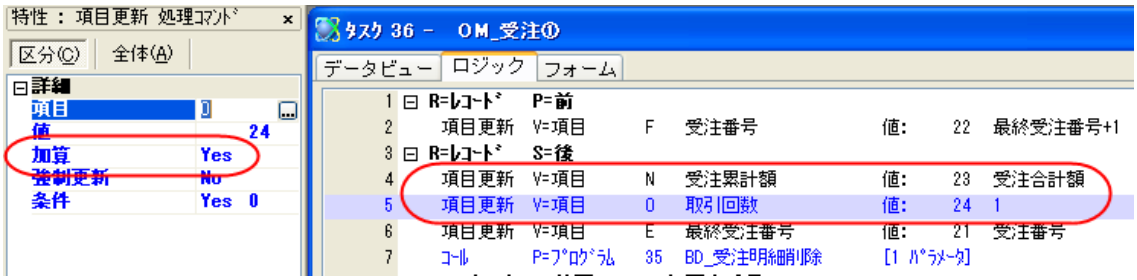
サンプルアプリケーションでは、顧客マスタに「受注累計額」（各顧客毎の注文額の合計）および「取引回数」（各顧客毎の注文件数）というカラムがあり、受注が入るたびに更新されます。これらの値に対しても、加算更新を使うのが便利です。

具体的には、次のようになっています。

1. ヘッダタスクで、これらの項目は、顧客マスタから「顧客番号」をキーとしてリンクにより取得されています（下図）。



2. これらの項目は、ヘッダタスクのレコード後処理で受注累計額および取引回数に対して、加算更新を行います（下図）。



   ここでは、次のように設定されています。
   - 「受注累計額」に対する加算更新の「値」欄は、「受注合計額」
   - 「取引回数」に対する加算更新の「値」欄は、「1」

ここで、「取引回数」に対する「値」欄が「1」という定数であるのは不思議なようにも思えますが、どうしてこのような設定になっているのでしょうか？

「取引回数」は次のように更新する必要があります。

- 受注レコードの新規登録時には、＋1 します。
- 既存の受注レコードを修正した場合には、プラスマイナスゼロです。
- 受注レコードが削除された場合には、－1 します。

「取引回数」への加算更新で、「値」が「1」とした場合、6.12.2「加算更新の実行ルール」に従うと、次のように動作します。

- 登録時には、初期値は0と見なされて、現在値が1となるので、差分は ＋1 になります。従って、1 が加算されます。
- 修正時には、初期値は1、現在値も1なので、差分は0です。従って、値は変わりません。
- 削除時には、初期値は1、現在値は0と見なされるので、差分は －1 です。従って、1 が減算されます。

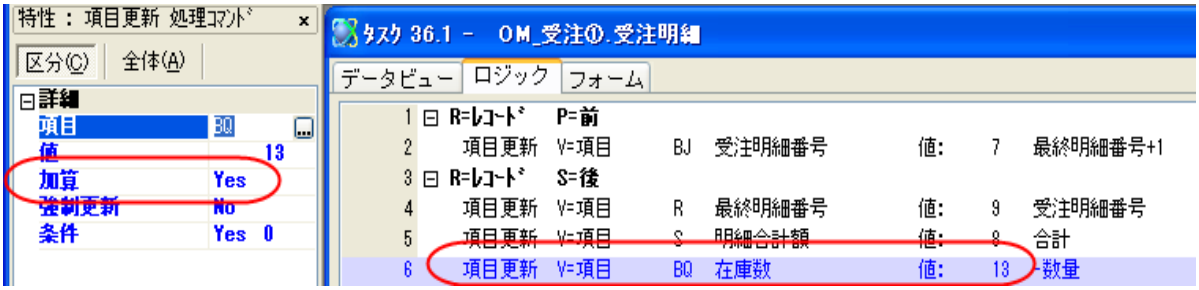
このように、「値」に「1」を指定した場合には、希望通りの動作になっていることがわかります。

6.12.4　商品マスタの在庫数

商品マスタには、「在庫数」というカラムがあり、注文がはいると、その個数分減らさなければなりません。この項目の更新にも、加算更新を使います。

1. 商品マスタは、明細タスクのデータビューで「商品番号」をキーとしてリンクされています。「在庫数」もリンク項目に含まれています（下図）。

2. 「在庫数」は明細タスクのレコード後処理で加算更新を使って更新します（下図）。



加算 が Yes と設定された場合の項目更新コマンドの動作については、以下のキーワードでリファレンスヘルプを検索してください。

- 加算更新

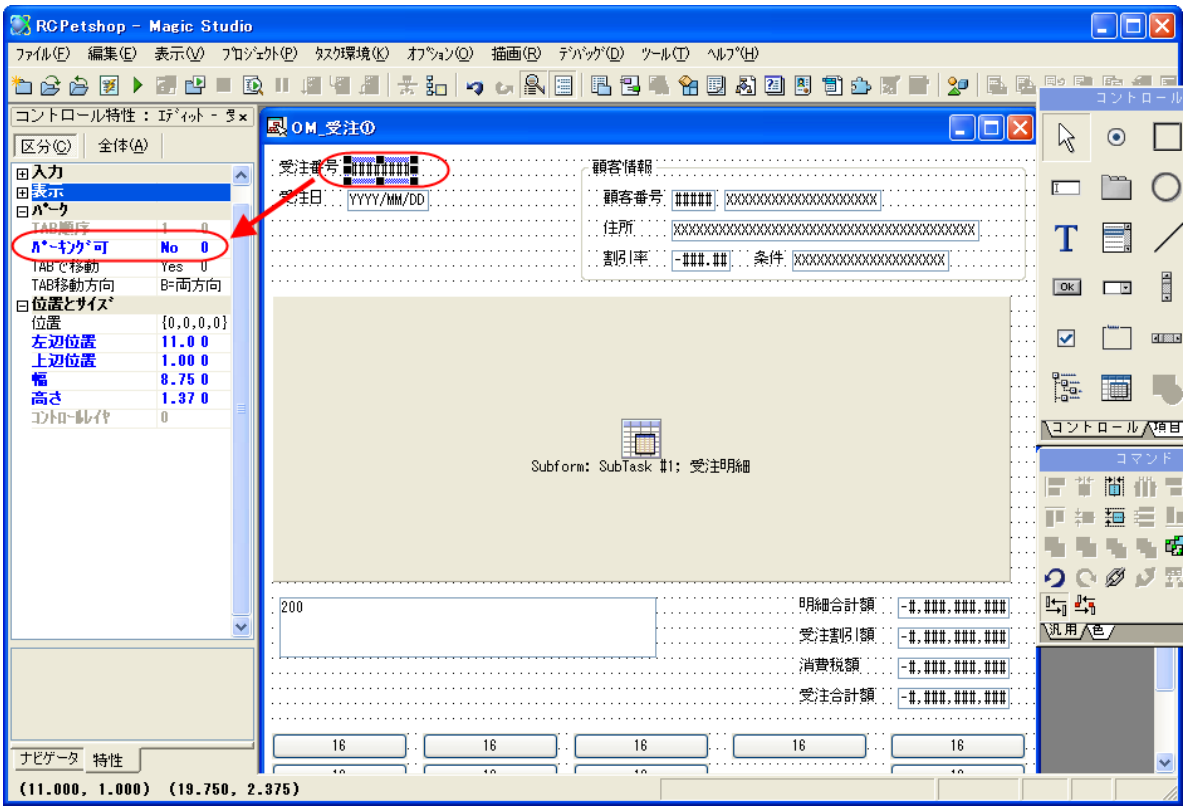
6.13 パークしない項目

実行時、画面上に表示されている項目には、カーソルがパークする項目とパークしない（すべきでない）項目とがあります。

例えば、サンプルプログラムでは、親タスクの「受注番号」、「住所」、「明細合計額」、「受注割引額」、「消費税額」、「受注合計額」等の項目にパークしないようになっています。

カーソルのパークの有無は、旧バージョンの Magic では、レコードメインの「セレクト」コマンドの「条件」によって制御していました。

V10 では、各コントロールの「パーキング可」特性で制御します。例えば、「受注番号」項目をパーク不可にするには、下図に見るように、フォームエディタで「受注番号」コントロールを選択し、「パーキング可」特性の値を No に設定します。



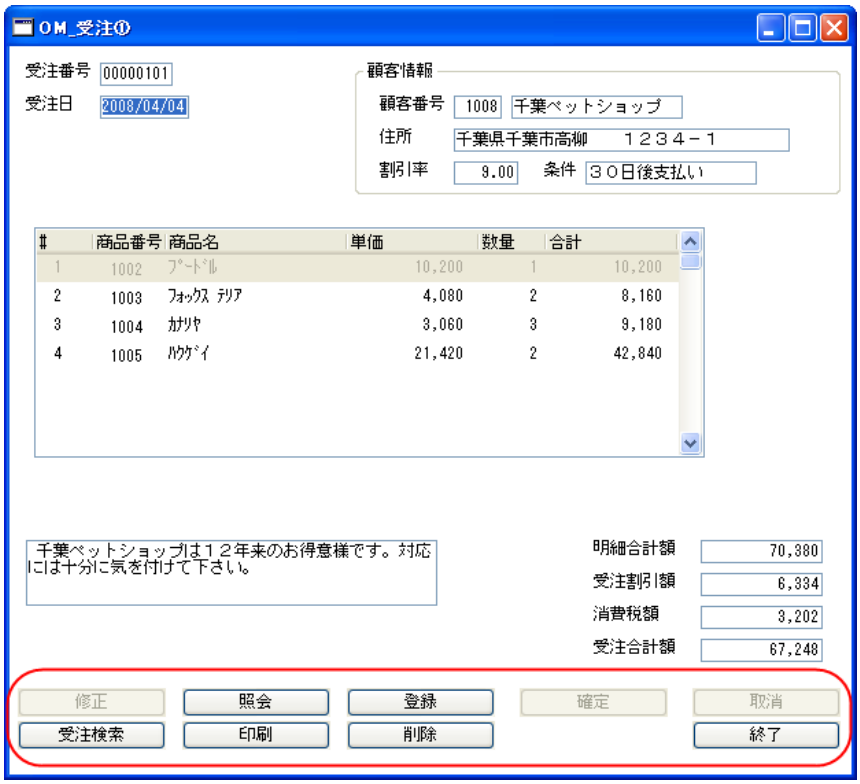
「パーキング可」特性は、式で指定することもできます。式で指定することにより、実行時にダイナミックにパークの可不可を制御することができるようになります。

「パーキング可」特性については、以下のキーワードでリファレンスヘルプを検索してください。

- パーキング可

6.14 プッシュボタンとイベント

サンプルアプリケーションでは、ユーザの操作の利便性のために、画面の下方にいくつかのボタンが配置されています。



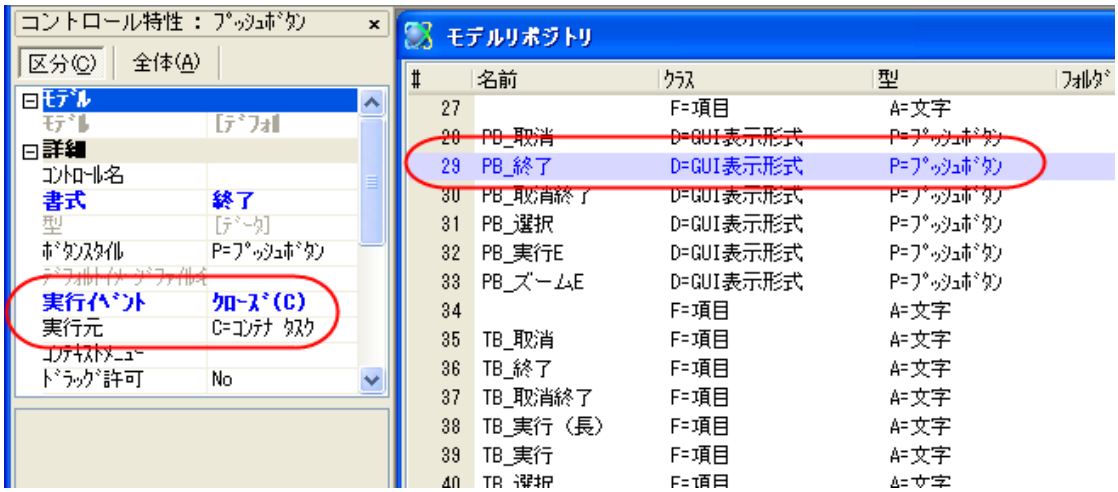
それぞれのボタンは、次のような機能を持っています。

ボタンラベル	機能	有効性
修正	修正モードに変わります。	すでに修正モードにあるときは無効。また、表示データに変更が加えられている場合には無効。（データ変更により無効になっている場合に有効にするには、確定、または取消を行ないます。以下同じ）
照会	照会モードに変わります。	すでに照会モードにあるときには無効。また、表示データに変更が加えられている場合には無効。
登録	登録モードに変わります。	すでに登録モードにあるときには無効。また、表示データに変更が加えられている場合には無効。
確定	データの登録・修正時に、データを確定（DBMS に書込み）します。	表示データに変更がされていないときには無効。
取消	データの登録・修正時に、データの修正内容を取り消します。	表示データに変更がされていないときには無効。
受注検索	受注一覧を表示し、ユーザが選択したら、その受注データを表示します。タスクのモードは変わりません。	登録モードの時には無効。また、表示データに変更が加えられている場合には無効。

印刷	現在表示されている受注データを印刷します。	登録モードの時には無効。また、表示データに変更が加えられている場合には無効。
削除	現在表示されている受注データを削除します。	登録モードの時には無効。また、表示データに変更が加えられている場合には無効。
終了	プログラムを終了します。データに修正が加えられている場合には、修正内容を DBMS に書き込みます。	

これらのボタンの機能が、どのように実現されているかについて、以下に説明します。

以下の説明では、ボタンの「実行イベント」特性に種々のイベントが設定されていますが、サンプルではすべてモデルリポジトリで設定されていて、変数を通して継承するようになっています。

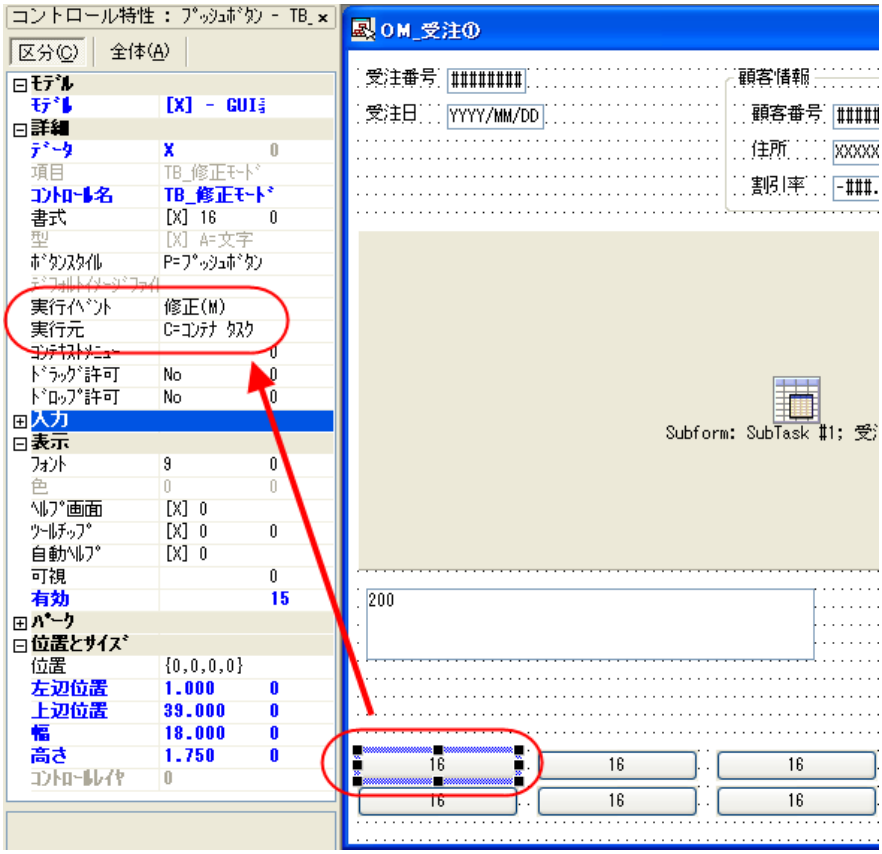


6.15 タスクモードの変更ボタン

「修正」「照会」「登録」ボタンは、それぞれ、タスクモードを変更するボタンです。

タスクモードを変更させるには、ボタンの「実行イベント」として、それぞれのタスクモード変更のための内部イベントを設定してやるだけで済みます。

右図は、「修正」ボタンの特性を、フォームエディッタ上で表示させたところです。ここでわかるように、「修正(M)」という内部イベントが設定されていますので、このボタンを押すと、タスクモードが修正モードに変わります。



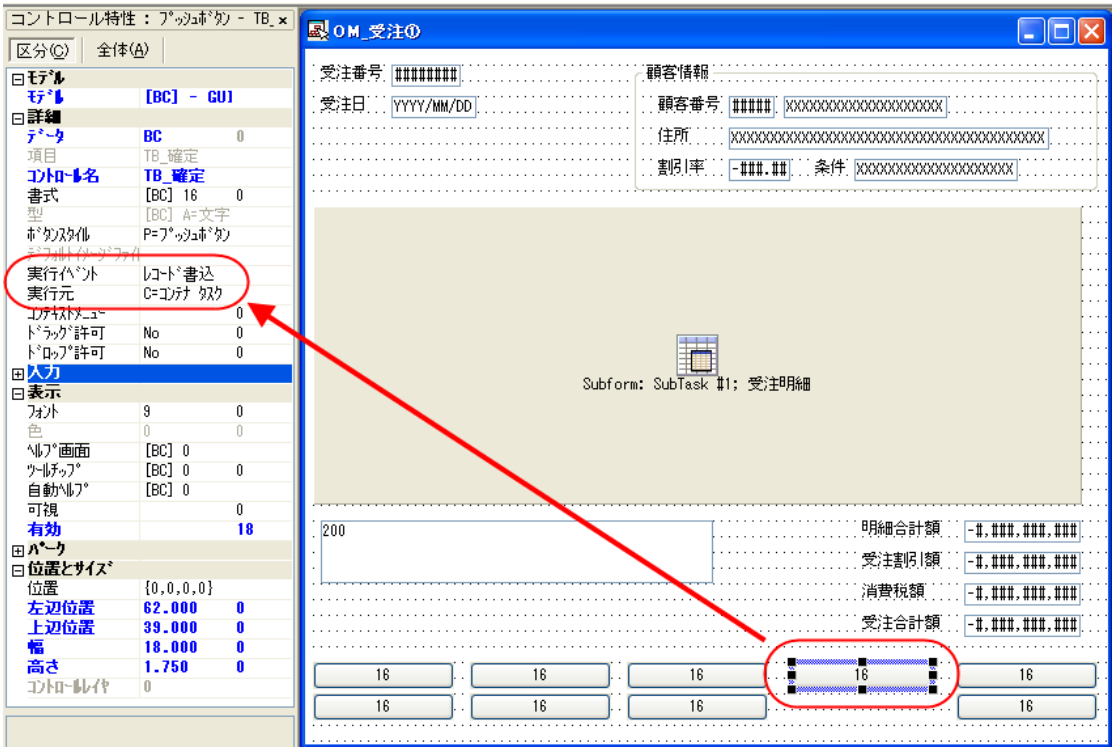
他、「照会」「登録」ボタンも同様です。

タスクモードを変更する内部イベントについては、以下のキーワードでリファレンスヘルプを検索してください。

- タスクイベント

6.16 入力の確定

ユーザ入力したデータを確定し、DBMS に書き込みたい場合には、「確定」ボタンを押します。
「確定」ボタンには、内部イベント「レコード書込」イベントが設定されています。



「レコード書込」内部イベントにより、次のことが起こります。

1. 現在カーソルがある項目から後ろにある項目のコントロール検証ハンドラが実行されます。
2. レコード後処理を実行します。
3. データベースにレコードを書き込みます。
4. 修正モードになって、データベースからレコードを読み直します。
5. 同じレコードのレコード前処理を実行します。
6. 最初にパークしていたカラムにカーソルが移動します。(この際、先頭からこの項目までの間のコントロール検証ハンドラが実行されます。)

このように、入力したレコードが DBMS に書込まれ、確定します。

「レコード書込」イベントを使うと、レコード書込み後、修正モードになります。
登録モードで連続して登録したい場合には、「レコード書込」ではなく、「次画面」内部イベントにしたらよいかもしれません。この場合、PgDn キーを押したのと同様、レコードを書き込んだ後、次の受注レコードに進みます。登録モードの場合には、現在のレコードを書込み、新しく登録モードでレコード入力を受け付ける状態になります。

「レコード書込」内部イベントについては、以下のキーワードでリファレンスヘルプを検索してください。

- レコード書込

6.17 入力データの取り消し（取消ボタン）

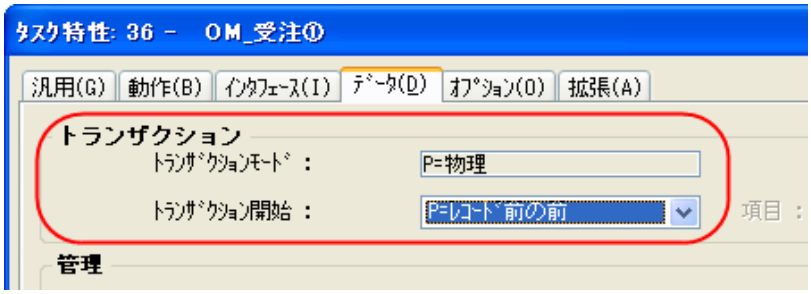
入力データを一度に全部、つまり明細行も含めて、取り消ししたい場合があります。このときのために、「取り消し」ボタンを設けています。

Magic には、「キャンセル(C)」という内部イベントがありますが、ここでの「取り消し」には使えません。「キャンセル(C)」内部イベントは、レコード単位でユーザの入力を取り消すことができるだけで、明細行まで含めた受注データの全体を取り消すことができないからです。

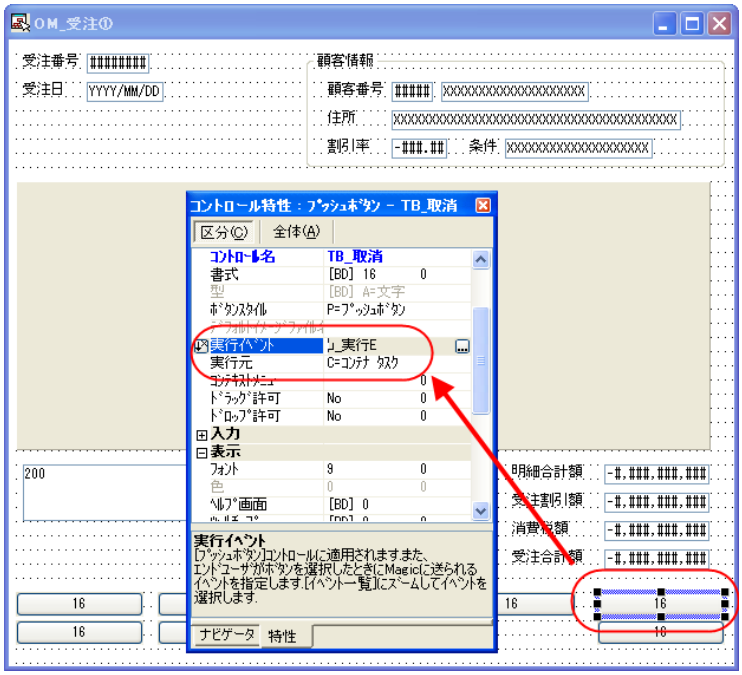
明細行も含め、全ての入力内容を取り消すために、次のように、トランザクションのロールバックを利用しています。

ひとつの受注レコードの処理を、トランザクションでくくります。

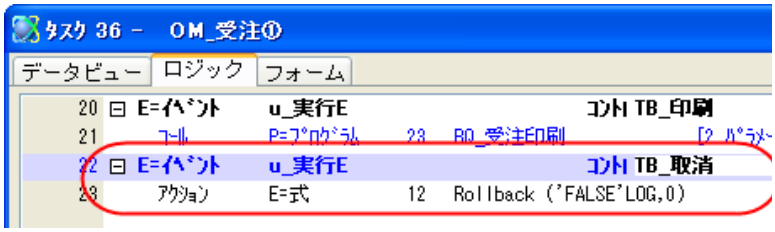
具体的には、ヘッダタスクにおいてレコードレベルのトランザクションを設定するために、「タスク特性 ⇒ データ(D)タブ」において、「トランザクション開始」を「P=レコード前の前」に設定します。



フォームエディッタでは、「取り消し」ボタンに、ユーザアクション「U_実行E」を設定します。



それに対するイベントハンドラにおいて、Rollback() 関数を用いてトランザクションをロールバックします。

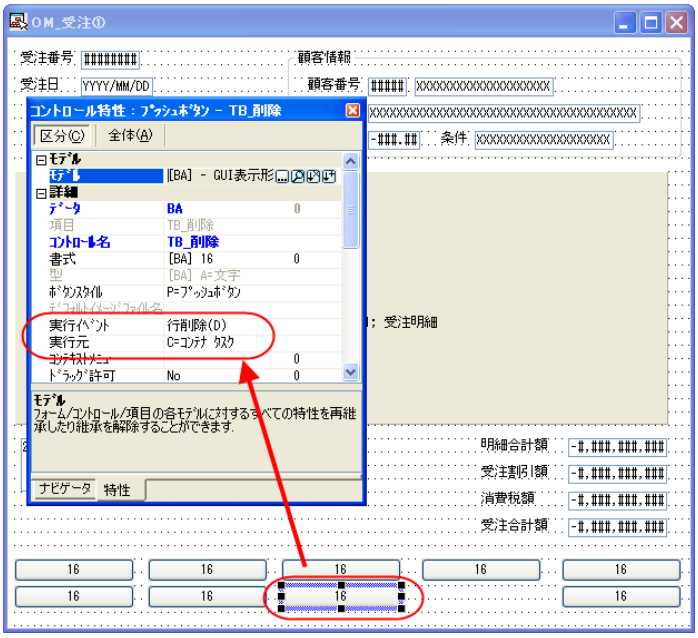


こうすることによって、トランザクション内の変更内容が、サブタスクの明細行の内容も含め、一度にすべて取り消すことができるようになります。

6.18 レコードの削除（削除ボタン）

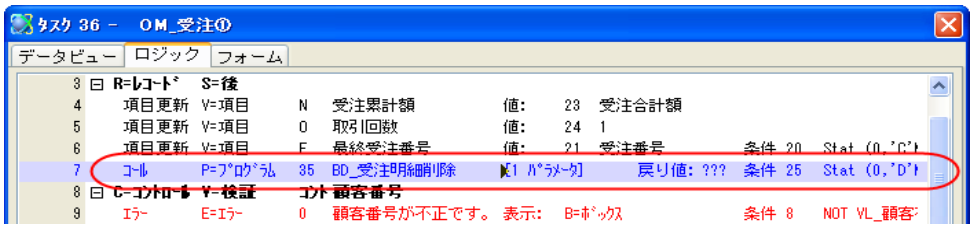
「削除」ボタンを押すと、現在表示中の受注データが、明細行も含めて削除されます。

Magic のプログラムでレコードを削除するには、「削除(D)」内部イベントを利用します。受注入力タスクでも、受注データを削除するために、「削除」ボタンに「削除(D)」内部イベントが設定されています（下図）。

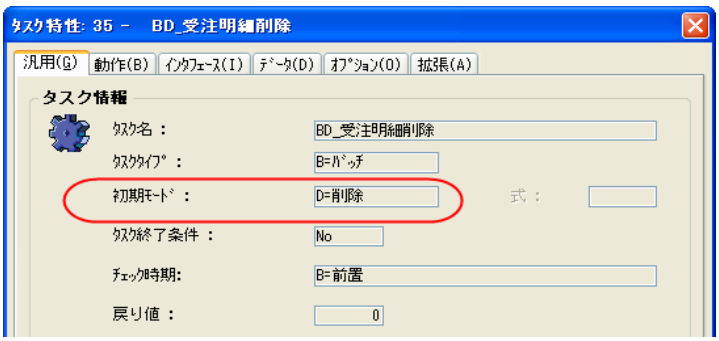


ただし、ヘッダ明細型のデータ構造の場合には、ヘッダタスクで「削除(D)」内部イベントが発行されても、削除されるのはヘッダタスクのレコードだけで、明細レコードは削除されません。このため、ヘッダタスクのレコード後処理で、削除モードの場合に、明細レコードを削除するバッチタスクを呼び出す必要があります。

このコールコマンドは、削除モードの場合にだけ呼び出すように、条件として Stat (0, 'D'MODE) が設定されています。



削除用のバッチタスク（プログラム番号 35 番「BD_受注明細削除」）は、タスクモードが「D=削除」であるバッチタスクです。



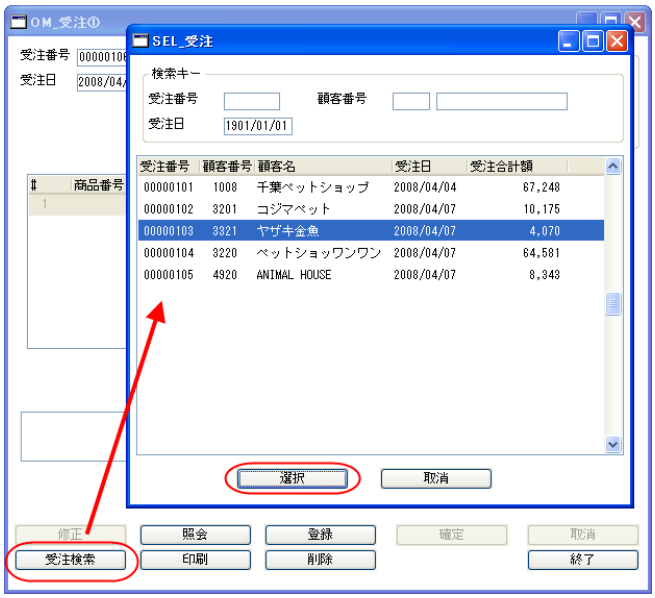
メインソースは「受注明細テーブル」で、受注番号をパラメータで受け取り、範囲指定されています。

また、このタスクでは、商品マスタをリンクして、各明細レコードで指定されている商品レコードの「在庫数」を調整しています。

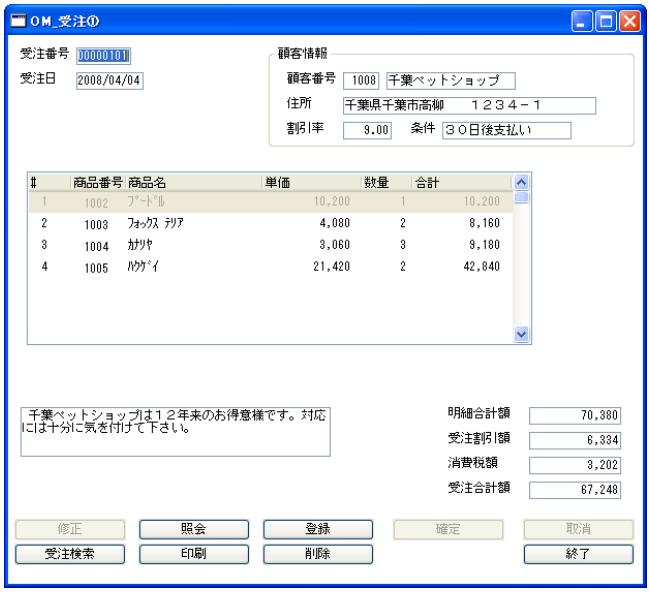
6.19 受注検索ボタン

「受注検索」ボタンを押すと、現在登録されている受注データを選択して、表示させることができます。

受注検索の機能では、照会モードあるいは修正モードの場合に、受注一覧画面を開いて、その中からユーザに選択させます。



すると、選択した注文の内容が表示されます。タスクのモードは以前のままです。



この機能を実現するには、「ビュー再表示」内部イベントを利用して、次のように設計します。

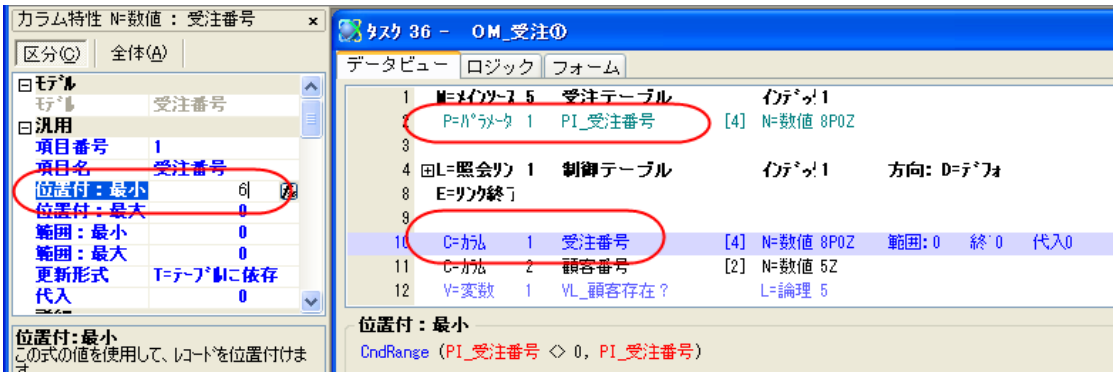
6.19.1 データビューの定義

1. ヘッダタスクに、受注番号指定のためのパラメータ項目「PI_受注検索」を定義します。

パラメータではなく変数項目でもかまいません。パラメータにすると、受注番号を指定してこのプログラム呼び出す、という使い方も可能になりますので、ここではパラメータにしました。

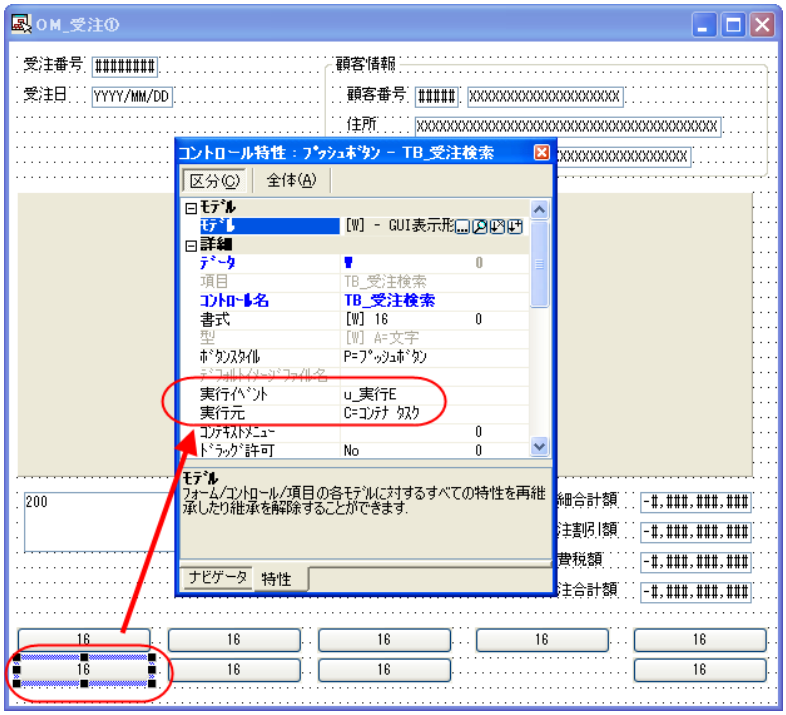
2. このパラメータを使って、受注テーブルの受注番号を位置付けします。

パラメータを指定せずに呼び出されることも可能にするため、位置づけ式としては、「CndRange (PI_受注番号 <> 0, PI_受注番号)」とします。



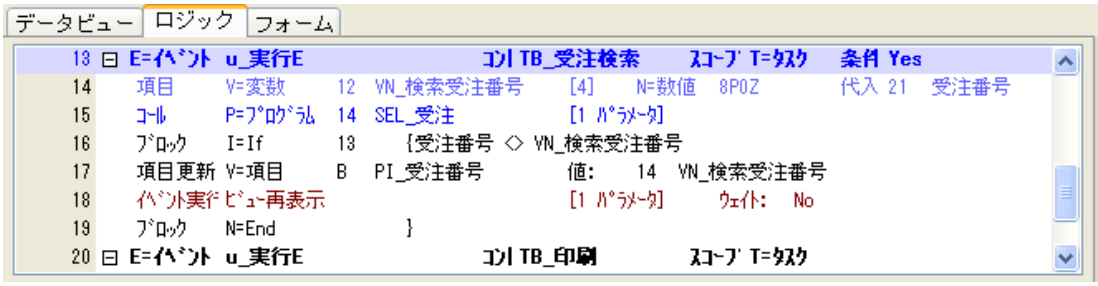
6.19.2 ボタンの定義

ボタンの「実行イベント」特性には、ユーザイベント「u_実行 E」を設定します。



6.19.3 イベントハンドラの定義

ロジックエディタで、このイベントに対するイベントハンドラを定義します。



ここでは、次のことを行います。

1. 受注選択プログラム（プログラム 14 番「SEL_受注」）を呼び出し、新しい受注番号をユーザに選択させます。
2. 新しい受注番号を、パラメータ PI_受注番号 に設定します。
3. パラメータとして 1 を指定して、「ビュー再表示」内部イベントを発行します。（ウェイトは No）。

このようにすると、受注番号の選択と位置づけができるようになります。

6.19.4 「ビュー再表示」イベント

ビュー再表示イベントには、数値パラメータをひとつ与えることができます。ここでは、パラメータとして 1 を与えていますが、パラメータとして 1 を与えると、タスクの位置づけ指定に基づいて再位置づけを行うようになります。

このタスクでは、PI_受注番号 を使って位置づけ指定がされているので、ビュー再表示イベントを実行することにより、一覧で選択した受注番号に位置づけられて受注データが表示されるようになります。

「ビュー再表示」内部イベントおよび CndRange 関数についてのより詳しい情報は、以下のキーワードでリファレンスヘルプを検索してください。

- ビュー再表示
- CndRange

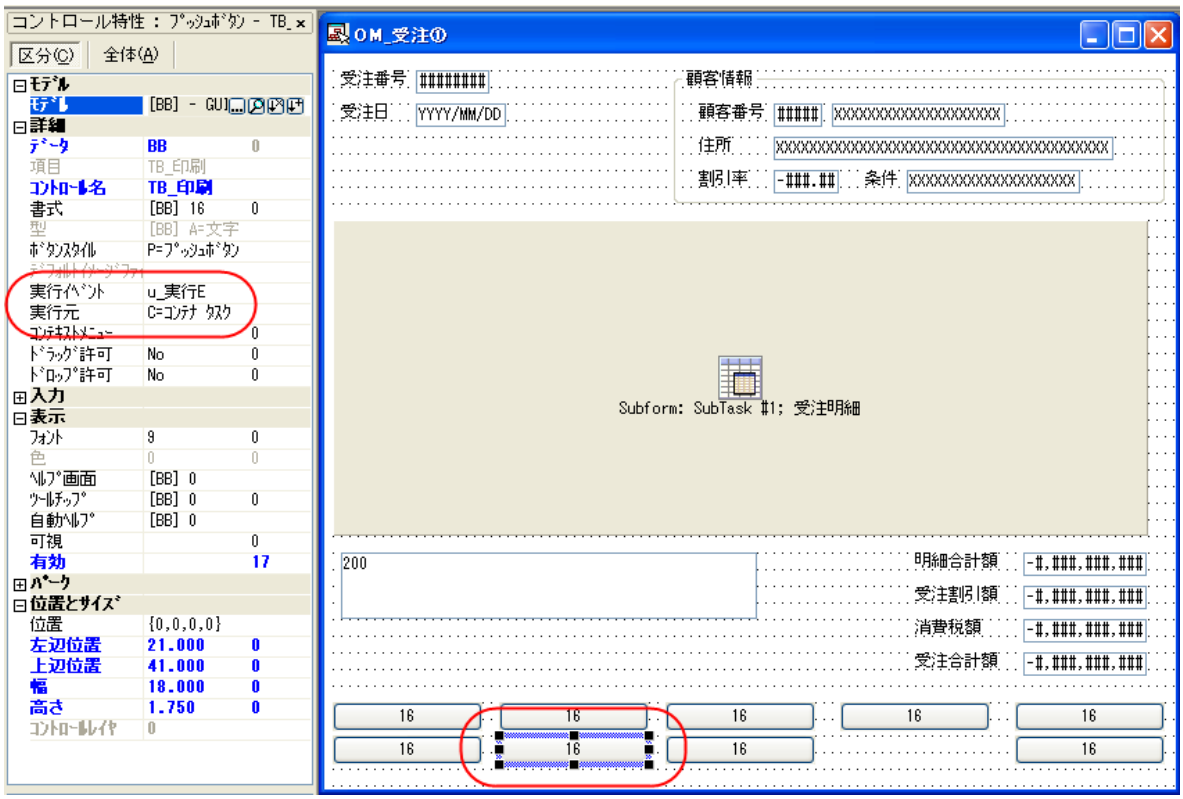
6.20 印刷ボタン

「印刷」ボタンを押すと、現在表示中の受注データがプリンタに印刷されます。

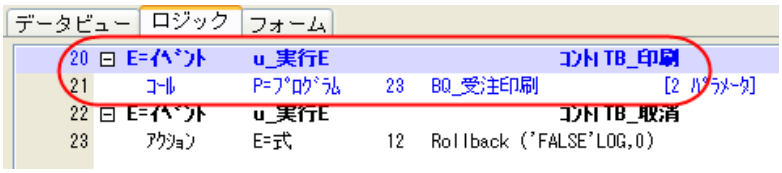
印刷については、印刷用のバッチプログラムがすでに用意されていれば、ボタンを押したらそのプログラムを呼び出すようにハンドラを定義するだけで済みます。

ここでは、印刷バッチプログラムそのものについての解説は省略します。

1. フォームエディッタでは、「印刷」ボタンに「u_実行 E」イベントを設定します。



2. ロジックエディッタで、これに対応するハンドラを定義し、印刷バッチプログラムを呼び出します。



印刷を行う場合には、表示データに修正が加えられていない状態でなければなりません。受注入力プログラムで入力途上の状態では、入力・修正が完了しておらず、DBMS に書込みがされていないからです。印刷プログラムは、DBMS 中のレコードを参照して印刷を行いますので、未確定の修正データは印刷されません。

もし、入力・修正途中であっても、表示されているデータの印刷を可能にするには、次のいずれかの方向で、プログラムを組みなおす必要があります。

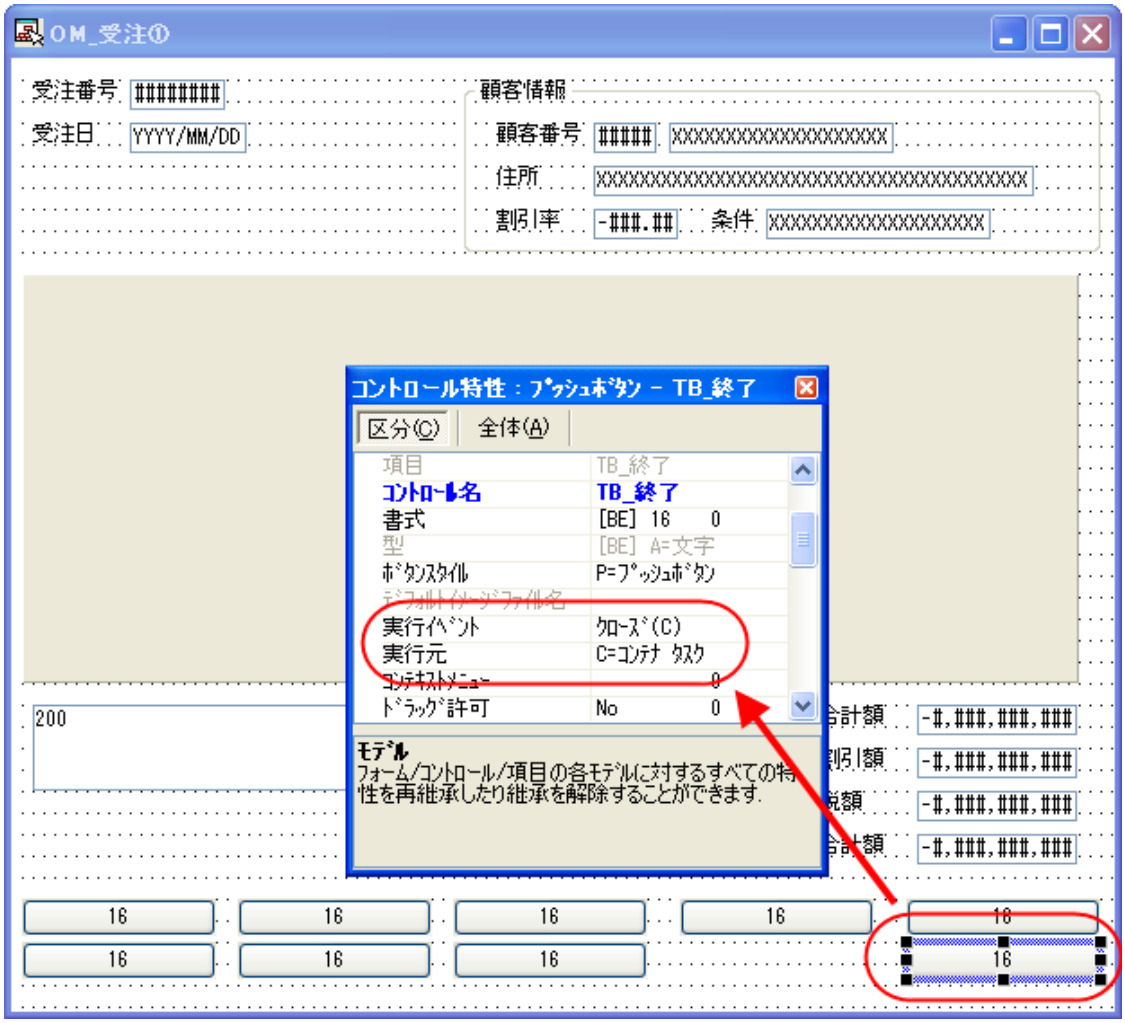
- 「レコード書込み」イベントなどを使って、いったんデータを DBMS に書き込んだ上で、印刷プログラムを呼び出すように、プログラムを書き直す。
- 修正途上のデータをパラメータとして印刷プログラムに渡すようにする

本書では、これらの方法についての詳細を省略します。

6.21 タスクの終了（終了ボタン）

「終了」ボタンを押すと、受注入力プログラムが終了します。もしこの時点で、表示データに修正が加えられていたら、DBMS にその修正が書き込まれます。

「終了」ボタンは、単に「ｸﾛｰｽﾞ(C)」イベントを発行するように設定しておくだけで OK です。



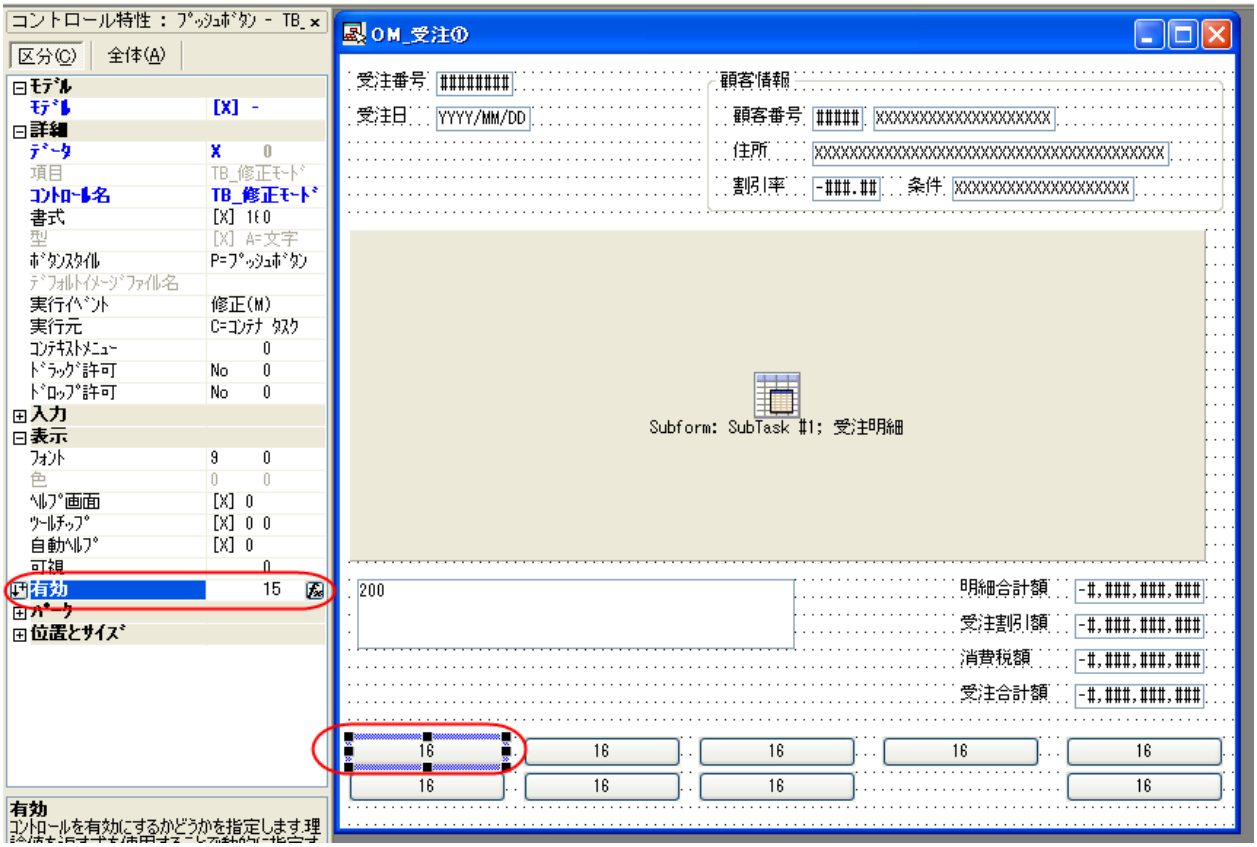
6.22 ボタンの無効化

前節までに説明したボタンは、常に有効にしておくのではなく、状況によって無効化しておきたい場合もあります。

例えば、タスクが修正モードになっている場合には、「修正」ボタンは有効化する必要はありません。また、登録モードでユーザが何も入力をしていない状態では「確定」ボタンや「取り消し」ボタンも無効にします。

6.22.1 ボタンの有効性の設定

ボタンの有効性は、フォームエディッタでのボタン特性において、「有効」特性によって制御します。この値は論理値の式で設定します。式の結果が真であればボタンが有効になります。偽であればボタンは無効化され、表示はグレー文字で表示され、また、ボタンを押しても「実行イベント」で指定されたイベントは発生しません。



6.22.2 各ボタンの有効性の判断

サンプルでは、次の二つの状態の組み合わせによって、各ボタンの有効・無効を判断しています。

- タスクモード（照会、修正、登録）
- 入力状態（未修正、修正あり）

登録モードでは、「入力状態」が「修正あり」だった場合に、さらに細分化して、正しい受注データとして必要最小限の項目が入力されたかどうか？も区別しています。

各ボタンの有効性と、タスクモード・入力状態の関係をマトリクスでまとめたのが次の表です。

ボタン	入力状態			タスクモード		
	未修正	修正あり	必要最小限	照会	修正	登録
修正	✔			✔		✔
照会	✔				✔	✔
登録	✔			✔	✔	
削除	✔				✔	
確定		✔（修正ﾓｰﾄﾞ）	✔（登録ﾓｰﾄﾞ）		✔	✔
取り消し		✔			✔	✔
受注検索	✔			✔	✔	
印刷	✔			✔	✔	

例えば、「修正」ボタンについて、入力状態が未修正であり、かつ、照会あるいは登録モードの場合に有効となります。

また、別の例としては、「確定」ボタンは、修正モードにおいて表示データに修正が加えられた状態、あるいは、登録モードにおいて必要最小限の入力内容が入力された場合に、有効となります。

6.22.3 状態の判定

前述の入力状態およびタスクモードは、Magic の内部で具体的にどう判定されるでしょうか？

タスクモードは、Stat 関数を使って簡単に判定することができますので、問題はないと思います。

入力状態については、ViewMod 関数を使って判定するのが比較的簡単です。

ただし、ViewMod 関数は、登録モードのときに、意図したような結果を返さないことがあるので、注意が必要です。

例えば、「受注日付」項目では、「今日」の日付を登録時のデフォルトとするために、「代入」式に「 Date() 」が設定されいてますが、このような場合には、ユーザが何も入力していない初期状態でも、Viewmod 関数は True を返します。これは、「デフォルト値と異なる」という意味で、「入力がされた」と判定されているようです。

また、サンプルでは使っていませんが、タブなどを使っている場合には、タブの切り替えを行うとタブ変数が変更されたとみなされて、ViewMod 関数が True になってしまいます。

このため、登録モードの場合に限り、アプリケーションの仕様を基にして、フラグをセットしてやることが必要になります。

具体的には、サンプルでは次のようになっています。

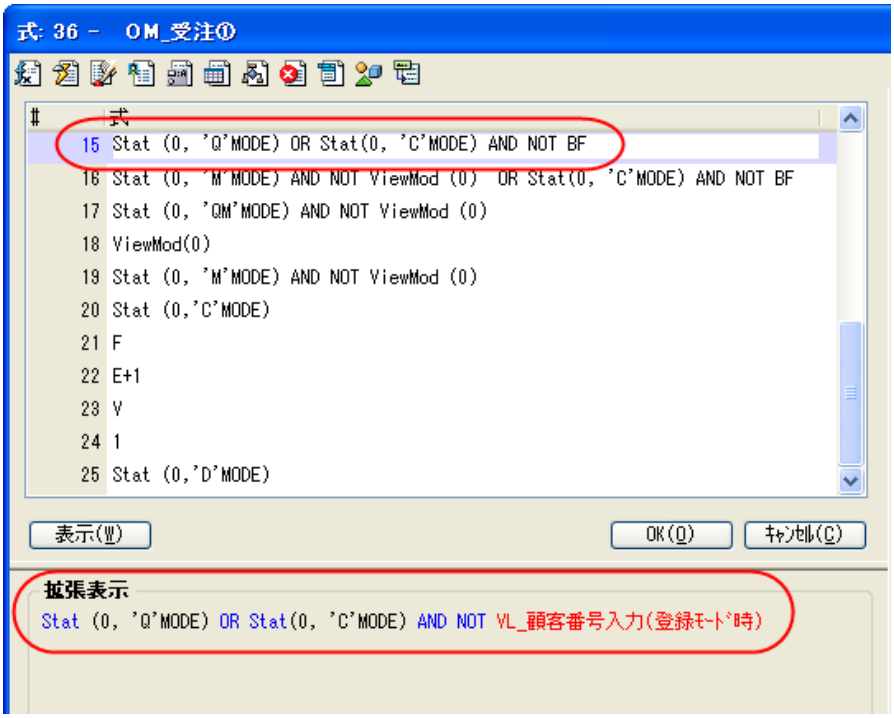
- 顧客番号が正しく設定されたら、「入力開始」とみなす。
- 明細行で、商品番号が正しく設定されたら、「必要最小限のデータが入力された」とみなす。

この考え方に従って、Magic で式を定義すると、例えば、「修正」ボタンの「有効」条件を判定する条件式は、

```
Stat (0, 'Q'MODE) OR Stat (0, 'C'MODE) AND NOT BF
```

というような式になります。ここで、BF というのは、「顧客番号が正しく設定されたか？」を判定するフラグ変数

であり、初期値は False、顧客番号のコントロール検証で True に設定されます。



6.23 トランザクション

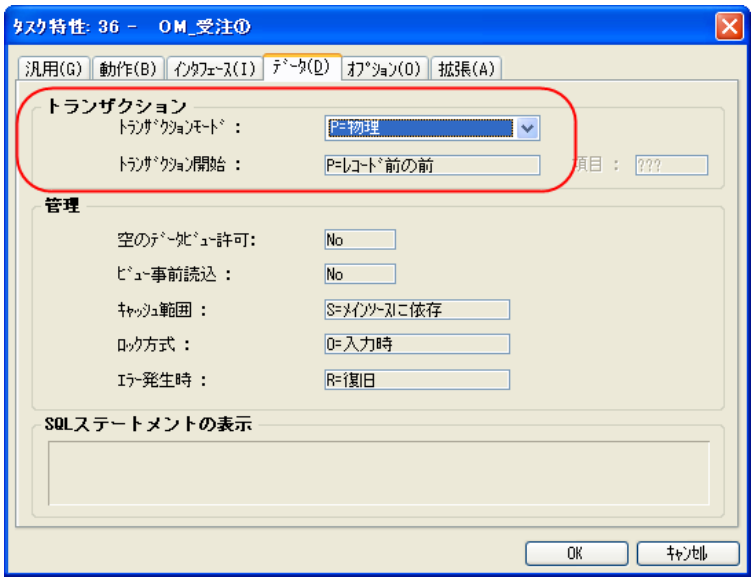
受注入力でのトランザクションの単位は、ひとつの受注伝票です。従って、トランザクションは、ヘッダタスクでのレコードレベルとなります。

基本形では、トランザクションとして物理トランザクションを使っています。

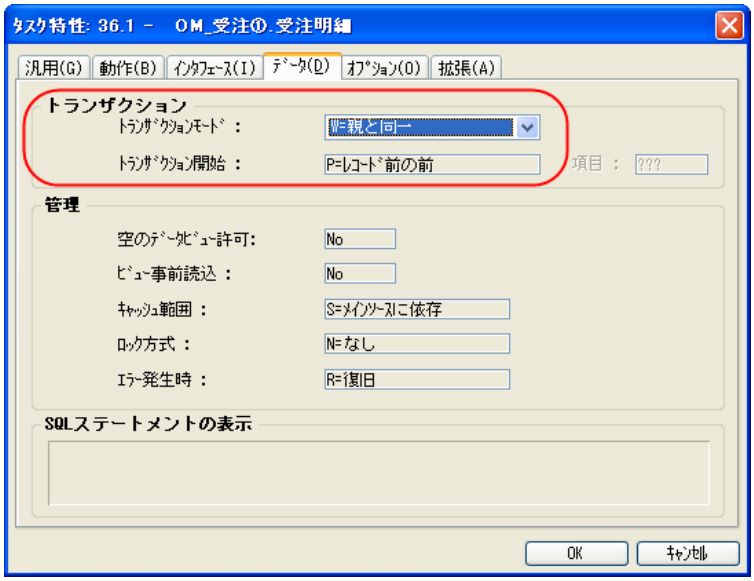
受注入力のようなヘッダ明細構造のタスクでは、ヘッダタスクにおいてレコードレベルのトランザクションを設定します。すなわち、タスク特性の「データ(D)」タブにおいて、

- トランザクションモード：P=物理
- トランザクション開始：「P=レコード前の前」

を設定します。



明細タスクでは、親タスクで開いたトランザクションの中で動作するように、トランザクションモードを「W=親と同一」にします。この場合、「トランザクション開始」の設定は無視されます。



6.24 テーブルのオープン

6.24.1 Magic におけるテーブルの「オープン」

SQL DBMS においては、「テーブルのオープン」という概念はありません。SQL 文を使えば、アクセス権限がある限り、どのようなテーブルでもアクセスすることができます。

しかし、Magic の実行エンジンにおいては、ISAM 系の Pervasive (Btrieve) をもとに発展してきた経緯があり、また、内部的にも、テーブルを利用するための内部情報の準備などの処理が必要になります。このような処理を行うために、「テーブルのオープン」という概念があります。

「テーブルのオープン」という概念は、さらに、テーブルの排他制御を行うためにも使われます。すなわち、テーブルのオープン時には、

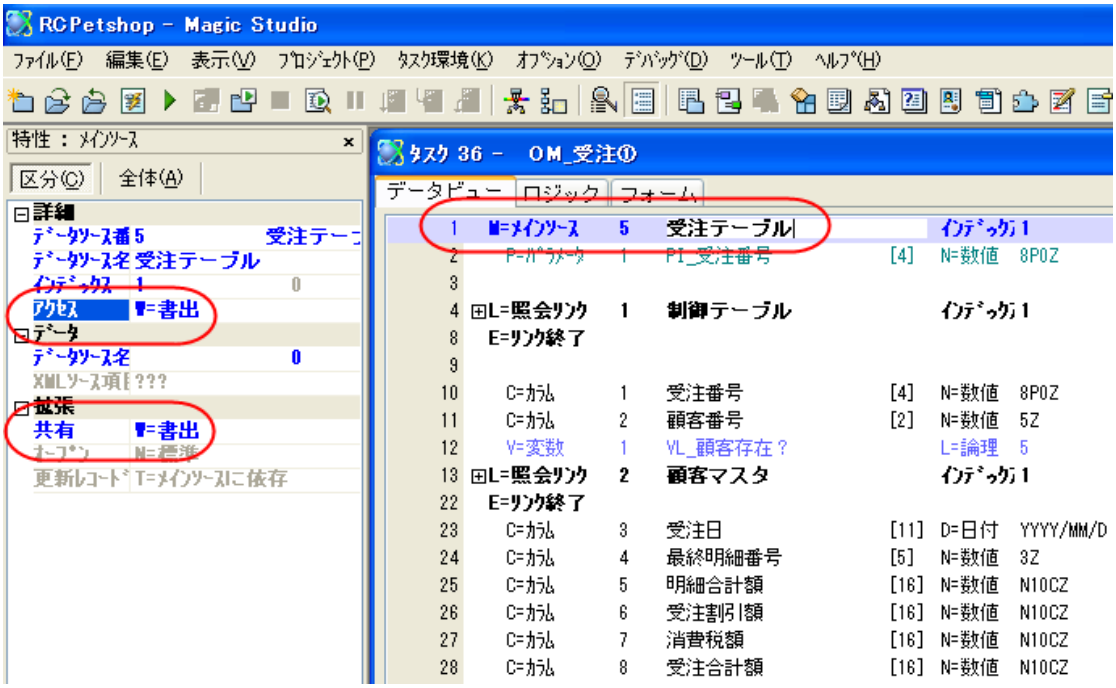
- このテーブルをこのタスクで読み込み専用とするか、あるいは読み書き両用を許すか、という「**アクセスモード**」
- 他のユーザにどのようなアクセスモードでの利用を許可するか、を指定する「**共有モード**」

の設定が行えます。これらのモード設定特性は、まとめて「**テーブルモード**」と呼びます。

6.24.2 テーブルモードの設定

テーブルモードの設定は、通常「データビュー」において、メインソースあるいはリンクの特性として、開発者が設定します。

例えば、下図は受注明細タスクにおいて、ヘッダタスクのメインソースである「受注テーブル」の特性を表示しているところです。



特性シートの中で、「アクセス」と「共有」という特性があり、この場合にはいずれも「W=書出」になっています。これは、それぞれ、

- アクセス ＝「W=書出」：この「受注テーブル」というテーブルが、このタスクで修正される可能性がある
- 共有 ＝「W=書出」：他のユーザも書出モードでアクセスしてもよい

ということを意味しています。

同様な特性設定が、「照会リンク」行（上図の 4 行目、13 行目）でも設定することができます。

テーブルモードは、テーブルを単位とした排他制御にも用いられます。排他制御の話題は多岐にわたり、また、利用している DBMS によっても機能に違いがあるため、本書ではこの話題についてこれ以上は説明しません。

6.26「複数ユーザ利用時の問題点」で、一般的な排他制御に関するトピックについて、種々の関連項目を参照していますので、それらの項目も参照してください。

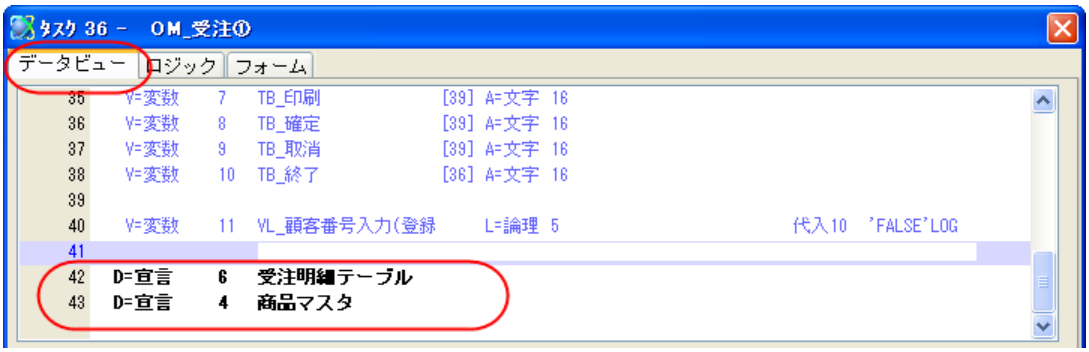
6.24.3 先行オープン

親子タスクの場合、次のような理由により、サブタスクで使うテーブルを親タスクであらかじめオープンしておきたいことがあります。

- トランザクションを親タスクで始める場合、トランザクション中で使うテーブルをあらかじめオープンしておいた方がよいため。（Pervasive を使う場合は、これが必須となります）
- オープン処理には、Magic 内部でオーバーヘッドがあるので、頻繁にサブタスクが呼び出される場合には、サブタスクが呼び出されるたびにオープン処理が行われると、遅くなることがあるので。

タスクでメインソースとしてもリンクとしても参照されていないテーブルを、あらかじめオープンしておくには、データビューエディタで「D=宣言」行を定義します。

本書でのサンプルプログラムでは、明細タスクで使う「受注明細テーブル」、「商品マスタ」が、ヘッダタスクで先行してオープンされています。



ここでは、テーブル6 とテーブル4とを先行オープンしておきましたが、後の章で説明する種々のバリエーションによっては、オープンするテーブルが異なることがあるので、サブタスクで使っているテーブルを確認して、「D=宣言」を定義してください。「D=宣言」で参照されているテーブルと、サブタスクで使われているテーブルとの照合は行われませんので、間違った設定をすると不要なテーブルがオープンされ、必要なテーブルがオープンされない、ということが実行時に起こります。

次の図は基本形のサブタスクで、テーブル6とテーブル4が使われています。

テーブルのオープンについてのより詳しい情報は、以下のキーワードでリファレンスヘルプを検索してください。

- 宣言定義
- テーブルモード

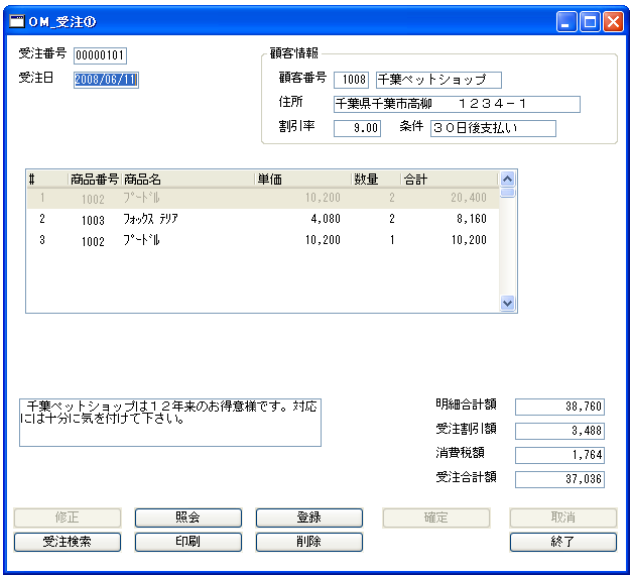
6.25 スクロール

受注画面を照会モードあるいは修正モードで、既存の受注データを表示しているとき、前の受注データ、あるいは後の受注データなど、別の受注データにスクロールしたくなることがあります。

基本形では、Magic のオンラインタスクが持っているスクロールの機能をそのまま使っています。具体的には、次のようになります。

初期状態で、先頭のデータが表示されます。

右図では、最初の受注レコード（受注番号 101 番）が表示されています。

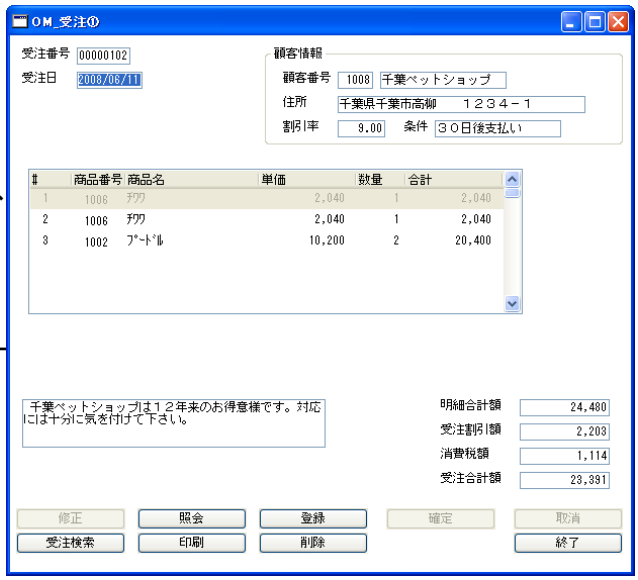


[PgDown] キーを押すと、次のレコード（受注番号 102 番）に移ります。

この状態で、[PgUp] キーを押すと、前のレコード（受注番号 101 番）に戻ります。

以下同様に、[PgDown] キーで次のレコードに移り、[PgUp] キーで前のレコードに戻ります。

また、[Ctrl+Home]キーで、先頭のレコードにジャンプします。同様に、[Ctrl+End]キーで、最後のレコードにジャンプします。



スクロールについては、Magic が本来持っている機能をそのまま使っているので、特別なプログラミングや設定などは必要ありません。

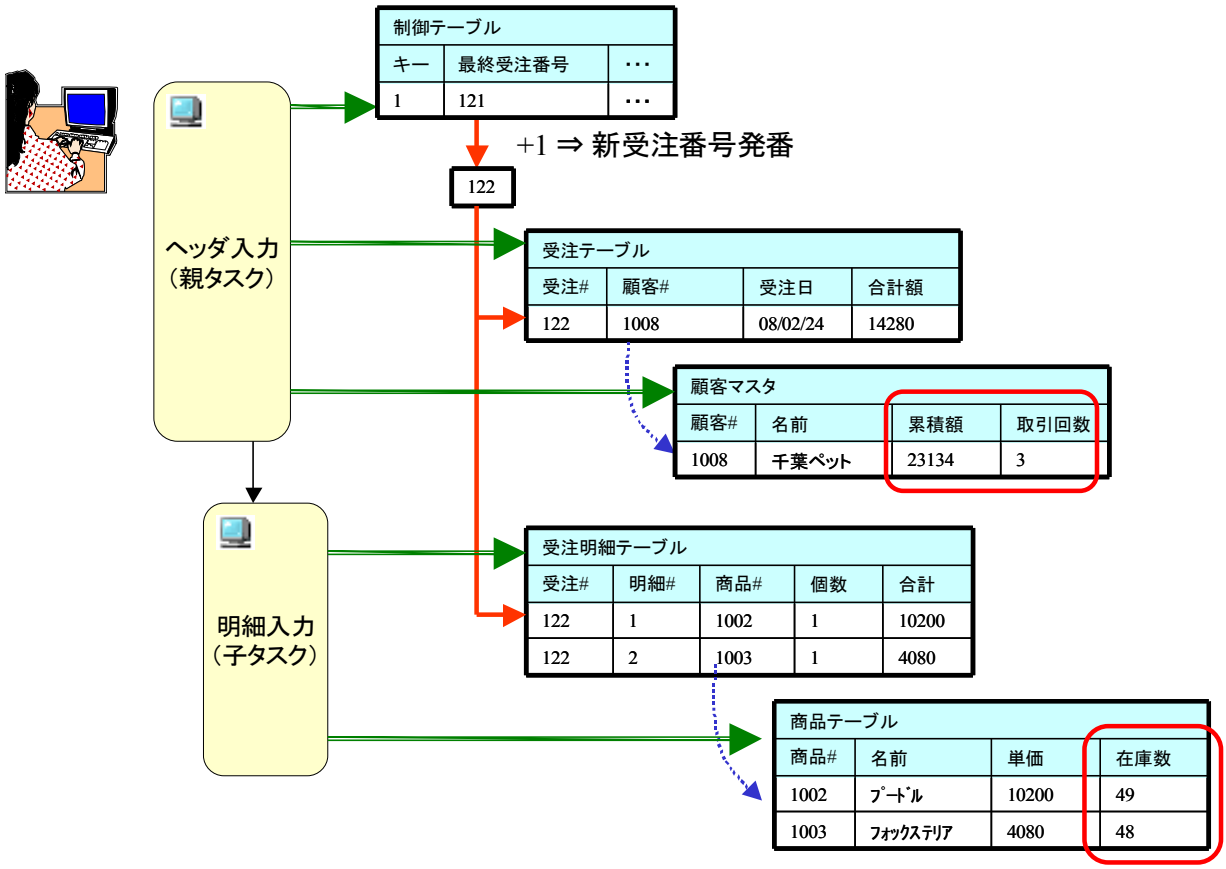
6.26 複数ユーザ利用時の問題点

以上、基本形において Magic の機能がいかに使われているかを説明してきましたが、この基本形で作ったプログラムは、複数のユーザが同時に受注入力を行う環境で、ロックの問題が起こります。

複数ユーザが同時に利用することを前提とすると、次のような要求事項が出てきます。

- 複数ユーザが同時に入力、修正、照会できること。
- データの整合性を正しく保つこと。例えば、
  - 受注番号は重複や歯抜けがなく、連続した番号が割り当てられること。
  - ヘッダ、明細の関連が保たれること。
  - データの依存関係が正しく維持されること（6.9「依存関係のある値を更新する」、6.12「累計値の更新」参照）
- ロック待ちは、あったとしても極力短時間に収まり、業務に支障を生じないこと。

下図は、基本形でのタスクと、そこで使われるテーブルとの関係を示した図です。本章の最初（37 ページ）で示したものと同じ図です。



ここでは、次のようなレコードがアクセスされます。

タスク	テーブル	レコード
親タスク	制御テーブル	キー ＝ 1
	受注テーブル	現在の受注番号のレコード（修正/照会モードのみ）
	顧客マスタ	受注顧客のレコード
サブタスク	受注明細テーブル	現在の受注番号を持つレコード
	商品マスタ	各明細行に指定されている商品のレコード

これらのレコードは、すべて アクセス＝書込み、共有＝書込み のモードでアクセスされるため、排他的なレコードロックがかかります。このため、2 人以上のユーザが使うと、次のようなロックの競合が起こります。

- 制御テーブル: キー ＝ 1 のレコードはひとつしかないので、一人が使い始めると他の人はロック待ちとなる。
- 顧客マスタ: 注文書がたまたま同一顧客からだった場合、顧客マスタのレコードのロックの競合が起こる。
- 商品マスタ: 注文された商品に同一商品が入っていたら、商品マスタのレコードのロックの競合が起こる。

特に、制御テーブルのレコードのロック競合のために、実質的には同時には一人でしか使えないということになります。

このような問題があるために、ロックの競合を避けるための工夫が必要となります。

マルチユーザ環境での適切な排他制御については、非常に多岐な話題になるため、本書では詳細には説明しません。

一般的なルチユーザ環境での考慮点については、リファレンスマニュアルの下記の項目を参照してください。

- マルチユーザ環境
- データ管理 ⇒ SQL に関する考慮事項 ⇒ 構成とパフォーマンス ⇒ ロック
- 設定 ⇒ 動作環境 ⇒ [マルチユーザ]タブ

また、排他制御機能の実際の動作については、利用している DBMS ごとに若干異なることがあります。各 DBMS に固有な事項については、リファレンスマニュアルの次の項目を参照してください。

- データ管理 ⇒ SQL に関する考慮事項 ⇒ Magic SQL データベース 以下

また、製品添付の README.CHM ファイルにも、DBMS に固有な情報があります。

- V10 追加情報 ⇒ データベース固有の追加情報 以下

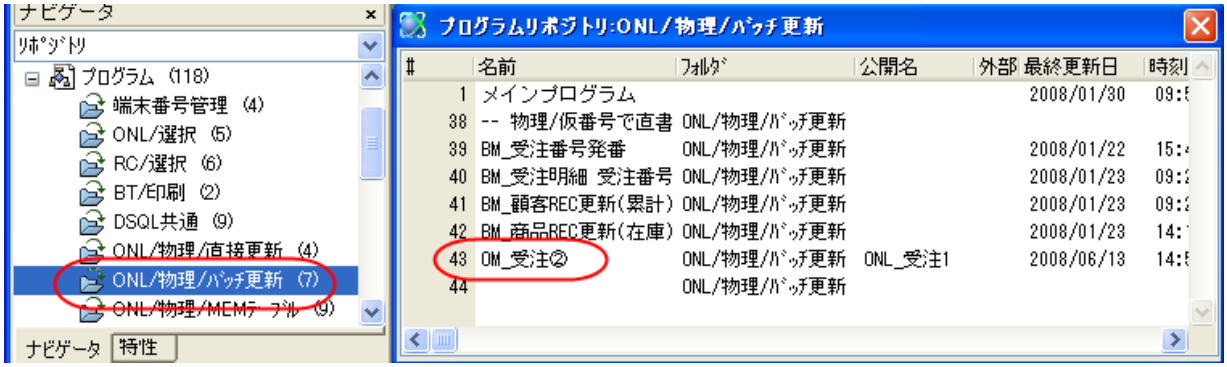
7 ONL/物理/バッチ更新

本章で説明する型は、基本形と同様、物理トランザクションを利用するオンラインプログラムですが、6.26「複数ユーザ利用時の問題点」で説明した、排他制御の問題を解決するための工夫をしたものです。

バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

このプログラムは、プログラムリポジトリで「ONL/物理/バッチ更新」というフォルダに格納してあります(右図)。受注入力プログラムは、プログラム 43 番「OM_受注②」であり、バッチタスクが 4 つ定義されています。



7.1 処理の概要

基本形での問題点は、ロックの競合が非常に頻繁に起こってしまう、ということでした。これを回避するためには、ロックをできるだけ短時間に抑え、ロックの競合による待ち時間が、実際上問題ないレベルに収まるように工夫が必要になります。

基本形では、リンクされているレコード（制御、顧客、商品レコード）がユーザの入力中の長い間ロックされたままになっていました。これを避けるためには、リンクされているレコードがロックされないようにする必要があります。

リンクされているレコードに対してのロックをさせないためには、リンクを「読込み専用」で行うようにします。具体的には、リンクコマンドの「アクセス」パラメータを「R=読込」に設定します。

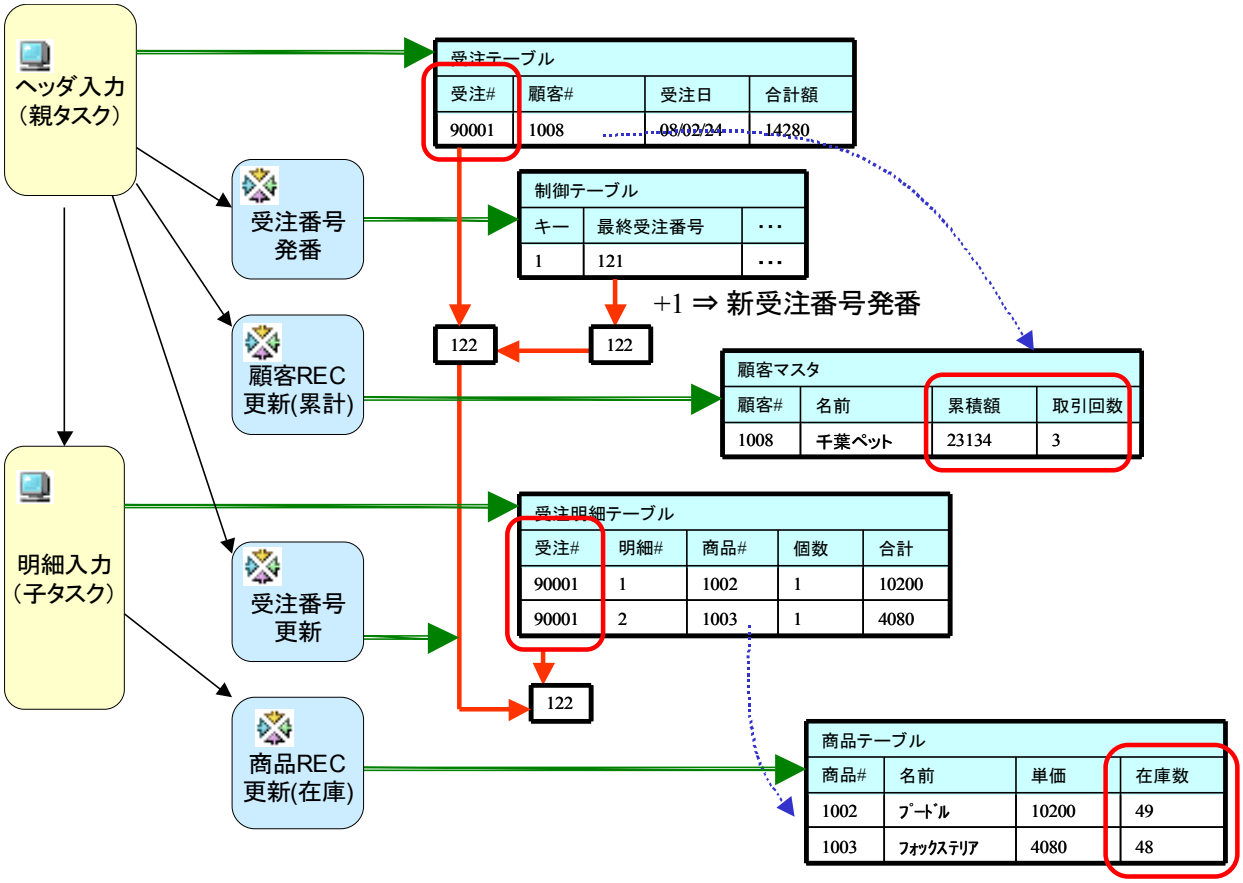
「アクセス」を「R=読込」にすることにより、ロックはかからなくなりますが、その代わり、データの更新もできなくなります。そうすると、レコード後処理で行っている累計データ（顧客マスタの受注累計額、および取引回数、商品マスタの在庫数、および制御テーブルの最終受注番号）を更新することができなくなってしまいます。

そこで、累計データの更新を行うため、レコード後処理で「項目更新」コマンドを実行して直接更新する代わりに、レコード更新用のバッチタスクを定義しておいて、このバッチタスクを呼び出すことにより更新を行うようにします。

このために、次のような修正用のバッチタスクを作成しました。

テーブル名	処理内容	修正用のバッチタスク
制御テーブル	制御テーブルのレコードを読み出し、最終受注番号に1を加え、新しい受注番号として、パラメータで返します。	BM_受注番号発番
顧客マスタ	顧客番号をパラメータとして受け取り、それをキーとして顧客マスタのレコードを読み取り、累計データ（受注累計額、取引回数）を更新します。	BM_顧客 REC 更新(累計)
商品マスタ	商品番号をパラメータとして受け取り、それをキーとして商品マスタのレコードを読み取り、在庫数を更新します。	BM_商品 REC 更新(在庫)

図にして示すと、次のページの図のようになります。


ヘッダ入力
（親タスク）
受注テーブル
受注#
顧客#
受注日
合計額
90001
1008
08/02/24
14280
受注番号
発番
制御テーブル
キー
最終受注番号
・・・
1
121
・・・
+1 ⇒ 新受注番号発番
122
122
顧客REC
更新(累計)
顧客マスタ
顧客#
名前
累積額
取引回数
1008
千葉ペット
23134
3
明細入力
（子タスク）
受注明細テーブル
受注#
明細#
商品#
個数
合計
90001
1
1002
1
10200
90001
2
1003
1
4080
受注番号
更新
122
商品REC
更新(在庫)
商品テーブル
商品#
名前
単価
在庫数
1002
ﾌﾟｰﾄﾞﾙ
10200
49
1003
ﾌｫｯｸｽﾃﾘｱ
4080
48

7.2 制御テーブルのレコードロック回避

基本形での排他制御で、一番基本的で深刻な問題は、「制御テーブル」へのレコードロックの競合です。制御テーブルには、レコードがひとつしかなく、これがロックされてしまうと、他のユーザがすべてロック待ちとなり、先に進めなくなってしまうからです。

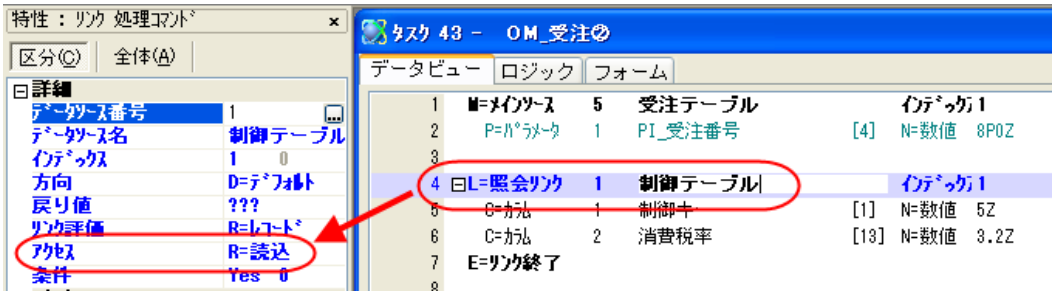
基本形において、制御テーブルのレコードにロックをかけなければならない理由は、登録時に新しい受注番号を発番するためでした。このレコードロックを回避するために、新しい受注番号を発番するロジックを、次のように変更します。

- 制御テーブルへのリンク: 制御テーブルへのリンクでは、「アクセス」＝「R=読込」として、ロックがかからないようにします。
- 仮受注番号: 最初は、仮受注番号を使って、レコードを登録していきます。仮受注番号は、各ユーザごとにユニークな値となるようにします。
- 正式受注番号: 確定時に、バッチタスクを使って新しい受注番号（正式受注番号）を発番させます。
- 明細行の受注番号: この際、明細行の受注番号も、この正式受注番号で置き換えます。

各々について、以下に説明していきます。

7.2.1 制御テーブルへのリンク

制御テーブルをリンクする際に、レコードロックを防止するため、「アクセス」パラメータを「R=読込」にします（下図）。



7.2.2 仮受注番号

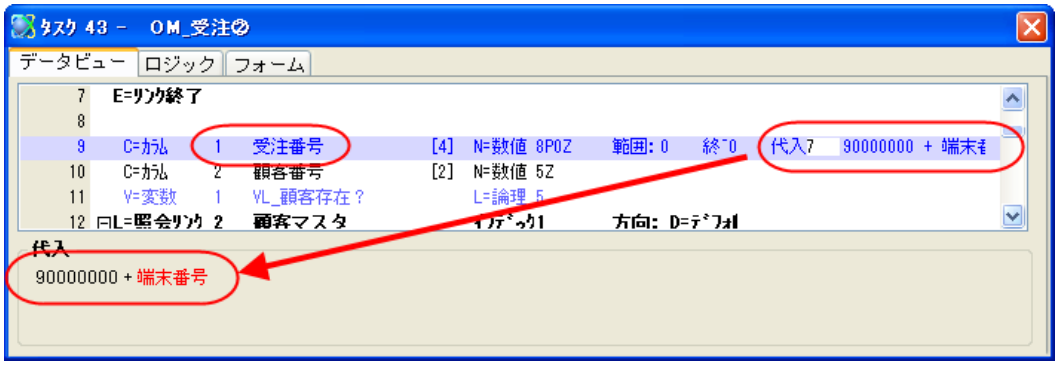
登録モードにおいて、仮受注番号を使ってレコードを登録してゆきます。

仮受注番号としては、次の 2 点が保障されなければなりません。

- 各ユーザごとにユニークになること。
- 既存の受注番号と異なること。

サンプルでは、各ユーザごとの「端末番号」に、実際の受注番号としては存在しないような非常に大きな数字 90000000 を加えた数を、仮受注番号としています。この値は、「受注番号」カラムの「代入」式で設定します。

この方式が正しく動作するには、端末番号が各ユーザごとにユニークであることが保障されなければなりません。この方法については、7.3「端末番号の割り当て」で説明します。

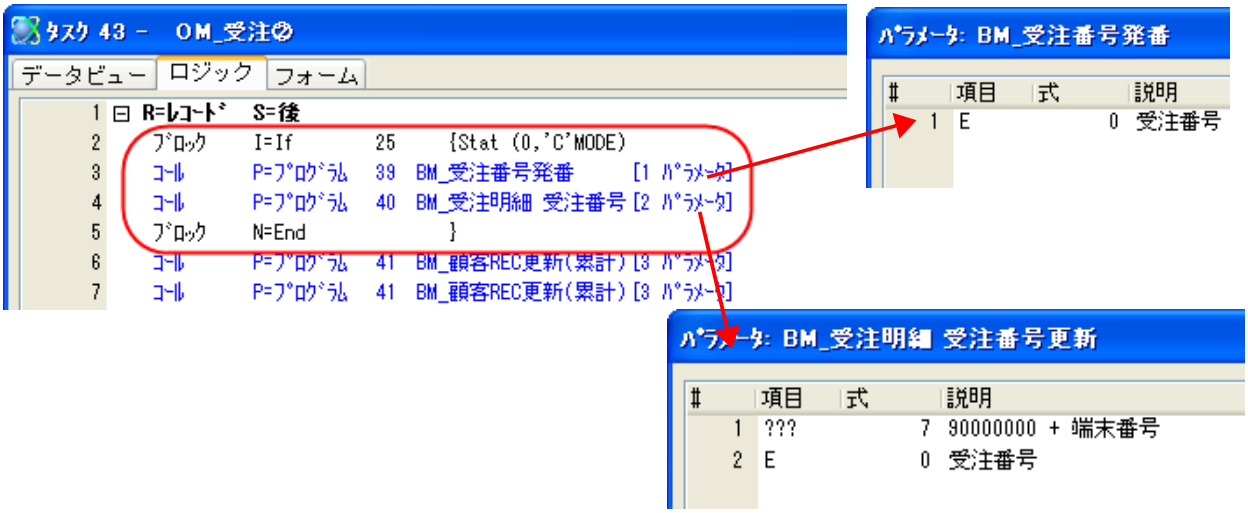


7.2.3 正式受注番号と明細行の受注番号

ユーザがひとつの伝票の入力を終え、確定（保存）する時には、仮受注番号を正式な受注番号に置き換えなければなりません。これには次のことを行います。

1. バッチタスクを使って新しい受注番号（正式受注番号）を発番させます。
2. 受注レコードの「受注番号」を、この正式受注番号で置き換えます。
3. 受注明細レコードの受注番号も、この正式受注番号で置き換えます。

この処理は、ヘッダタスクのレコード後処理で行っています（下図）。



- この処理は、登録モードのときにだけ行うので、全体をブロック If で囲んでいます（2 行目）。
- プログラム 39 番「BM_受注番号発番」は、正式受注番号を発番するバッチタスクです。このバッチタスクの処理内容は、
  - 制御テーブルの「最終受注番号」に1を加える。
  - その番号を、パラメータに設定して返す。

  という単純なものです。このタスクに、受注番号がパラメータとして渡されて、正式受注番号が設定されて返ってきます。
- プログラム 40 番「BM_受注明細 受注番号更新」は、仮受注番号と正式受注番号をパラメータとして受け取り、受注明細行の受注番号を、仮受注番号から正式受注番号に付け替えるものです。

このような処理にすることにより、制御テーブルへのレコードロックは、レコード後処理からレコードが更新されてトランザクションがコミットされるまでの、ごく短い時間に抑えることができるようになります。

7.3 端末番号の割り当て

7.2.2「仮受注番号」で説明したように、本章の方式では、各端末ごとにユニークな番号（端末番号）をもとにして仮受注番号を作成する必要があります。端末番号としては、連続している必要はないのですが、一定の範囲の数値の中で、各端末ごとに重複がないことが保障されていなければなりません。万一、二つの端末に同一の端末番号が振られると、別のユーザの一時データが混同されてしまい、正しい処理が行われません。

従来は、各 PC の起動用バッチファイルに、Magic 実行版へのコマンドラインパラメータとして、/TERM=n を指定し、プログラムからは Term() 関数を使って取得する、という方法がよく使われていました。しかしこの方法は、次の点で好ましくありません。

- システム管理者が、各端末ごとに違う番号を割り振って個々に設定しなければならない。これは管理者の負担になると同時に、手作業による誤設定に起因する誤動作の原因となる。
- リッチクライアントでのシステムにしようとすると、ひとつのサーバインスタンスが複数のユーザを担当して処理するので、Term() 関数の結果がすべて同じになってしまい、ユニークな番号の設定を行うことができない。

このことから、プログラムを使って端末番号を自動的に割り振るアルゴリズムを採用することが望ましいこととなります。

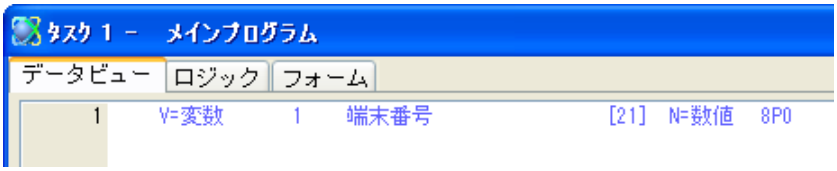
いろいろな方法が考えられると思いますが、ここでは、Lock/Unlock 関数を使って、ユニークな番号を探し出す方法を使いました。

端末番号は、ユーザがアプリケーションを開始する時点で割り振り、アプリケーションを終了する時点で解放する必要があります。このような処理を行うのは、「メインプログラム」のタスク前処理、およびタスク後処理が最適です。

リッチクライアントシステムの場合でも、メインプログラムは必ず各コンテキスト（ユーザ）ごとに実行されるので、メインプログラムで端末番号を割り当て/解放するようにすれば、ユーザごとにユニークな番号を割り当てることができるようになります。

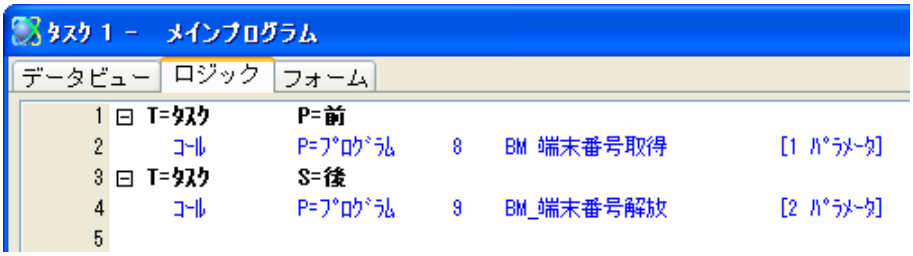
具体的には、次のようになっています。

1. メインプログラムのデータビューで、端末番号を格納するための数値変数を定義します。

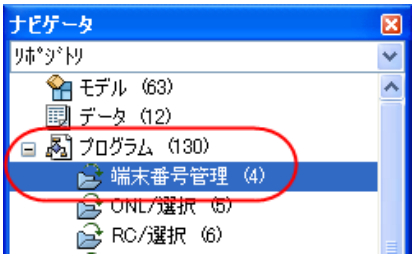


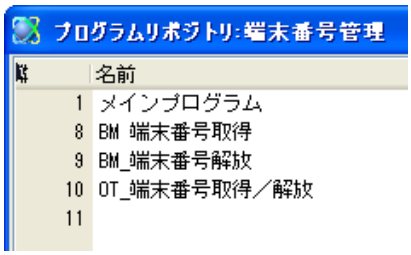
   この変数は、メインプログラムで定義されているので、アプリケーション全体で利用できるグローバル変数となります。

2. タスク前処理で、端末番号取得を行うバッチプログラムを呼び出します。また、タスク後処理で、端末番号を解放するバッチプログラムを呼び出します。



端末番号の取得・解放のためのバッチプログラムは、プログラムリポジトリの「端末番号管理」フォルダにあります。





プログラム番号	名前	内容
8	BM_端末番号取得	ユニークな端末番号を生成します。
9	BM_端末番号解放	現在利用中の端末番号を解放し、他のユーザが利用できるようにします。
10	OT_端末番号取得／解放	上記プログラムのテスト用プログラムです。

ユニークな端末番号を生成する上で、キーとなるのが、Lock() 関数です。この関数の仕様は、次のようになっています。(リファレンスヘルプより)

Lock リソースをロック

一定の時間内に一人のユーザのみによって占有される仮想的な要素(リソース)を作成し、テーブルの行やタスクをロックします。

構文: Lock (リソース, タイムアウト)

パラメータ:

リソース	任意の文字列。長さは 0～128。リソース名はユニークでなければいけません。
タイムアウト	リソースが別のユーザによりロックされている場合の待ち時間(秒)。負の値を指定した時は、無制限に待ちます。

戻り値:

0	ロックが成功
1	同一セッションで同じリソースに既にロックがかかっている場合
2	別のセッションで同じリソースにロックがかかっていて、待ち時間がタイムアウトを越えた場合(ロックは失敗)

注意事項:

- 通常項目更新の式で設定します。
- Lock したリソースは必ず Unlock して解放する必要があります。

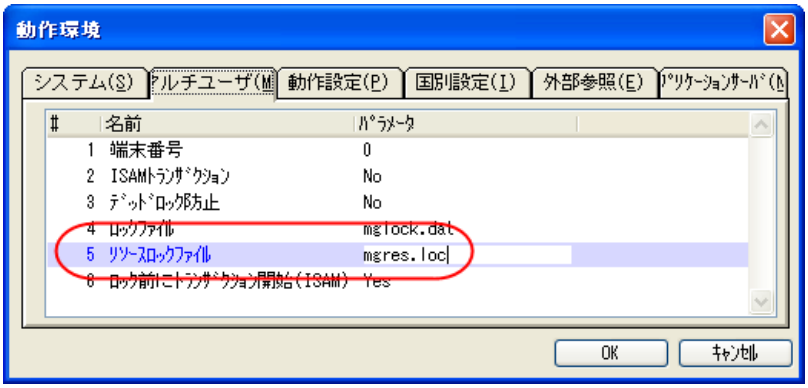
関連項目: UnLock

この関数を使って、次のような簡単なループにより、現在未使用の端末番号を見つけ出します。プログラム「BM_端末番号取得」は、この方式を使っています。

1. 番号を、仮に1とする。
2. 次の関数を実行する: Lock (Str(番号, '8P0'),0)
3. 戻り値が 0 の場合には、この番号を端末番号とする。
4. 戻り値が 0 でない場合には、すでに他の人が使っているものなので、番号を＋1して、2 に戻る。

Lock 関数は、リソースロックファイルを使って、OS レベルのロックを行うことにより実装されています。従って、Magic エンジンがこのファイルを共有するようになっていることが大前提です。

このファイルは、「動作環境 ⇒ マルチユーザ タブ ⇒ リソースロック」パラメータで設定されます（下図）。



デフォルトの設定では、Magic ディレクトリの下に mgres.loc という名前で作成されます。

アプリケーションをクローズする場合には、Lock によって取得した端末番号を解放することが必要です。これは「BM_端末番号解放」プログラムが行いますが、実際には単に、Unlock 関数を呼び出して、ロックを解除しているだけです。

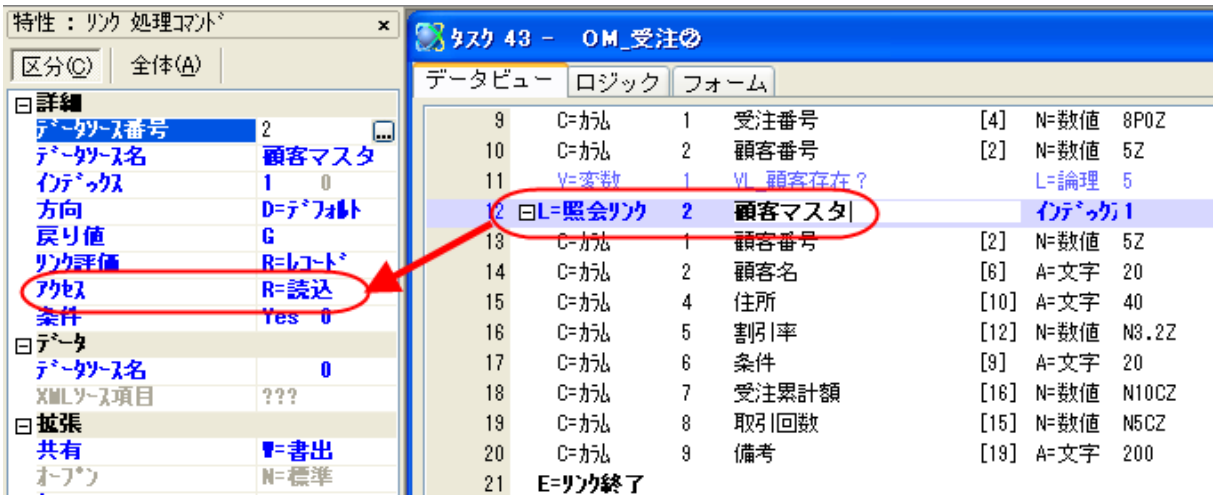
本節の内容についてのより詳しい情報は、以下のキーワードでリファレンスヘルプを検索してください。

- Lock
- Unlock
- リソースロックファイル

7.4 顧客マスタのレコードロック回避

7.4.1 顧客マスタへのリンク

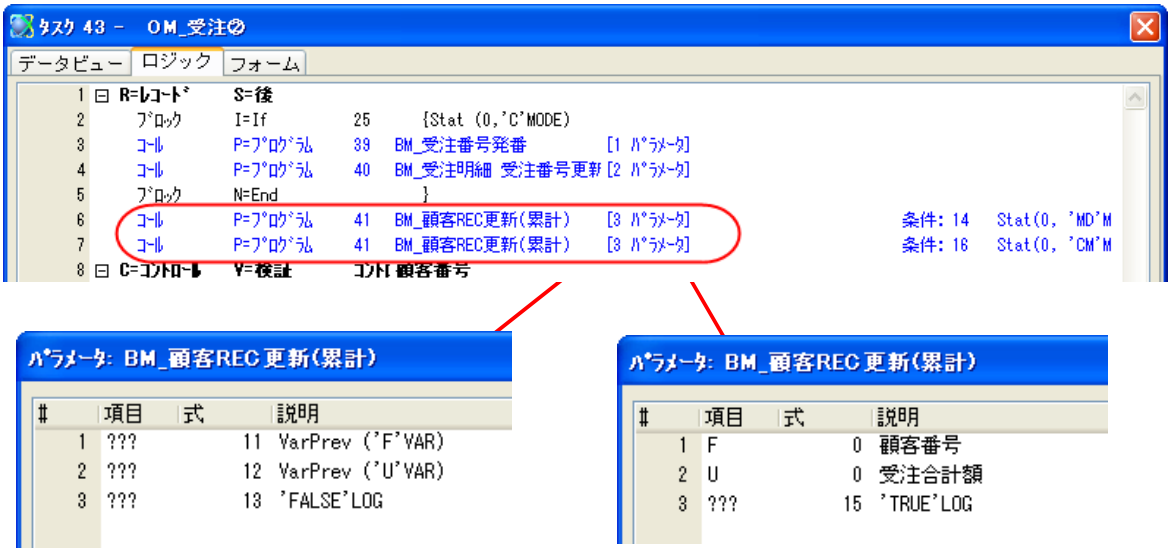
顧客マスタへのレコードロックを回避するために、リンク時の「アクセス」特性は「R=読込」とします（下図）。



7.4.2 顧客マスタの累計データの更新

顧客マスタにある累計データ（受注累計額、取引回数）は、基本形では、ヘッダタスクのレコード後処理で直接「項目更新」コマンドで更新していました（6.12「累計値の更新」 55 ページ参照）。

ここでは、顧客マスタへのリンクを「R=読込」としてしまったので、ヘッダタスクでデータを直接更新することはできません。このため、別途データ更新のためのバッチタスク「BM_顧客 REC 更新(累計)」（プログラム 41 番）を作成し、それをレコード後処理から呼び出して、累計データの更新を行っています（下図）。



図中、バッチタスク「BM_顧客 REC 更新(累計)」が 2 回呼び出されていますが、これは 6.12.2「加算更新の実行ルール」(56 ページ)で説明したような、差分を加算させるためのロジックを実現するためです。即ち、「差分」を加算するために、

- 初期値の減算
- 最終値の加算

という二段階を踏んでいます。

#	図中の行番号	処理内容	条件
1	6 行目	受注レコードの初期値を減算する。	修正モード、および削除モードで実行
2	7 行目	受注レコードの最終値を加算する。	登録モード、および修正モードで実行

このバッチタスクは、3 つのパラメータをとります。

1. 処理対象となる顧客レコードの顧客番号
2. 受注合計値の値
3. 加算するか減算するかのフラグ

また、1 回目の呼び出しでパラメータに渡す「初期値」は VarPrev 関数を使って求めています。

明細タスクにおける商品マスタへのロックの競合を避ける方法も、顧客マスタで行った方法と全く同様ですので、ここでは説明を省略します。

差分を計算するために、上記のように初期値の減算と最終値の加算の 2 回に分ける方法を使っていますが、差分を計算するだけならば、単に差を引き算で計算すればよいのではないか、無駄なことをしているのではないか、と思われるかもしれません。しかし、このようにしているのには、また別の理由があります。

プログラムでは、顧客番号を変更することも許しています。例として、ある受注レコードにおいて、顧客番号が 1008 (千葉ペットショップ)だったものを、顧客番号 1234 (ペットセンター神田)に変更し、さらに、追加注文によって受注合計額が 37,036 円から 46,782 円になったとします。このような場合には、次のような処理を行う必要があります。

- 顧客番号 1008 の顧客レコードについて、受注合計額の初期値(37,036 円)を差し引き、取引回数を1減らす。
- 顧客番号 1234 の顧客レコードについて、受注合計額の最終値(46,782 円)を加え、取引回数を1増やす。

この処理は、簡単な数式で表現することはできません。しかし、上記のような減算と加算を二つに分けて行う方法を採用すれば、問題なく処理することができます。

8 ONL/物理/MEM テーブル

本章では、Memory データベースに一時テーブルを使う方法を説明します。

バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形(6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

前章の「ONL/物理/バッチ更新」の方法では、受注データおよび受注明細データを作成するために、まず仮番号で登録しておいて、確定時に正式受注番号で置き換える、という方法をとっていました。

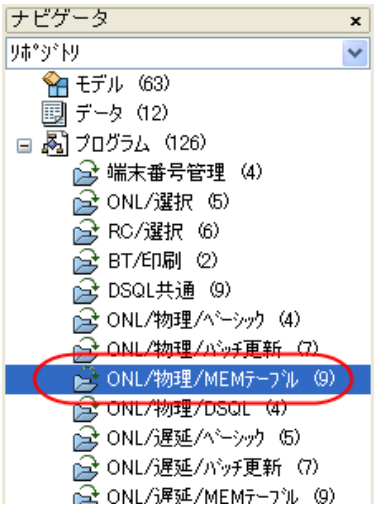
この方式では、次のような懸念があります。

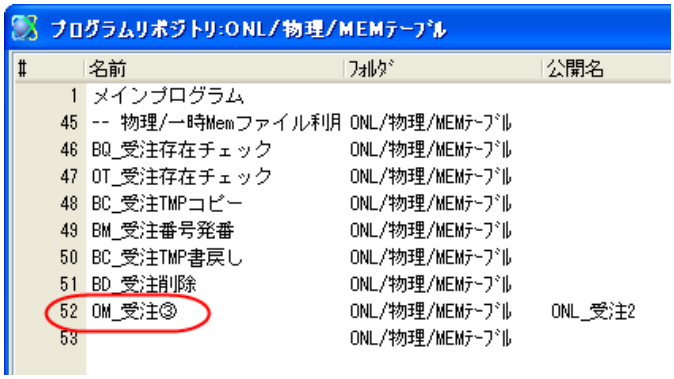
- 仮受注番号で登録中のデータはまだ確定されていないデータですが、このような中途半端なデータが、受注データという重要なテーブルの中に混在することは望ましくありません。もちろん、プログラムロジックが正しく作られ、トランザクションの設定も正しければ、このデータが不正に残ることはありません。しかし、万一プログラムロジックやトランザクション設定にミスがあれば、ごみデータが受注テーブルに混じりこんでしまうことになります。プログラムが複雑になれば不具合の発生する可能性も高くなることを考慮すれば、極力ごみデータが混入する可能性は少なくしておきたいものです。
- 受注番号は、受注テーブルのキー項目であり、受注明細テーブルでもキー項目の一部となっています。このようなキー項目に関わるカラムを更新するのは、データモデルの観点からも好ましくないし、またパフォーマンス上も問題の出る可能性があります。DBMS 設計時に参照性制約などが設定されていたら、変更すること自体が許されない場合もあります。

このような問題点を解決するには、本章で説明するような、一時テーブルを利用するのが一番簡単です。

一時テーブルを利用することにより、ユーザの入力・修正途上の中間データに対する細かな取り扱いに対しても自由度が高くなる、という利点も出てきます。

一時テーブルを利用する方法を使ったプログラムは、プログラムリポジトリの「ONL/物理/MEM テーブル」という名称のフォルダに収められています。この中で「OM_受注③」(プログラム 52 番)が受注入力プログラムであり、これから呼び出されるバッチプログラムが 5 つほど定義されています。

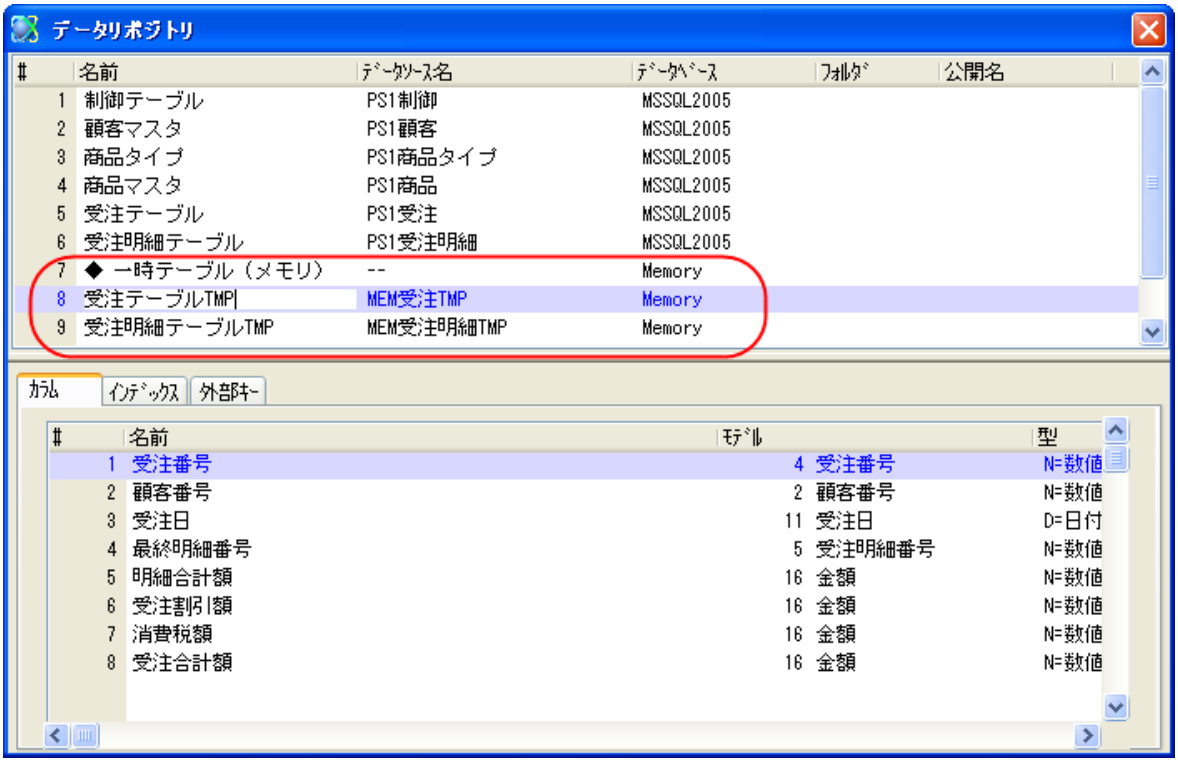

ナビゲータ
ﾘﾎﾟｼﾞﾄﾘ
モデル (63)
データ (12)
プログラム (126)
端末番号管理 (4)
ONL/選択 (5)
RC/選択 (6)
BT/印刷 (2)
DSQL共通 (9)
ONL/物理/ﾍﾞｰｼｯｸ (4)
ONL/物理/ﾊﾞｯﾁ更新 (7)
ONL/物理/MEMﾃｰﾌﾞﾙ (9)
ONL/物理/DSQL (4)
ONL/遅延/ﾍﾞｰｼｯｸ (5)
ONL/遅延/ﾊﾞｯﾁ更新 (7)
ONL/遅延/MEMﾃｰﾌﾞﾙ (9)


プログラムリポジトリ:ONL/物理/MEMﾃｰﾌﾞﾙ
名前 ﾌｫﾙﾀﾞ 公開名
1 メインプログラム
45 -- 物理/一時Memファイル利用 ONL/物理/MEMﾃｰﾌﾞﾙ
46 BQ_受注存在チェック ONL/物理/MEMﾃｰﾌﾞﾙ
47 OT_受注存在チェック ONL/物理/MEMﾃｰﾌﾞﾙ
48 BC_受注TMPコピー ONL/物理/MEMﾃｰﾌﾞﾙ
49 BM_受注番号発番 ONL/物理/MEMﾃｰﾌﾞﾙ
50 BC_受注TMP書戻し ONL/物理/MEMﾃｰﾌﾞﾙ
51 BD_受注削除 ONL/物理/MEMﾃｰﾌﾞﾙ
52 OM_受注③ ONL/物理/MEMﾃｰﾌﾞﾙ ONL_受注2
53 ONL/物理/MEMﾃｰﾌﾞﾙ

8.1 処理の概要

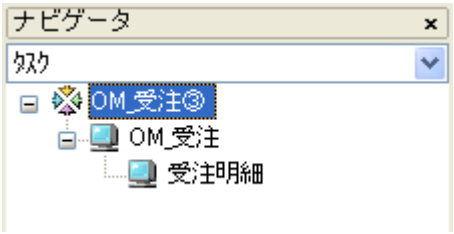
8.1.1 データリポジトリ

一時テーブルを利用するためには、まず、データリポジトリに一時テーブル定義をする必要があります。一時テーブルとしては、受注テーブル、および明細テーブルと全く同じ定義内容のものを、Memory データベースとして登録します（下図、テーブル 8 および 9）。



8.1.2 プログラム構造

基本形での受注入力プログラムは、親子の 2 階層のプログラム構造を持っていました。

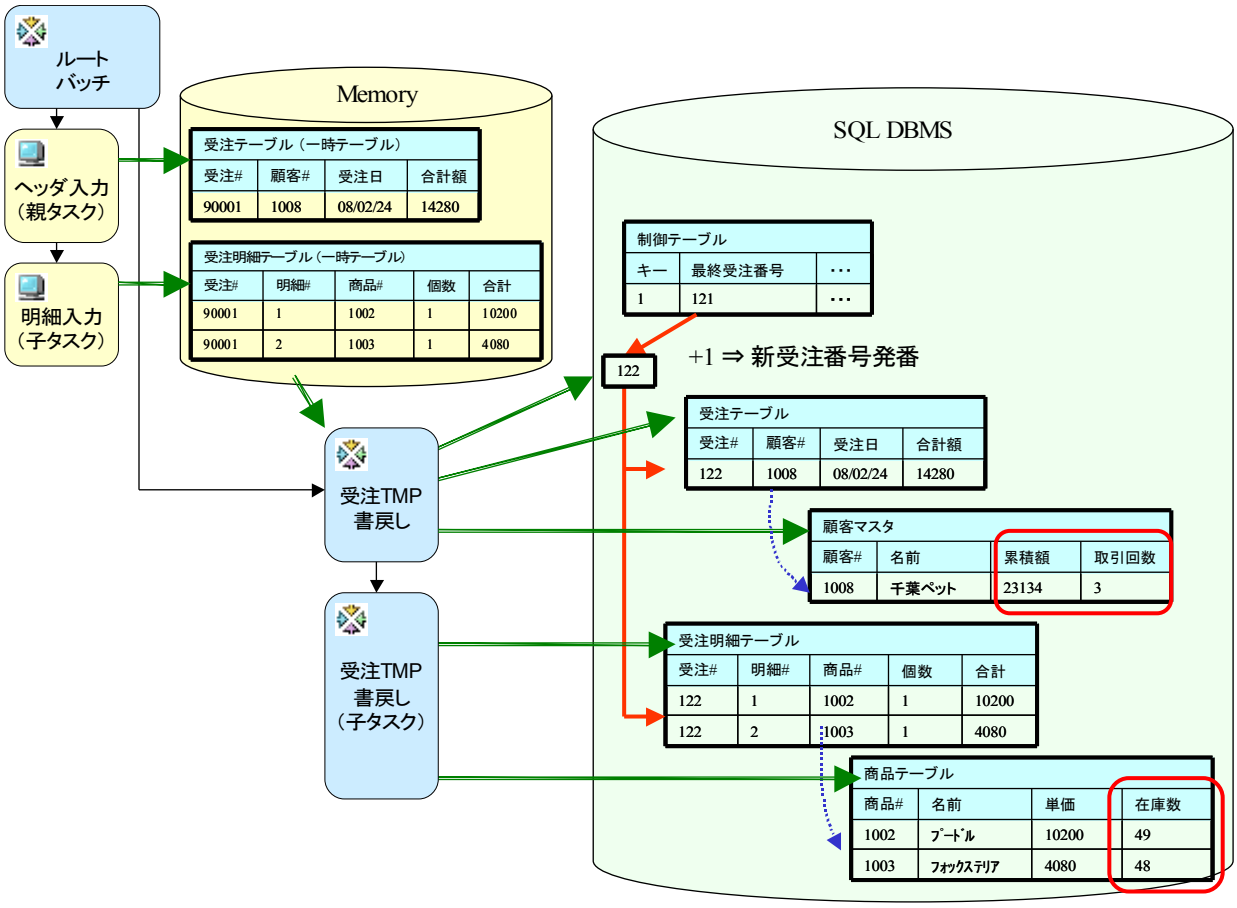


一時テーブルを利用する方法では、これにルートのタスクとしてバッチタスクを追加した、3 階層の構成とします（右図）。従って、この構造では、サブタスクがヘッダタスク、孫タスクが明細タスクになります。

ヘッダタスクおよび明細タスクは、基本形と同様に、受注データおよび受注明細データを、それぞれメインソースとして利用し、ユーザが入力・修正できるようになっています。ただし、このメインソースとしては、DBMS にある受注テーブル/受注明細テーブルではなく、Memory データベース上にある受注テーブル・受注明細データの一時テーブルを使います。

8.1.3 登録モードの時の処理の流れ

新規登録時の処理の流れは、大略、次の図のようになります。



プログラム設計において、次のようになっています。

- ヘッダタスク(サブタスク)は、Memory データベース上の受注テーブル (一時テーブル)をメインソースとしています。
- 明細タスク(孫タスク)は、同じく、Memory データベース上の受注明細テーブル (一時テーブル)をメインソースとしています。
- ヘッダタスクと明細タスクにおいて、ユーザが受注データを入力します。ここでのロジックは、基本形と似ています。
- ただし、受注番号は仮受注番号を使い、顧客マスタや商品マスタの累計データの更新はここでは行いません。従って、制御テーブル、顧客マスタ、商品マスタへのリンクでは、「アクセス」=「R=読込」とします。これにより、制御テーブル、顧客マスタ、商品マスタに対するレコードロックは発生しなくなります。

処理の流れは、次のようになります。

1. 最初に、ルートバッチタスクで、一時テーブルの内容を空にしておきます。
2. ヘッダタスクを呼び出します。
3. ヘッダタスクおよび明細タスクでは、ユーザが入力を行います。ユーザがひとつの伝票の入力を終え、入力を確定・保存する時には、ヘッダタスクは終了し、処理の流れがルートバッチタスクに戻ります。
4. ルートバッチタスクにおいて、一時テーブルの内容を DBMS に書き戻しします。書き戻しには、バッチタスクを使います。(プログラム 50 番「BC_受注 TMP 書き戻し」)。この中で、正式受注番号の発番も行います。
5. DBMS への書き込みを行ったら、最初に戻ります。

以上は新規登録の場合の処理の流れです。

8.1.4 修正・照会モードの時の処理の流れ

既存の受注データを修正する場合には、前項の 1 の段階において、一時テーブルを空にするだけではなく、DBMS から受注/受注明細レコードを一時テーブルにコピーしておく必要があります。

このときに、DBMS 中の受注レコードをすべて一時テーブルにコピーしていたら、時間もかかるし、Memory データベースも膨大になるし、コピー後に他ユーザが変更しても一時テーブルには反映されないということになってしまうので、一時には1つの受注レコードだけをコピーするようにします。

このために、ルートバッチタスクでは、「現在の受注番号」というものを管理しておくようにします。「現在の受注番号」の値は、

- 登録時には、仮受注番号として、実在しない大きな値 (9999999) とします。
- 修正・照会時には、初期値として、最終受注データの受注番号を自動的に検索して設定します。
- ただし、ユーザが受注検索プログラムを使って、受注番号を選択して、「現在の受注番号」を変更できるようにします。

とします。

ここでは、仮受注番号として、固定値 (9999999) を使っています。前章で説明した「ONL/物理/バッチ更新」の場合とは異なり、一時テーブルを使う方法では、仮受注番号をユーザごとに分ける必要がありません。一時テーブルは Memory データベースに作られるので、別ユーザと共有する可能性はないので、同じ仮受注番号であったとしても、別ユーザの中間データと混同する可能性がないからです。

8.2 トランザクションの設定

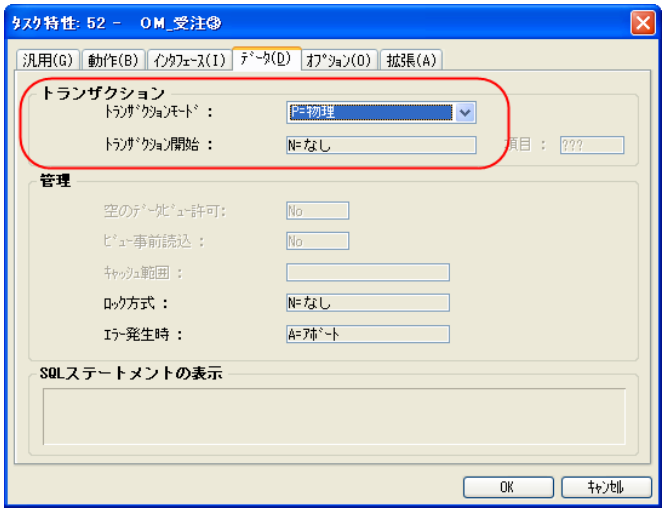
ひとつの受注伝票単位でトランザクションをコミットするために、トランザクションの設定は、次のようになっています。

ルートタスクはトランザクションの外に置いておきます。このため、ルートタスクのトランザクション設定は、

「トランザクションモード」＝「P=物理」

「トランザクション開始」＝「N=なし」

とします。

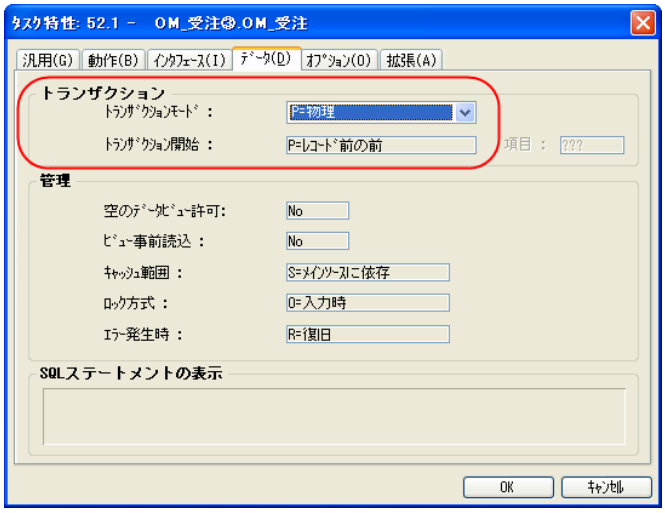


トランザクションをヘッダタスクで始めます。従って、ヘッダタスク（サブタスク）のトランザクション設定は、

「トランザクションモード」＝「P=物理」

「トランザクション開始」＝「P=レコード前の前」

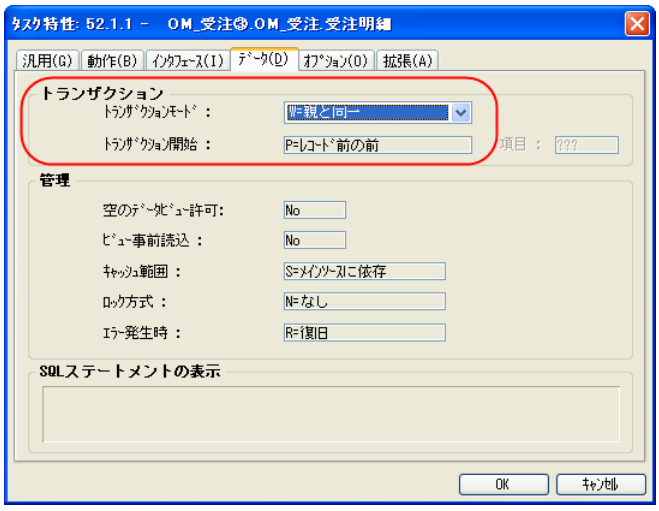
とします。



明細タスクは、ヘッダタスクで開始されたトランザクションの中で実行します。従って、

「トランザクションモード」＝「W=親と同一」

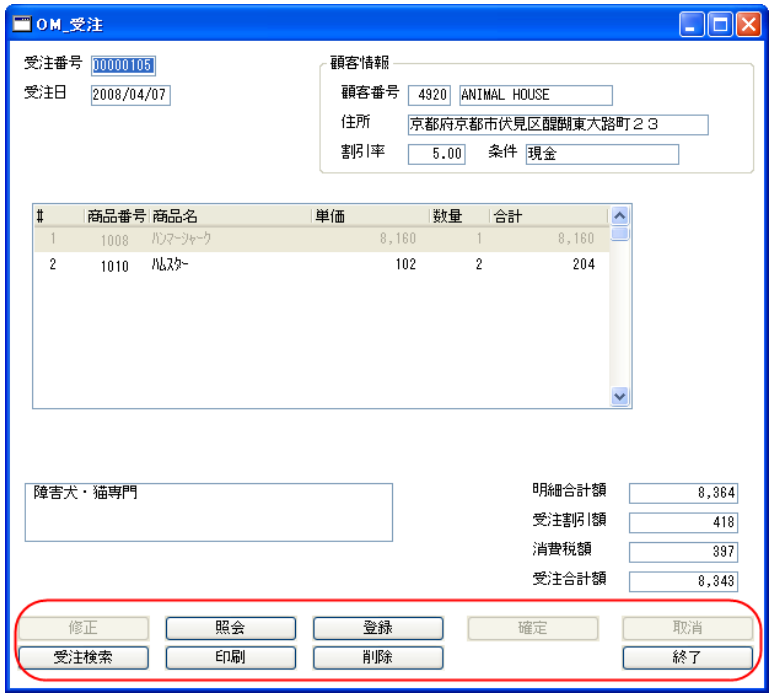
とします。



8.3 ルートバッチタスクの制御変数

ルートバッチタスクが制御すべき機能としては、次のようなものがあります。

- タスクモードの変更
  （修正、照会、登録）
- 入力の確定と取り消し
- 受注検索
- 印刷
- 削除
- 終了



画面には、各機能を実行するためのボタンが配置されます。

また、修正・照会モードにおいては、「現在の受注番号」を内部的に管理しておく必要があります。

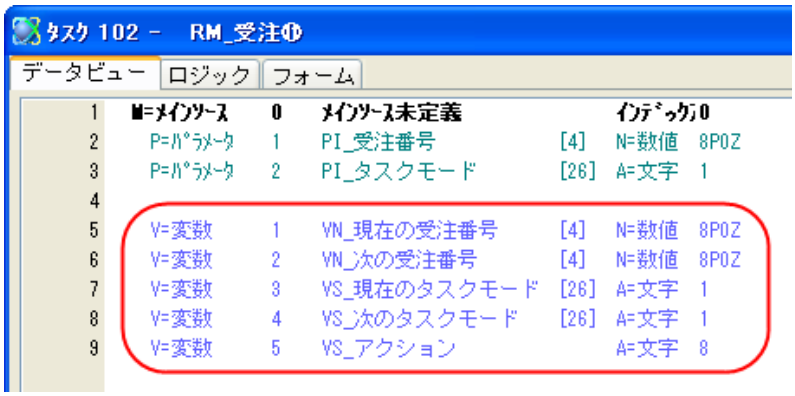
これらのことを実現するために、以下の変数を使って、ルートタスクにおける実行の流れを制御します。

- 現在の受注番号
- 現在のタスクモード
- アクション（ユーザ入力後に行うべき処理。DBMS への書き戻し、印刷、受注検索、レコード削除など）

また、制御を容易にするために、受注番号とタスクモードについては、「現在」のものと「次のループ」でのものとの 2 種類に分けておいたほうがよいので、次の変数も使います。

- 次の受注番号
- 次のタスクモード

以上まとめると、ルートバッチタスクでは、右図のような 5 つの変数を使って、プログラムの流れの制御を行います。



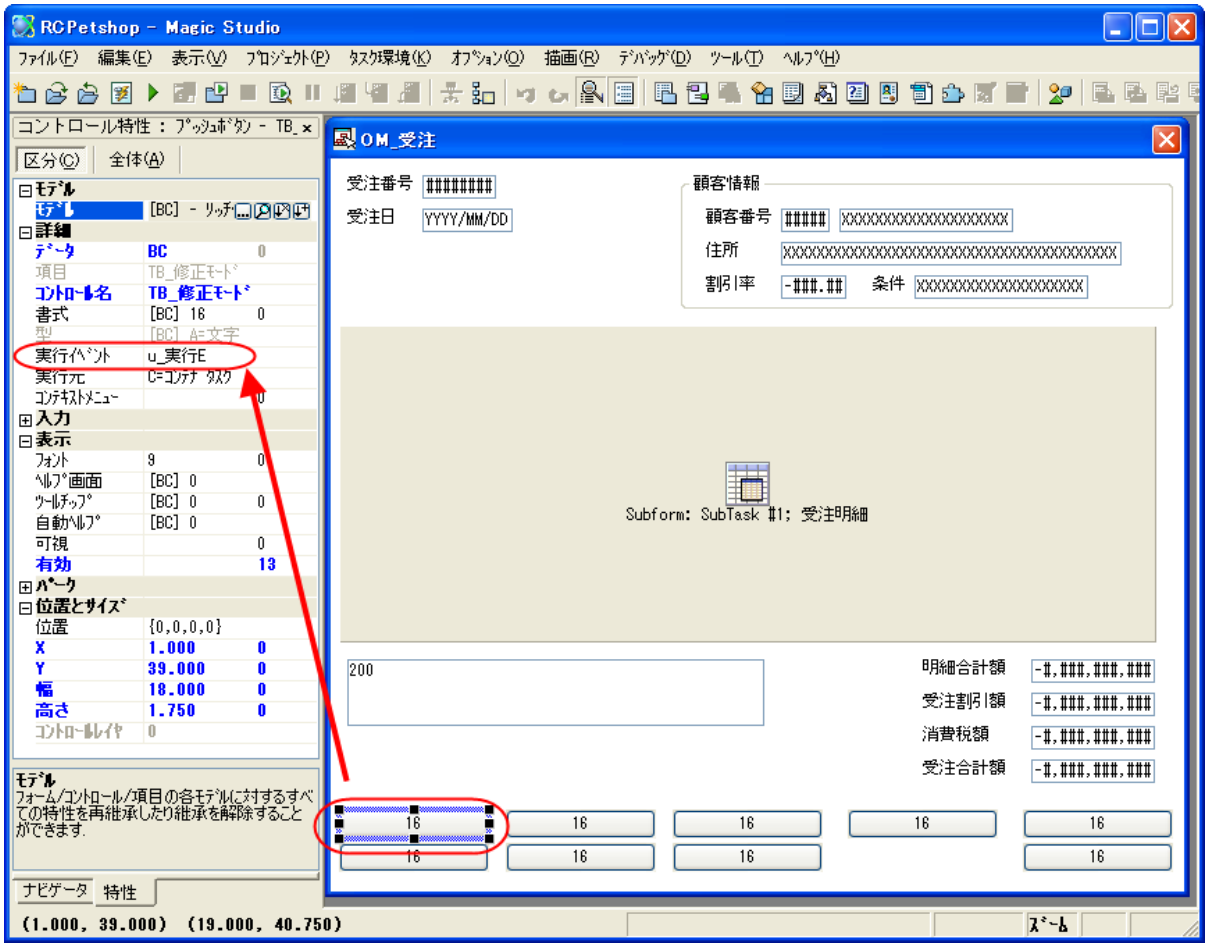
8.4 ボタンとイベントハンドラ

ユーザがプログラムを制御するためのボタンは、すべてヘッダタスクのフォームに配置されています（下図）。

基本形においては、各ボタンに内部イベントを設定していました。例えば、「修正」ボタンには「修正(M)」内部イベントを、「終了」ボタンには「クローズ(C)」内部イベントを設定していました。

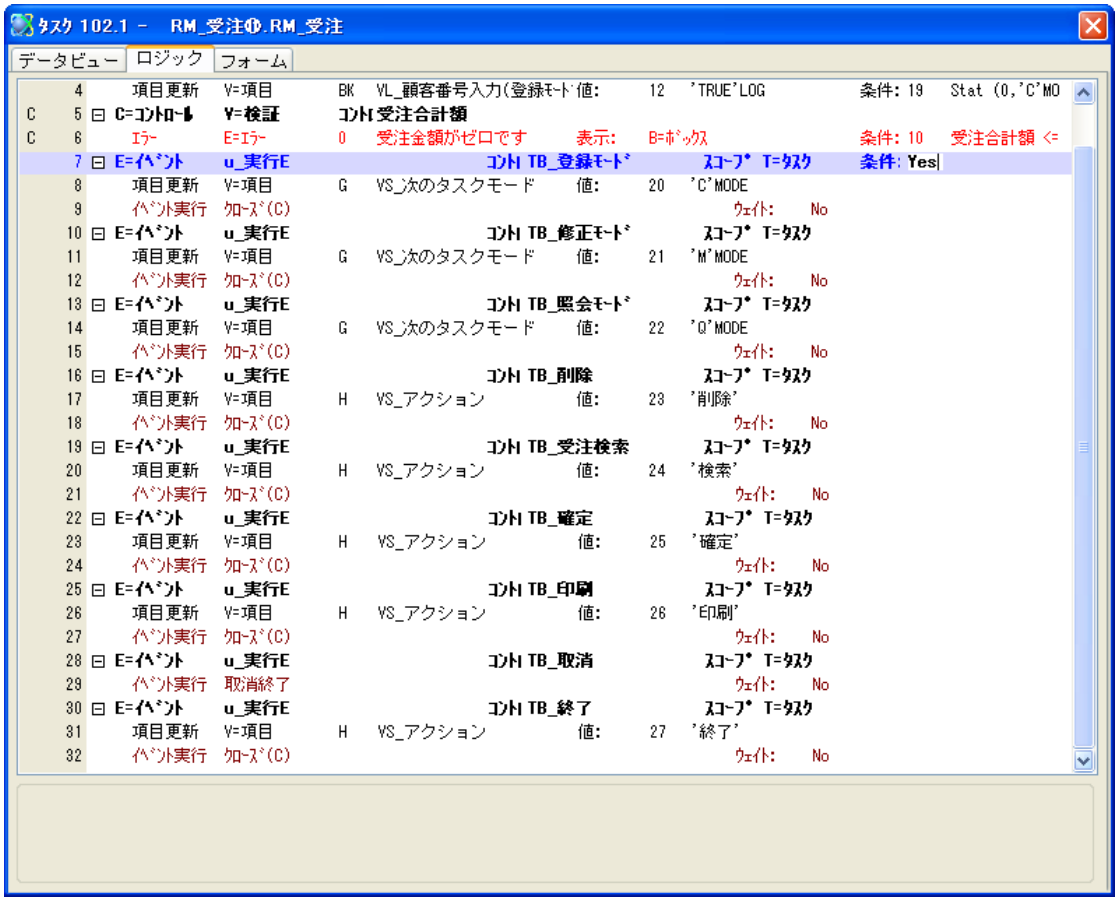
一時テーブルを使う方法では、ボタンに直接内部イベントを設定することはしません。それぞれのボタンにはユーザイベント「u_実行 E」を設定して、各ボタンごとにハンドラを作成し、そこでルートバッチタスクの制御変数を適当に設定することにより、全体の流れを制御します。

下図は、ヘッダタスクのフォームエディッタです。特性シートには、「修正」ボタンのコントロール特性が表示されていますが、ここで見るように、「実行イベント」には「u_実行 E」が設定されているのがわかります。



次の図は、ヘッダタスクの「ロジック」エディタに定義された、イベントハンドラです。

ここで見るように、ボタンの押下に対応して、ユーザイベント「u_実行 E」のハンドラがボタン分だけ定義されています。どのボタンで押された場合にどのハンドラが走るのかは、「コントロール名」の設定により区別されます。

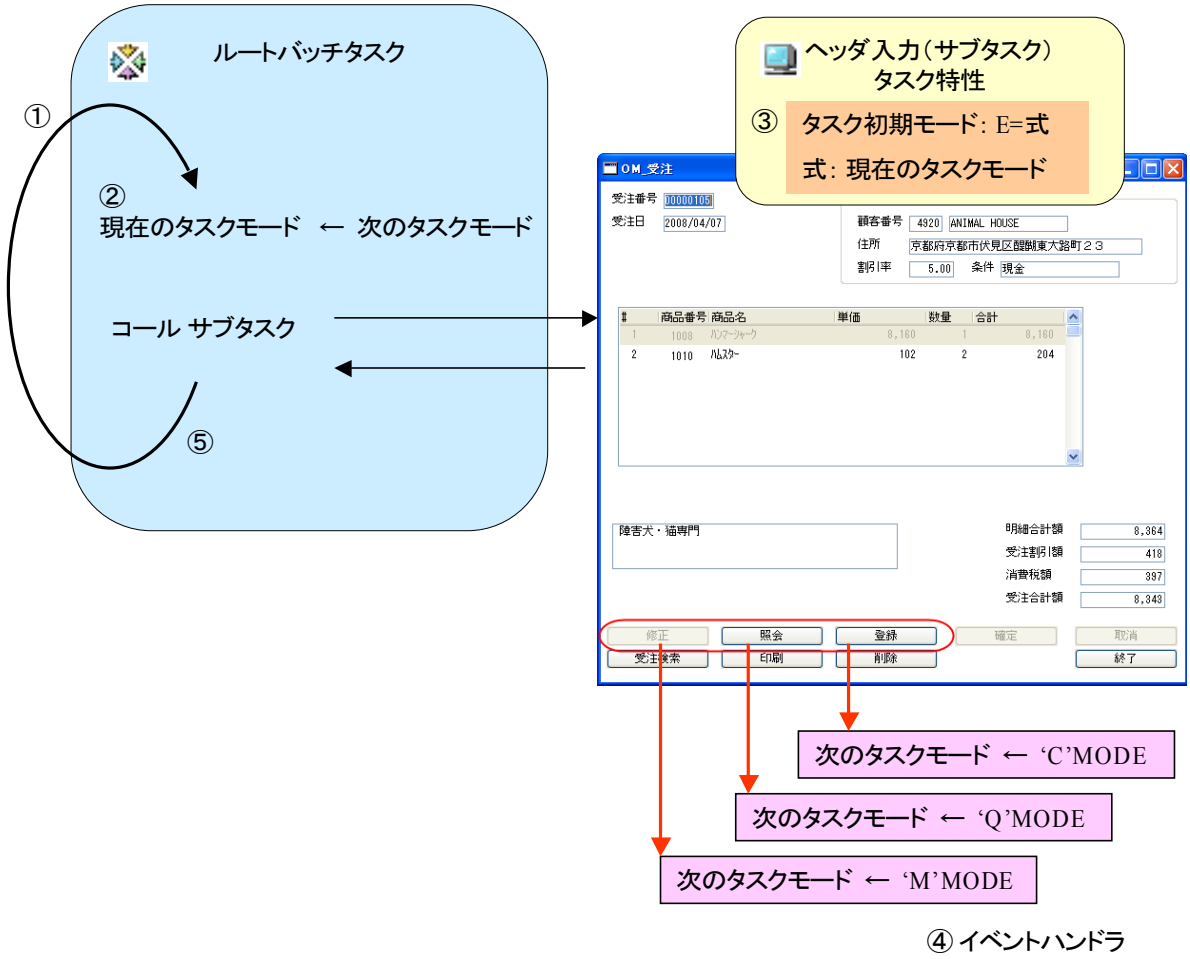

タスク 102.1 - RM_受注@.RM_受注
データビュー
ロジック
フォーム
4 項目更新 V=項目 BK VL_顧客番号入力(登録ﾓｰﾄﾞ 値: 12 'TRUE'LOG 条件: 19 Stat (0,'C'MO
C 5 C=コントロール V=検証 コントロール 受注合計額
C 6 エラー E=エラー 0 受注金額がゼロです 表示: B=ボックス 条件: 10 受注合計額 <=
7 E=イベント u_実行E コントロール TB_登録ﾓｰﾄﾞ スコープ T=タスク 条件: Yes
8 項目更新 V=項目 G VS_次のタスクモード 値: 20 'C'MODE
9 イベント実行 クローズ(C) ウェイト: No
10 E=イベント u_実行E コントロール TB_修正ﾓｰﾄﾞ スコープ T=タスク
11 項目更新 V=項目 G VS_次のタスクモード 値: 21 'M'MODE
12 イベント実行 クローズ(C) ウェイト: No
13 E=イベント u_実行E コントロール TB_照会ﾓｰﾄﾞ スコープ T=タスク
14 項目更新 V=項目 G VS_次のタスクモード 値: 22 'Q'MODE
15 イベント実行 クローズ(C) ウェイト: No
16 E=イベント u_実行E コントロール TB_削除 スコープ T=タスク
17 項目更新 V=項目 H VS_アクション 値: 23 '削除'
18 イベント実行 クローズ(C) ウェイト: No
19 E=イベント u_実行E コントロール TB_受注検索 スコープ T=タスク
20 項目更新 V=項目 H VS_アクション 値: 24 '検索'
21 イベント実行 クローズ(C) ウェイト: No
22 E=イベント u_実行E コントロール TB_確定 スコープ T=タスク
23 項目更新 V=項目 H VS_アクション 値: 25 '確定'
24 イベント実行 クローズ(C) ウェイト: No
25 E=イベント u_実行E コントロール TB_印刷 スコープ T=タスク
26 項目更新 V=項目 H VS_アクション 値: 26 '印刷'
27 イベント実行 クローズ(C) ウェイト: No
28 E=イベント u_実行E コントロール TB_取消 スコープ T=タスク
29 イベント実行 取消終了 ウェイト: No
30 E=イベント u_実行E コントロール TB_終了 スコープ T=タスク
31 項目更新 V=項目 H VS_アクション 値: 27 '終了'
32 イベント実行 クローズ(C) ウェイト: No

8.5 タスクモードの制御

「修正」、「照会」、「登録」のボタンを押すと、タスクモードが変更されます。

基本形では、それぞれのボタンに、モード切り換えを行う内部イベント「修正(M)」、「照会(Q)」、「登録(C)」を設定して、直接タスクモードの変更を行っていました。

それに対し、ここでは、ルートバッチタスクに定義されている変数「VS_現在のタスクモード」および「VS_次のタスクモード」を使って、タスクモードを制御します。下図にこの様子を示してあります。図中では簡単のため、変数名の接頭辞「VS_」を省略しています）



1. ルートバッチタスクは、レコードループによって、繰り返し実行されます。（終了条件については、後述します）
2. 最初に、「VS_次のタスクモード」の値を「VS_現在のタスクモード」に設定します。「VS_次のタスクモード」の初期値は 'C'MODE となっています。
3. ヘッダタスク（サブタスク）では、タスク特性で、「タスクの初期モード」が「E=式」に設定されています。そして、式の値としては、「VS_現在のタスクモード」を参照しています。
4. 「修正」ボタンが押されたら、イベントハンドラで、「VS_次のタスクモード」を 'M'MODE に設定して、タスクを終了します。同様に、「照会」ボタンのイベントハンドラでは' Q'MODE、「登録」ボタンのイベントハンドラでは 'C'MODE にそれぞれ設定し、タスクを終了します。
5. ルートバッチタスクに戻ったら、2 に戻り、次の繰り返しに入ります。

8.6 受注番号の制御

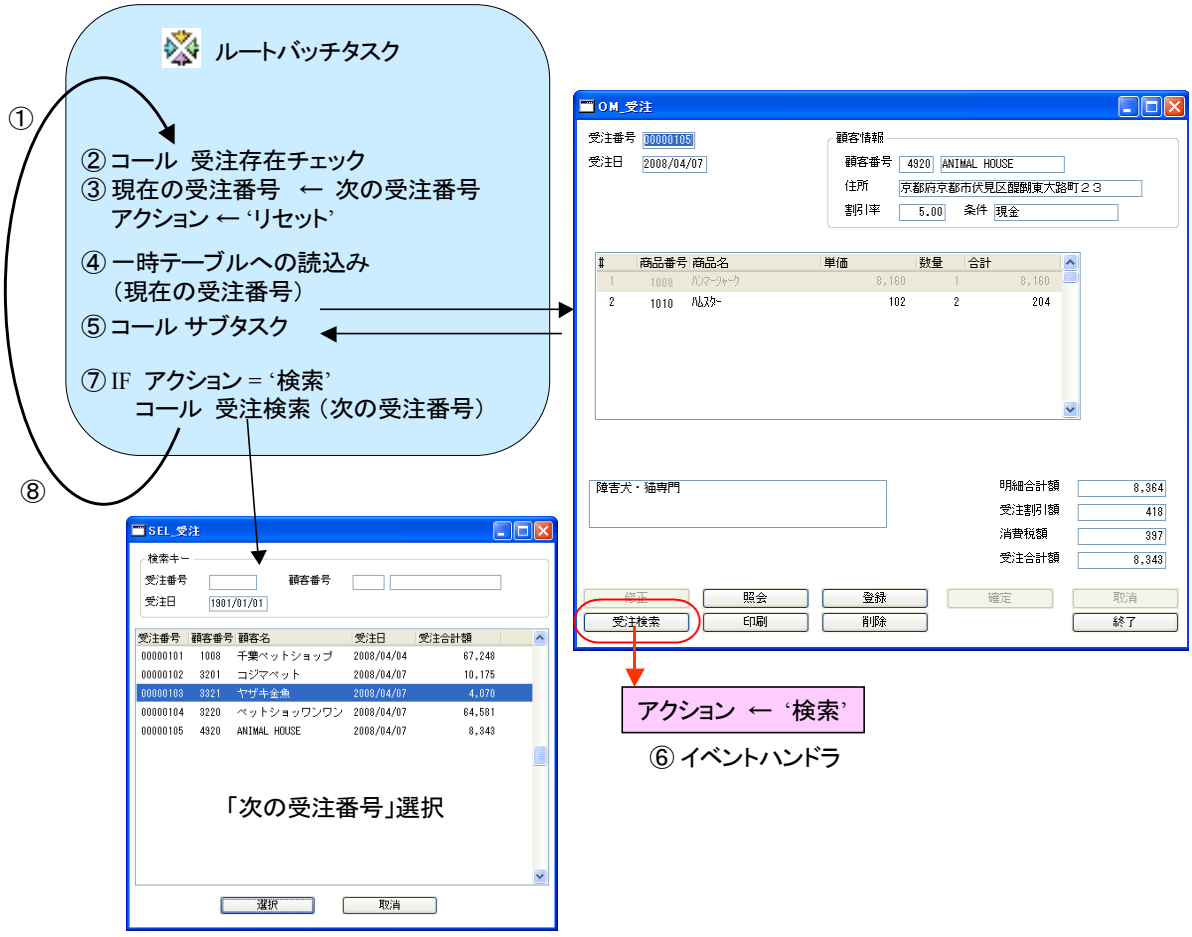
修正モードおよび照会モードの時、受注検索ボタンを押すと、受注一覧画面が表示され、ユーザがその中から選ぶと、その受注内容に位置づけされて表示されます。

基本形では、パラメータに受注番号を設定し、「ビュー再表示」イベントを利用して、位置づけを行っていました。（6.19「受注検索ボタン」参照）

一方、ここでは、ルートバッチタスクの変数「VN_現在の受注番号」と、「VN_次の受注番号」、および「VS_アクション」によって制御しています。

8.6.1 受注番号制御の概観

下図にこの様子を図示しています。（前節と同様、変数名の接頭辞「VS_」および「VN_」は省略しています。）



1. ルートバッチタスクは、レコードループによって、繰り返し実行されます。
2. 最初に、プログラム「BQ_受注存在チェック」(プログラム 46 番) を呼び出します。このプログラムは「VN_次の受注番号」と「VS_次のタスクモード」をパラメータとしてとり、受注レコードが存在するかをチェックして、必要に応じパラメータ値を調整するものです。（後述の 8.6.2「受注存在チェック プログラム」において、より詳しく説明します。）
3. 「VS_次の受注番号」の値を「VS_現在の受注番号」に設定します。
4. 「VS_現在の受注番号」の受注レコードを、一時テーブルにコピーします。（ヘッダ、明細とも、プログラム 48 番「VC_受注 TMP コピー」により行います）。
5. ヘッダタスク（サブタスク）を呼び出します。

6. ヘッダタスクで「検索」ボタンが押されたら、イベントハンドラで、「VS_アクション」を '検索' に設定して、タスクを終了します。
7. ルートバッチタスクに戻ったら、「VS_アクション」の値により、処理を行います。「VS_アクション」＝'検索' の場合には、受注検索プログラム（プログラム 14 番 「SEL_受注検索」)を呼び出します。このタスクには「VS_次の受注番号」をパラメータとして渡し、ユーザの選択結果を受け取ります。
8. 2 に戻り、次の繰り返しに入ります。

8.6.2 受注存在チェック プログラム

制御用の変数の値は、DBMS 中のデータと依存関係があり、正当性を確認する必要があります。

例えば、「VS_次のタスクモード」が「修正」の場合には、「VS_次の受注番号」には、DBMS 中に存在する受注レコードの受注番号が設定されていなければなりません。存在しない受注番号、あるいは空文字列などが設定されていたとすれば、それは正しくありません。

このような正当性を確認し、必要があれば適当に調整するための処理が必要になりますが、これが、ルートバッチタスクにおいて、ループの最初に呼び出されている「BQ_受注存在チェック」というバッチプログラムです。これは簡単なプログラムですが、受注番号の制御において重要な役目を果たします。

このプログラムは、次の二つのパラメータを受け取ります。

1. タスクモード
2. 受注番号

このパラメータを使って、受注データをチェックし、下表のように適宜パラメータの値を調整してからリターンします。

実行前			実行後	
タスクモード	受注番号		タスクモード	受注番号
C	−	⇒	C	（仮受注番号）
M/Q	（対応する受注レコードが存在する）	⇒	（そのまま）	（そのまま）
M/Q	（対応する受注レコードが存在しない）	⇒	（そのまま）	（下記本文参照）

- 「タスクモード」が C（登録モード）の場合には、「受注番号」のもとの値にかかわらず、仮受注番号を設定します。
- 「タスクモード」が M（修正モード）または Q（照会モード）の場合には、「受注番号」パラメータの値を使って、受注データがデータベースに存在するかどうかを確認します。
  - もし受注レコードが存在すれば、パラメータの値はそのままとします。
  - もし受注レコードが存在しなければ、不正な受注番号とみなして、次のように受注番号を変更します。
    - その受注番号より大きい受注データが存在すれば、その中から最小のものを取って、「受注番号」パラメータを設定します。
    - その受注番号より大きい受注データが存在しなければ、最新の受注データ（存在する最大の受注番号）を設定します。

このプログラムを最初にコールして、受注番号が適正なものであるかを確認することによって、存在しない受注番号が設定されてプログラム全体の制御がおかしくなってしまうことを防止します。

また、タスクモードが「登録」モードから「修正」あるいは「照会」モードに変更になるときには、「受注番号」は非常に大きな仮受注番号になっているはずなので、上のアルゴリズムにより、自動的に、最大の受注番号（すなわち、最新の受注レコード）を選択するようにもなっています。

厳密に考えると、受注レコードがまだ 1 件も存在しない場合、ということも考えられ、その場合には、強制的に「タスクモード」＝'C'、「受注番号」＝仮受注番号 にする必要があります。実際のプログラムではそのロジックも入っています。

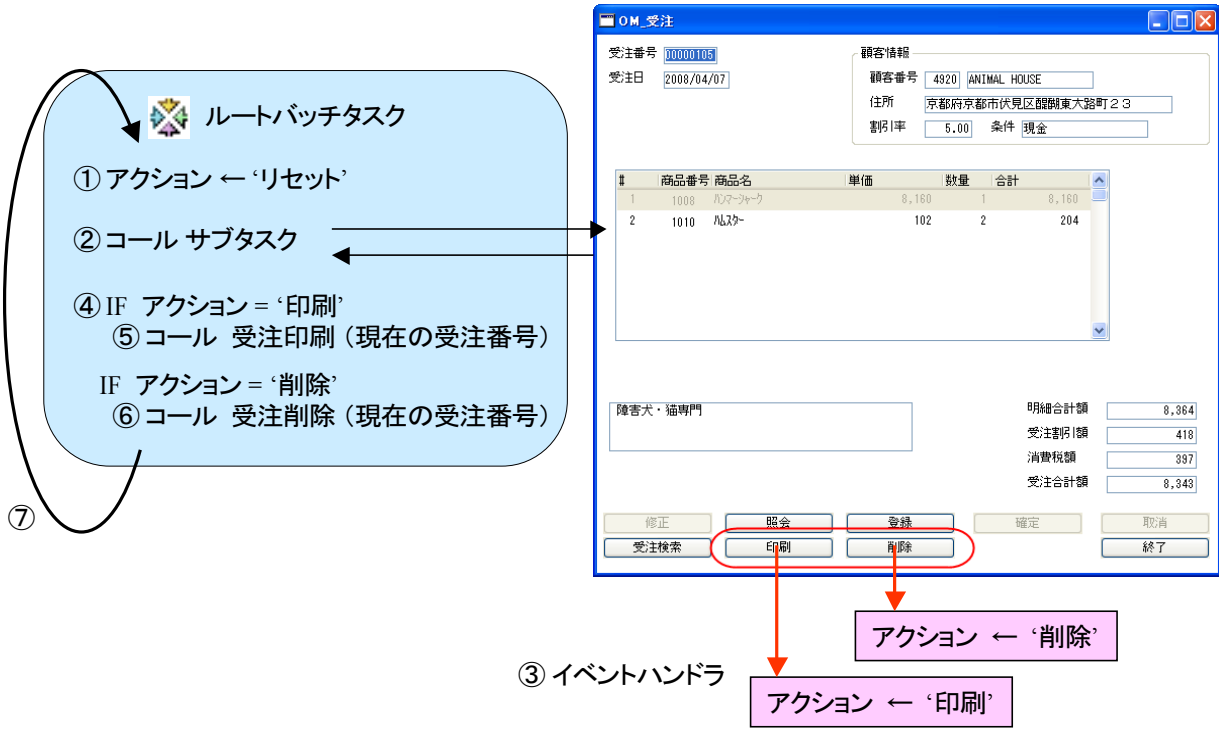
8.7 印刷および削除

「印刷」ボタンおよび「削除」ボタンは、次のような動作を行います。

- 「印刷」ボタンを押したら、現在表示中の受注データの内容がプリンタに出力されます。
- 「削除」ボタンを押したら、現在表示中の受注データが、明細データも併せて、削除されます。

このボタンは、いずれも、照会モードのとき、あるいは修正モードでまだデータへの修正が加えられていない状態でだけ有効化されています。

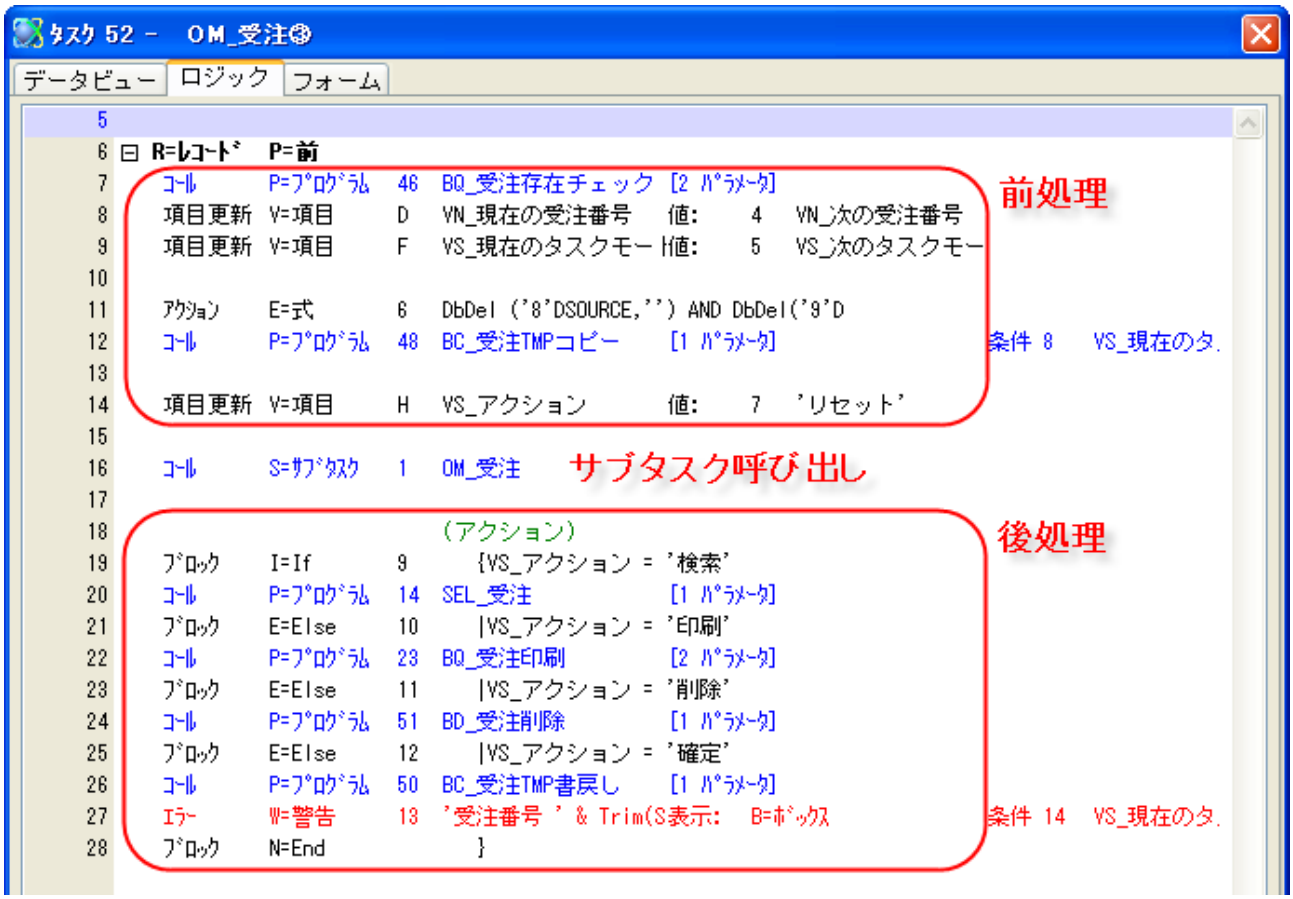
この処理は非常に単純で、いずれも、ルートバッチタスクの変数「VS_アクション」を使って制御します。すなわち、次のような処理になります（下図参照）。



1. ルートバッチタスクでは、「VS_アクション」を 'リセット' に設定する。
2. ヘッダタスク（サブタスク）を、コールコマンドで呼び出す。
3. ヘッダタスクでは、「印刷」ボタンのハンドラで、「VS_アクション」を '印刷' に設定して、タスクを終了する。
   同様に、「削除」ボタンのハンドラでは、「VS_アクション」を '削除' に設定して、タスクを終了する。
4. コールコマンドから戻ってきた後で、「VS_アクション」の値により、条件分岐を行う。
5. '印刷' になっていたら、印刷用のバッチタスクを呼び出す。
6. '削除' になっていたら、削除用のバッチタスクを呼び出す。
7. 次のループに進む。

8.8 ルートバッチタスクのロジック

最終的に、ルートバッチタスクのロジックは、下図のようになります。



この方式では、[PgUP]/[PgDown] キーによる受注レコードのスクロールはできません。別の受注レコードの内容を表示させるには、受注検索ボタンで受注一覧を表示させ、その中から選択する、という操作が必要になります。

9 ONL/物理/DSQL

本章で解説するバリエーションは、前章の「ONL/物理/MEM テーブル」と同様ですが、次の点が異なります。

- 一時テーブルとして、Memory ではなく、MSSQL Server のテーブルを用います。
- 受注テーブル/受注明細テーブルと、一時テーブルとのデータのコピーおよび書き戻しのために、Magic のバッチタスクではなく、MSSQL のストアドプロシージャを用います。

バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

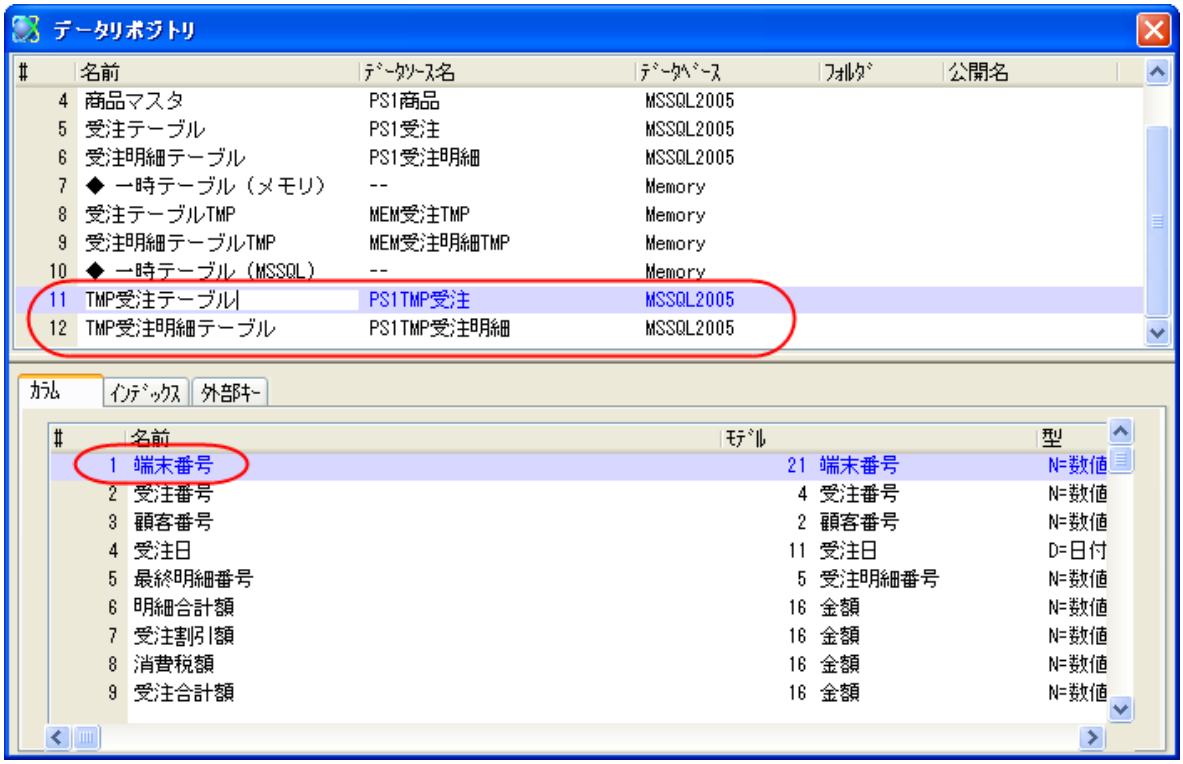
この方法は、Memory テーブルを使う方法に比べて、次のような長所短所があります。

- 一時データと本データとの間のデータ操作は、ストアドプロシージャにより DBMS 内で行われるので、高速に実行される。
- ストアドプロシージャを作成するスキルを持った開発者が必要となる。また、ストアドプロシージャの文法は、DBMS ごとに大きく異なるので、アプリケーションの移植性が悪くなる。

従って、この方法は、利用する DBMS が決まっていて移植性があまり重要でない場合、あるいは、SQL を得意とする開発者と、Magic 開発者とが共同でプロジェクトを進めるような場合に適した方法であると言えます。

9.1 データリポジトリ

データリポジトリでは、一時テーブルを MSSQL 上に定義します(下図)。

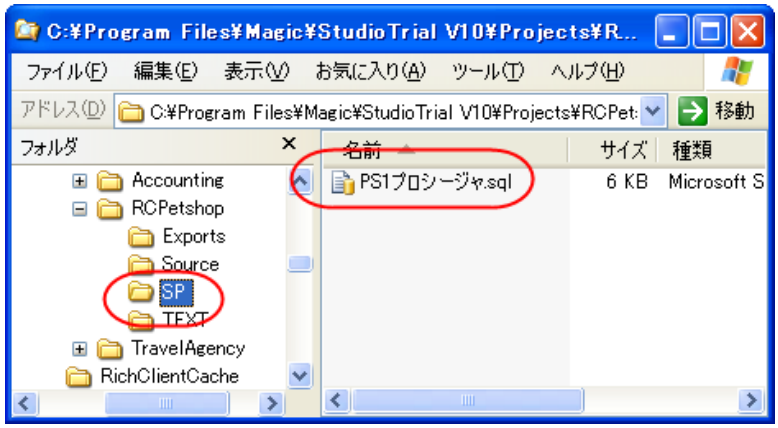


一時テーブルには、ヘッダと明細テーブルに対応して、それぞれ、「TMP 受注テーブル」(テーブル 11 番)と、「TMP 受注明細テーブル」(テーブル 12 番)とがあります。

それぞれのテーブルのカラム定義は、対応する元テーブルと基本的に同じですが、複数ユーザが共有するテーブルなので、ユーザを区別するために、「端末番号」カラムが追加されています。

9.2 ストアドプロシージャ

ストアドプロシージャは、「2.10 ストアドプロシージャの作成」で説明したように、プロジェクトディレクトリの下にある SP サブディレクトリに格納されています。



このファイルでは、次のようなストアドプロシージャが定義されています。

#	ストアドプロシージャ名	パラメータ	処理概要
1	PS1TMP 受注クリーン	端末番号	端末番号に対応する一時テーブル中のレコードを削除します。
2	PS1TMP 受注にコピー	端末番号 受注番号	指定された受注番号のレコード(ヘッダ/明細ともに)を、元テーブルから一時テーブルにコピーします。
3	PS1 受注更新確定	端末番号	一時テーブルの内容を、元テーブルに反映します。
4	PS1 受注削除確定	端末番号	現在一時テーブルに格納されている受注番号を使い、元テーブルの受注レコード(ヘッダ/明細とも)を、元テーブルから削除します。
5	PS1 受注登録確定	端末番号	一時テーブルに格納されているデータを、元テーブルにコピーします。これは登録モードの場合に使われます。

上表のうち、#3 ～ #5 のストアドプロシージャは、元テーブルのレコードを更新しますが、この場合同時に、顧客レコードの累計データ(受注累計額、取引回数)、商品レコードの在庫数の調整も行います。

以下は、ストアドプロシージャのソースです。

```
drop procedure PS1TMP受注クリーン
drop procedure PS1TMP受注にコピー
drop procedure PS1受注更新確定
drop procedure PS1受注削除確定
drop procedure PS1受注登録確定
go

-- ------------------------------------------------
create procedure PS1TMP受注クリーン
 @端末ID int
```

```
as
  delete from PS1TMP受注 where 端末番号 = @端末ID
  delete from PS1TMP受注明細 where 端末番号 = @端末ID
go

-- ------------------------------------------------
create procedure PS1TMP受注にコピー
 @p_端末ID int,
 @p_受注番号 int
as
  if (@p_受注番号 > 0)
    begin
     insert into PS1TMP受注
       select @p_端末ID, * from PS1受注 where 受注番号 = @p_受注番号
     if (@@rowcount > 0)
      begin
        insert into PS1TMP受注明細
           select @p_端末ID, * from PS1受注明細 where 受注番号 = @p_受注番号
      end
   end
go

-- ------------------------------------------------
create procedure PS1受注更新確定
 @p_端末ID int
as
declare @受注番号 int
declare @顧客ID int
declare @受注額 int
declare @商品ID int
declare @商品個数 int

 select @受注番号 = 受注番号, @顧客ID = 顧客番号, @受注額 = 受注合計額
   from PS1TMP受注
   where 端末番号 = @p_端末ID

 if (@@rowcount > 0)
  begin
   -- とりあえず古い受注明細を削除
   -- - 商品マスタの在庫数を調整
   update PS1商品
    set 在庫数 = i.在庫数 + m.数量
    from PS1商品 i, PS1受注明細 m
    where m.受注番号 = @受注番号
    and m.商品番号 = i.商品番号

   -- - 受注明細レコードを削除
   delete from PS1受注明細 where 受注番号 = @受注番号

   -- 新しい受注明細を挿入
   -- - 商品マスタの数量を調整
   update PS1商品
    set 在庫数 = i.在庫数 - m.数量
```

```
    from PS1商品 i, PS1TMP受注明細 m
    where m.受注番号 = @受注番号
    and   m.商品番号 = i.商品番号
    and   m.端末番号 = @p_端末ID

   -- - 受注明細レコードをTMPからコピー
   insert into PS1受注明細
     select 受注番号,受注明細番号,商品番号,商品タイプ,数量,単価,合計
     FROM PS1TMP受注明細
     where 端末番号 = @p_端末ID

   -- 受注レコードの情報を更新
   -- - 顧客マスタの情報を調整
   update PS1顧客
    set   受注累計額 = 受注累計額 - o.受注合計額,
          取引回数 = 取引回数 - 1
    from  PS1顧客 c, PS1受注 o
    where o.受注番号 = @受注番号
    and   o.顧客番号 = c.顧客番号

   update PS1顧客
    set   受注累計額 = 受注累計額 + n.受注合計額,
          取引回数 = 取引回数 + 1
    from  PS1顧客 c, PS1TMP受注 n
    where n.受注番号 = @受注番号
    and   n.顧客番号 = c.顧客番号
    and   n.端末番号 = @p_端末ID

   -- - 受注レコード更新
   UPDATE PS1受注
    SET
      顧客番号 = t.顧客番号,
      最終明細番号 = t.最終明細番号,
      受注日 = t.受注日,
      明細合計額 = t.明細合計額,
      受注割引額 = t.受注割引額,
      消費税額 = t.消費税額,
      受注合計額 = t.受注合計額
   from PS1受注 j, PS1TMP受注 t
   WHERE  t.受注番号 = @受注番号
    and   j.受注番号 = @受注番号
    and   t.端末番号 = @p_端末ID

  end
go

-- ------------------------------------------------
create procedure PS1受注削除確定
 @端末ID int
as
declare @受注番号 int
declare @顧客ID int
declare @受注額 int
```

```
 select @受注番号 = 受注番号, @顧客ID = 顧客番号, @受注額 = 受注合計額
   from PS1TMP受注
   where 端末番号 = @端末ID

 if (@@rowcount > 0)
  begin
   -- 受注明細を削除
   -- - 商品マスタの受注数を調整
   update PS1商品
    set 在庫数 = i.在庫数 + m.数量
    from PS1商品 i, PS1受注明細 m
    where m.受注番号 = @受注番号
    and m.商品番号 = i.商品番号

   -- - 受注明細レコードを削除
   delete from PS1受注明細 where 受注番号 = @受注番号

   -- 受注レコードの情報を更新
   -- - 顧客マスタの情報を調整
   update PS1顧客
    set 受注累計額 = 受注累計額 - o.受注合計額,
        取引回数 = 取引回数 - 1
    from PS1顧客 c, PS1受注 o
    where o.受注番号 = @受注番号
    and   c.顧客番号 = @顧客ID

   -- - 受注レコード削除
   delete from PS1受注
   WHERE 受注番号 = @受注番号

  end
go
-- ------------------------------------------------
create procedure PS1受注登録確定
 @p_端末ID int
as
 declare @受注番号 int
 declare @顧客ID int
 declare @受注額 int
 declare @商品ID int
 declare @商品個数 int

 select
    @顧客ID = 顧客番号,
    @受注額 = 受注合計額
   from PS1TMP受注
   where 端末番号 = @p_端末ID

 if (@@rowcount > 0)
  begin

   update PS1制御
```

```
    set    最終受注番号 = 最終受注番号 + 1
    where  制御キー = 1

  select
    @受注番号 = 最終受注番号
    from PS1制御
    where 制御キー = 1

  update PS1TMP受注
    set  受注番号 = @受注番号
    where 端末番号 = @p_端末ID

  update PS1TMP受注明細
    set  受注番号 = @受注番号
    where 端末番号 = @p_端末ID

  insert into PS1受注
    SELECT 受注番号, 顧客番号, 受注日, 最終明細番号, 明細合計額, 受注割引額, 消費税額, 受注合計額
    FROM   PS1TMP受注
    where  端末番号 = @p_端末ID

  insert into PS1受注明細
    SELECT 受注番号, 受注明細番号, 商品番号, 商品タイプ, 数量, 単価, 合計
    FROM   PS1TMP受注明細
    where  端末番号 = @p_端末ID

  update PS1顧客
    set
      受注累計額 = 受注累計額 + @受注額,
      取引回数 = 取引回数 + 1
    where
      顧客番号 = @顧客ID

  update PS1商品
    set   在庫数 = i.在庫数 - m.数量
    from  PS1商品 i, PS1TMP受注明細 m
    where m.端末番号 = @p_端末ID
      and m.商品番号 = i.商品番号

 end
go
```

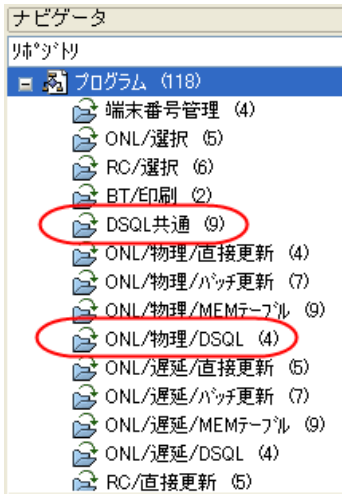
9.3 プログラム構成

9.3.1 フォルダ構成

右図は、プログラムリポジトリの中で、本方式に関連するプログラムを格納するフォルダを示したものです。

ここに見るように、以下の二つのフォルダ中のプログラムを主に使っています。

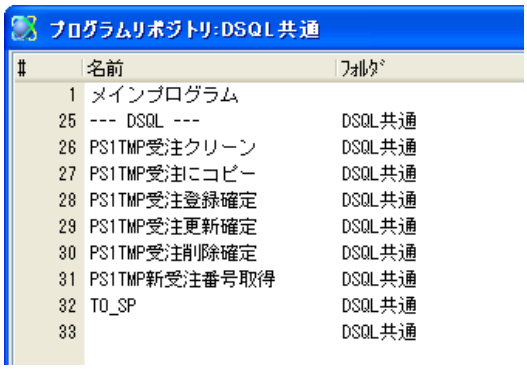
- DSQL 共通：ストアドプロシージャを呼び出す、埋め込み SQL のバッチタスクが収められています。
- ONL/物理/DSQL：本方式での受注入力プログラムが収められています。

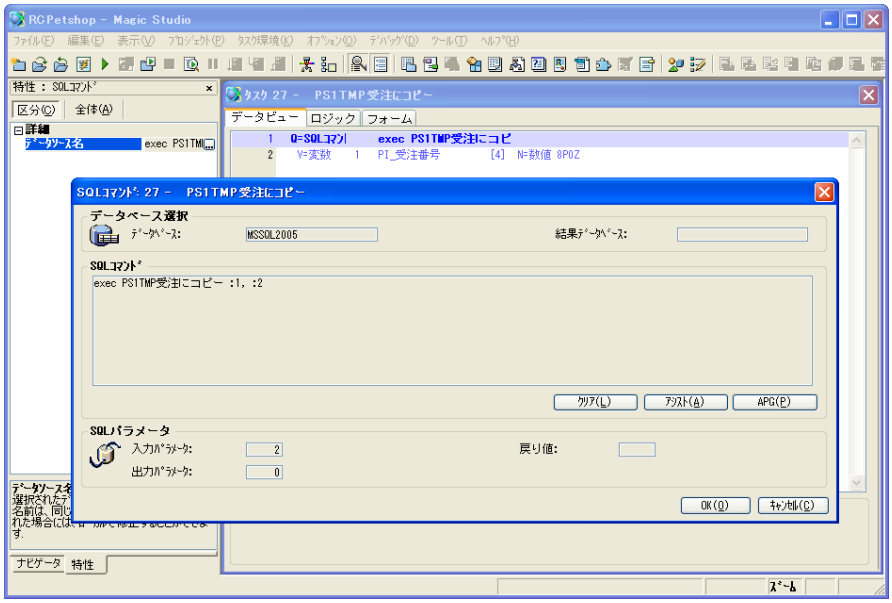


9.3.2 「DSQL 共通」フォルダ

「DSQL 共通」フォルダには、前節で説明したストアドプロシージャを呼び出すための、埋め込み SQL のバッチタスクが収められています。各ストアドプロシージャに、ひとつのバッチタスクが対応しています。

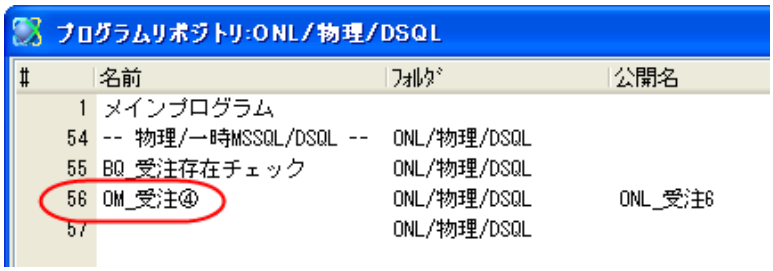
例えば、下図は、「PS1TMP 受注にコピー」ストアドプロシージャを呼び出すタスクの、埋め込み SQL 文を示したものです。



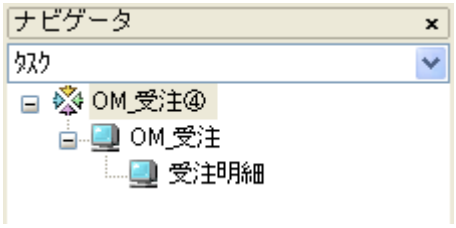


9.3.3 ONL/物理/DSQL フォルダ

「ONL/物理/DSQL」フォルダには、受注入力のためのプログラムが定義されています。(プログラム 56 番)

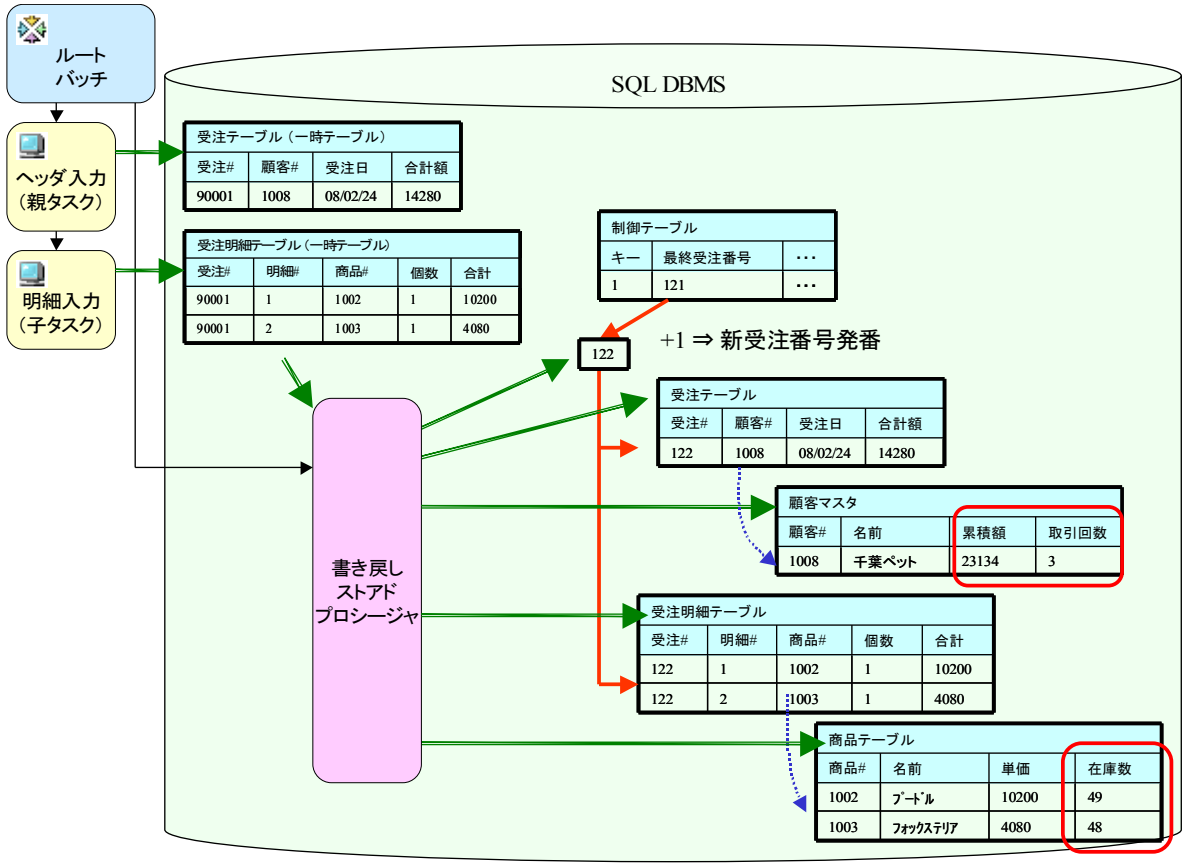


このプログラムは、右図で見るように、「ONL/物理/MEM テーブル」のパターンと同様、ルートバッチタスクを持つ 3 階層のタスクからなっています。



9.3.4 処理の流れ

処理の流れも、前節の「ONL/物理/MEM テーブル」のパターンとほとんど同じで、下図のようになります。一時テーブルが SQL DBMS 内に作成され、ストアドプロシージャで操作される点だけが異なります。



9.3.5 プログラム上の違い

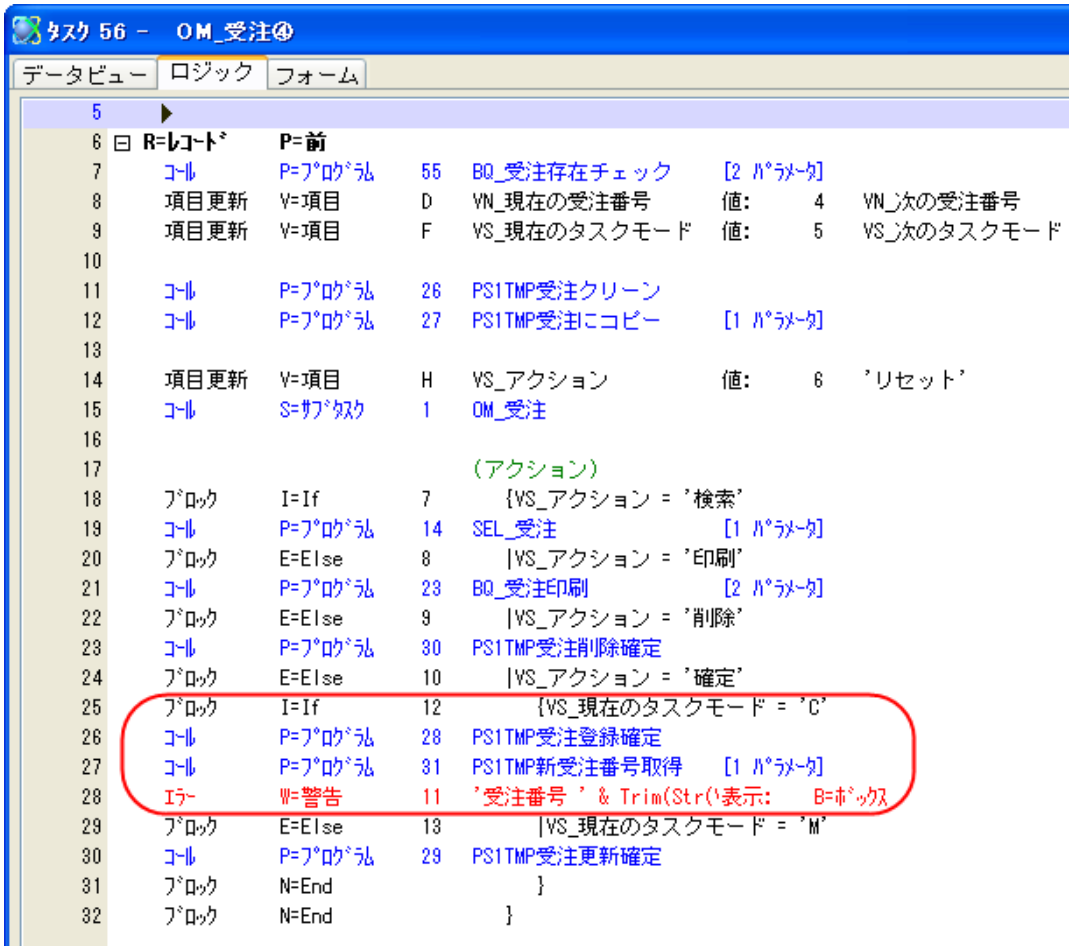
プログラム上の相違点としては、一時データ操作用に呼び出すバッチプログラムが異なる点はもちろんですが、それ以外に、登録時、新しい受注番号を別途取得する必要があります。これは次のような理由によります。

新規受注登録の場合、新しい受注番号は、ストアドプロシージャ「PS1 受注登録確定」の中で作成します。しかし、Magic の埋め込み SQL タスクの制限として、ストアドプロシージャからの戻り値などを受け取ることができず、作成された受注番号が Magic からはわかりません。

このため、次のようにして、作成された受注番号を Magic から取得するようにします。

1. ストアドプロシージャ「PS1 受注登録確定」では、新規作成した受注番号を、一時テーブル「PS1TMP 受注」の受注番号に設定します。
2. 別の埋め込み SQL バッチタスクを作り、単純な SELECT 文によって、「PS1TMP 受注」の受注番号を取得します。

下図に、ルートバッチタスクでこの処理を行っている部分を示します。



ストアドプロシージャからの値の受け取りについては、DBMS によって制限内容が異なり、Oracle などでは INOUT あるいは OUT パラメータもサポートされています。この場合には、上のようなことをする必要はありません。

10 オンライン・遅延トランザクション

本章では、前章と同じく、オンラインタスク（クライアント・サーバ）のタスクを扱いますが、前章とは異なり、物理トランザクションではなく、遅延トランザクションを使います。

第 5 章「実装方法のいろいろ」で示した表で言えば、下表の赤色の四角で囲まれたパターンになります。

アルゴリズム	タスクとトランザクション		
	ONL/物理	ONL/遅延	RC
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

遅延トランザクションを使ったプログラムの作成は、それぞれのパターンに対応する、物理トランザクションを使ったプログラムをコピーして、トランザクション設定を変更し、必要な修正を施す、という形で移植していきます。例えば、「ONL/遅延/直接更新」は、「基本形」を原型としてコピーして、トランザクションの設定を変更することにより移植します。

一般には、遅延トランザクションに変更することにより、次のような修正が必要となります。

- 排他制御の方法が変わる（悲観的ロックから、楽観的ロックになる）ため、排他制御が適切に行われるように修正が必要になることがある。
- バッチタスクを呼び出す場合には、未コミットのデータを正しく操作するために、バッチタスクのトランザクション設定も変更が必要になることがある。
- 登録モードでの重複チェックが即時に行われない場合があるので、重複チェックのためのロジックを追加する必要がある場合がある。

サンプルのような簡単なアプリケーションでは、遅延トランザクションにしたことに伴う変更はそれほど多くありません。そのため、遅延トランザクションを使ったパターンについては、本章ですべて4パターンとも説明します。

遅延トランザクションの概念と排他制御、トランザクションの範囲、移植時の注意事項等については、「Magic eDeveloper V10 遅延トランザクション」（弊社 Web サイトの「Magic スキルアップセンター」よりダウンロードできます）に詳しく説明してあるので、そちらを参考にしてください。この本に書いてある内容については、本書では、繰り返し詳述しません。

Magic スキルアップセンターの URL は、次の通りです。

http://www.magicsoftware.co.jp/training/introduction/introduction.html

10.1 ONL/遅延/直接更新

この型は、第 6 章「ヘッダ・明細型プログラムの基本形」を原型として、トランザクションの設定を、「物理」から「遅延」に変更しました。

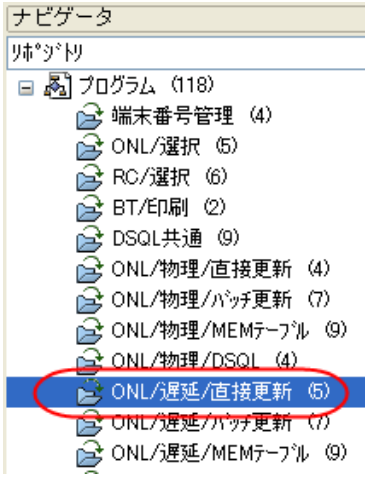
バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

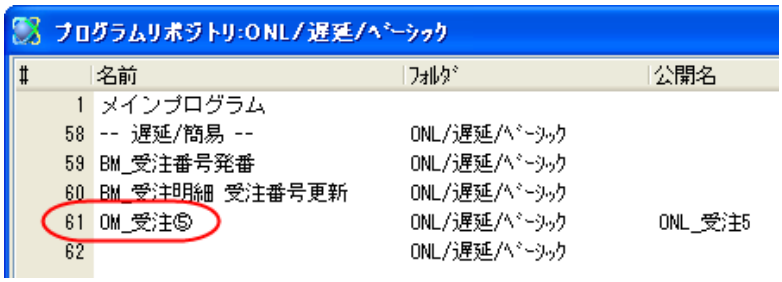
アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

このパターンは、タスク構造が簡単でありながら、マルチユーザ環境にも対応できるので、遅延トランザクションを使った場合のお勧め形です。

10.1.1 プログラム

このプログラムは、プログラムリポジトリで「ONL/遅延/直接更新」というフォルダに格納されています（下図）。

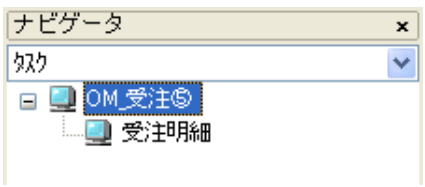




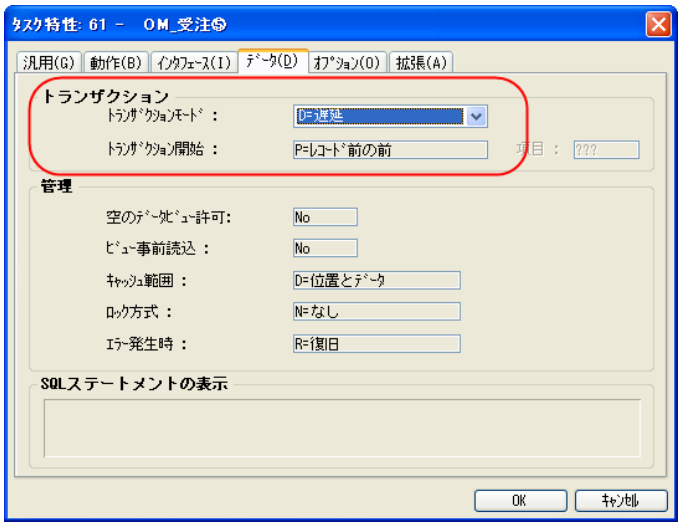
10.1.2 プログラム構造

プログラムの構造は、基本形と同じく、2 階層のオンライン親子タスクとして作成してあります。

親タスクがヘッダタスクで、子タスクが明細タスクになります。

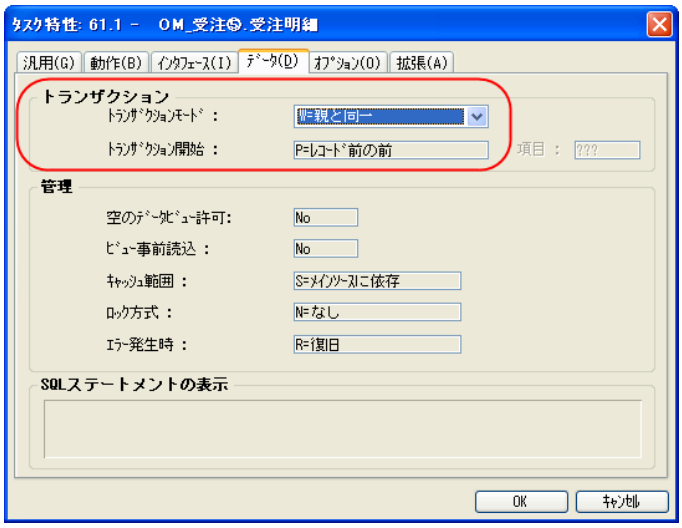


トランザクション設定は、ヘッダタスクでは「D=遅延」にします。



明細タスクでは、基本形と同じく、「W=親と同一」のままです。

この設定により、ヘッダタスクで遅延トランザクションが開始され、明細タスクはその遅延トランザクションの中で動作することになります。



10.1.3 排他制御

基本形では、6.26「複数ユーザ利用時の問題点」で説明したように、排他制御に問題があり、実質、同時には 1 ユーザだけしか利用することができませんでした。

遅延トランザクションの場合には、「楽観的ロック」を使うので、ロックによる同時実行の問題が大幅に緩和されます。楽観的ロックというのは、簡単に言えば、

- ユーザの入力時には、DBMS に対するロックをかけない。
- レコードを DBMS に書き込むタイミングで、更新レコードが他のユーザによって変更されていないかを確認する。

という方法です。

そのため、基本形から移行した、本節のパターンでも、同時に複数ユーザが利用できるものを作ることができます。

10.1.4 リンクのアクセスパラメータ

遅延トランザクションでは、レコードロックがかからないので、ロックの衝突により他のユーザが利用不能になってしまうということがありません。従って、リンクの「アクセス」パラメータは「W=書込」のままで構いません。

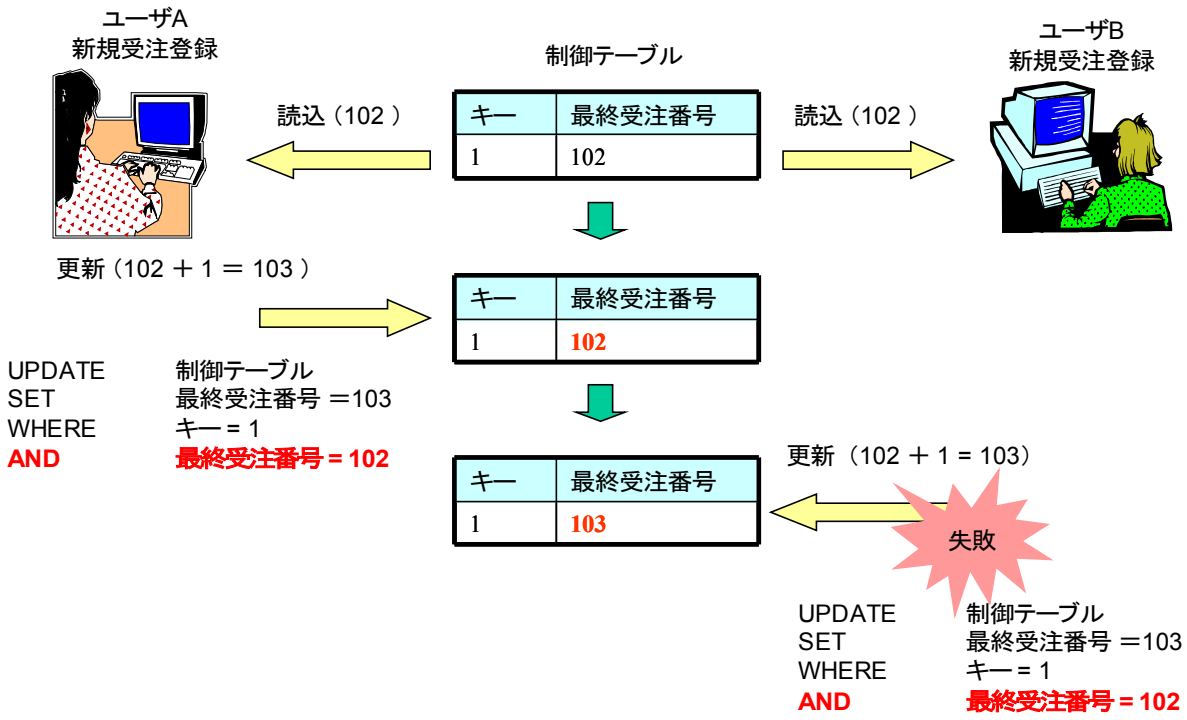
下図は、親タスクの「顧客マスタ」へのリンクです。制御テーブル、サブタスクの商品マスタなども同様です。

10.1.5 受注番号の発番

基本形では、受注番号の発番を行うのに、制御テーブルをリンクして、その最終受注番号に ＋1 を行って載番していました。

このままの方式で遅延トランザクションにすると、レコードロック解除待ちは起こらなくなりますが、確定のタイミングで「他のユーザが更新しました」のエラーが多発してしまいます。

例えば、次の図は、エラーの起こる様子を例で示したものです。



1. 最初、ユーザ A が新規受注登録を始めます。このとき、制御テーブルの「最終受注番号」は 102 です。
2. 次に、ユーザ B が新規受注登録を始めます。このときも、制御テーブルの「最終受注番号」は 102 です。
3. ユーザ A が受注内容の入力を終え、確定します。このタイミングで、最終受注番号が ＋1 されて、103 となります。このとき、「楽観的ロック」のメカニズムによって、最終受注番号の値が他のユーザによって

変更されていないかがチェックされるので、データ更新には次のような UPDATE 文が発行されます。

UPDATE 制御テーブル SET 最終受注番号 ＝103 WHERE キー = 1 **AND 最終受注番号 = 102**

この UPDATE 文は、ここでは問題なく成功し、「最終受注番号」は 103 になります。

4. 次に、ユーザ B が受注内容の入力を終え、確定しようとします。受注番号は、先に読み込んできた最終受注番号 102 に ＋1 した値 103 となりますが、この番号はすでにユーザ A により使われており、制御テーブルの最終受注番号は 103 になってしまっています。ここで、ユーザ B が制御テーブルに書込みを行おうとすると、「楽観的ロック」のメカニズムによって、ユーザ A と同様、次の UPDATE 文が発行されます。

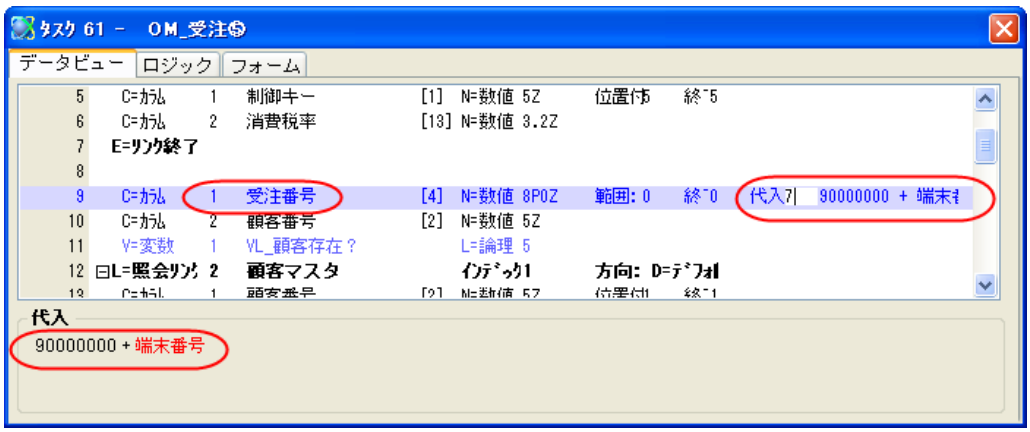
   UPDATE 制御テーブル SET 最終受注番号 ＝103 WHERE キー = 1 **AND 最終受注番号 = 102**

   このときには、最終受注番号がすでに 103 になっているので、この UPDATE 文は失敗となります。

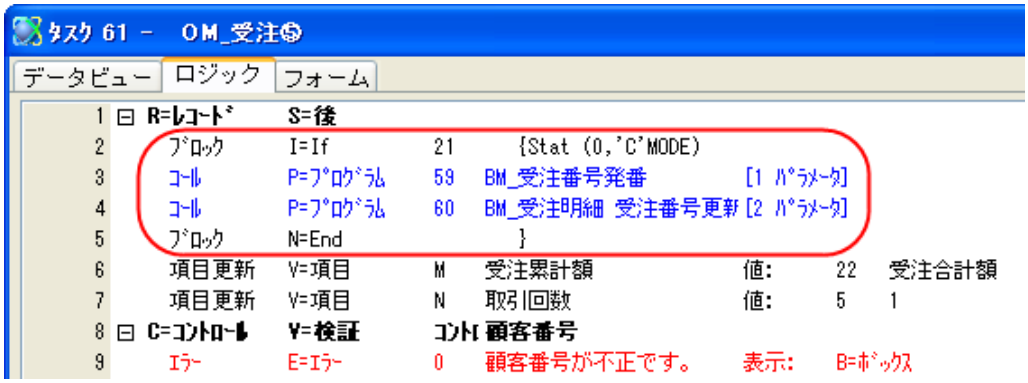
UPDATE 文が失敗すると、遅延トランザクションのコミットの失敗となり、一般的にはユーザ B のトランザクションをロールバックして、最初から入力しなおしということになります。

このエラーが多発するようでは、実用にならないので、受注発番のロジックは ONL/遅延/バッチ更新 の方式を採用します。すなわち次のようにします。

1. 最初は、仮番号で登録します。(仮番号としては、実存する可能性のない大きな番号 ＋ 各端末ごとにユニークな番号、とします)



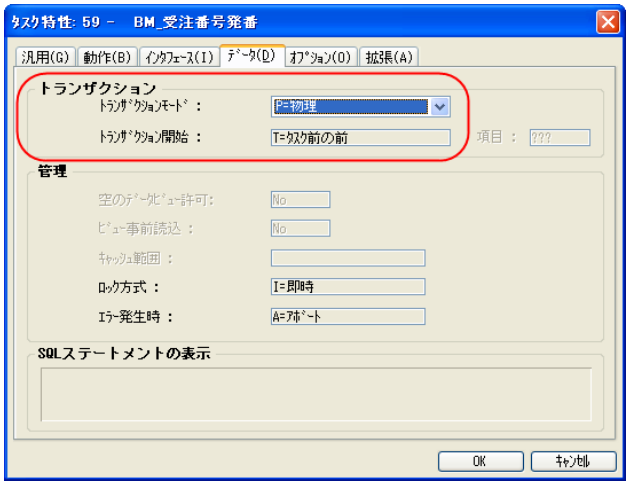
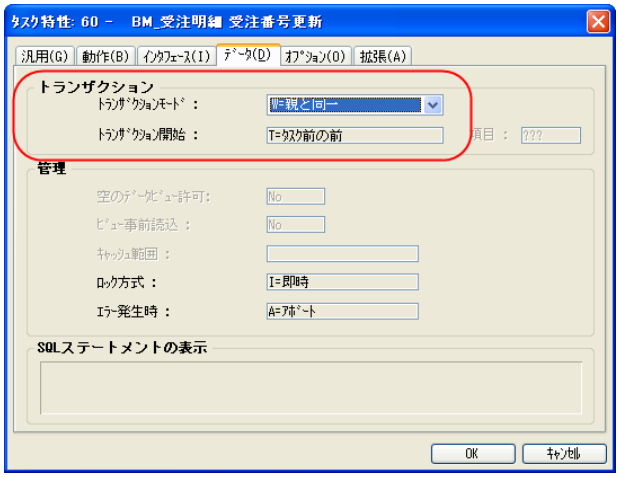
2. 確定時に、受注番号発番のためのバッチタスクを呼び出し、受注番号をつけかえます。このとき、明細行の受注番号も付け替えます。



ここで、以下の二つのバッチタスクを呼び出します。

- 受注番号の発番
- 受注明細番号の受注番号更新

このバッチタスクでのトランザクションの設定は重要です。結論から言えば、次のように設定します。

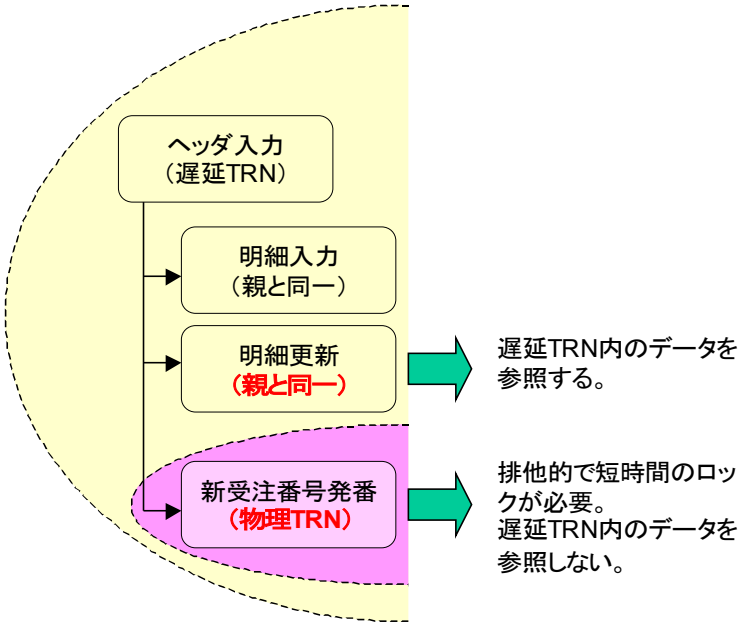
バッチタスク	トランザクション設定
受注番号の発番	P=物理/T=タスク 
受注明細番号の受注番号更新	W=親と同一 

このように設定する理由は、次の通りです。

- 「受注番号の発番」タスクは、悲観的・楽観的に関わらず、ロックの期間は極力短くしておかなければなりません。また、トランザクションキャッシュ中の未コミットデータを参照することはありませんから、同一遅延トランザクションで実行させる必要はありません。このような用途には、物理トランザクションが最適です。
- 受注明細番号の受注番号更新のバッチタスクは、ユーザが遅延トランザクション中に入力した受注明細データを参照します。この明細データは、このタイミングではまだコミットされていないので、DBMS には存在しておらず、Magic のトランザクションキャッシュの中にだけ存在します。従って、このデータを参照するには、同一遅延トランザクションの中で動作させる必要がありますから、トランザクション設定は

「W=親と同一」にする必要があります。

下図は、タスクの呼び出し構造と、トランザクションの範囲とを図示したものです。ヘッダ入力、明細入力、明細更新バッチが同一の遅延トランザクション内で動作する必要があり、受注番号発番バッチが別の物理トランザクションで実行されることを示しています。



このように設定することにより、エラーを起こすことなく、受注番号の発番を正しく行うことができるようになります。

10.2 ONL/遅延/バッチ更新

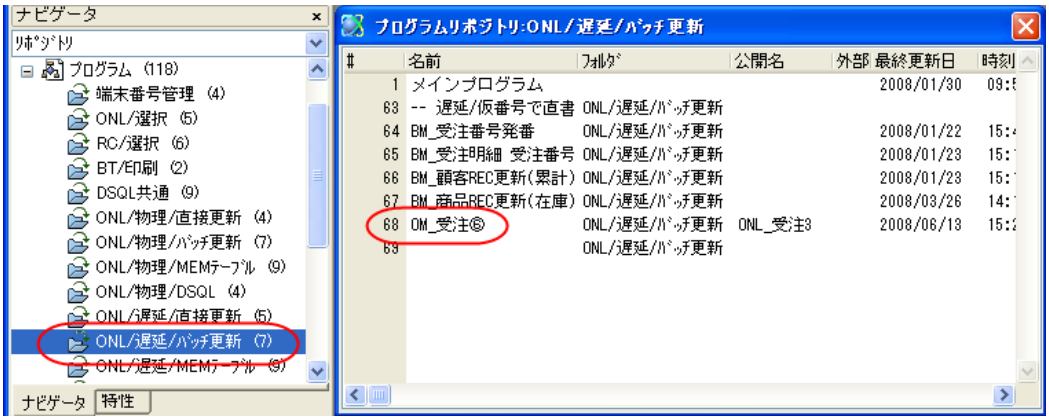
次には、「ONL/物理/バッチ更新」のタスクから、トランザクション設定のみを変更して、「ONL/遅延/バッチ更新」に変更します。

バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

10.2.1 プログラム

プログラムリポジトリの「ONL/遅延/バッチ更新」フォルダに受注入力プログラムと関連するバッチプログラムが格納されています。



10.2.2 トランザクション設定

受注入力オンラインプログラム (プログラム 68 番「OM_受注実際⑥」) は、トランザクションの設定を変更するだけです。

バッチプログラムについては、前節「ONL/遅延/直接更新」と同様の考え方により、次のような設定になっています。

- BM_受注番号発番: P=物理/T=タスクレベル
- その他のプログラム: W=親と同一

このパターンでは、すでに物理トランザクションを使った場合でも複数ユーザの同時利用に対応しているので、遅延トランザクションにするメリットがあまり生かされない形になります。出来上がった結果としては、必要以上に複雑になりますが、既存の物理トランザクションを使ったプログラムを、工数をかけずに単純移行したい場合にはこのパターンを使うのもよいと思います。

10.3 ONL/遅延/MEM テーブル

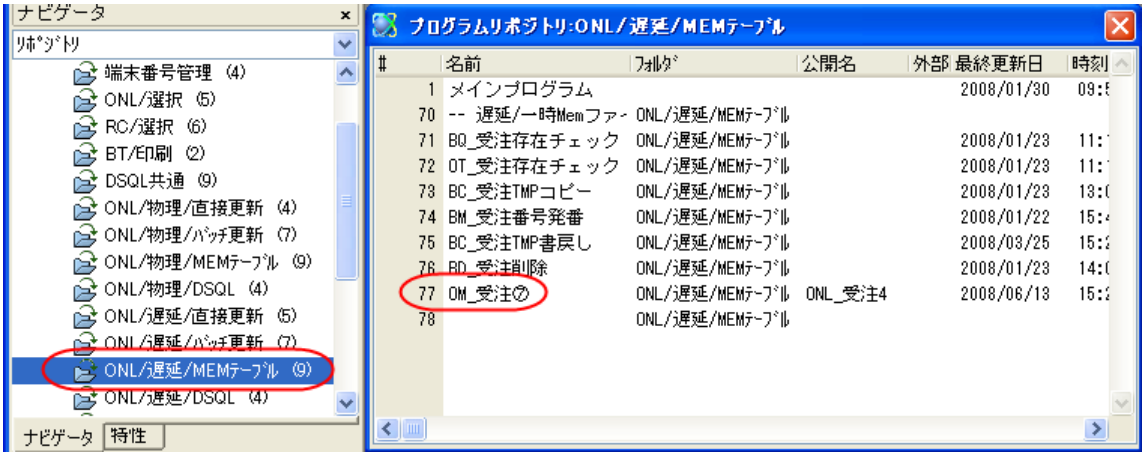
ここでは、「ONL/物理/MEM テーブル」のタスクから、トランザクション設定のみを変更して、「ONL/遅延/MEM テーブル」に変更します。

バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

10.3.1 プログラム

プログラムリポジトリの「ONL/遅延/MEM テーブル」フォルダに受注入力プログラムと関連するバッチプログラムが格納されています。

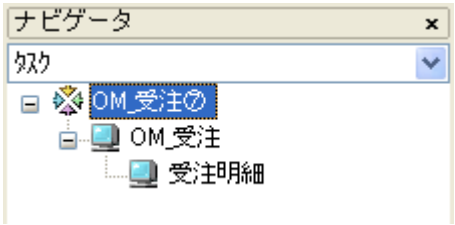


10.3.2 プログラム構造

タスク構造やロジックなど、ほとんどそのままです。右図は、プログラム 77 番「OM_受注⑦」のタスク構造ですが、

- ルートバッチタスク
- ヘッダ入力用のサブタスク（ヘッダタスク）
- 明細入力用の孫タスク（明細タスク）

という構成からなっています。



10.3.3 トランザクション設定

このプログラムでは、トランザクションの設定を次の表のように変更しています。

レベル	タスク名	タスクタイプ	メインソース	トランザクション設定
親（ルート）	OM_受注⑦	バッチ	なし	物理/なし
子	OM_受注	オンライン	受注受注テーブル TMP	遅延
孫	受注明細	オンライン	受注明細テーブル TMP	親と同一

また、このプログラムから呼び出すバッチタスクのトランザクション設定は 、次表の通りです。（プログラム72「OT_受注存在チェック」は、テスト用のオンラインタスクなので、省略しています。）

#	タスク名	用途	トランザクション設定
71	BQ_受注存在チェック	受注番号やタスクモードのチェックと確定	物理（遅延でも可）
73	BC_受注 TMP コピー	DBMS から一時テーブルにコピー	物理（遅延でも可）
74	BM_受注番号発番	登録時、新受注番号を発番	物理（必須）
75	BC_受注 TMP 書戻し	一時テーブルから DBMS にコピー	親と同一（必須）
76	BD_受注削除	明細行を削除	物理（遅延でも可）

10.3.4 まとめ

結果として、このパターンでは、遅延トランザクションを使ってはいますが、遅延トランザクションの利点をほとんど利用していません。また、遅延トランザクション自体に、トランザクションキャッシュという、一時テーブルに相当する機能のものがあるので、一時テーブルに遅延トランザクションを利用するというのは、屋上屋を重ねるような形になります。

従って、このパターンは、すでに物理トランザクションで一時テーブルを扱うプログラムがある場合に、工数をかけずに遅延トランザクションに単純移行したい場合には使うとよいと思います。

また、別の利点として、一時テーブルに対する細かな操作・制御を行える利点がある。エラー発生時のエラーハンドリング時に、きめ細かな制御を行いたいときにも、この方法を使うメリットがありましょう。

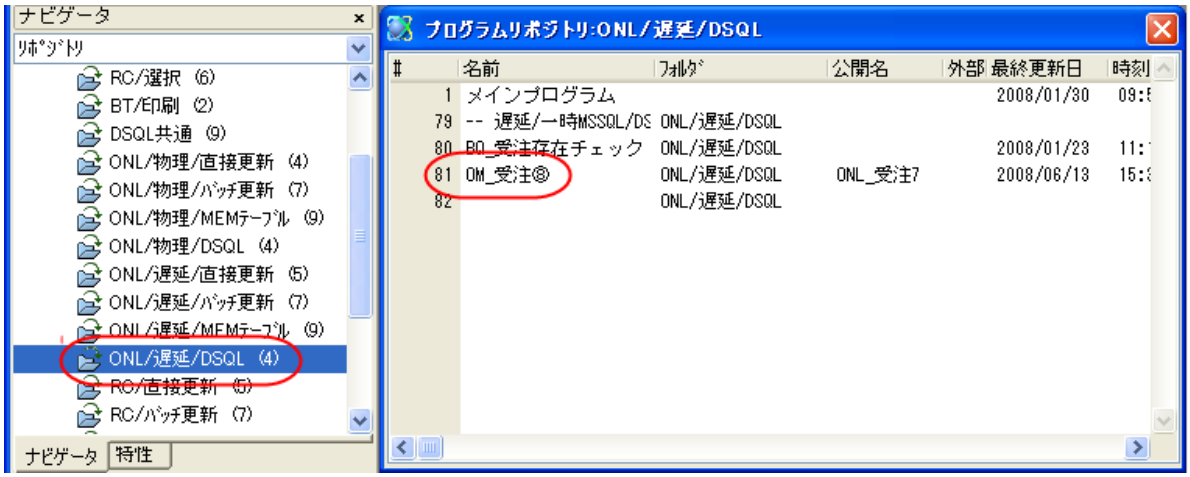
10.4 ONL/遅延/DSQL

ここでは、「ONL/物理/DSQL」のタスクから、トランザクション設定のみを変更して、「ONL/遅延/DSQL」に変更します。
バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

10.4.1 プログラム

プログラムリポジトリの「ONL/遅延/DSQL」フォルダに受注入力プログラムと関連するバッチプログラムが格納されています。

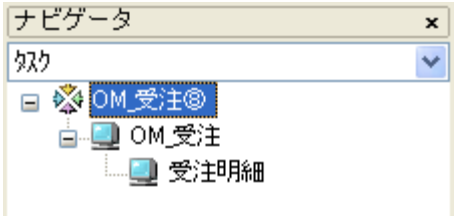


10.4.2 プログラム構造

タスク構造やロジックなど、ほとんどそのままです。右図は、プログラム 77 番「OM_受注⑧」のタスク構造ですが、

- ルートバッチタスク
- ヘッダ入力用のサブタスク（ヘッダタスク）
- 明細入力用の孫タスク（明細タスク）

という構成からなっています。



10.4.3 トランザクション設定

「オンライン/物理/DSQL」のタスクで、トランザクション設定のみ変更すればできあがりです。次の表に、トランザクションの設定を示します。

レベル	タスク名	タスクタイプ	メインソース	トランザクション設定
親	OM_受注⑧	バッチ	なし	物理/なし
子	OM_受注	オンライン	受注受注テーブル TMP	遅延/レコード前の前
孫	受注明細	オンライン	受注明細テーブル TMP	親と同一

10.4.4 ストアドプロシージャ呼び出し

ストアドプロシージャを呼び出す埋め込み SQL のバッチタスクもそのまま使っています。

10.4.5 まとめ

このパターンでも、前節のパターンと同様、一時テーブルを使っている上に遅延トランザクションを使っているので、結果として必要以上に複雑となりますが、既存のプログラムを遅延トランザクション対応に単純移行する場合に使えましょう。また、前節と同様、一時テーブルの細かな制御が必要な場合にもよいでしょう。

11 リッチクライアント

本章では、前章までに作成したオンラインプログラムを、V10.1SP4b での新機能であるリッチクライアントに移行してみます。

本書はリッチクライアントの解説書ではないので、リッチクライアントの基本的な事項の説明はしません。次のようなドキュメントを参照してください。

- リファレンスヘルプ ⇒ Web 開発 ⇒ リッチクライアントアプリケーション
- インタラクティブなリッチクライアントの開発と実行（製品添付 PDF）

オンラインタスクとリッチクライアントタスクの相違点については、第 13 章「リッチクライアントとオンラインとの違い」にまとめておきました。この内容は、10.1SP4b の製品 README.CHM の以下の記述から転載したものです。

- V10 追加情報 ⇒ 参考技術情報 ⇒ リッチクライアントのオンラインとの違い

オンラインとリッチクライアントとでは、プログラムの基本的な開発方法は似ているのですが、異なるところもあります。主要な相違点を挙げれば、以下のようなものがあります。

1. リッチクライアントの場合、トランザクション設定は 遅延トランザクションのみで、物理トランザクションは設定できません。

   従って本章では、前章で説明した遅延トランザクション対応のプログラムを基にして、リッチクライアントに移行するようにします。
2. リッチクライアントでは、クライアント側とサーバ側との通信が発生します。通信回数とデータ量とが、パフォーマンスに大きな影響を与えるので、この点の考慮が必要になります。

   この話題は重要な話題ですが、本書の範囲を超えてしまうので、本書では扱いません。
3. レコードメイン互換レベルはサポートされていません。すべて、イベントを使ってプログラムロジックを書きます。

   本書のサンプルは、はじめからレコードメイン互換を使っていないので、この点は問題になりませんが、もし V8 以前から移行してきたクライアントサーバのアプリケーションをリッチクライアントに移行する場合には、まず、オンラインプログラムでレコードメイン互換をイベントで書き直して、その後、リッチクライアントの移行するようにしてください。
4. ファントムタスクはサポートされていません。すべて、サブフォームを使って書きます。

   本書のサンプルでは、最初からサブフォームを使っており、ファントムタスクを使っていないのでこの点も大丈夫ですが、過去のバージョンから移行してきたアプリケーションでは、オンラインプログラムで、サブフォームを使って書き直してから、リッチクライアントに移行するようにしてください。
5. また、オンラインとリッチクライアントで、サブフォームの動作が異なりますので、この点も考慮を払う必要があります。

   本書のサンプルは単純なロジックなので、サブフォームの動作の差異によって影響を受けることがあまりありませんが、画面のインターフェースを作りこんであるオンラインプログラムをリッチクライアントに移行したら、サブフォームまわりの制御を調整する必要があります。

   11.2「リッチクライアントのサブフォーム」で、この点について簡単に触れます。
6. 印刷やテキスト入出力を行う場合には、クライアント側とサーバ側の処理の違いについて考慮が必要になり、書き直しが必要になります。これも重要な話題ですが、本書の範囲を超えてしまうので、省略します。
7. そのほか、オンラインでサポートされていて、リッチクライアントでサポートされていない機能があります。ここではすべて列挙することはできませんが、適当な形で同等のことを行うように、プログラムを書きなおす必要があります。

例えば、本書のサンプルでは、「選択プログラム」特性や「フローモード」特性がリッチクライアントでサポートされていないので、同等のことを行うよう、プログラムに手を加えています。11.1「RC/直接更新」でこの点について簡単に触れています。

オンラインタスクとリッチクライアントタスクの相違点については、第 13 章「リッチクライアントとオンラインとの違い」にまとめておきました。この内容は、10.1SP4b の製品 README.CHM の以下の記述から転載したものです。

- V10 追加情報 ⇒ 参考技術情報 ⇒ リッチクライアントのオンラインとの違い

11.1 RC/直接更新

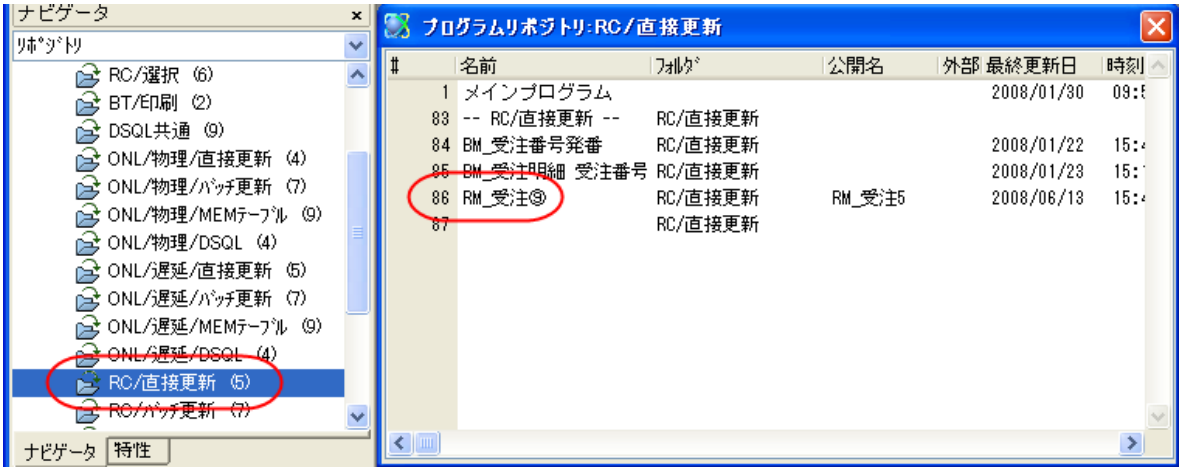
ここでは、リッチクライアントの受注入力の基本形を扱います。このプログラムは、オンラインで遅延トランザクションを使った「ONL/遅延/直接更新」(10.1節参照)より移行して作りました。

バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

11.1.1 プログラム

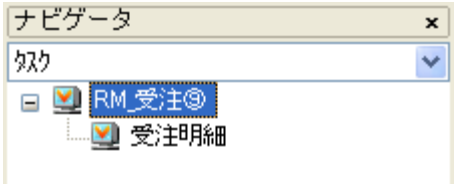
このプログラムは、プログラムリポジトリ「RC/直接更新」というフォルダに格納されています。



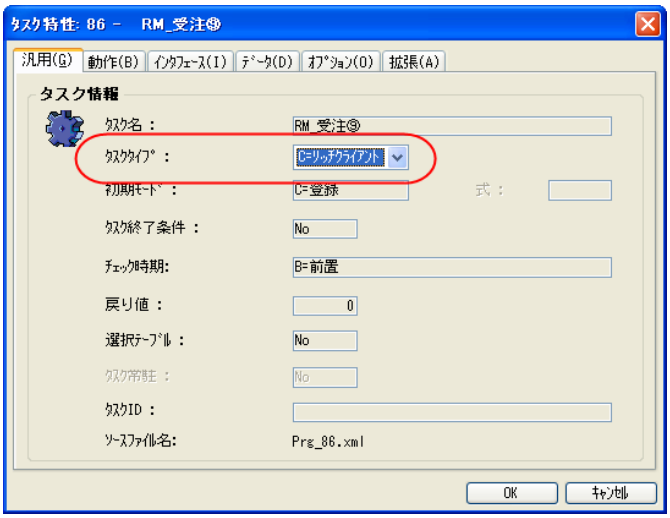
RC タスクでは、公開名が必須になります。上図では、「RM_受注 5」という公開名が設定されています。

11.1.2　プログラム構造

プログラムの構造は、「ONL/遅延/直接更新」と同じく、オンラインの 2 階層です。



タスクタイプをオンラインからリッチクライアントに変更しました。サブタスクのタスクタイプもリッチクライアントに変更します。



このプログラムでは、これだけでリッチクライアントの受注入力プログラムになります。マルチユーザ環境にも対応しています。

11.1.3　リッチクライアント非サポート機能

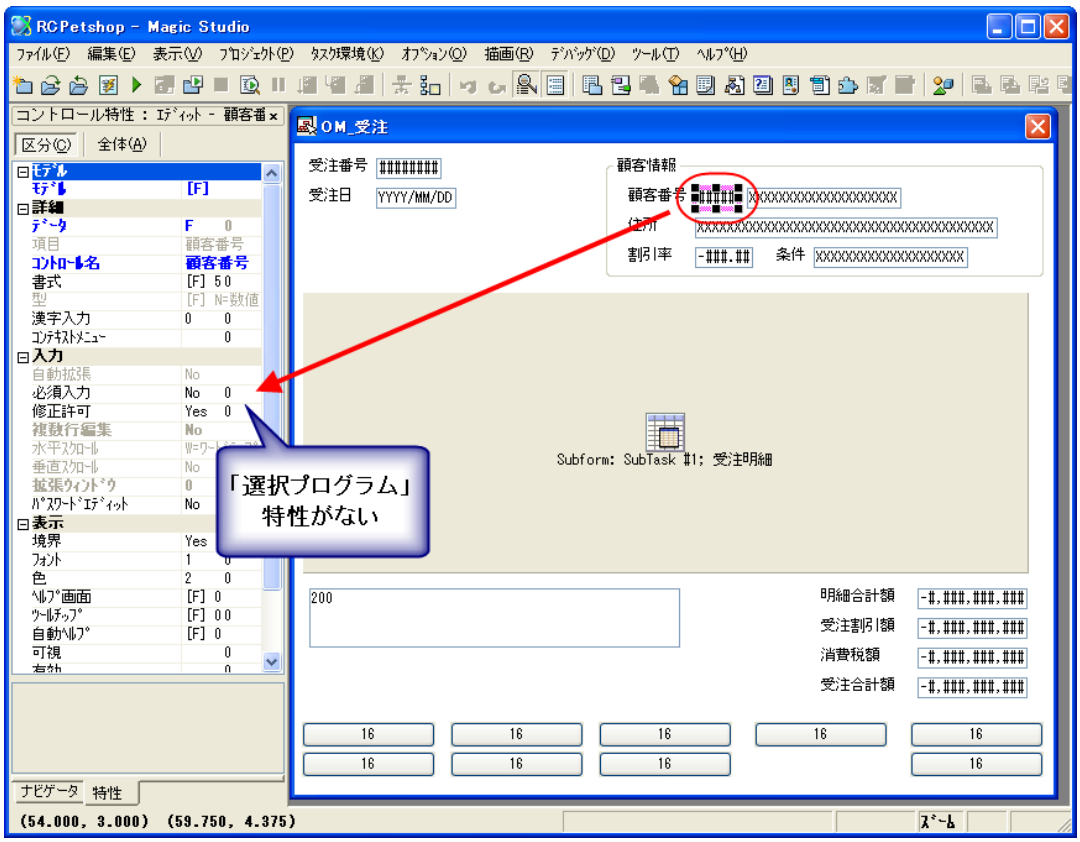
オンラインからリッチクライアントに移行したとき、リッチクライアントでサポートされていない機能は削除されてしまうので、適当な形で書きなおす必要があります。

ここでのサンプルでは、次の機能が非サポートとなり、機能が削除されているので、修復します。

- 選択プログラム
- フロー特性

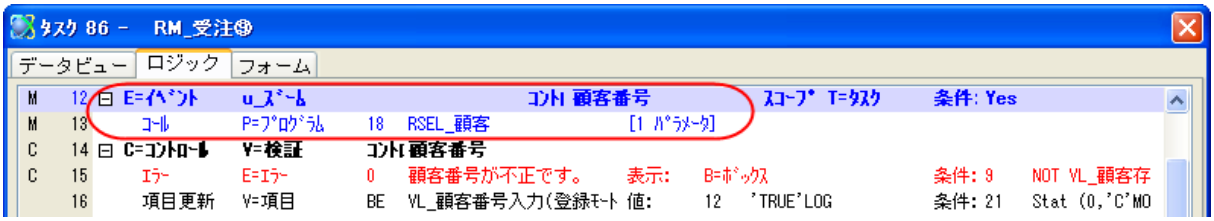
11.1.4　選択プログラム

次の図は、リッチクライアントに移行した直後の、親タスクのフォームエディッタを開いたところです。



ここで「顧客番号」項目の特性を見てみると、オンラインでは存在していた「選択プログラム」特性が、リッチクライアントではなくなっていることがわかります。この結果、実行時には F5 キーやダブルクリックによるズーム機能が効かなくなってしまいます。

これに対応するために、ズームイベントハンドラを追加します（下図）。

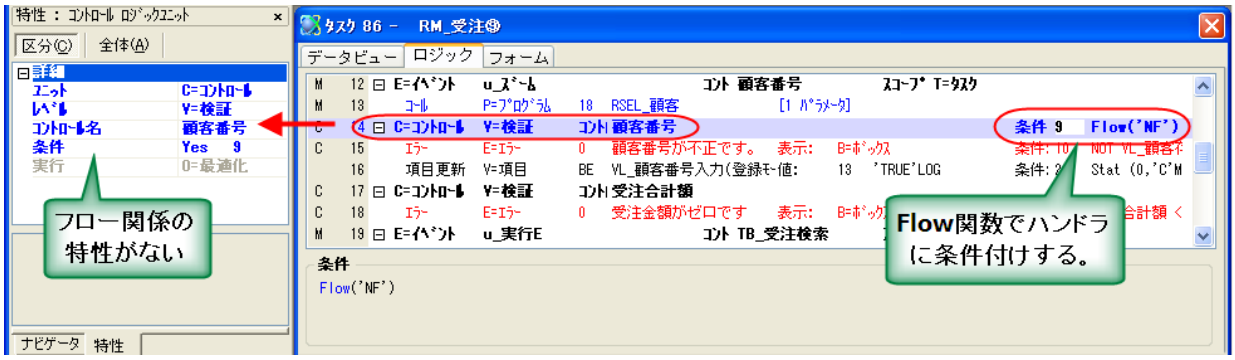


同様に、サブタスクの「商品番号」にも「選択プログラム」特性がなくなってしまっているので、サブタスクにズームイベントハンドラを追加します。（図は省略します）

11.1.5 フロー特性

リッチクライアントタスクでは、コマンドの「フローモード」「フロー方向」といったフロー特性がサポートされていません。オンラインからリッチクライアントに移行した場合、これらの特性は削除されてしまうので、実行時に動作が違ってきてしまいます。

これに対応するために、Flow 関数を使った式で、条件付けをします。下図の例では、Flow ('NF') を条件として指定することにより、オンラインで「フロー方向」＝「F=前方向」と指定してあったのと同じ動作となるようにしています。



Flow 関数についてのより詳しい情報は、以下のキーワードでリファレンスヘルプを検索してください。

- Flow 関数

11.2 リッチクライアントのサブフォーム

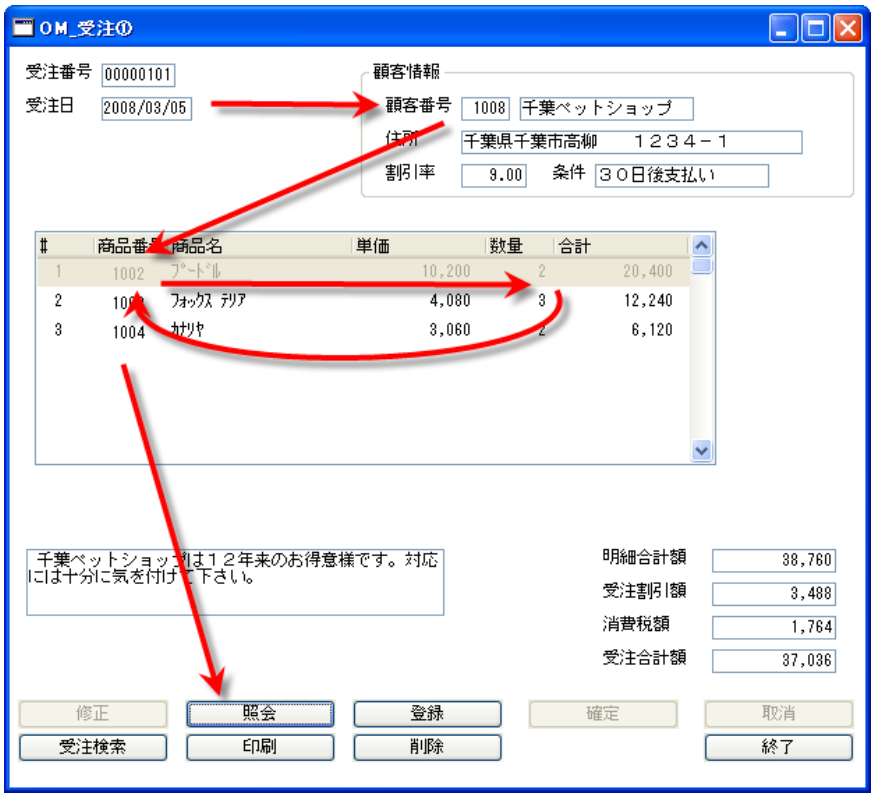
オンラインとリッチクライアントとで、サブフォームの動作が異なります。この点について、サンプルを使って確認しておきましょう。

11.2.1 カーソルの動き

最初に Tab キーを押した場合のカーソルの動きについて調べてみます。

オンラインの場合

オンラインプログラムとして、基本形 (プログラム 36 番「OM_受注①」) を実行してください。ここで、画面操作をすると、下図のように動きます。

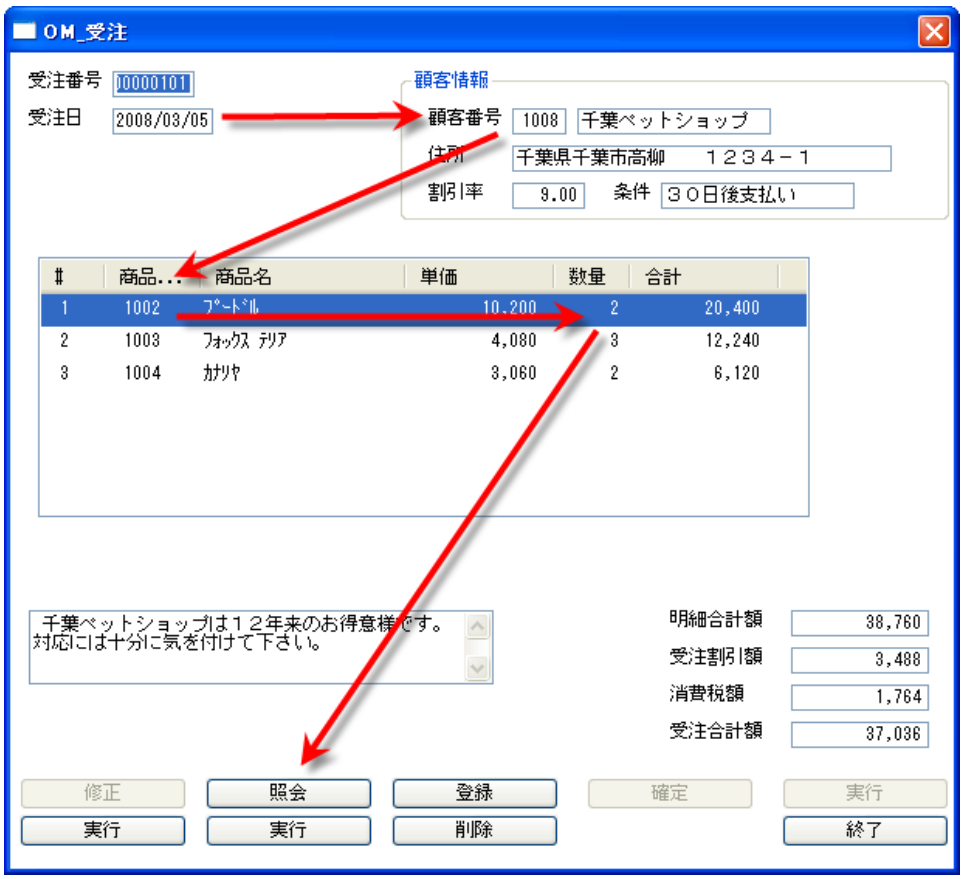


1. 初期状態は登録モードなので、「修正」ボタンを押して修正モードになります。
2. Tab キーを押すと、「受注日」 ⇒ 「顧客番号」 ⇒ サブフォームに入り、「商品番号」という順にカーソルが移動します。
3. さらに Tab キーを押すと、「商品番号」 ⇒ 「商品名」⇒「数量」⇒「商品番号」 ⇒ ・・・ と、明細行レコードの中を循環します。
4. ESC キーを押すと、サブフォーム内の明細行の項目から、親タスクの次の項目 (「照会」ボタン) に移行します。

この動きは、ファントムの技法を使って行っていたときと同じ動きであり、Magic 開発者にとってはなじみの深いものです。

リッチクライアントの場合

リッチクライアントの場合には、若干変わった動きになります。リッチクライアントの代表として、前節で説明した、リッチクライアントの基本形 「RC/直接更新」を実行し、同じ操作を行って、動作を見てみます。



1. 初期モードが登録なので、「修正」ボタンを押して、修正モードになります。
2. Tab キーを押していくと、「受注日」 ⇒ 「顧客番号」 ⇒ サブフォームに入り、「商品番号」という順にカーソルが移動していきます。ここまではオンラインの場合と同じです。
3. 明細行の上で、Tab キーを押します。すると、「商品番号」 ⇒ 「数量」の後、明細行の先頭には戻らず、親フォームの「照会」ボタンに進みます。
4. ここで、マウスを使って、明細行の「商品番号」にカーソルを置いてください。この状態で ESC キーを押すと、オンラインタスクでは、カーソルが明細行から離れ、親フォームの「照会」ボタンに行きましたが、リッチクライアントの場合には、受注入力プログラム全体が終了し、ウィンドウがクローズしてしまいます。

このように、リッチクライアントの場合には、サブフォームを使うと、ヘッダタスクと明細タスクとが一体となって、ひとつのタスクになったかのようなカーソルの動きとなります。

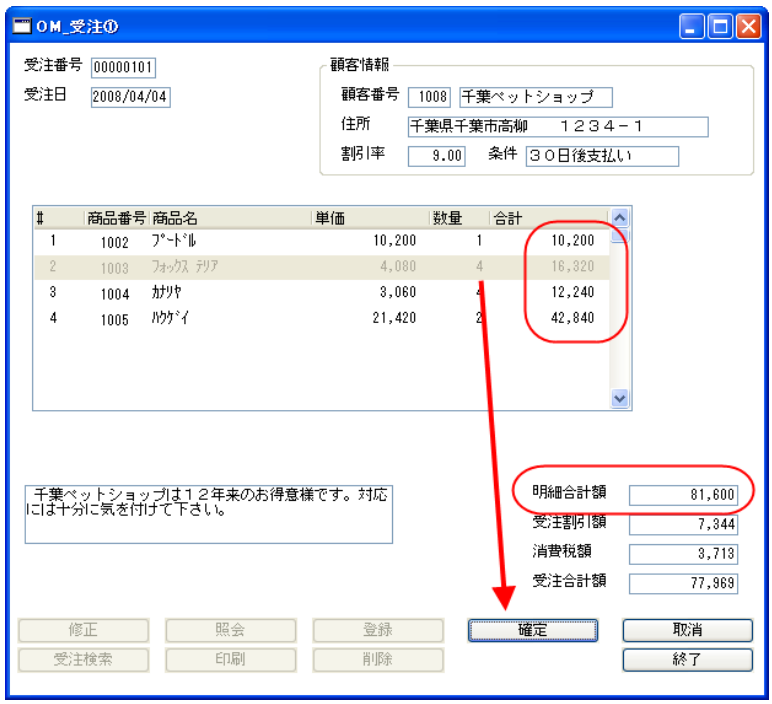
11.2.2　レコード後処理の実行タイミング

レコード後処理などのレベルの実行されるタイミングも異なります。

オンラインのプログラムと、リッチクライアントのタスクで、それぞれ、明細行で「数量」の値を変更してみてください。数量の値が変わると、明細行の「合計」値が変わりますが、それに伴い、ヘッダタスクの「明細合計額」も変わります。ヘッダタスクの「明細合計額」は、明細タスクのレコード後処理で「加算」モードの「項目更新」コマンドにより更新されますから、ヘッダタスクの「明細合計額」が変わるタイミングを見れば、明細タスクのレコード後処理が実行されるタイミングを判別することができます。

オンラインの場合

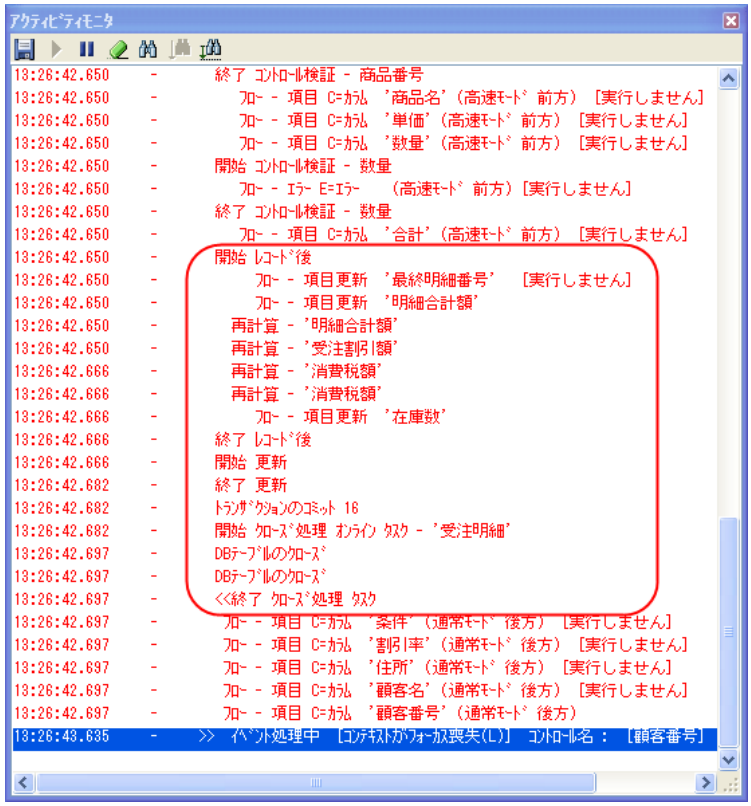
オンラインの場合には、明細行でESCキーを押して、カーソルがヘッダタスクに移動したタイミング（下図）で、明細行の「合計」値が更新されると共に、「明細合計額」が変わります。



実際には、このタイミングで、エンジン内部では

明細タスクのレコード後処理
⇒ 明細タスクのレコード書込み
⇒ 明細タスクのタスク後処理
⇒ 明細タスク終了

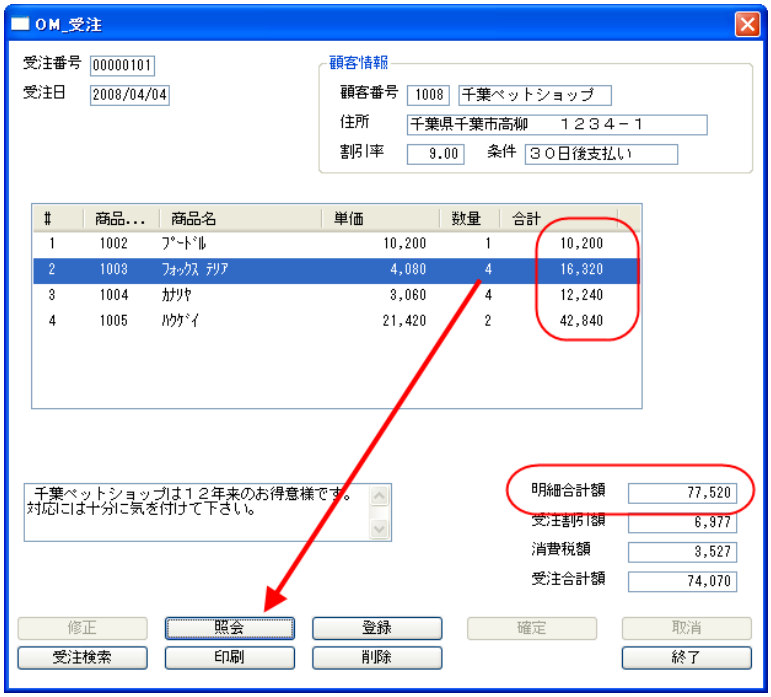
という動作が起こっています。このことは、アクティビティモニタを見てみると確かにそのような流れになっていることを確認できます。



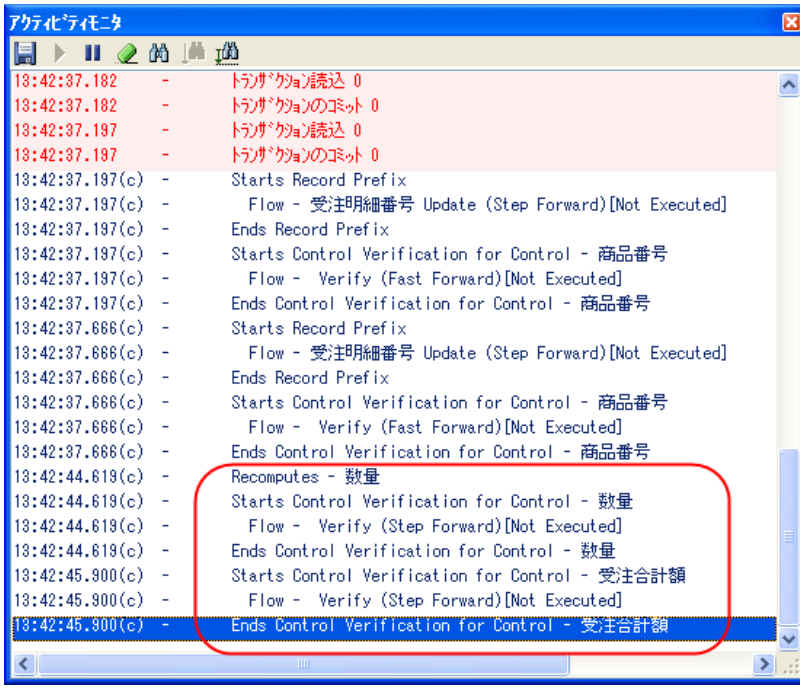
リッチクライアントの場合

一方、リッチクライアントの場合には、明細行の「数量」を変更したあと、Tab キーを押すと、カーソルがヘッダタスクのボタンに進みますが、このタイミングでは、明細行の「合計」欄は、再計算が行われて、値が変わるものの、ヘッダタスクの「明細合計額」は変わりません。

これは、このタイミングでは明細タスクのレコード後処理が行われていないことを意味します。



アクティビティモニタを見てみても、このタイミングでは、「数量」の変化に伴う再計算 (Recomputes と記録されている) はあるものの、レコード後処理やタスク後処理などは実行されていないことが確認できます。



ところで、この状態では、上の画面を見てもわかるように、明細行の「合計」の合計値は 81600 なので、「明細合計額」もこの値になっていなければならないのですが、実際には以前の値のままになっています。このため、表示上は不整合なデータ値が表示されていることになります。

最終的には、「確定」ボタンを押したタイミングで明細タスクのレコード後処理も行われて、DBMS に書き込まれる値は正しい値になるのですが、途中の表示上このような状態になることがあることに注意が必要です。

サブフォームの違いについてより詳しい情報は、リファレンスヘルプの次の項目を参照してください。

- （オンラインの場合）
  表示フォーム ⇒ GUI 表示フォーム ⇒ GUI コントロール
  ⇒ GUI 表示コントロール特性 ⇒ サブフォームコントロール
- （リッチクライアントの場合）
  表示フォーム ⇒ リッチクライアントフォーム ⇒ リッチクライアントコントロール
  ⇒ サブフォームコントロール特性

11.3 RC/バッチ更新

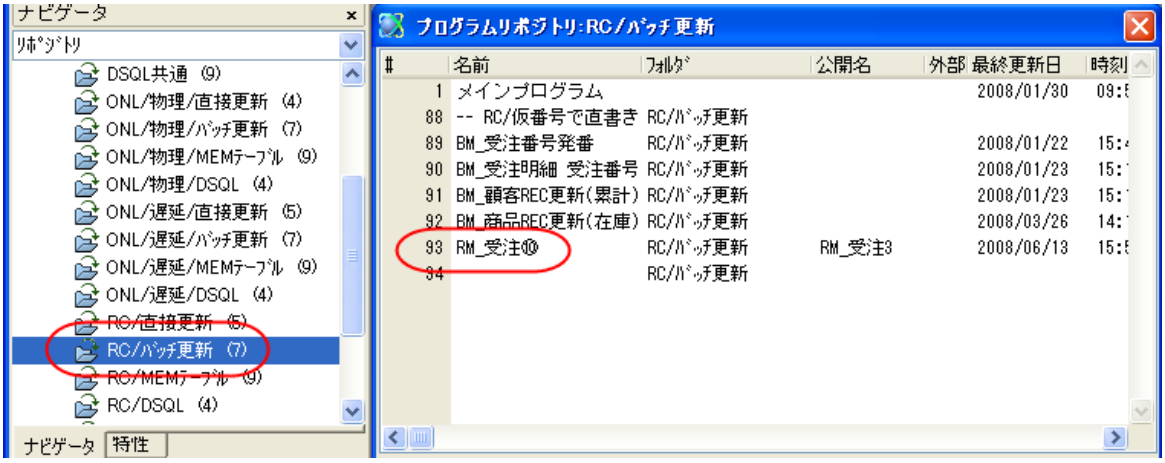
これは、オンラインの「ONL/遅延/バッチ更新」(10.2 節参照)をもとにして、リッチクライアントに単純移行したものです。

バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

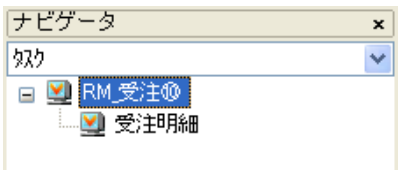
11.3.1 プログラムの構成

このパターンは、プログラムリポジトリの「RC/バッチ更新」というフォルダに格納されています。



11.3.2 プログラム構造

プログラムの構造は、もととなるオンラインの「ONL/遅延/バッチ更新」と同じく、二階層の親子タスクからなっています。
親タスクがヘッダタスク、サブタスクが明細タスクです。



移行の方法はきわめて単純で、以下のような修正を行うだけです。

- ヘッダ、明細タスク共に、タスクタイプを、オンラインからリッチクライアントにする。
- 「選択プログラム」特性の代わりに、ズームハンドラを作成する。

- 「フロー方向」が削除されてしまうので、検証ハンドラで、Flow 関数で条件づけする。

2 番目と 3 番目の修正については、リッチクライアントの基本形「RC/直接更新」（11.1 節参照）と全く同じなので、詳しくはそちらを参照してください。

サブフォームの動作については、前節（11.2）「リッチクライアントのサブフォーム」で説明したのと全く同じです。

11.4 RC/MEM テーブル

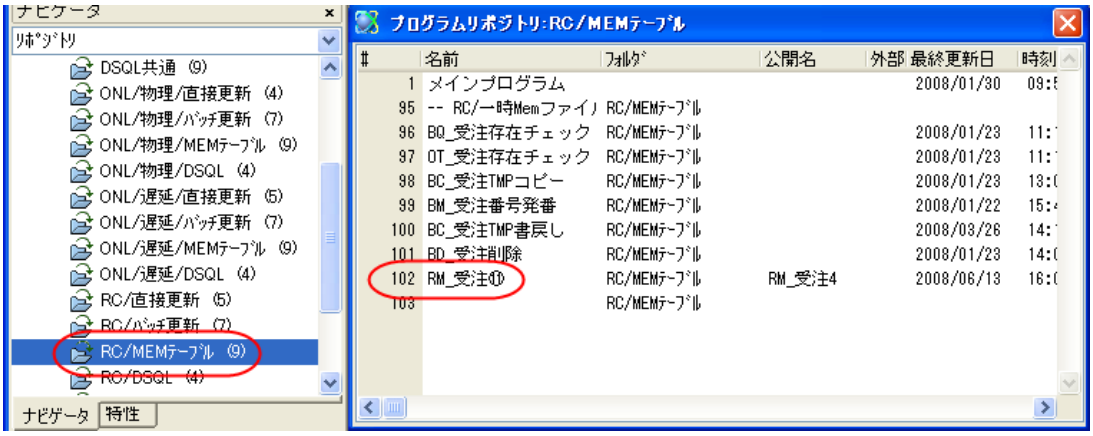
次には、一時テーブルを利用するパターンを説明します。これは、オンラインで一時テーブルと遅延トランザクションを使うパターン「ONL/遅延/MEM テーブル」(10.3 節)をもとにして、リッチクライアントへ移行しました。

バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

11.4.1 プログラムの構成

このパターンは、プログラムリポジトリの「RC/MEM テーブル」フォルダに格納されています。



11.4.2 プログラム構造

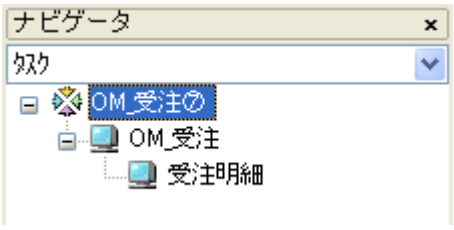
このパターンでは、プログラム構造に工夫が必要になります。

オンラインプログラムの場合には、ルートにバッチタスクがあり、子と孫タスクがそれぞれヘッダ、明細テーブルを扱っていました。すなわち、バッチ ⇒ オンライン ⇒ オンラインという三階層の構造になっていました。

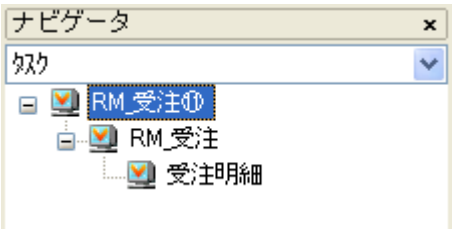
これをリッチクライアントにそのまま移行すると、バッチタスク ⇒ リッチクライアント ⇒ リッチクライアント という構造になりますが、この形ではエラーになってしまいます。リッチクライアントには、「リッチクライアントタスクは、リッチクライアントタスクからしかコールできない」という制限があるからで、ルートタスク(バッチタスク) ⇒ サブタスク (リッチクライアント) という呼び出しがこの制限に引っかかるからです。

この制限を避けるため、ルートバッチタスクも、リッチクライアントタスクに変更します。すなわち、リッチクライアントばかりの、三階層のタスク構造になります。

オンラインの場合



リッチクライアントの場合

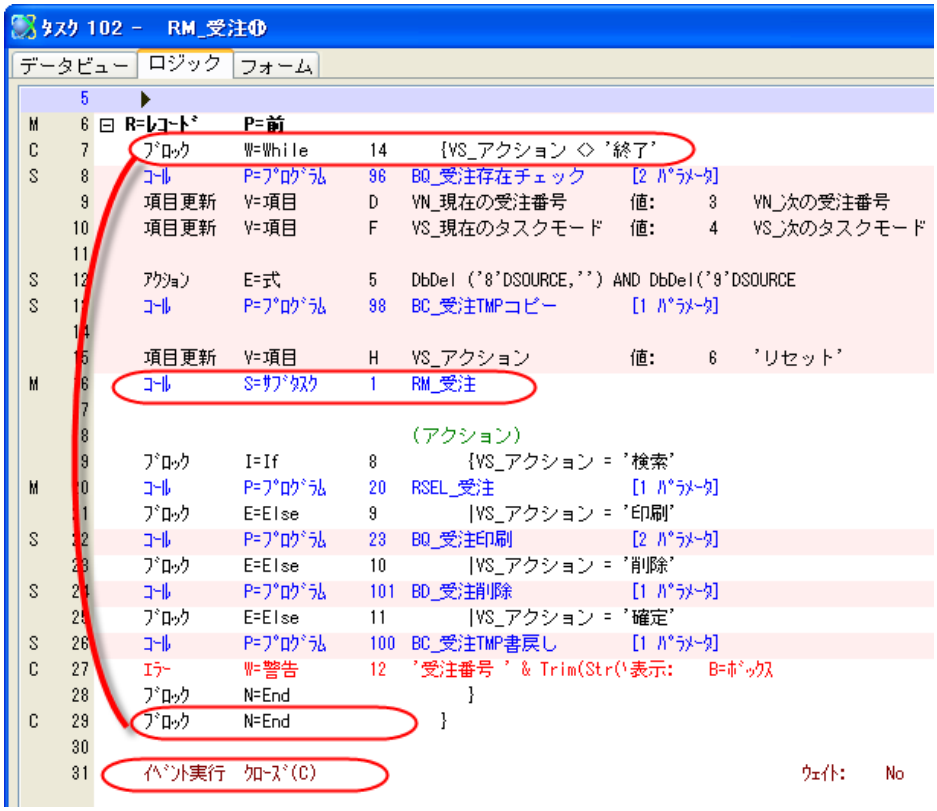


このとき、ルートのリッチクライアントタスクは、実行時にはあたかもバッチタスクのように動作させる必要があります。これは、次のようなトリックを使うことにより、実現することができます。

11.4.3 ロジック

オンラインの場合には、ルートバッチタスクは、レコードサイクルを使って、ループを作っていました。

リッチクライアントの場合には、ルートタスクの「ロジック」で、レコード前処理の中で、ブロック While コマンドを使って、ループを作ります。



そして、このループの中で、次のことを行います。

- 一時ファイルのクリアと読み込み、
- サブタスクの呼び出し（ユーザが受注データの入力を行う）
- 呼び出し後の後処理

このループは、「VS_アクション」変数が「終了」になったら終了します。

オンラインタスクの場合には、タスクの「終了」条件にこれを設定していました。

リッチクライアントの場合には、「ブロック While」のループの条件として設定してあります。

そして、「ブロック While」ループを抜けただけでは、タスクが終了しないので、「クローズ(C)」イベントを発行して、タスクを終了させるようにします。

11.4.4 フォーム

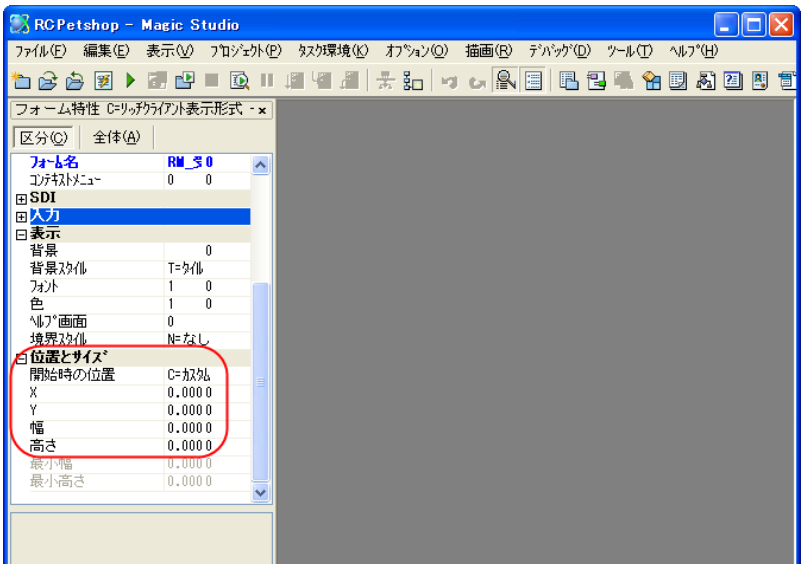
ルートタスクのフォームは表示しません。

オンラインの場合には、ルートバッチタスクのタスク特性で「ウィンドウを表示」パラメータを No にしておくことで、フォームを表示しないようにすることができました。

リッチクライアントタスクの場合には、「ウィンドウ表示」パラメータは無効化されており、No を設定することができません。すなわち、フォームを表示しない、ということはできず、必ず何らかのフォームを表示しなければなりません。

しかし、このフォームは、実際にはユーザの目に見えるようにしたくありません。このため、次のような特殊な設定をしたフォームとします。

- 内容は空（コントロールをひとつも配置しない）
- 境界スタイルは「N=なし」
- サイズはすべてゼロ



このようなフォームにすることにより、表示はされているものの、実質上ユーザの目からは見えないフォームとすることができます。

オンラインタスクの場合には、フォームにパーク可能なコントロールがひとつもない場合、実行時にエラーとなってしまいますが、リッチクライアントの場合には、空のフォームでもエラーなしに実行可能です。

11.4.5 その他の修正事項

その他には、特に大きく変更すべきことはありません。リッチクライアントの基本形「RC/直接更新」(11.1 節)でやったのと同様に、次の修正を行えば、移行完了です。

- 「選択プログラム」の代わりに、ズームハンドラを作成する。
- 「フロー方向」の代わりに、Flow 関数で条件づけする。

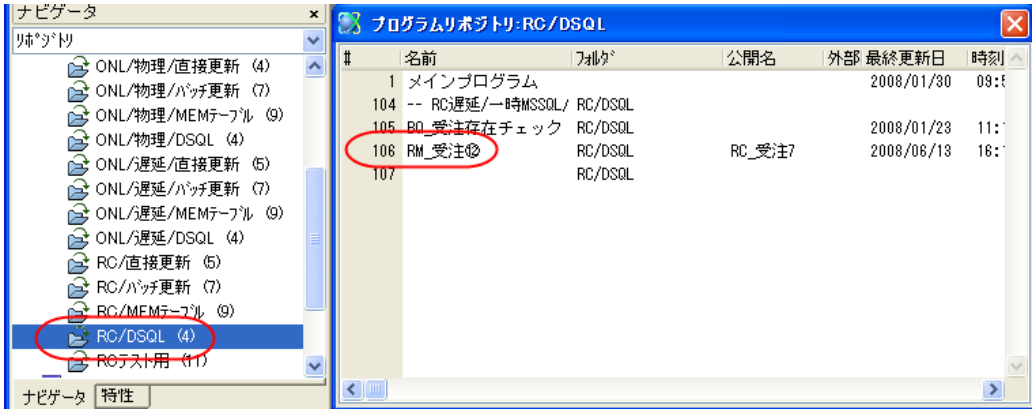
11.5 RC/DSQL

ここでは、一時テーブルをストアドプロシージャで操作する方式で、リッチクライアントプログラムを作成します。
このプログラムは、オンラインの「ONL/遅延/MEM テーブル」(10.3 節)をもとにして、「RC/直接更新」(11.1 節)や、「RC/MEM テーブル」(11.4 節)で説明した修正内容を、同様に適用して移植します。
バリエーションの分類で言うと、下図の赤枠で囲まれた部分になります。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

11.5.1 プログラム

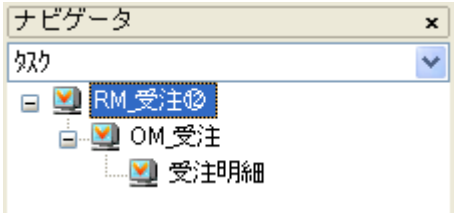
このプログラムは、プログラムリポジトリの「RC/DSQL」フォルダに格納されています。



11.5.2 プログラム構造

前節の「RC/MEM テーブル」と同様、リッチクライアントばかりの三階層のプログラム構造になります。

ルートのリッチクライアントタスクについても、「RC/MEM テーブル」と同様、レコード前処理で「ブロック While」を使って、ループを作り、バッチタスクのような動きを作っています。



そのほか、「選択プログラム」特性の代わりにズームハンドラを作ること、「フロー方向」パラメータの変わりに Flow 関数で条件づけすることなど、前節の「RC/MEM テーブル」と全く同じです。

12 実装方法の選択

以上、簡単な受注入力を行うプログラムを実装するにも、いろいろな方法があることがあり、それぞれに長所短所があることが理解していただけたことと思います。下表に、実装方法のマトリクスを再掲載します。

アルゴリズム	タスクタイプ/トランザクション設定		
	ONL/物理	ONL/遅延	RC (/遅延)
直接更新	基本形 (6)	ONL/遅延/直接更新 (10.1)	RC/直接更新 (11.1)
バッチ更新	ONL/物理/バッチ更新 (7)	ONL/遅延/バッチ更新 (10.2)	RC/バッチ更新 (11.3)
MEM テーブル	ONL/物理/MEM テーブル (8)	ONL/遅延/MEM テーブル (10.3)	RC/MEM テーブル (11.4)
DSQL	ONL/物理/DSQL (9)	ONL/遅延/DSQL (10.4)	RC/DSQL (11.5)

ここではまとめを兼ねて、どのような要件の場合にどの方式を採用すればよいかについて、簡単に指針を挙げたいと思います。

クライアントサーバかリッチクライアントか？

システム開発の大前提として、システムをクライアントサーバとして作成するか、リッチクライアントとして作成するかの決定があると思います。これは、開発・保守の工数や、システムリソース、要求仕様などを総合的に勘案して決めます。

クライアントサーバとして開発されるシステムであれば、タスクタイプはオンライン(ONL)でなければなりません。一方、リッチクライアントシステムとして開発されるシステムであれば、タスクタイプはリッチクライアント(RC)となります。

新規システム開発か既存システムの移行か？

次に、新規システム開発か、既存システムの移行か？という選択肢があります。これも、システム開発時の大前提として決定されているものと思います。

既存システムの移行であれば、すでに動いているプログラムの構造やロジックをできるだけ修正しない形で移行するほうが、開発・テストの工数を削減するためにベストの選択となります。すなわち、上の表で言えば、すでにあるプログラムのアルゴリズム（直接更新、バッチ更新、MEM テーブル、DSQL）は変更せずに、タスクタイプおよびトランザクション設定だけを変更する、というのが一番簡単です。

例えば、既存のシステムで「ONL/物理/MEM テーブル」のアルゴリズムでプログラムが作成されていて、これをリッチクライアントに移行しようとするならば、アルゴリズムはそのままに、トランザクションタイプを遅延とし、タスクタイプをリッチにして、「RC/MEM テーブル」の形にします。この形は「必要最低限」という意味では最適ではないかもしれませんが、工数最小化という意味ではベストの選択になります。

一方、新規システム開発であれば、要求仕様を満たす限り、できるだけ簡単な形で実装することが、開発においても、以後の保守においても、工数削減のためにベストの選択となります。これについては、次に説明します。

ユーザ入力があるか、表示専用か？

新規にプログラムを作成する場合、どれがベストな方法かの選択が必要になってきますが、この選択にあたっては、ユーザ入力があるか、表示専用かが、大きな分かれ目になります。

というのは、さまざまなアルゴリズムのバリエーションが必要になってくるのは、プログラムでユーザの入力（登

録、修正、削除）があるときには、DBMS における不正更新を防止するために、並列制御を行わなければならないからです。

もし、データの表示しか行わないプログラムであるならば、不正更新について考慮する必要がありませんので、もっとも簡単なアルゴリズムを選択すれば十分です。すなわち、「直接更新」のアルゴリズムを使った方式となります。

しかも、更新は行わないのですから、第 6 章「ヘッダ・明細型プログラムの基本形」で説明した項目の中で、データ更新について考慮した多くの点も実装する必要がなく、非常に簡単なプログラムになります。

すなわち、オンラインの場合には「基本形」あるいは「ONL/遅延/直接更新」、リッチクライアントの場合には「RC/直接更新」の方式となります。オンラインの場合、トランザクションは物理でも遅延でも違いはありませんので、いずれを選んでも OK です。

なお、検索系のプログラムでは、検索パラメータをユーザが入力しますが、この入力データは DBMS のデータを更新するためには使われず、純粋に範囲付けの条件としてだけ使われるので、やはり「表示専用」と同じことになります。

一方、ユーザの入力（DBMS への更新）がある場合には、各方式の中から、最適なものを選択する必要があります。次にそれについて説明します。

一時テーブルを利用するかしないか？

まず、「基本形」はマルチユーザに対応していないので、複数ユーザが同時利用して DBMS への更新がある場合には、選択の対象からはずれます。

次に、一時テーブルを使うか否か？が分かれ目になります。

- 一時テーブルを使わないアルゴリズム（「直接更新」および「バッチ更新」）では、プログラムはシンプルになりますが、トランザクションの範囲についてよく考慮して設計しなければならない場面が出てきます。親タスクでトランザクションが始まるので、ほとんどの処理を同一トランザクション内で処理しなければならなくなるからです。
- 一時テーブルを使うアルゴリズム（「MEM テーブル」および「DSQL」）では、データリポジトリでの一時テーブルの定義と、コピーおよび書き戻しのためのバッチプログラムの作成を行わなければならないため、プログラムが多くなります。その分工数が増えます。

  しかし、ルートバッチタスクはトランザクションの外になっているので、ルートバッチタスクから呼び出すタスクは、ヘッダ・明細入力時に必要となるトランザクションとは切り離されることになり、トランザクションの設定の自由度が上がります。これにより、複雑になりがちなレコードロックの干渉について、設計上の考慮事項が簡単化されます。

  また、一時テーブルにユーザが入力したデータが記憶されているので、まだ DBMS に反映されていない一時データに対する細かな制御も可能になります。例えば、入力途上の一時データを BLOB の形にしてハードディスク上のファイルに保存しておいて、後日、ファイルから復元して入力の続きを行い、最後に DBMS に反映させる、というような使い方も可能になります。（ただし、このようなことをする場合には、ファイルに保存してある間に、他のユーザが DBMS の内容を変更してしまわないように、設計・運用上の考慮が必要になります）。

選択の指針

いずれを採用するかは、開発者のスキルや慣れ、プログラムに対する要求仕様により決められることですが、単純化して言えば、次のようなことが言えると思います。ここではオンラインについてのみ言及していますが、リッチクライアントでも考え方は同じです。

- 細かな制御が必要でない場合には、「オンライン/遅延/直接更新」の形が一番簡単です。ただし、遅延トランザクションについて、ある程度の理解が必要です。
- 遅延トランザクションの利用にまだ慣れていない場合には、「オンライン/物理/バッチ更新」の形がその次に簡単です。
- 細かな制御が必要になる場合には、一時テーブルを使う「オンライン/物理/MEM テーブル」がオーソドックスな形です。この方法は、従来の Magic システムでも多用されてきた方法であり、Magic 開発者にとっても一番なじみのある方法と思われます。
- ストアドプロシージャを多用することが前提であるプロジェクトの場合、例えば、Magic 以外の言語系ツールで開発されるシステムと DBMS を共用しているとか、あるいは SQL DBMS に経験豊富な技術者が多数いて、ストアドプロシージャを多用することを好まれるような場合には、「オンライン/物理/DSQL」の方式を採用することもよいでしょう。ただし、この場合には、DBMS の移植性がなくなり、また、テーブル定義の変更時の保守性が低下することが欠点となります。

以上のような点を考慮して、開発するシステムに最適な方式を採用してください。

13 リッチクライアントとオンラインとの違い

本章では、移植のための参考情報として、リッチクライアントとオンラインとの違いについて説明します。紙面の関係上、個々の項目について詳しく説明することはできませんが、リファレンスヘルプなどで関連項目について参照してください。

本章の内容は、V10.1SP4b の README.CHM の「参考技術情報 ⇒ リッチクライアントのオンラインとの違い」を転載したものです。

13.1 動作環境

- Magic RichClient Server のライセンス(MGRCX1)では、では、JNLP ファイルから起動されたリッチクライアントタスクのみを呼び出すことができます。Web マージ、リモートコール、ブラウザクライアントなどのリクエストを処理することはできません。
- ライセンスのスレッド数は、[動作環境]の[最大並行ユーザ数]に定義された値が Magic エンジンの起動時に消費されます。[最大並行ユーザ数]が「0」かライセンス上のユーザ数を越えている場合は、ライセンスのユーザ数分が消費されます。
- ActiveDirectory 認証は利用できません。LDAP 認証を使用して Active Directory サーバへの認証を行うようにしてください。

13.2 動作が異なる機能

以下の機能は、オンラインタスクと動作が異なります。またこれ以外でも、インターネット環境を利用して実行する特性上、動作が異なる場合があります。

#	内容	対応
1	数値項目の入力時の最初のキャレット位置が右側になります。(オンラインプログラムでは左側。)	
2	クローズイベント発生時、[クローズ]イベントのみ発行されて、タスクが終了します。(オンラインタスクでは、[クローズ]イベントの後に[終了]イベントが発生します。)	[クローズ]イベントを[終了]イベントに変更します。
3	画面左上隅の[X] ボタンを押したとき、[終了]イベントが発生します。(オンラインでは[クローズ]イベントが発生。)	[クローズ]イベントを[終了]イベントに変更します。
4	サブフォームタスクの[タスク前]／[タスク後]の実行フローが異なります。また最後の項目から次項目を行うと、親タスクの最初の項目に位置付きます。	
5	[トランザクションモード]特性のオプション	
6	コンボボックスの選択肢が多い場合、選択リストの開始位置がコンボボックスより上の位置に変更されます。	[コントロール特性／表示行数]に適切な行数を設定します。

13.3 サポートされない機能

以下の機能は、現在リッチクライアントアプリケーションではサポートされていません。

これ以外でも、リッチクライアントアプリケーションは、処理内容によってサーバ側またはクライアント側のどちらかでしか処理されない場合があるため、設定箇所によっては定義できない（関数などの）オブジェクトがあります。詳細は、『インタラクティブなリッチクライアントの開発と実行』を参照してください。

13.3.1 タスク/ロジック定義

#	内容	対応
1	[RM 互換]ロジックユニット	[イベント]ロジックユニットに移動します。
2	バッチタスクからの呼び出し	プログラム構造の変更が必要です。
3	照会モード位置付け	
4	実行時のオプションメニュー（範囲、位置付け、ソート、インデックス変更）	
5	数値項目の入力時の最初のキャレット位置が右側。（オンラインプログラムでは左側。）	
6	日本語のプロジェクト名	半角英数字のプロジェクト名を定義します。
7	OS コマンドによるデータファイル(*.txt, *.doc, *.xls など)、または OS の標準コマンドの指定。	C:¥WINDOWS¥system32¥cmd.exe /c (ファイル名) と指定します。
8	マウスのスクロール機能	
9	[入出力ファイル]テーブル	
10	[タスク特性]の以下のオプション • タスク常駐 • レコード削除 • ウィンドウ再表示 • ウィンドウ表示 • ウィンドウ消去 • キャッシュ範囲 • ロック方式 • 位置付 • 範囲 • インデックス変更 • ソート • 入出力ファイル • インデックス最適化 • 照会モード位置付 • データ出力	
11	以下のデータ型	

	• ActiveX 型 • OLE 型	
12	以下のコントロール • （テキスト／グループ以外の）スタティック • ライン • スライダ • OLE • リッチテキスト • リッチエディット	
13	以下の処理コマンド • コール COM • フォーム	
14	処理コマンドの以下の特性 • フローモード • フロー方向 [コール]処理コマンドの以下の特性 • フォーム • ロック • 同期 • コンテキスト ID [イベント実行]処理コマンドの以下の特性 • 出力先コンテキスト名 [コール OS コマンド]処理コマンドの以下の特性 • 表示 • ウェイト（[実行]特性が「クライアント」の場合）	

13.3.2 フォーム/コントロール

#	内容	対応
1	[テーブル]コントロールでのマルチマーキング	
2	内部形式/Windows 形式のヘルプ	URL によるヘルプ、またはツールチップ、自動ヘルプに変更します。
3	ドラッグ &ドロップ	
4	イメージコントロールの BLOB 型項目設定	ファイル名（URL 形式）を格納した文字型項目に変更します。
5	以下のフォーム特性 • ウィンドウリストに表示 • 寸法単位（「ダイアログ」固定） • フォーム状態 ID	

	• ドロップ許 • パレット最適化 • 分割 • 位置	
6	[エディット]コントロールの以下の特性 • ドラッグ許可 • ドロップ許可 • 選択プログラム • 起動モード • スクロールバーの表示 • CR 許可（複数行が有効な場合は、「Yes」固定） • スタイル • 境界スタイル • 垂直整列 • Tab 移動方向（「両方向」固定）	
7	[ラベル]コントロール（GUI 表示フォームの[テキスト]コントロール）の以下の特性 • RTF 許可 • スタティックタイプ • ドラッグ許可 • ドロップ許可 • スタイル • 境界スタイル • 垂直整列 • 線種 • 線幅	
8	[プッシュボタン]コントロールの以下の特性 • ボタンスタイル（「プッシュボタン」固定） • デフォルトイメージファイル • ドラッグ許可 • ドロップ許可 • 選択プログラム • 起動モード • Tab 移動方向（「両方向」固定）	
9	[コンボボックス]コントロールの以下の特性 • ドラッグ許可 • ドロップ許可 • 選択プログラム • 起動モード • スタイル	

	• 境界スタイル • 垂直整列 • Tab 移動方向（「両方向」固定）	
11	[リストボックス]コントロールの以下の特性 • ドラッグ許可 • ドロップ許可 • 選択モード（「シングル」固定） • 選択プログラム • 起動モード • スタイル • 境界スタイル • 水平整列 • Tab 移動方向（「両方向」固定）	
12	[ラジオボタン]コントロールの以下の特性 • ソーステーブル • 表示項目 • リンク項目 • インデックス • 範囲 • ドラッグ許可 • ドロップ許可 • 選択プログラム • 起動モード • 表示方法（「ラジオ」固定） • スタイル（「平面」と「凹立体」のみ） • 複数行 • 境界スタイル • 垂直整列 • 水平整列 • イメージファイル名 • Tab 移動方向（「両方向」固定）	
13	[イメージ]コントロールの以下の特性 • ドラッグ許可 • ドロップ許可 • ドラッグ許可 • ドロップ許可 • スタイル • 境界スタイル • イメージ効果 • Tab 移動方向（「両方向」固定）	

	サポートするイメージファイルは、BMP, GIF, ICO, JPEG, PNG, TIFF　です。	
14	[チェックボックス]コントロールの以下の特性 • ドラッグ許可 • ドロップ許可 • 選択プログラム • 起動モード • スタイル（「平面」と「凹立体」のみ） • 3 ステータス • 複数行 • 境界スタイル • 垂直整列 • 水平整列 • Tab 移動方向（「両方向」固定）	
15	[グループ]コントロールの以下の特性 • スタティックタイプ • ドラッグ許可 • ドロップ許可 • スタイル • 境界スタイル • 垂直整列 • 水平整列 • 線種 • 線幅	
16	[タブ]コントロールの以下の特性 • ソーステーブル • 表示項目 • リンク項目 • インデックス • 範囲 • ドラッグ許可 • ドロップ許可 • 選択プログラム • 起動モード • スタイル • 垂直整列 • タブラベル位置（「上」と「下」のみ） • ホットトラック • タブ幅 • 複数行	

	• Tab 移動方向(「両方向」固定)	
17	[テーブル]コントロールの以下の特性 • ドラッグ許可 • ドロップ許可 • スタイル • 境界スタイル • スクロールバー • 区切り • ハイライト行のスタイル • タイトル高さ • 下辺位置 • カラムの区切り線 • 最終区切り線 • ウィンドウ内テーブル 以下のコントロールは、[テーブル]コントロールに配置できません。 • ラジオボタン • グループ • タブ • リストボックス • サブフォーム	
18	[カラム]コントロールの以下の特性 • カラムタイトルの複数行入力 • 上境界線 • 右境界線 • 垂直整列 • フォント • ソート可 • カラムのマーキング	
19	[サブフォーム]コントロールの以下の特性 • 自動再表示(「Yes」固定) • 境界スタイル • Tab 移動方向(「両方向」固定)	

13.3.3 関数

以下の関数は利用できません。また、リッチクライアントでのみ有効な関数もあります。詳細は、『インタラクティブなリッチクライアントの開発と実行』を参照してください。

Blob2Req	ClientCertificateAdd	COM 関数
CleftMDI	ClientCertificateDiscard	Counter

CTop/CTopMDI
CtrlHWND
CurrPosition
DateFormat
Drag&Drop 関数
EOF/EOP
ExpCalc
File2Req
GetNextRecNum
HitZOrder
IOCurr
KbGet/KbPut
Line
MarkedTextSet
MDate
マルチマーク関数
MnuAdd/MnuRemove/MnuReset
Page
RqHTTPHeader
SetContextFocus
SNMPNotify
SplitterOffset
SubformExecMode
Text
TransMode
UDF 関数
Variant 関数
WinHelp/WinHWND
WsProviderAttachmentAdd
WsProviderAttachmentGet
XML 関数

13.3.4 内部イベント

以下の内部イベントは利用できません。「ユーザアクション」は、ユーザイベントに変更する必要があります。詳細は、『インタラクティブなリッチクライアントの開発と実行』を参照してください。

OS ｺﾏﾝﾄﾞ
MAGIC 情報
ｱﾌﾟﾘｹｰｼｮﾝを開く
ｲﾝﾃﾞｯｸｽ
ｵﾌﾞｼﾞｪｸﾄの挿入
ｿｰﾄ
ﾃﾞｰﾀ出力
ﾄﾞﾗｯｸﾞ開始
ﾄﾞﾛｯﾌﾟ
ﾉｰﾄﾞ縮小
ﾉｰﾄﾞ名を編集
ﾉｰﾄﾞ展開
ﾏｰｸを反転
ﾏｰｸ/ﾏｰｸ解除
ﾏｰｸを全て解除
ﾏﾙﾁﾏｰｷﾝｸﾞで次ﾍﾟｰｼﾞ
ﾏﾙﾁﾏｰｷﾝｸﾞで次行
ﾏﾙﾁﾏｰｷﾝｸﾞで前ﾍﾟｰｼﾞ
ﾏﾙﾁﾏｰｷﾝｸﾞで前行
ﾕｰｻﾞｱｸｼｮﾝ
子ﾉｰﾄﾞ作成
全て削除
再表示
開始値
終了値
位置付／次候補
位置付
全てﾏｰｸ
次ﾍﾟｰｼﾞにﾏｰｸ
前ﾍﾟｰｼﾞにﾏｰｸ
項目先頭までﾏｰｸ
最後までﾏｰｸ
先頭までﾏｰｸ
次行でﾏｰｸ
前行でﾏｰｸ
親のﾉｰﾄﾞに移動
出力ﾀﾞｲｱﾛｸﾞ 次へ
出力ﾀﾞｲｱﾛｸﾞ 戻る
照会
範囲
入出力ﾌｧｲﾙ
子のﾉｰﾄﾞに移動
次のﾉｰﾄﾞに移動
前のﾉｰﾄﾞに移動

Magic eDeveloper V10

Magic eDeveloper V10
タスク基本構造（ヘッダ明細入力）

第 1 版　2008 年 7 月 7 日

発行　〒151-0053 東京都渋谷区代々木三丁目二十五番地三号
あいおい損保新宿ビル 14 階
マジック ソフトウェア・ジャパン（株）
http://www.magicsoftware.co.jp/