Magic eBusiness Platform V9Plus
チュートリアル

SQL 編

Magic eDeveloper™ Plus V9

初版　　２００５年１２月３１日

マジックソフトウェア・ジャパン株式会社

本書についてお気づきの点、ご意見ご希望等ございましたら、メールで japan_support@magicsoftware.com までお送りください。個々のメールにご返事することはできませんが、今後の改訂の際に参考にさせていただきます。

目次

1. はじめに

本書では、リレーショナルデータベース管理システム(以下 RDBMS、あるいは SQL データベースと略)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。サンプルとしては、チュートリアルで作成したペットショップデモを利用します。ペットショップデモは非常に単純なサンプルですが、Magic におけるデータのハンドリングに必要な基本テクニックがカバーされるので、Magic の基本を理解するには最適です。実際のアプリケーションが複雑だったとしても、基本的な部分はペットショップでのテクニックの応用で実現できます。

リレーショナルデータベースとひとくちに言っても、製品ごとに細かな仕様が異なります。本書では MS-SQL Server (MSDE)を使って SQL 化を行っていきます。他の RDBMS を使う場合には、命名規則やロック・トランザクションの動作などに微妙な違いがありますが、プログラムの作り方についての基本的な考え方は同じです。

参考：
チュートリアル入門編では、データベースとして、Magic にバンドルされている Pervasive.SQL を使ってきました。Pervasive.SQL データベースは、その名の通り、SQL 文をインターフェースとする「リレーショナル・アクセス」といわれる方式と、昔からの Btrieve と互換性を持った、ISAM タイプのインターフェースを持つ「トランザクショナル・アクセス」と呼ばれる方式とがあります。
Magic の Pervasive ゲートウェイは、このうちトランザクショナル・アクセスを利用して Pervasive.SQL のデータをアクセスしていますので、Pervasive.SQL をデータベースエンジンとして使っていても、実際には ISAM のインターフェースでアクセスしていることになります。
このため、Magic から Pervasive ゲートウェイを介して Pervasive.SQL にアクセスする場合には、トランザクションの利用が必ずしも必要ではなく、実際に Magic のアプリケーションではトランザクションを使わずに組んでいる場合が多いようです。
この場合には、「第 9 章 トランザクション」(62 ページ)で説明するような、トランザクションを使うことによる制約事項について考慮する必要がないので、アプリケーションの作成が簡単に済みます。一方、トランザクションのデータ整合性保障のメリットが得られないことになります。

1.1. 目標

本書の目標は、次の通りです。

- MS-SQL を例にとり、SQL データベースの設定方法を理解する。
- Pervasive.SQL で作ったペットショップデモを MS-SQL Server を使ったものに移植する。
- 移植後のアプリケーション（特に、受注入力プログラム）が、マルチユーザ環境でも正しく動作するようにする。

Magic の SQL データベースサポートに関しては、非常にきめ細かで幅広い内容がありますが、本書ではペットショップデモを MS-SQL Server できちんと動かすことに限定して説明を進めていきます。

1.2. 前提条件

本書では、読者と動作環境について、次のことを前提としています。

1.2.1. Magic について

- 読者は Magic の概念と操作についての基礎についての知識を持っていることを仮定しています。具体的には、Magic のチュートリアルを勉強し、Pervasive でペットショップデモを作成できる程度の知識があることが前提です。
- お使いになる PC には、Magic をインストールしておいてください。製品版 の Magic eDeveloper をお持ちでない場合には、体験版を弊社ホームページなどからダウンロードしてインストールしてください。
- また、PC には Pervasive.SQL (2000i 以降)をインストールしておいてください。Magic eDeveoloper の製品 CD にはバンドルされていますが、もしお持ちでなければ、弊社ホームページなどから体験版と一緒に Pervasive.SQL 体験版(V8SP2。90 日期限付き)をダウンロードし、インストールしてください。
- 本書では、SQL データベースの例として、MS-SQL Server 2000 (SP3 以降) を使います。もしご利用になる PC に MS-SQL Server を利用する環境がなければ、「第 2 章 SQL 利用準備」(12 ページ)を参考にして、MS-SQL Server (評価版、あるいは Desktop Engine)を入手し、インストールしてください。

もし、MS-SQL Server ではなく Oracle や DB2/UDB を利用したい場合には、本書の内容はほぼそのまま応用できますが、データベースの設定でデータベースごとの細かな違いが出てきます。詳しくはリファレンスマニュアル第 25 章「SQL に関する考慮事項」、開発者ガイド第 1 章「Magic の環境設定 → SQL データベースの定義」あるいは、Readme.chm の「データベース固有の追加情報」などを参照してください。

1.2.2. SQL について

読者は、SQL データベースについて基本的なことを知っていることを仮定しています。具体的には、次のような簡単な SQL 文です。

- DDL 文
  - CREATE/DROP TABLE
  - CREATE/DROP INDEX

- DML 文
  - SELECT ・・・ FROM ・・・ WHERE ・・・ ORDER BY ・・・
  - UPDATE
  - INSERT
  - DELETE

- ロック (SELECT 文へのヒント)

- UPDLOCK　NOWAIT

- トランザクション制御
  - BEGIN TRANSACTION
  - COMMIT TRANSACTION
  - ROLLBACK TRANSACTION

これらの DDL/DML 文は SQL データベースのごく基本ですが、もし知らなければ数多くの書籍がありますのでそちらでまず勉強してください。

なお、本書では MS－SQL Server を使いますので、MS SQL Server の知識と経験があればなおわかりやすくなります。

1.3. 高度な話題

本書で説明する内容に関連して、より高度な説明がリファレンスマニュアルにありますので、以下の章も参考にしてください。

機能	リファレンスマニュアルの箇所
トランザクション一般に関する話題	第 11 章 データ管理
SQL に関する考慮事項	第 25 章 SQL に関する考慮事項
マルチユーザ環境での考慮事項	第 23 章 マルチユーザ環境

また、Magic の SQL データベースサポート機能の中で、本書では説明しない主なものは、以下のようなものがあります。詳しくは、セミナーコースに参加されるか、あるいは以下のリファレンスマニュアルを参照してください。

機能	リファレンスマニュアルの箇所
テーブル定義の詳細	第 4 章 テーブル
埋め込み SQL	第 6 章 プログラム → 埋め込み SQL
SQL WHERE 句	第 6 章 プログラム → 範囲と位置付ダイアログ → [SQL Where]タブ
遅延トランザクション	第 11 章 データ管理 → 遅延トランザクション

2. SQL 利用準備

本書では、SQL DBMS として MS-SQL Server を利用します。MS-SQLServer が今使っている環境で利用できるようになっていなければ、ここでインストールしておいてください。
MS-SQL Server をお持ちでない場合には、以下のいずれかの方法で、MS-SQL Server Desktop Engine (MSDE)、あるいは期間限定の評価版を入手することができます。

- Magic 製品 CD から MSDE をインストール
- Microsoft サイトからダウンロード

それぞれについて、以下に説明します。

2.1. Magic 製品 CD から MSDE をインストール

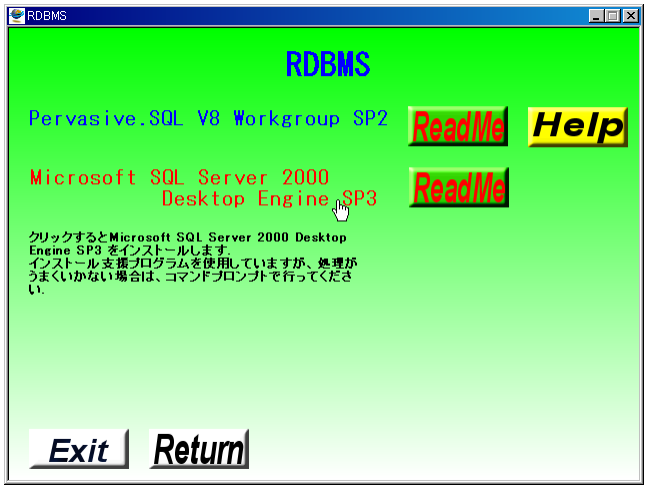
Magic の製品 CD をお持ちであれば、MSDE2000 SP3 がバンドルされているので、CD からインストールすることができます。
インストール時には以下の情報が必要となるので、予め決定しておいてください。

設定値	デフォルト値
管理者 sa のパスワード	password
セキュリティモード	Windows 認証

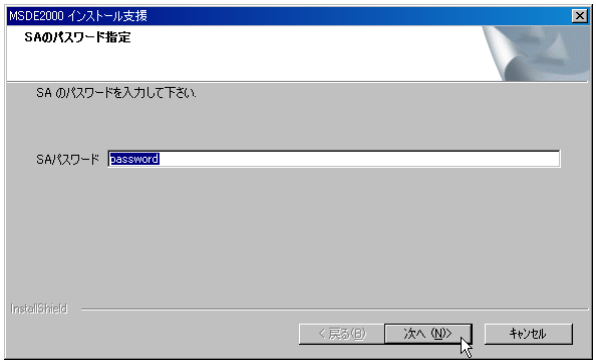
1. Magic 製品の CD-ROM を CD-ROM ドライブにセットします。自動的にインストーラが開始されます。

2. RDBMS をクリックしてください。RDBMS のインストール画面が表示されます。

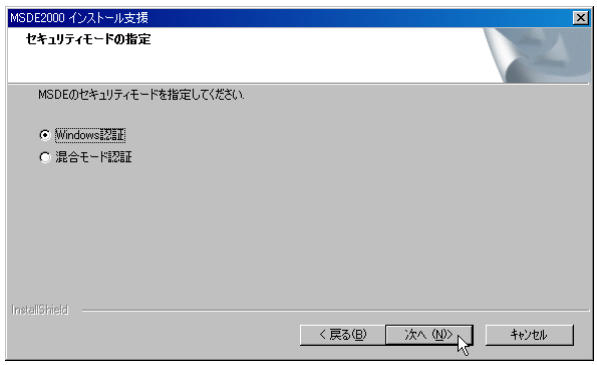
3. 「Microsoft SQL Server 2000 Desktop Engine SP3」をクリックしてください。インストーラが開始します。



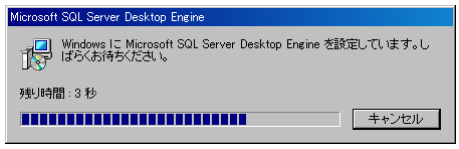
4. 管理者 sa のパスワードを聞いてきます。適当なパスワードを設定してください。



5. セキュリティモードを聞いてきます。Windows 認証か、混合モード認証を選択してください。
   セキュリティモードについては、MS-SQL Server の技術資料を参照してください。簡単なテストの場合には、デフォルトの Windows 認証で良いでしょう。



6. インストールが実行されます。



7. Magic という名前のデータベースを作成します。確認ダイアログが出ますので、「はい」で答えてください。

以上で MSDE のインストールが完了です。

2.2. Microsoft サイトからダウンロード

Magic の製品 CD-ROM をお持ちでない場合には、MS-SQL Server の期限限定の評価版を Microsoft 社のサイトからダウンロードすることができます。また、Microsoft SQL Server 2000 Desktop Engine (MSDE 2000)は、簡単な登録を行うことにより通常版をダウンロードすることができます。

これらの製品に関する情報およびダウンロードは、Microsoft 社の Microsoft SQL Server のページ http://www.microsoft.com/japan/sql/default.mspx を参照してください。
また、インストールおよび利用・再配布の条件に関しては、それぞれの製品に関するドキュメントを参照してください。

2.3. データベースの作成

注意： Magic の製品 CD-ROM より MSDE をインストールした場合には、インストーラにより MAGIC という名前のデータベースがすでに作成されているので、ここでの設定は不要です。「2.4 簡単な SQL 操作」(17 ページ)に進んでください。

インストールした直後では、システムが使うデータベースしか登録されていませんので、テスト用に別途データベースを作成します。ここでは以下のようにして MAGIC という名前のデータベースを作成します。

1. コマンドプロンプトを開きます。
2. 以下のコマンドを入力してください。(データベースのサイズは、ここでは 10MB としていますが、データ量に応じて適当に増減してください)

```
C:¥>osql -E
1> create database MAGIC
2> go
CREATE DATABASE: ディスク 'MAGIC' に 0.75 MB 割り当てています。
CREATE DATABASE: ディスク 'MAGIC_log' に 0.49 MB 割り当てています。
1> alter database MAGIC modify file (name='MAGIC', size=10MB)
2> alter database MAGIC modify file (name='MAGIC_log', size=10MB)
3> go
1> quit
```

参考： 上のコマンドのうち、alter database ... は、データ領域を予め増やしておくために実行します。本書で出てくる小さなテーブルのみを使う場合には、実行しなくとも構いません。

以上でデータベース MAGIC が MS SQL Server 上に作成されました。

2.4. 簡単な SQL 操作

インストールが済んだら、動作確認を兼ねて、OSQL を使って、簡単な DDL/DML 文を実行させてみましょう。すでに動作を確認してある場合にはスキップしてかまいません。

1. コマンドラインから OSQL コマンドを起動します。

```
C:¥> osql -E
```

2. データベース MAGIC を選択します。

```
1> use magic
2> go
```

3. テーブル mytbl を作成します。

```
1> create table mytbl (a char(10))
2> go
```

4. レコードを一件登録します。

```
1> insert into mytbl values ('いろは')
2> go
(1 件処理されました)
```

5. データを見てみます。

```
1> select * from mytbl
2> go
 a
 ----------
 いろは
(1 件処理されました)
```

6. データを更新します。

```
1> update mytbl set a = 'あいう' where a = 'いろは'
2> go
(1 件処理されました)
```

7. 再度データを見てみます。

```
1> select * from mytbl
2> go
 a
 ----------
 あいう
(1 件処理されました)
```

8. レコードを削除します。

```
1> delete mytbl where a = 'あいう'
2> go
(1 件処理されました)
```

9. 再度データを見てみます。

```
1> select * from mytbl
2> go
 a
 ----------

(0 件処理されました)
```

10. テーブルを削除します。

```
1> drop table mytbl
2> go
```

3. Magic での MS-SQL 設定。

MS-SQL Server のインストールが済んだので、次には Magic での MS-SQL に関する、次のような設定を行います。

- ゲートウェイのインストール
- DBMS テーブル
- データベーステーブル

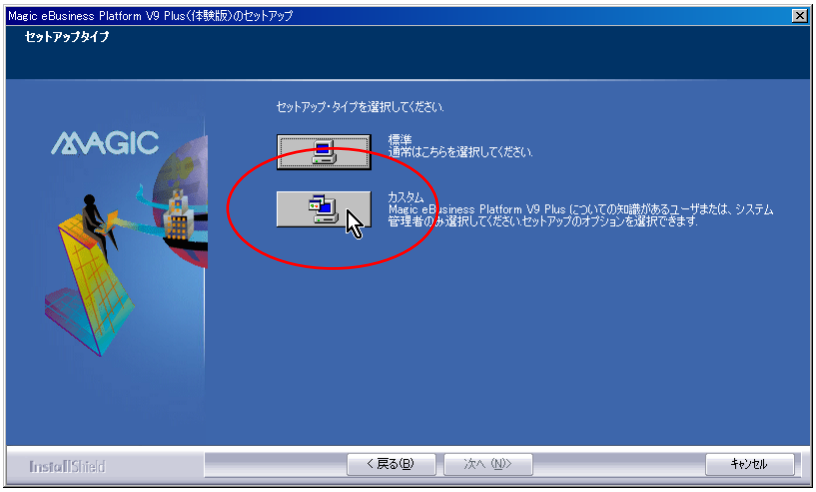
3.1. ゲートウェイのインストール

Magic から MS SQL Server にアクセスするには、Magic の MS SQL Server 用ゲートウェイをインストールし、Magic エンジンが実行時にロードできるようにしておく必要があります。

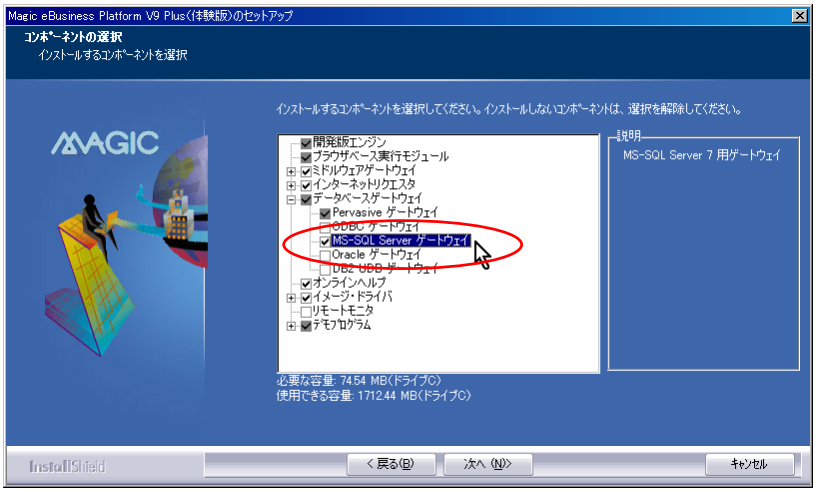
3.1.1. Magic の新規インストールの場合

MS SQL Server 用ゲートウェイは、標準のインストールを行った場合にはインストールされません。新規にインストールする場合には、「カスタム」インストールを選択し、MS SQL Server ゲートウェイを選択します。

1. インストール中、インストールタイプの選択画面に来たら、「カスタム」を選びます。



2. コンポーネントの選択の画面で、データベースゲートウェイから MS-SQL Server ゲートウェイを選択します。



3. その他は通常通りのインストールです。

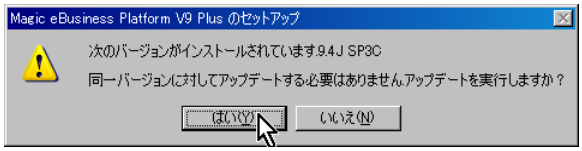
3.1.2. アップデートインストールの場合

Magic を標準インストールでインストールした環境がすでにある場合には、MSSQL のゲートウェイがインストールされていないので、アップデートインストールでインストールする必要があります。

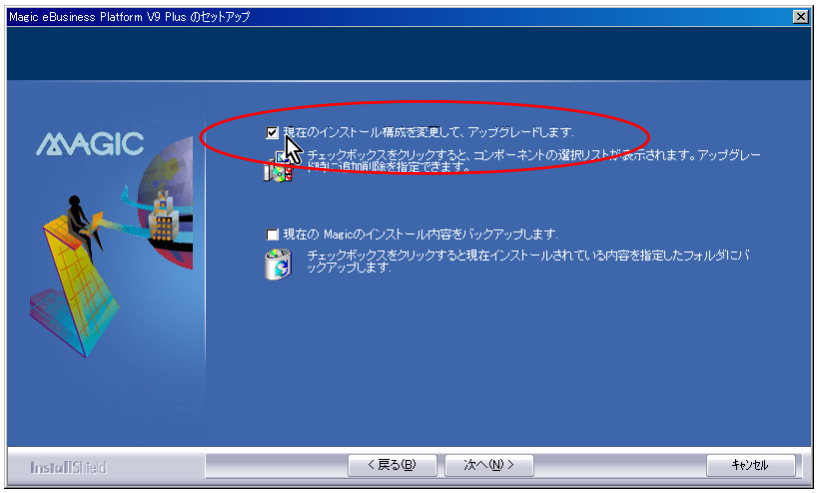
アップデートインストールは、インストーラを用いて行います。

1. インストールしたときと同じようにインストーラを起動します。

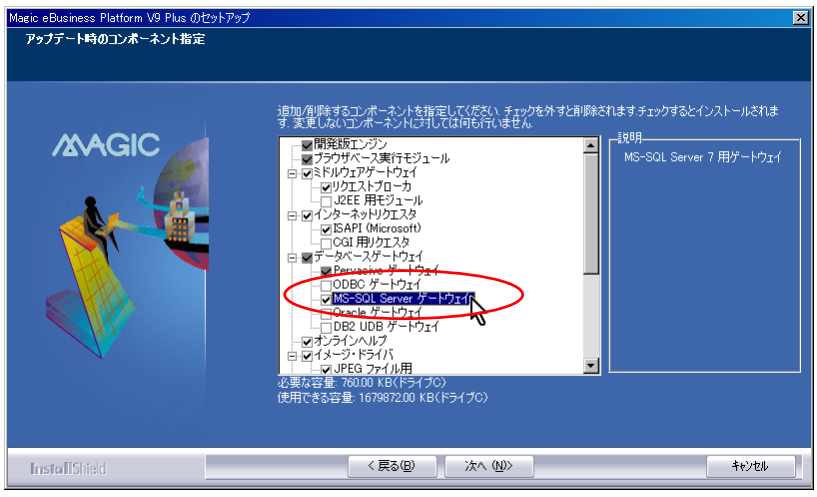
2. 右図のような警告メッセージが出ますので、「はい(Y)」を押します。



3. 続いて、右のような画面が出たら、「現在のインストール構成を変更してアップグレードします」を選択し、「次へ(N)」を押します。



4. 「アップデート時のコンポーネント指定」画面で、データベースゲートウェイから MS-SQL Server ゲートウェイを選択します。



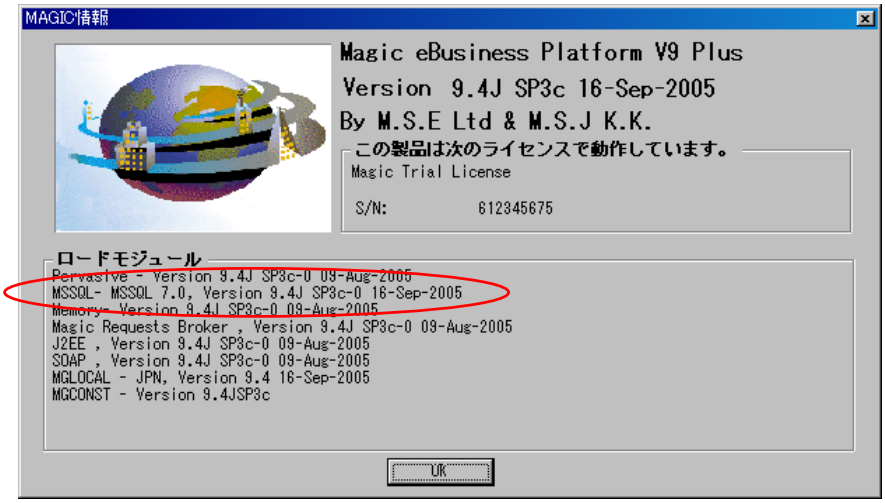
5. 後は、通常通りインストールを続けてください。

アップデートインストールが完了すると、MS-SQL Server ゲートウェイがインストールされています。

3.1.3. ゲートウエイの確認

Magic を起動して、MSSQL ゲートウェイがロードされていることを確認しましょう。

1. Magic を起動します。
2. メニュー 「ヘルプ(H)」から「Magic 情報(M)」を選びます。
3. バージョン情報のダイアログ(右図)が表示されますが、ロードモジュール一覧の中に MSSQL があることを確認してください。あれば MS-SQL Server ゲートウェイが正常にロードされています。

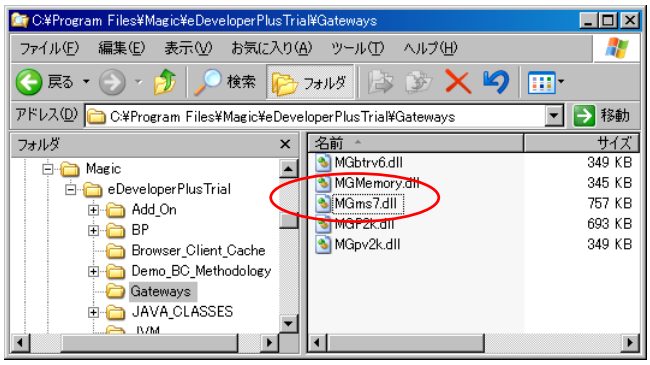


参考： もしロードされていなければ・・・

もしロードモジュール一覧に MSSQL がなければ、以下の点を確認してください。

ゲートウェイモジュールがあるか？

Magic のゲートウェイモジュールは、Magic をインストールしたディレクトリの下の Gateways サブディレクトリに DLL の形で格納されています。ここを見て、MGms7.dll というモジュールがあるかどうかをチェックしてください。もしなければ、前記インストールを再度行ってみてください。



MAGIC.INI にゲートウェイが登録されているか？

Magic がインストールされたディレクトリに、MAGIC.INI というファイルがありますが、これは Magic の動作を制御する環境設定パラメータが多数定義されています。この中に[MAGIC_GATEWAYS] というセクションがあり、ここでロードするゲートウェイを指定しています。MS-SQL Server ゲートウェイは MGDB20 というキーワードで指定されますが、これが正しくゲートウェイモジュールを参照していることを確認してください。特に、行頭にセミコロン「;」文字があると、コメントとして無視されますので、コメントがはずれていることを確認してください。

```
[MAGIC_GATEWAYS]
;MGCOMM01=mgwsock.dll
MGDB00=Gateways¥mgpv2k.dll
;MGDB03=mdcisam.dll
;MGDB05=Gateways¥mgcache.DLL
;MGDB06=Gateways¥mgdb400.DLL
;MGDB13=Gateways¥mgOra8.dll
;MGDB14=Gateways¥mginf.dll
;MGDB16=Gateways¥mgeac32.dll
;MGDB18=Gateways¥mgdb2.DLL
;MGDB19=Gateways¥mgodbc.dll
MGDB20=Gateways¥mgms7.dll
MGDB21=Gateways¥mgmemory.dll
```

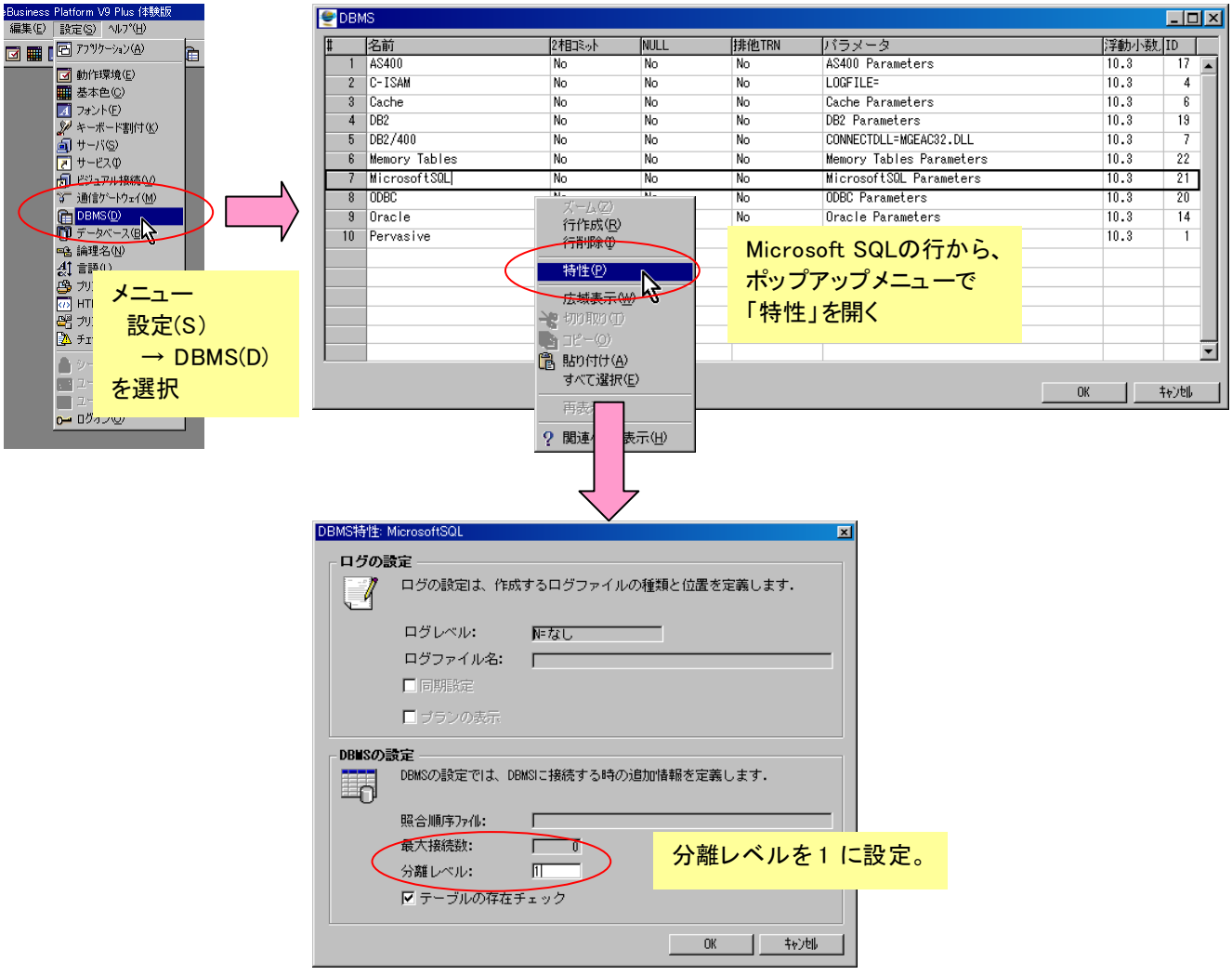
3.2. DBMS テーブル

MS-SQL Serve にアクセスするには、Magic でいくつかの設定をする必要があります。最初に設定するのは、**DBMS テーブル**です。

DBMS テーブルは、各 DBMS (MS-SQL、Oracle、Pervasive.SQL、DB2UDB、など)ごとに共通な設定を行うものです。MS-SQL Server の場合には一箇所だけ、**分離レベル**の設定を変更します（図 3-1 参照)。

1. DBMS テーブルは、Magic のアプリケーションを閉じた状態で、メニュー「設定(S)」から「DBMS(D)」を選んで開くことができます。
2. 名前欄が「MicrosoftSQL」の行にカーソルを置き、右クリックでポップアップメニューを出して「特性」を選択します。(あるいは、Ctrl+P でも同じです)。DBMS 特性ダイアログが開きます。
3. 「分離レベル」を 1 にします。ほかの部分は変更しなくてかまいません。

図 3-1 DBMS テーブルの設定



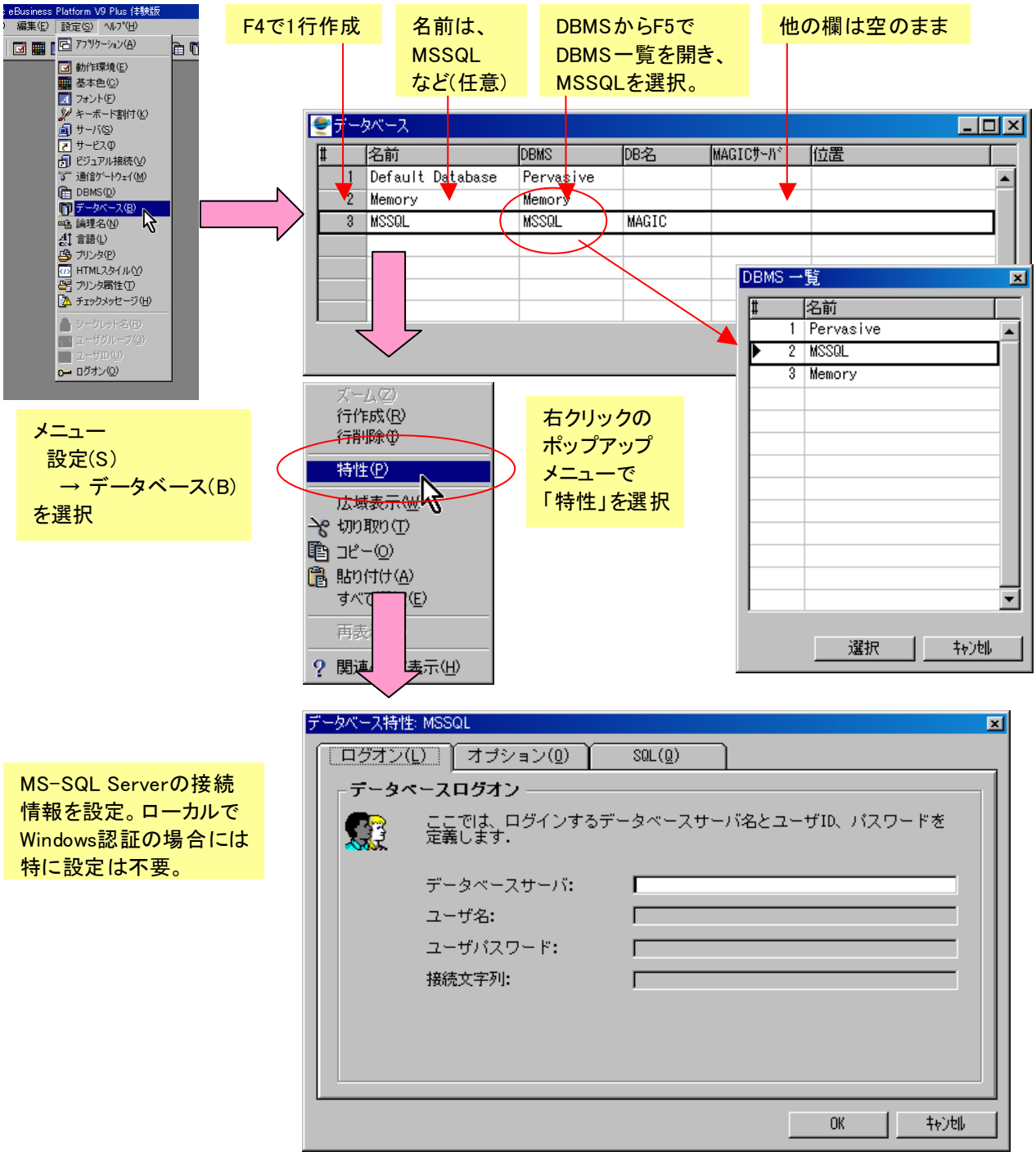
参考： 分離レベルについて

分離レベルとは、複数ユーザが同時にトランザクションを行っているときに、トランザクションの間の独立性を決めるものです。詳細は「9.3 分離レベル」（66 ページ）を参照してください。デフォルトは 0（Read Uncommited）で、これは並列性が高いがデータ整合性に問題があるので、1（Read Commited）の方が適当です。

3.3. データベーステーブル

データベーステーブルは、特定の DBMS 上に作成されたデータベースを参照するもので、同時に接続情報(サーバマシン名、接続時のユーザ ID、パスワード)、その他のパラメータを設定します。
本書では、ローカルマシンにある MS-SQL Server 上に作成された、MAGIC という名前のデータベースを参照します（図 3-2)。

図 3-2 データベーステーブルの設定



1. メニュー「設定(S)」から「データベース(B)」を選択します。データベーステーブルが開きます。
2. F4 で 1 行新規作成します。
3. 「名前」欄には「MSSQL」などとします。この名前は Magic のテーブルリポジトリの中から参照されるもので、任意の名前を設定できます。普通はデータベースの目的がわかるような名前とつけます。
4. 「DBMS」欄から F5 キーでズームすると、現在ロードされているデータベースゲートウェイの一覧が表示されます。今は MS-SQL Server をアクセスするので、MSSQL を選択します。
5. そのほかの欄は空欄のままにします。
6. 右マウスクリックでポップアップメニューを出し、「特性」を選択します。データベース特性ダイアログが開きます。
7. 「ログオン」タブには、データベース接続情報を記入します。ローカルマシンに Windows 認証で接続する場合には、すべて空欄のままでかまいません。別マシンにある場合には、マシン名と接続時のログオン ID、パスワードをここに指定します。

以上でデータベーステーブルの設定は終わりです。

4. MSSQL テーブルを Magic から扱ってみる

本書の最終目標は、ペットショップデモを MS-SQL Server 対応にすることですが、最初に Magic の SQL 対応機能について理解するため、テスト用のアプリケーションを作成して以下のように実験してみます。

- 新規アプリケーションの作成
- テーブル定義
- テーブル作成とデータ登録
- データ再編成

また、MS-SQL Server を利用する場合の注意事項として、以下の点について説明します。

- 命名規則
- 名前の対応
- 一意インデックスの定義

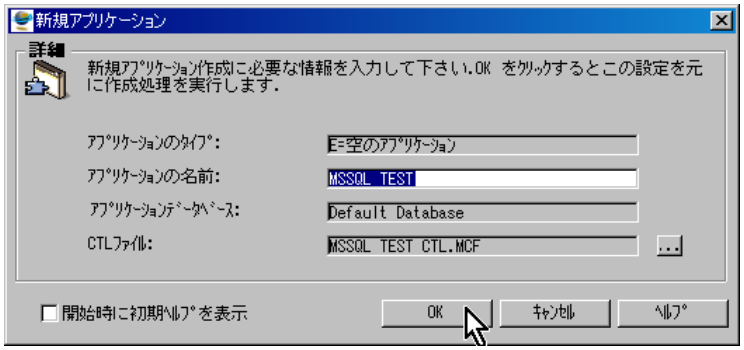
4.1. 新規アプリケーションの作成

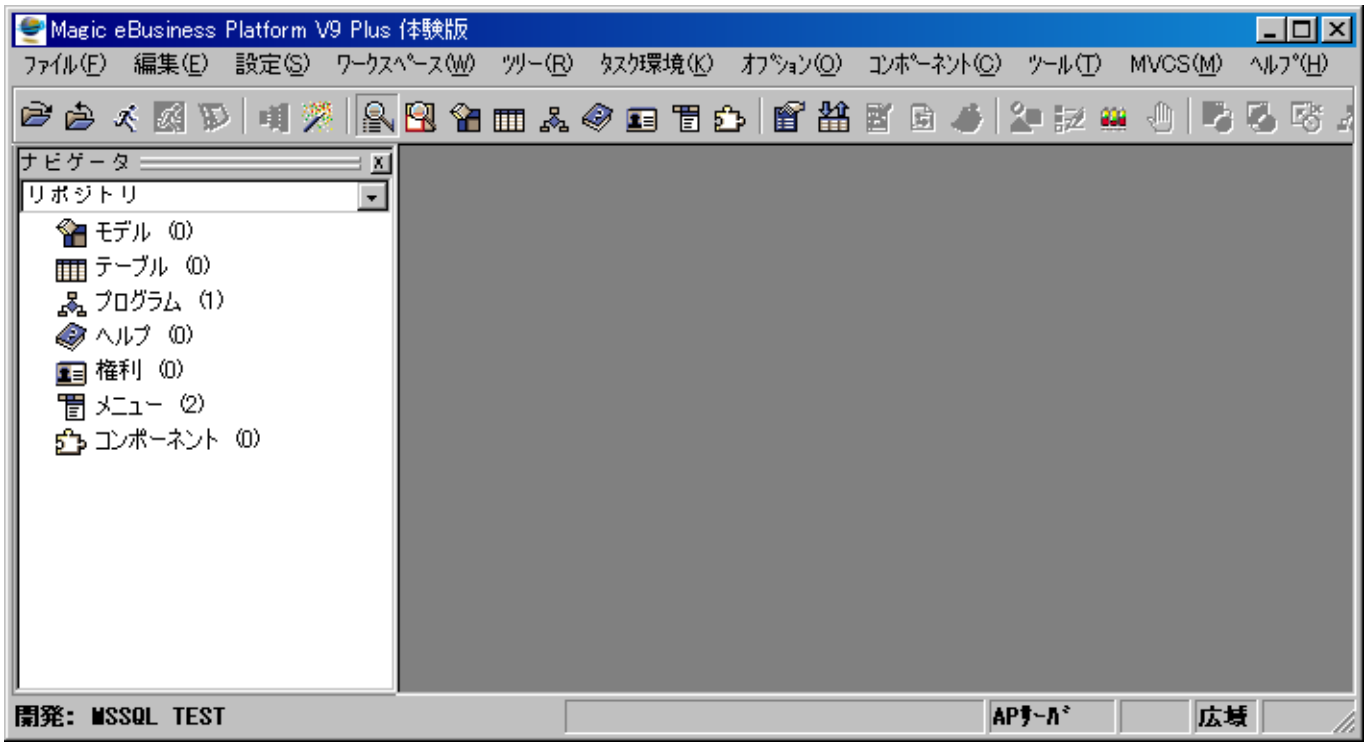
Magic で新規アプリケーションを作成するには、以下のようにします。

1. メニュー「ファイル(F)」から「新規作成(N)」を選びます。新規アプリケーションダイアログが開きます。

2. 「アプリケーションの名前」欄に適当な名前を指定します。ここでは、「MSSQL TEST」とします。

3. OK ボタンを押すと、新規アプリケーション作成され、オープンされます。





4.2. テーブル定義

まずは、テーブルリポジトリで、簡単なテーブルを新規定義しましょう。

1. テーブルリポジトリを開きます。(メニュー「ワークスペース(W)」から「テーブル(B)」を選ぶか、Shift+F2で開きます。
2. 次のようなテーブルを定義してください。

名前： テスト、DB テーブル名： テスト、データベース： MSSQL			
カラム定義			
#	カラム名	型	書式
1.	数値型	N=数値	N5
2.	文字型	A=文字	10
インデックス定義			
#	インデックス名	セグメント番号	セグメントカラム名
1	BY 数値	1	数値型

Magic のテーブルリポジトリでは、下図のように定義します。

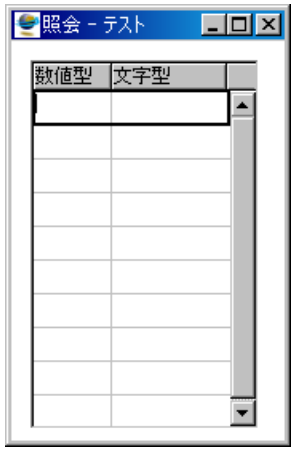
- カラム定義

- インデックス定義

4.3. テーブル作成とデータ登録

テーブルリポジトリに定義したテーブルは、この時点では Magic 上に定義だけしか存在しません。ここで MS-SQL Server 上にテーブルを作成しましょう。

テーブル作成について、特別な手続きは必要ありません。単に APG などでテーブルを開くだけで、Magic が自動的にテーブルを作成します。つまり、Pervasive.SQL の場合と同様、Magic はテーブルのオープンに先だってテーブルが存在しているかをチェックし、もし存在しなければ自動的にテーブルを作成します。



テーブルリポジトリで、APG でテーブルを開いてみてください。最初はレコードが登録されていないので、空のテーブルが表示されます(右図)。この時点で、すでに MS-SQL Server 上にテーブルは作成されています。

今の例では、内部的に次のような CREATE TABLE 文が発行され、テーブルとインデックスが作成されます。

```
CREATE TABLE MAGIC..テスト (数値型 INT NOT NULL,文字型 CHAR(10) NOT NULL)
CREATE UNIQUE NONCLUSTERED INDEX BY数値 ON MAGIC..テスト (数値型)
```

ここでは、MAGIC というユーザ ID で Windows にログインしているので、テーブルのオーナ名が MAGIC になっています。この DDL 文は Magic が内部的に自動的に生成・実行するので、Magic のアプリケーション開発者はこれを意識する必要がありません。

何件かレコードを登録します (右図)。終わったら、ESC キー(あるいはウィンドウ右上の X ボタン)で閉じてください。

OSQL から、テーブルの中身を見てみましょう。データが正しく作成されていることを確認してください。

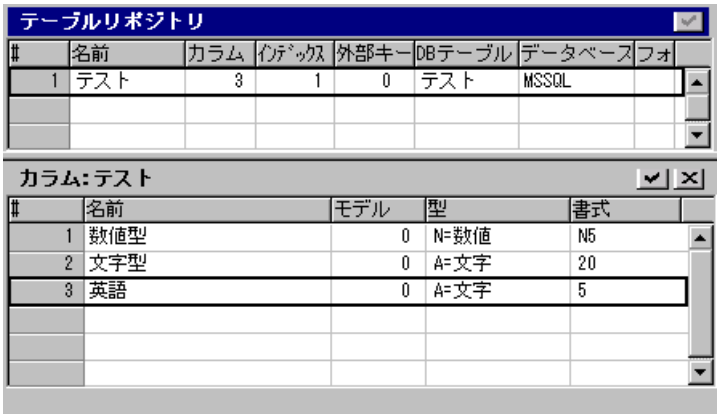
```
1> use magic
2> select * from テスト
3> go
 数値型          文字型
 ----------- ----------
           1 赤巻紙
           2 青巻紙
           3 黄巻紙
(3 件処理されました)
```

4.4. データ再編成

Magic のテーブルリポジトリの定義を変更すると、MS-SQL Server 上のテーブルも自動的に変更されます。これは Magic の**自動データ再編成機能**です。

4.4.1. テーブルを変更してみる

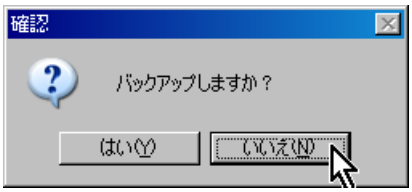
今の例で、文字型カラムを 10 文字から 20 文字に増やし、さらに、第 3 のカラムとして「英語」というカラムを追加しましょう。データ型は A=文字、書式は 5 です。



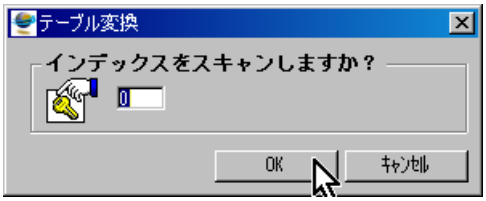
Enter キーを 2 回押してテーブルリポジトリを閉じるか、別のテーブルのエントリに移動しようとすると、自動データ再編成機能が開始されます。

1. 最初に確認画面が出ますので、「はい（Y)」で答えてください。

2. 次に、もとのテーブルをバックアップするかを聞いてきます。安全のためにはバックアップを取っておくほうが良いですが、スペースが余計に必要になります。今はテストですので「いいえ(N)」で答えてください。



3. テーブル変換オプションが出てきます。今はこのままで OK ボタンを押してください。今回はデータ件数が少ないので、すぐに終了するはずです。



これで、MS-SQL Server 上のテーブルも変更されています。

Magic から APG をかけて、テーブルを開いてください。「文字型」のカラムが広がり、新たに「英語」カラムができています。

参考：
新しく作成された「英語」カラムのデータはすべて NULL となります。

ここでデータを修正して、新しく作成された「英語」カラムにもデータを設定しましょう。

1. APG で「テスト」テーブルを開きます。
2. APG でテーブルを開いた状態ではタスクは照会モードなので、データを修正することができません。修正モードに変更してください。
   参考： 修正モードに変更するには、メニュー「オプション(O)」から「修正(M)」を選ぶか、あるいは Ctrl+M キーを押します。
3. 「英語」カラムに適当にデータを入力します。
4. ESC で閉じます。

OSQL からこのテーブルを見て、変更が正しく反映されていることを確認してください。

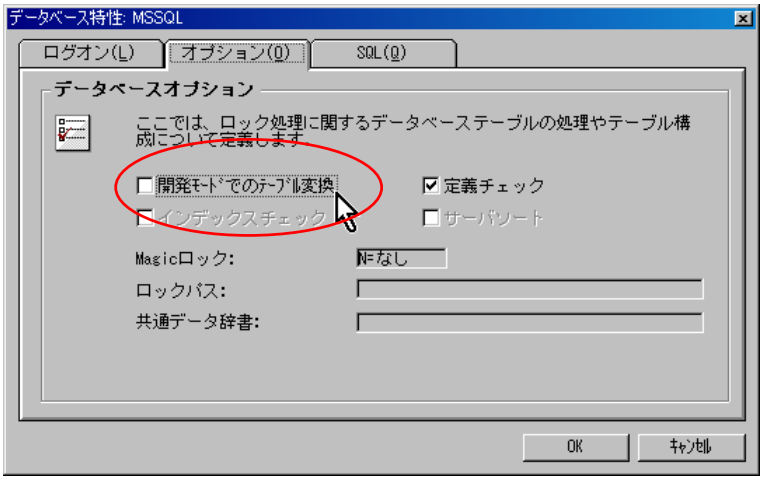
```
1> select * from テスト
2> go
 数値型          文字型                英語
 ----------- -------------------- -----
           1 赤巻紙                Red
           2 青巻紙                Blue
           3 黄巻紙                Yello
(3 件処理されました)
```

4.4.2. データ再編成機能の無効化

このように、Magic の自動データ再編成機能を使うと、テーブルの定義変更に伴うデータ再編成に非常に簡単に対処することができ、Magic 上の定義と実際の MS-SQL Server 上のテーブルとを常に同期させることができます。また、上では触れませんでしたが、データ再編成機能の一環として、Magic のテーブルリポジトリ上でのテーブル定義を削除すると、それにあわせて、MS-SQL Server 上のテーブルも削除されるようになっています。これは、開発途上でしばしばテーブル定義を変更したり、テーブルの追加削除をしなければならない状況では大変便利です。
しかし、逆に言うと、不注意でテーブル定義を変更してしまった場合にも MS-SQL Server 上のテーブルが

変更されてしまい、意図せぬデータの破壊になってしまう可能性もあります。これは実際の運用環境で起こったら大変な問題になります。
これを防ぐため、Magic ではデータベーステーブルの設定により、自動データ再編成機能を無効化させることができます。

1. アプリケーションを閉じてください。(メニュー「ファイル(F)」から「アプリケーション・クローズ(C)」で閉じます)
2. データベーステーブルを開いてください (メニュー「設定(S)」から「データベース(B)」を選びます)。
3. 「MSSQL」のエントリに移動し、データベース特性ダイアログを開いてください。(マウスの右クリックでポップアップメニューを開き「特性」を選ぶか、あるいは Ctrl+P で開きます)。
4. 「オプション(O)」タブを開きます。
5. 「開発モードでのテーブル変換」のチェックをはずします。



これで、この MSSQL データベースに対しては、自動データ再編成機能が無効となります。

注意：
自動データ再編成機能を無効にした場合には、Magic のテーブルリポジトリでテーブル定義を変更しても MS-SQL Server 上の実際のテーブルに反映されません。これによりテーブル定義の不整合が発生し、アプリケーション実行中にエラーが発生したり、意図したとおりに動作しなくなる可能性もあります。
このように、自動データ再編成機能の有効/無効は、作業の目的に合わせて適切に選んでください。

4.5. 命名規則

Pervasive.SQL の場合には、DB テーブル名は OS 上のファイル名(Pervasive.SQL 形式)となり、OS のファイル名として許容される名前ならば任意に指定することができました。
MS-SQL Server などの SQL 系の DBMS を利用する場合には、DB テーブル名、カラム名、インデックス名などに、各 DBMS に固有な命名規則があり、Magic のテーブルリポジトリでテーブルを定義する場合にも、それに則って命名する必要があります。

例えば、MS-SQL Server 2000 の場合には、命名に関して次のような一般規則があります。

1. 最初の文字が次のいずれかである必要があります。
   - Unicode Standard 2.0 で定義されている文字。Unicode の文字定義には、各国言語の文字のほかに、ラテン文字 a ～ z と A ～ Z も含まれます。
   - アンダースコア (_)、アット マーク (@)、または番号記号 (#)。
2. 名前の先頭以外では、次の文字を使用できます。
   - Unicode 規格 2.0 で定義されている文字
   - Basic Latin スクリプトまたはそのほかの各国スクリプトの 10 進数
   - アット マーク、ドル記号 ($)、番号記号、またはアンダースコア
3. Transact-SQL 予約語を識別子として使用することはできません。SQL Server では予約語は大文字、小文字ともに予約されています。
4. 埋め込み型スペースおよび特殊文字は使用できません。
5. これらの規則に従わない識別子を Transact-SQL ステートメントで使用する場合は、二重引用符または角かっこで区切る必要があります。

テーブル名、カラム名、インデックス名などには、それぞれまた細かな制限があります。詳細は、MS-SQL Server のリファレンスマニュアルを参照してください。

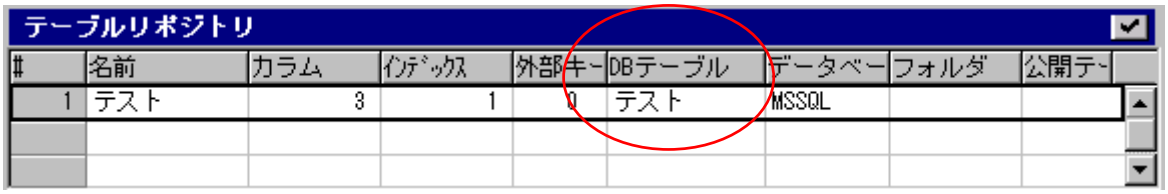
一般的な指針としては、特殊文字の使用はアンダースコア「_」程度に止めておくことと、あまり長い名前は避けることです。

4.6. 名前の対応

Magic の場合、テーブルリポジトリ上に定義する名前と、DBMS 上に作成されるテーブルでの名前とを異なるものとして定義することが可能です。ここでは、Magic のテーブルリポジトリでの名前の定義と DBMS 上での名前との対応関係を説明します。

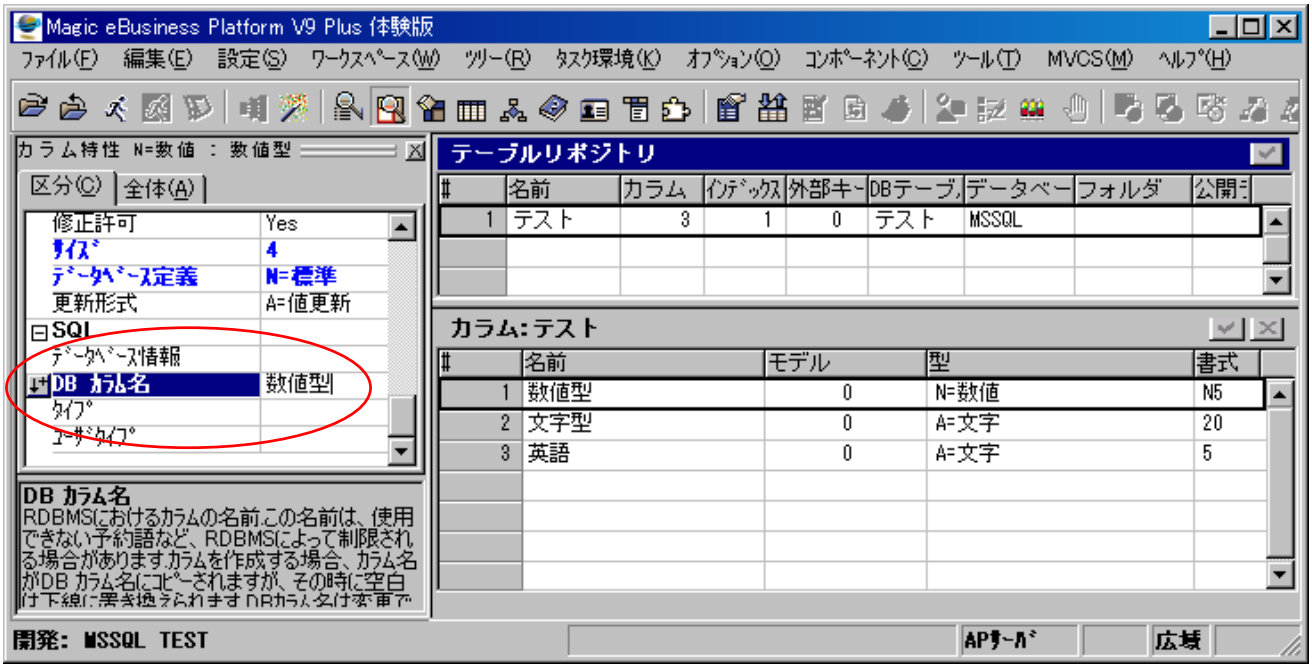
4.6.1. テーブル名

テーブルリポジトリの「名前」欄は、Magic 内部だけで使う名前で、任意の名前をつけることができます。一方、DBMS 上に作成されるテーブルの名前は、「DB テーブル」欄に指定したものが採用されます。デフォルトでは、「名前」欄に指定したのと同じ名前が「DB テーブル」欄にも自動的に設定されますが、異なるものに変更することが可能です。



4.6.2. カラム名

テーブルリポジトリの下半分のカラムテーブルでは、「名前」欄でカラム名を指定しますが、この名前は Magic 内部でだけ使う名前で、任意の名前をつけることができます。DBMS 上に作成されるテーブルのカラム名は、カラム特性の「DB カラム名」が採用されます。
デフォルトでは「名前」欄に指定したのと同じ名前が「DB カラム名」欄にも自動的に設定されますが、異なるものに変更することが可能です。



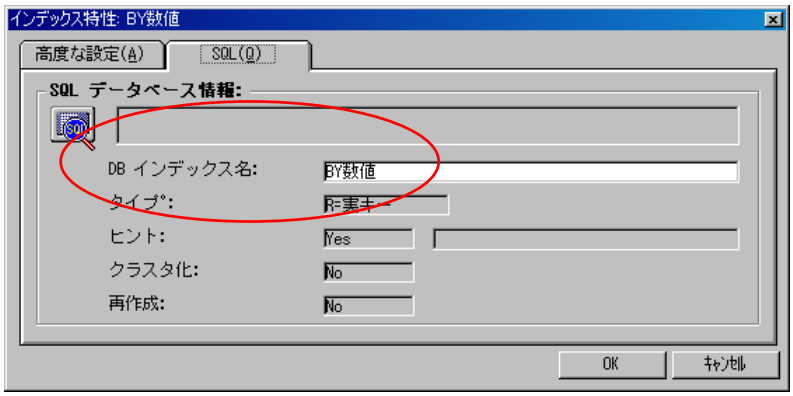
4.6.3. インデックス名

テーブルリポジトリで、カーソルを「インデックス」欄に置くと、そのテーブルに定義されているインデックスが表示されます。

ここで、インデックスの「名前」欄に指定された名前は Magic 内部でのみ使う名前で、任意の名前をつけることができます。

DBMS 上で作成されるインデックスの名前は、インデックス特性で定義されます。

1. インデックステーブル上のエントリ上にカーソルを置いてください。
2. インデックス特性ダイアログを開きます。(右マウスクリックでポップアップメニューを開き、そこから「特性」を選びます。または、Ctrl+P キーを押します)。
3. 「SQL(Q)」タブを開きます。ここに「DB インデックス名」欄があります。



デフォルトで、インデックスの「名前」欄と同じ名前が設定されますが、あとから変更することもできます。

このように、Magic では DBMS 上での名前と Magic 内部での名前とを別々に設定できるようになっているので、DBMS での命名規約を守りつつ、Magic の中ではそれに縛られないわかりやすい名前をつけることができるようになっています。このことは、開発・保守時のプログラムのわかりやすさを向上させるとともに、DBMS の移植性(異なる DBMS にアプリケーションを移行させる)を向上させるのにも役立っています。

4.7. 一意インデックスの定義

MS-SQL Server 上にあるテーブルを Magic からアクセスする場合に、ひとつ重要な約束ごとがあります。それは、「ユニークインデックスを必ず一つは定義しておかなければならない」ということです。

Magic では、インデックスの定義で「タイプ」欄を「U=重複不可」に設定すると、MS-SQL Server 上でユニークキーが作成されます。例えば、テーブル「テスト」のインデックスの定義をもういちど見てみると、「BY 数値」という名前のインデックスが「U=重複不可」に設定されています。このようなインデックスが一つでもあれば OK です。

各テーブルに最低一つのユニークキーを定義しておく必要があるかどうかは、DBMS により異なります。表 4-1 に、DBMS ごとの必要性をまとめました。

表 4-1 ユニークインデックスの必要の有無

ユニークインデックスを必要とする	必要としない
MS-SQL Server	Pervasive.SQL
DB2/UDB	Memory
ODBC	Oracle

このようなことから、例えば、Pervasive.SQL で作成したアプリケーションを MS-SQL Server に移植するような場合にユニークインデックスの有無のチェックが必要で、必要に応じてユニークインデックスを付加してください。

なお、この制限は Magic の実行方式と DBMS の機能との兼ね合いから来るもので、MS-SQL Server 自身の制限ではありません。

5. 定義取得

前章では、Magic のテーブルリポジトリで定義したテーブルを MS-SQL Server 上に作成する、という方法を説明しました。
本章ではその逆に、MS-SQL Server にすでに存在するテーブルを Magic に取り込む、**定義取得**の機能について説明します。

まず、実験の準備として、テーブルをいくつか MS-SQL Server 上に作成しておきましょう。OSQL を使って、簡単なテーブルを二つほど定義しておきます。

1. テキストファイルに以下の内容を書き出しておいてください。名前は「顧客商品.sql」などとしておきます。

```
USE MAGIC
CREATE TABLE 顧客 (顧客番号 INT NOT NULL,顧客名 CHAR(20),顧客名ヨミ CHAR(20),住所 CHAR(60))
CREATE UNIQUE INDEX BY顧客番号 ON 顧客 (顧客番号)
CREATE INDEX BY顧客名 ON 顧客 (顧客名)
INSERT INTO 顧客 VALUES(100231,'山口サービス','ﾔﾏｸﾞﾁｻｰﾋﾞｽ','東京都武蔵野市山田町2-1-23')
INSERT INTO 顧客 VALUES(100240,'岡村不動産','ｵｶﾑﾗﾌﾄﾞｳｻﾝ','東京都豊島区南大塚1-2-3')
INSERT INTO 顧客 VALUES(100252,'富山薬局','ﾄﾔﾏﾔｯｷｮｸ','東京都渋谷区西3-15-5')
CREATE TABLE 商品 (商品番号 INT NOT NULL,商品名 CHAR(20),単価 INT,有効 BIT)
CREATE UNIQUE INDEX BY商品番号 ON 商品 (商品番号)
CREATE INDEX BY商品名 ON 商品 (商品名)
INSERT INTO 商品 VALUES(5023,'ウマカコーヒー',500,0x01)
INSERT INTO 商品 VALUES(5055,'山梨観光地割引切符',23500,0x01)
INSERT INTO 商品 VALUES(6034,'加湿器 SVKR-703',13200,0x01)
GO
```

2. コマンドラインから、次のコマンドを実行します。これは、指定したファイルから SQL 文を読み込み、すべて実行するものです。

```
C:¥Program Files¥Magic¥eDeveloperPlusTrial> osql -E -i 顧客商品.sql -n
(1 件処理されました)
(1 件処理されました)
(1 件処理されました)
(1 件処理されました)
(1 件処理されました)
(1 件処理されました)
C:¥Program Files¥Magic¥eDeveloperPlusTrial>
```

これで、MS-SQL Server 上に「顧客」と「商品」というテーブルが作成され、それぞれ 3 レコードづつデータが挿入されました。

5.1. 複数のテーブルの定義取得

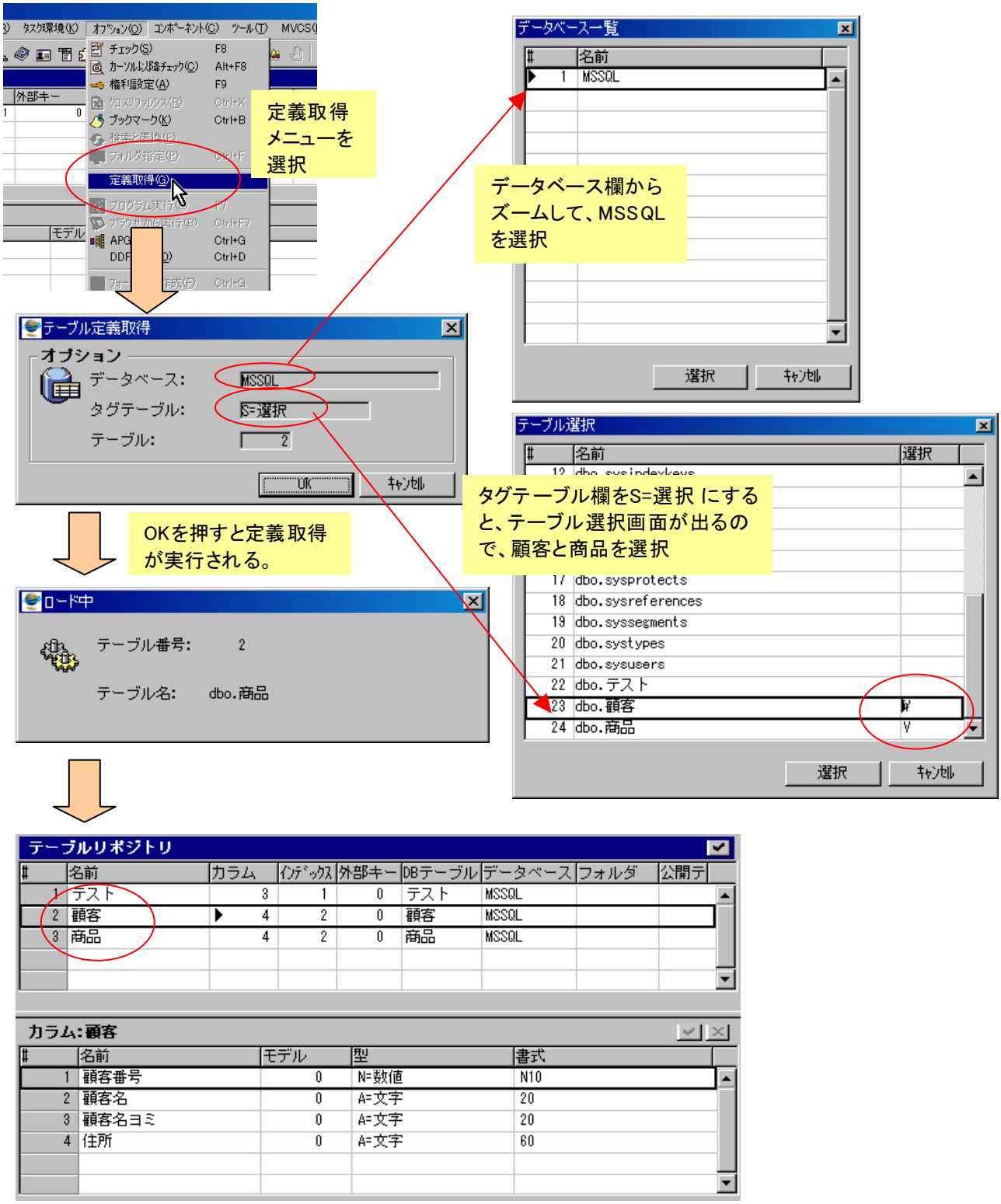
準備が整ったので、Magic から定義取得をしてみましょう。ここでは、MS-SQL Server 上に定義されているテーブルの一覧をまず取得し、そこから実際に定義取得するテーブルを選択する方法を示します。

1. アプリケーション「MSSQL TEST」を開きます。
2. テーブルリポジトリを開きます。
3. カーソルをタイトル行においておきます。先頭の行にカーソルがある状態で、さらに ↑ キーを押すと、カーソルがタイトル行に移動します。右図のように、カーソルが三角形になって表示されます。

4. メニュー「オプション(O)」から、「定義取得(G)」を選択します。定義取得ダイアログが開きます。(以下、図 5-1 参照)
5. 「データベース」欄からズーム(F5 キー、またはダブルクリック)すると、データベーステーブルに登録されているデータベースの中から、定義取得の機能をサポートするものの一覧が表示されます。今の場合、「MSSQL」のみですので、これを選択します。
6. 「タグテーブル」欄を「S=選択」にすると、テーブル選択画面が表示されます。
7. 「dbo.顧客」と「dbo.商品」とを選択します。テーブルの選択をするには、選択するテーブルの行にカーソルを合わせ、スペースキーを押すと、選択を示すチェック記号「V」がトグルします。
8. 「選択」 ボタンを押すと、選択したテーブルの数が「テーブル」欄に表示されます。今の場合は、二つのテーブルを選択しているので、「２」が表示されています。
9. OK ボタンを押すと、定義取得が実行されます。
10. 定義取得が完了したら、テーブルリポジトリに「顧客」と「商品」のエントリが追加されているはずです。

正しく定義取得ができたかを確認するために、APG でテーブルを開いて内容を確認してください。

図 5-1 定義取得の手順


定義取得
メニューを
選択
データベース欄から
ズームして、MSSQL
を選択
テーブル定義取得
オプション
データベース:
MSSQL
タグテーブル:
S=選択
テーブル:
2
OK
キャンセル
OKを押すと定義取得
が実行される。
タグテーブル欄をS=選択 にする
と、テーブル選択画面が出るの
で、顧客と商品を選択
ロード中
テーブル番号: 2
テーブル名: dbo.商品
データベース一覧
名前
1 MSSQL
選択
キャンセル
テーブル選択
名前
選択
17 dbo.sysprotects
18 dbo.sysreferences
19 dbo.syssegments
20 dbo.systypes
21 dbo.sysusers
22 dbo.テスト
23 dbo.顧客
24 dbo.商品
テーブルリポジトリ
名前 カラム インデックス 外部キー DBテーブル データベース フォルダ 公開テ
1 テスト 3 1 0 テスト MSSQL
2 顧客 4 2 0 顧客 MSSQL
3 商品 4 2 0 商品 MSSQL
カラム:顧客
名前 モデル 型 書式
1 顧客番号 0 N=数値 N10
2 顧客名 0 A=文字 20
3 顧客名ヨミ 0 A=文字 20
4 住所 0 A=文字 60

5.2. 特定のテーブルの定義取得

前節での定義取得の方法は、MS-SQL Server に定義されているテーブルの一覧をいったん取得し、その中から実際に定義取得したいものを選択して定義取得する、という方法でした。この方法は便利なのですが、MS-SQL Server に定義されているテーブルが何百何千とあるような場合には、一覧を取得するのに時間がかかるし、また、その中から選択するにも時間がかかります。
もし、すでに名前のわかっているテーブルを一つだけ定義取得したい、というような場合には、一覧を取得してその中から選択する、というのは却ってわずらわしいこともあります。このような場合には、一覧を表示させずに、テーブル名を直接指定して定義取得する、ということもできます。

1. テーブルリポジトリで、F4 キーにより新規の行を作成します。
2. 「DB テーブル」欄に、定義取得したいテーブルの名前を指定します。
3. 「データベース」欄からズームし、「MSSQL」を選択します。

4. この行にカーソルがある状態で、メニュー「オプション(O)」から「定義取得(G)」を選択します。定義取得が実行されます。
5. 定義取得が完了したら、カラムやインデックスに定義が設定されているはずです。(右図)

同じく、定義取得が正しく行われたかを APG で確認してください。

5.3. 定義取得の結果を見てみる

定義取得して取得したテーブルのエントリの内容を見てみましょう。

「4.6 名前の対応」(35 ページ)で説明したように、MS-SQL Server 上に定義されている名前と、Magic 上での名前の対応付けは、表 5-1 のようになっています。

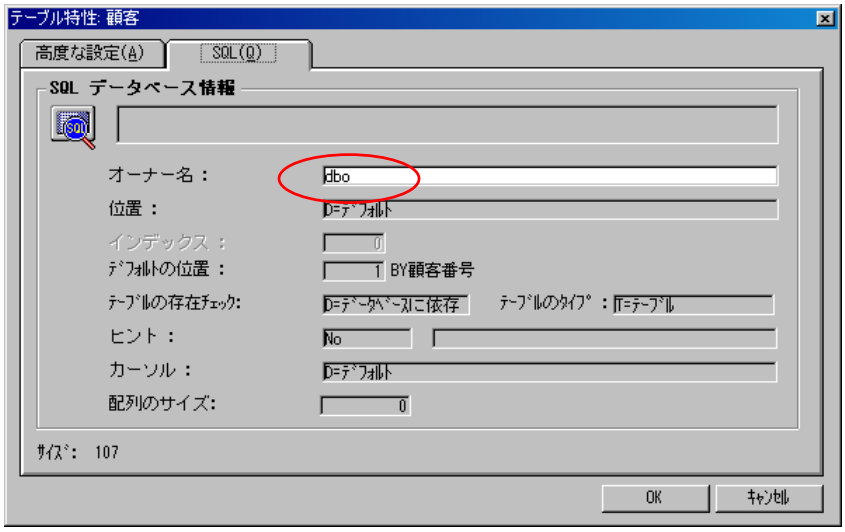
表 5-1 DBMS での名前と Magic での名前の対応表

MS-SQL Server 上での定義	Magic での定義
テーブル名	「DB テーブル」欄
カラム名	カラム特性の「DB カラム名」
インデックス名	インデックス特性の「SQL(Q)」タブの「DB インデックス名」欄

テーブルリポジトリで、定義取得したテーブルのエントリを見て、上の表のように設定がされていることを確認してください。

ここでひとつ抜けている項目があります。それはデータベースの**オーナー名**です。先ほど定義取得したテーブル「顧客」と「商品」は、OSQL で CREATE TABLE するときにオーナー名を指定せずに作成したので、デフォルトでオーナーは dbo となっています。それで定義取得時の「選択テーブル」一覧にも「dbo.顧客」のような名前で表示されていました。このオーナー名「dbo」はどこに行ってしまったのでしょうか？
答えは、「テーブル特性」です。次のようにしてオーナー名が設定されていることを確認してください。

1. 定義取得したテーブルのエントリにカーソルを置きます。
2. テーブル特性ダイアログを開きます。テーブル特性ダイアログは、右マウスボタンクリックでポップアップメニューを開き「特性」を選択するか、あるいは Ctrl+P キーにより開きます。
3. 「SQL(Q)」タブを開きます。「オーナー名」欄に「dbo」と設定されているはずです。



5.4. ビューの定義取得

SQL データベースでは、実際のデータを格納している実テーブルのほかに、実テーブルの上に仮想的に定義された**ビュー**があります。Magic では、DBMS に定義されているビューも定義取得することができます。

実験をするにあたり、簡単なビュー**高額商品**を定義しておきましょう。

```
1> use magic
2> go
1> create view 高額商品 as select 商品番号,商品名 from 商品 where 単価 > 10000
2> go
1> select * from 高額商品
2> go
 商品番号        商品名
 ----------- --------------------
        5055 山梨観光地割引切符
        6034 加湿器 SVKR-703
(2 件処理されました)
```

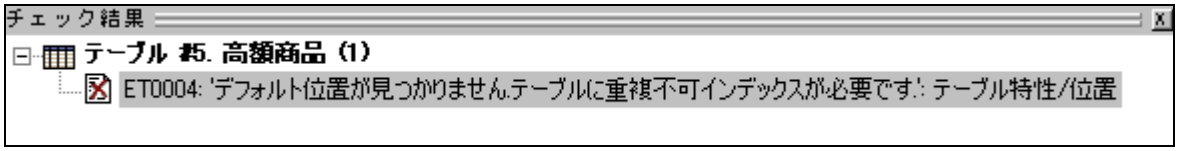
5.4.1. ビューの定義取得

ここで高額商品ビューを定義取得してみましょう。やりかたは前節で説明したのと同じです。

1. テーブルリポジトリに、F4 で新規行を作成する。
2. 「DB テーブル」欄を「高額商品」とし、「データベース」は「MSSQL」とする。
3. メニュー「オプション(O)」から「定義取得(G)」を選び、定義取得を実行する。

定義取得した結果を右図に示します。これを見ると、カラム情報は取得されていますが、インデックス欄は「0」であり、ひとつもありません。ビューにはインデックスを定義することができないので、当然な結果です。

しかし、「4.7 一意インデックスの定義」(37 ページ)で説明したとおり、MS-SQL Server のデータを Magic から扱う場合には、ユニークインデックスが最低一つが必要です。従って、この状態でシンタックスチェック(F8 キー)をかけるか、あるいは APG を行おうとすると、下図のようにエラーがでます。

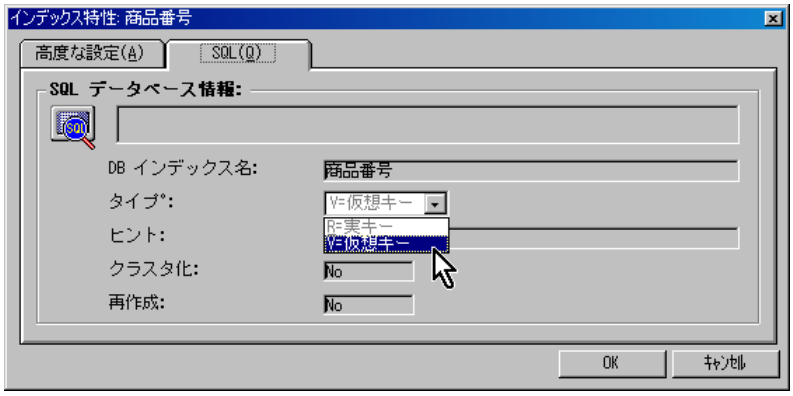


では、どうすれば良いのでしょうか？このような状況に対応するために、Magic には**仮想インデックス**が定義できます。

5.4.2. 仮想インデックスの定義

仮想インデックスというのは、Magic のテーブルリポジトリ上で、Magic プログラムの実行を目的としてだけ定義される仮想的なインデックスで、DBMS 上には対応するインデックスが実際になくとも構いません。Magic のプログラムからは、通常の実インデックスと同じように扱うことができます。
今の例では、「商品番号」カラムが重複不可のカラムであることがわかっているので、「商品番号」カラムを重複不可の仮想インデックスとして定義します。仮想インデックスの定義は次のように行います。

1. テーブルリポジトリの「インデックス欄」にカーソルをあわせ、通常のインデックスを定義する場合とまったく同じようにインデックスとインデックスセグメントとを定義します。ここでは名前を「商品番号」とします。
2. このインデックスのインデックス特性ダイアログを開きます。インデックス特性ダイアログを開くには、インデックス「商品番号」のエントリにカーソルを置いて、右クリックでポップアップメニューを開き、「特性」を選択します。(あるいは、Ctrl+P キーでもできます)。
3. 「SQL(Q)」タブを開きます。
4. 「タイプ」欄を「V=仮想キー」に選びます。



このように仮想キーを定義しておけば、ビューであっても、あたかもユニークキーが存在しているかのように扱いますので、Magic からテーブルを開いて見ることができるようになります。

5.4.3. 仮想インデックス利用上の注意

仮想キーは、あくまで Magic の中でだけ仮想的に定義されているものなので、実キーのある場合と全く同じではありません。仮想インデックスを利用する際には、いくつかの点に注意する必要があります。

- 範囲指定検索時のパフォーマンス上のメリットが出るとは限らない。一般に、実インデックスが定義されているカラムに対して範囲を指定してレコード検索を行うと、インデックスの効果により高速にレコードをアクセスすることができます。しかし、仮想インデックスの場合には、Magic の中で仮想的に定義されているだけであり、DMBS 上に実際にインデックスがあるわけではないので、範囲を指定してもインデックスを使った高速検索が行われず、全数検索になってしまうこともあります。
- 重複値のチェックが行われるとは限らない。ユニークな実インデックスのある場合には、重複値のチェックが DBMS により必ず行われ、重複値のあるレコードを挿入・更新しようとすると、DBMS がエラーを出します。しかし、仮想キーの場合には重複値のチェックのメカニズムがないため、DBMS による重複値のチェックが働きません。
- 重複不可の仮想キーを定義する場合には、定義したインデックスセグメントの組み合わせで、必ずレコードが一意に識別されるようになっていなければなりません。これを保証するのはアプリケーションの開発者の責任となります。Magic には仮想キーの一意性をチェックするメカニズムはありません。
  例えば、実際には重複がありうるカラムを使って、重複不可の仮想インデックスを定義することは可能です。しかし、そのようにインデックスの一意性に不整合がある場合には、実行結果は予測できないものになります。

なお、仮想インデックスは、ビューだけでなく実テーブルに対しても定義することが可能です。この場合には、Magic が内部的に MS-SQL Server に発行する SELECT 文に、仮想インデックスに定義されているセグメントを指定して ORDER BY ・・・ 句がつけられて実行されます。実行時になんらかの実インデックスが使われるか否かなどは、DBMS のオプティマイザが決定します。
実テーブルに対する仮想インデックスの利用の用途としては、あるテーブルが特定のレコード順に処理されるケースが多いのに、実インデックスが定義されていない場合などに、仮想インデックスを定義して、タスク特性の「インデックス」にその仮想インデックスを指定する、というような使い方が考えられます。

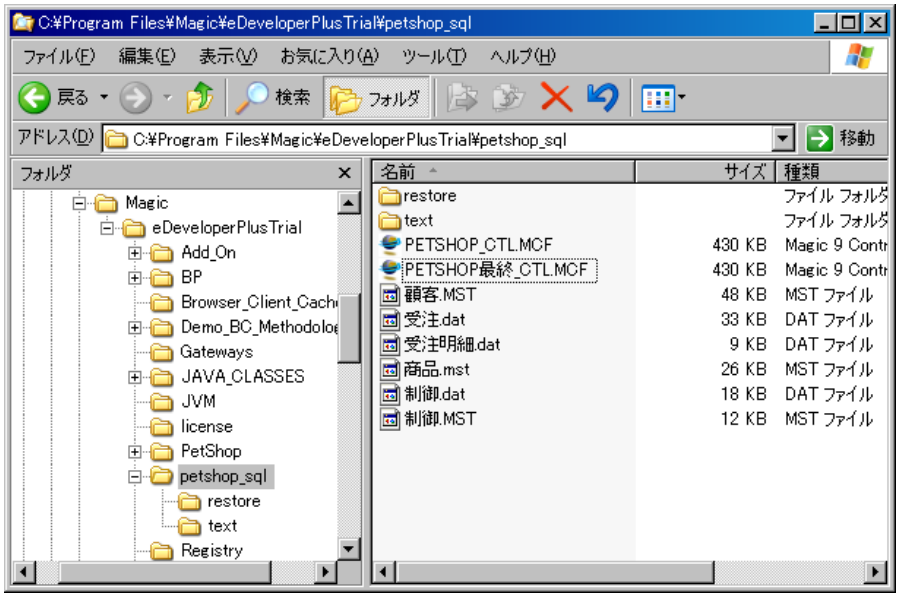
6. ペットショップサンプルの設定

前章までで Magic の SQL 対応機能の基本の説明をしましたので、本章からは、ペットショップデモを Pervasive.SQL から MS-SQL Server に移植する方法を説明します。

最初に、サンプルアプリケーションを設定する方法を説明します。

6.1.1. サンプルについて

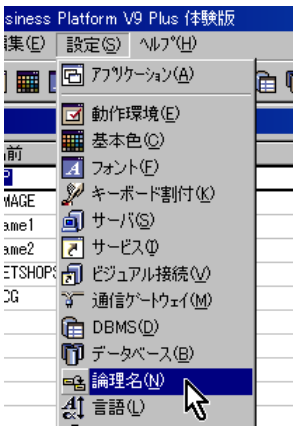
本チュートリアルと一緒に配布されるサンプルは、ZIP 形式でパックされています。ファイルを Magic ディレクトリで解凍すると、サブディレクトリとして petshop_sql が作成され、その下に二つの MCF ファイルと、その他のファイル、サブディレクトリがあります。



ファイル/ディレクトリ名	意味
PETSHOP_CTL.MCF	Pervasive 版のペットショップ。これが第 7 章「データ移行」(49 ページより)の出発点となります。
PETSHOP 最終_CTL.MCF	MS-SQL 対応版のペットショップ。第 7 章以降に従い順を追って修正していった最終形です。
.MST、.dat ファイル	Pervasive 形式のデータファイルです。
restore ディレクトリ	このディレクトリのコピーおよびリポジトリ出力形式が格納されています。このファイルをコピーしなおしたり、リポジトリ入力しなおしたりすることにより、いつでも初期状態に戻すことができます。
text	マスターテーブルのサンプルデータをテキストファイルとして格納したものが格納されています。

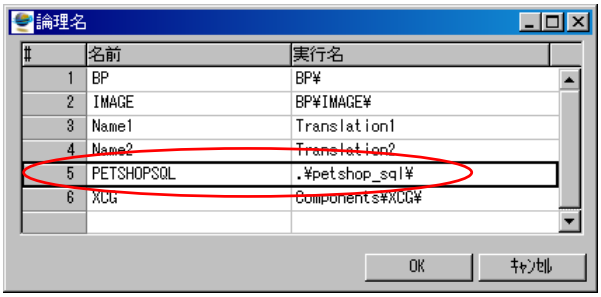
参考： MCF ファイルは、V9Plus 9.40JSP3 形式です。異なるバージョン/SP の Magic をお使いの場合には、restore ディレクトリにあるリポジトリ出力形式より、リポジトリ入力してください。

6.1.2. 論理名の設定



次に、Magic を起動して、**論理名**を設定します。
論理名というのは、Magic アプリケーションの移行・配布を容易化するための機能のひとつです。DOS や UNIX での環境変数に似た機能で、実行時にファイル名などの置き換えを行います。
論理名と実行時に置き換えられる名前(実行名)は、**論理名テーブル**で定義します。
論理名テーブルは、メニュー「設定(S)」から「論理名(N)」で開くことができます(右図)。

論理名テーブルで、F4 キーで新規に行を追加し、「名前」を「PETSHOPSQL」、「実行名」を「.¥petshop_sql¥」と設定してください。
このように設定しておくと、Magic プログラムの中で、「%PETSHOPSQL%」と指定してあるところでは、実行時にこれが「.¥petshop_sql¥」に変換されます。



例えば、SQL 化前の Pervasive のペットショップアプリケーションでは、テーブルリポジトリの「DB テーブル」の欄で使われています。

図 6-1 Pervasive 版のテーブルリポジトリでの論理名の使用

例えば、制御テーブルの DB テーブル欄は「%PETSHOPSQL%制御.DAT」となっていますが、これが実行時には「.¥petshop_sql¥制御.DAT」として認識されます。

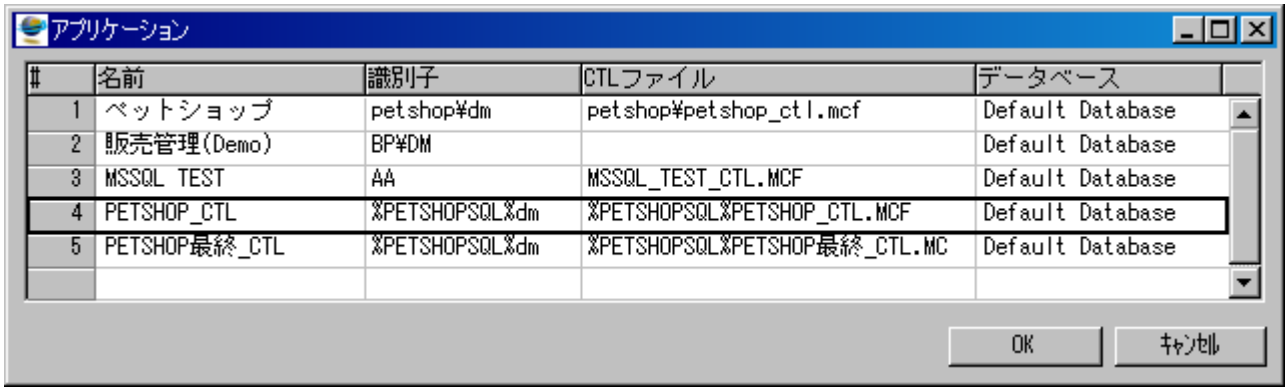
なお、論理名テーブルに定義した論理名と実行名の対応は、MAGIC.INI ファイルの[MAGIC_LOGICAL_NAMES]セクションに格納されます。

Pervasive を使った場合、DB テーブル欄にはファイル名を指定するのですが、ここで論理名を使って指定しておけば、実際の運用環境で、開発時とは異なるディレクトリにデータファイルを格納する場合にも、アプリケーションを変更することなく、MAGIC.INI の設定の変更だけで対応することができます。

6.1.3. アプリケーションテーブルの追加

提供されるサンプルアプリケーションを、アプリケーションテーブルに登録してください。
アプリケーションテーブルは、メニュー「設定(S)」→「アプリケーション(A)」で開くことができます。今の場合には既存の MCF を指定してアプリケーション登録をするので、図 6-2 のように登録してください。ここでも論理名を使うのが便利です。

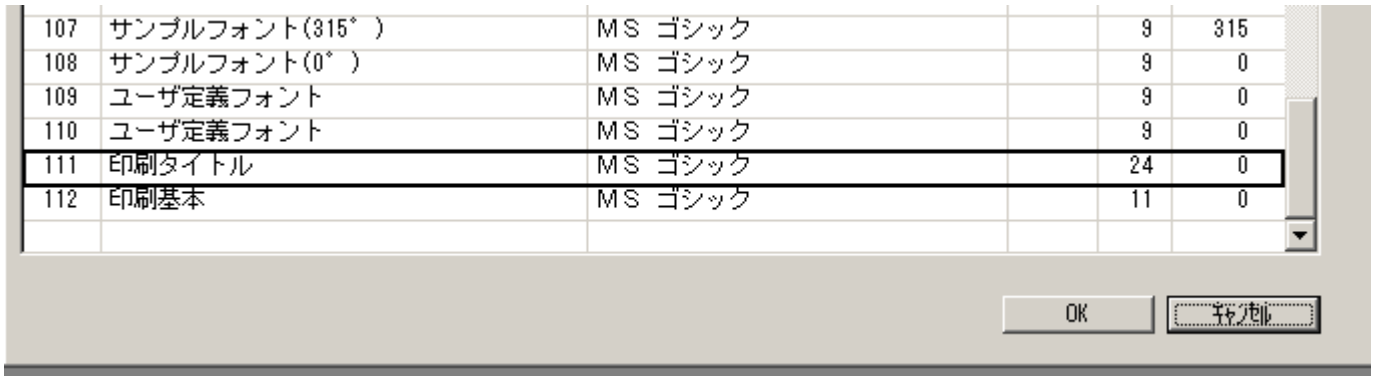
図 6-2 アプリケーションテーブルへの登録



- #4 の「PETSHOP_CTL」が Pervasive 版のスタートポイントとなる MCF で、第 7 章以降では、これを修正していきます。
- #5 の「PETSHOP 最終_CTL」は、本書に従って作成していったときの最終形です。読み進むうちにわからないところがあったら、参考にしてください。

6.1.4. フォントテーブルの追加

ペットショップアプリケーションでは、印刷プログラムでフォント 111 番と 112 番とを参照しています。標準で Magic をインストールした場合には、フォントは 110 番までしか定義されていないので、ここでフォント 111 番と 112 番とを登録してください（下図参照)。111 番は印刷のタイトルですので、少し大きめのフォント(24 ポイントなど)とし、112 番は通常印刷フォントなので、11 ポイントくらいがいいでしょう。



7. データ移行

アプリケーション移行の第一歩として、Pervasive 版の Petshop デモのデータを MS-SQL に移行しましょう。

Magic は各 DBMS ごとに開発された **Magic データベースゲートウェイ**を使うことにより、DBMS ごとの違いをできるだけ吸収するように設計されています。そのため、Magic のテーブルリポジトリにいったん定義されてしまえば、実際の DBMS が Pervasive.SQL であっても MS-SQL Server であっても、あるいはその他の DBMS であっても、Magic プログラムからは同じように参照し利用することができます。そのため、データ移行においても、たとえ異なる DBMS への移行であっても、Magic であれば比較的容易に行うことができます。

Magic を使ったデータ移行には、次のような方法があります。

- Magic のデータ再編成機能を使う
- テキスト出力・入力で移行
- データ移行用のプログラムを作成する

これらについて、本章で説明していきます。

参考： 以下の説明に先立って、「PETSHOP_CTL」アプリケーションを開いておいてください。

7.1. Magic のデータ再編成機能を使う

Magic のデータ再編成機能については、「4.4 データ再編成」(31 ページ)に説明しました。そこではカラム定義を変更する場合について説明しましたが、データ再編成機能はデータベースを変更する場合にも有効になります。つまり、テーブルリポジトリで「データベース」欄を変更することだけで、データの移行が自動的に行われます。

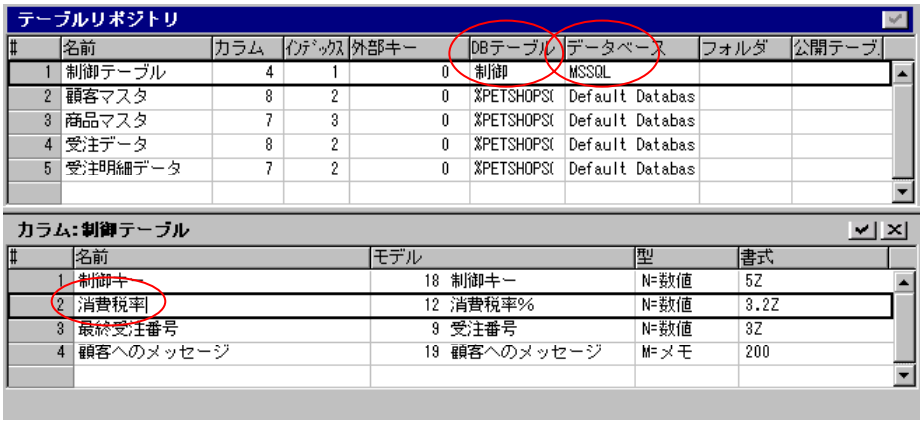
実際には、テーブルやカラムの命名規約など DBMS 固有の制限に基づくいくつかの細かなことを注意する必要がありますが、逆に言えば、それさえ押さえておけば設定一つで移行ができてしまう、というのは、開発時に大変便利な機能といえます。

7.1.1. データ再編成によるデータベース移行の手順

次に、ペットショップデモのテーブルを Pervasive.SQL から MS-SQL Server へ移行する手順を「制御テーブル」を例にして説明します。

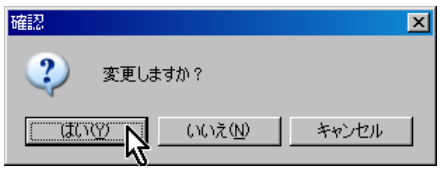
1. テーブルリポジトリを開き、「制御テーブル」の行にカーソルをおきます。
2. Tab キーを押して、カーソルを「カラム」欄に置きます。
3. F5 キーを押して、カラムテーブルに移動します。
4. 各カラムについて、カラム特性の「DB カラム名」が MS-SQL Server の命名規則に反していないかを確認します。

   「制御テーブル」の場合には、「消費税%」カラムの「%」はダメなので、DB カラム名を「消費税率」に変更します。他は OK です。

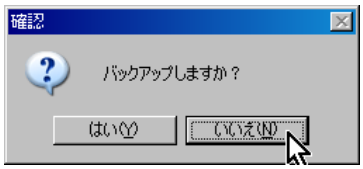


5. Enter キーを一度押して、カーソルを上の「カラム」欄に戻します。
6. Tab キーを 3 回押して、「DB テーブル」欄に移動します。
7. 「DB テーブル」の名前は「%PETSHOPSQL%制御.DAT」ですが、MS-SQL Server の規約にあわせ、単に「制御」とします。
8. Tab キーを押して、「データベース」欄に移動します。
9. 「データベース」欄からズーム(F5 キー)して、データベースを「Default Database」から「MSSQL」に変更します。
10. ここで APG を実行してみます。(あるいは、別のテーブルにカーソルを移動します)。このタイミングで自動データ再編成機能が働きます。
11. 「4.4 データ再編成」に書いたのと同様に、次のような確認ダイアログが出てきます：

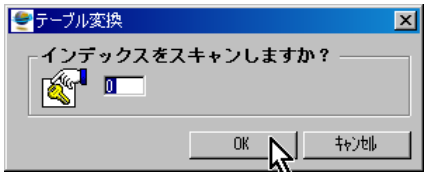
(ア) 「変換しますか？」には「はい(Y)」で答えます。



(イ) 「バックアップしますか？」には、「いいえ(N)」で答えます。(バックアップを取っておきたい場合には、はいで答えてください。)



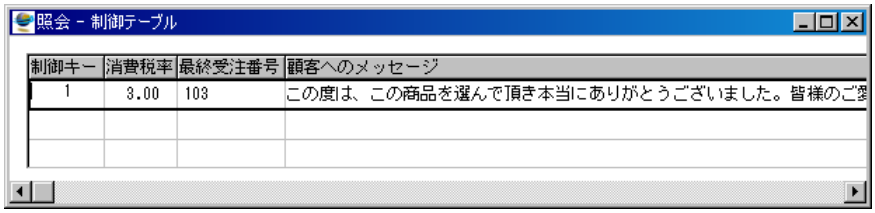
(ウ) 「インデックスをスキャンしますか？」には、そのまま OK ボタンを押します。



以上が完了したら、データは MSSQL に移行されたはずです。

7.1.2. データベースの確認

再度 APG を実行して、テーブルの中身を確認してください。Pervasive.SQLの時とまったく同じようにデータが表示されます。



念のため、OSQL からもテーブルが確認されデータが移行されたことを確認してください。

```
C:¥>osql -E -w 1000
1> use magic
2> select * from 制御
3> go
 制御キー   消費税率 最終受注番号        顧客へのメッセージ
 --------  -------- ------------  -------------------------
        1      3.0         103     0xAA0082B182CC937882CD81418...
(1 件処理されました)
1>
```

注意： 実際には、OSQL の結果は、上の図よりももっと見にくい形で表示されます。上の図は見やすくするために「顧客へのメッセージ」の部分のうしろの方を省略しています。

7.1.3. 「顧客へのメッセージ」のデータ型について

ここで、「顧客へのメッセージ」は、バイナリ 16 進形式で表示されますが、これはこのカラムが Magic 上でメモ型として定義されているためです。メモ型は Magic に固有なタイプなので、MS-SQL Server 上では varbinary 型にマッピングさるため、OSQL で表示するとこのようにバイナリ 16 進形式で表示されます。

「制御」テーブルを Magic だけからアクセスする場合には、このような形式で格納しておいても問題ありませんが、Magic 以外のアプリケーションとデータを共有したい場合には、varbinary 型よりも char 型で格納

しておくほうが適当です。このような場合、一番簡単には、このカラムをメモ型として定義するのではなく、通常の文字型として定義すれば対応できます。また、場合によって Magic がデフォルトで設定する型と異なる型で MS-SQL Server 上で定義したいと思うことがありますが、この方法については本書の範囲を超えてしまうので、リファレンスマニュアルの第 4 章「テーブル」、第 25 章「SQL に関する考慮事項」などを参照してください。

7.1.4. 自動再編成がうまくいかない場合

さて、この自動再編成機能は便利には違いありませんが、ちゃんとできたでしょうか？自動再編成機能は、実際には慎重に操作しなければ失敗しがちです。例えば、まだ修正が全部済んでいないのにマウスクリックを間違えて別のテーブルのエントリをクリックしてしまうと、そこで再編成が始まってしまうので、キャンセルしなくてはなりませんが、この状態は中途半端な状態なので、ここから再度定義を修正しなおして再編成をかけても、うまくいかない場合があります。この場合には、データ（ファイル）と定義を最初に戻して、再度やりなおしをしなければなりません。

慣れてくると、失敗しないやりかたのコツをのみこめてくると思いますが、慣れないうちには、あるいは失敗によるやりなおしをなるべく避けるには、次に説明するデータ出力・入力を使ってデータ移行を行う方が確実です。

7.2. テキスト出力・入力で移行

この方法は、データ再編成機能によるデータベース移行を行うのではなく、

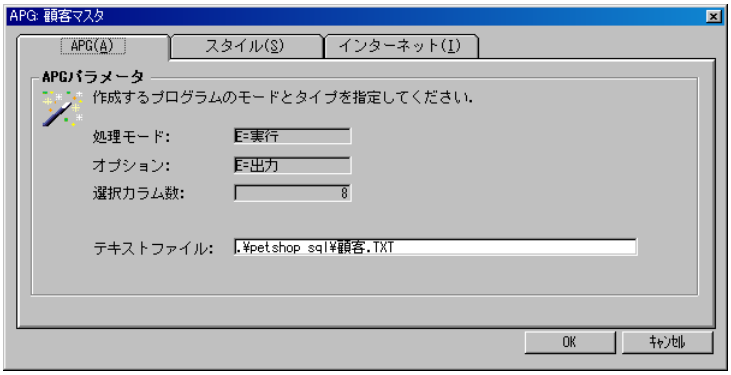
1. 変更前、もとのデータベース(Pervasive.SQL)でデータをテキストファイルで出力する。
2. データベースを変更する。この際、データ再編成機能は実行しない。
3. 変更後、テキストファイルからデータを入力する。

という手順をとります。

7.2.1. テキスト出力・入力によるデータベース移行の手順

ここでは、顧客マスタをテキスト出力・入力方式によりデータベース移行してみましょう。

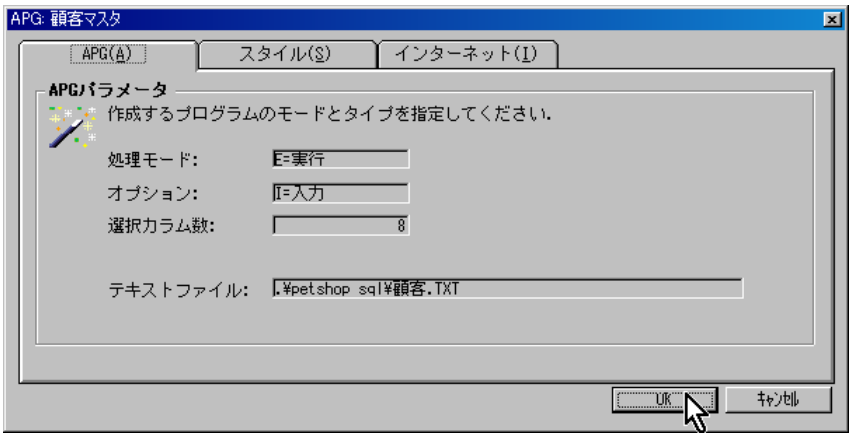
1. テーブルリポジトリを開きます。
2. APG を起動します。(メニュー「オプション(O)」から「APG(G)」を選ぶか、あるいは Ctrl+G キーにより起動します)
3. 「オプション」を「E=出力」として、データを「.¥petshop_sql¥顧客.TXT」というファイルにテキスト出力します。



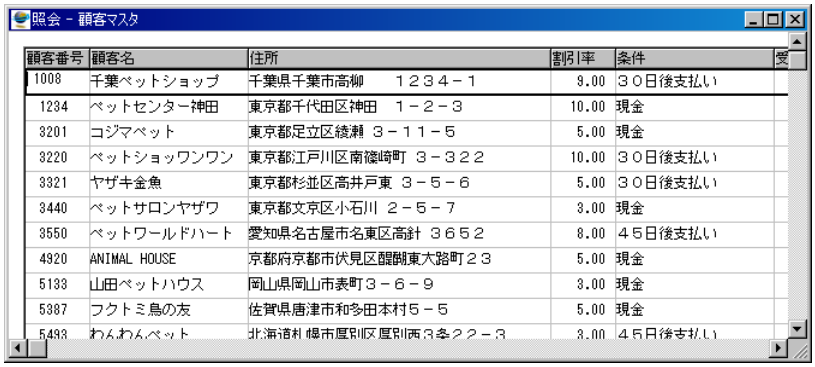
4. 前の例と同様に、カラム名が MS-SQL Server の命名規約に違反していないかをチェックします。顧客マスタの場合にはいずれのカラムも OK です。
5. 「DB テーブル」欄を、Pervasive.SQL での名前「%PETSHOPSQL%顧客.MST」から、MS-SQL Server の命名規約に適合したテーブル名「顧客マスタ」に変更します。

6. 「データベース」欄を「Defautl Database」から「MSSQL」に変更します。ここまでは、前の例と同じです。
7. APG を再度起動します。ここで、データ再編成機能が働き、「変更しますか？」の確認ダイアログが出ますが、今回はデータ再編成機能を使わないので「いいえ(N)」で答えます。
8. APG の起動がキャンセルされてしまうので、再度 APG を起動します。

9. 今度は「オプション」として「I=入力」とし、「テキストファイル」には、先ほどデータ出力したファイル名「.¥petshop_sql¥顧客.TXT」を設定します。
10. OK ボタンを押すと、データが入力されます。



11. データの確認のため、再度 APG を起動してくだしい。今度は「オプション」として「B=照会」とします。データが正しく格納されていることを確認してください。



12. 念のため、OSQL コマンドでもデータを確認してください。(下に示した実行結果で、実際には「条件」以後にもカラムが続きますが、見易さのために省略しています)。

```
c¥> osql -E -w 1000
1> use magic
2> select * from 顧客マスタ
3> go
 顧客番号    顧客名               住所                              割引率          条件
 ---------- ---------------- ------------------------------ -------------- ----------------
       1008 千葉ペットショップ   千葉県千葉市高柳      １２３４－１              9.0 ３０日後支払い
       1234 ペットセンター神田   東京都千代田区神田    １－２－３               10.0 現金
       3201 コジマペット         東京都足立区綾瀬  ３－１１－５                5.0 現金
       3220 ペットショッワンワン 東京都江戸川区南篠崎町  ３－３２２           10.0 ３０日後支払い
       3321 ヤザキ金魚           東京都杉並区高井戸東  ３－５－６              5.0 ３０日後支払い
       3440 ペットサロンヤザワ   東京都文京区小石川  ２－５－７                3.0 現金
   (省略)
(24 件処理されました)
```

以上で、テキスト出力・入力によるデータ移行ができました。

7.3. データ移行用のプログラムを作成する

前節に説明したテキスト出力・入力による方法は、ミスによるやり直し(最悪の場合データ損失)がなくなる、という利点がありますが、手順がワンステップ多くなるので大量のデータがある場合には時間がかる上に、テキストデータ用のディスクスペースが必要になる、という欠点があります。データ量の少ない開発・テスト環境では良いですが、時間的にもハードウェア資源の点でも制約がありがちな実際の運用環境で行うのは問題になることもあります。
そのため、大量のデータをなるべく早く間違えなく移行するためには、移行用のプログラムを作成するのが便利です。移行プログラムと言っても、単にデータをコピーするだけのバッチプログラムですから、Magic を使えば簡単に作成できます。このようにしておけば、テーブル数が多い場合にも、ひとつづつ手作業でする必要がなくなります。

また、データ移行/入力用のバッチタスクを作っておくことは、テスト時にも有効です。テスト時には、ある操作を行った前後のデータ値が正しいかを確認することが必要になりますが、常にデータを初期状態に復元できるようにプログラムをひとつ作っておくと確認が容易になります。
サンプルアプリケーションでは、テスト用にデータ入力バッチタスクが作成されています。プログラム 34 番の「データ入力(受注は空)」というもので、これを実行すると、制御テーブル、顧客・商品マスタはテキストファイルから入力して初期状態になり、受注および受注明細データは空になります。このプログラムは、受注入力の動作確認のために後でよく使います。

では、いずれの方法でも構いませんので、ペットショップアプリケーションのテーブルリポジトリをすべて MSSQL 用に移行してください。


テーブルリポジトリ

#	名前	カラム	インデックス	外部キー	DBテーブル	データベース	フォルダ	公開テーブル
1	制御テーブル	4	1	0	制御	MSSQL		
2	顧客マスタ	8	2	0	顧客マスタ	MSSQL		
3	商品マスタ	7	3	0	商品マスタ	MSSQL		
4	受注データ	8	2	0	受注	MSSQL		
5	受注明細データ	7	2	0	受注明細	MSSQL		

カラム:顧客マスタ

#	名前	モデル	型	書式
1	顧客番号	1 顧客番号	N=数値	5Z
2	顧客名	2 顧客名	A=文字	20
3	住所	3 住所	A=文字	40
4	割引率	13 その他%	N=数値	N3.2Z
5	条件	4 条件	A=文字	20
6	受注累計額	16 金額（8桁）	N=数値	N8CZ
7	取引回数	15 注文／取引回数	N=数値	N5CZ
8	顧客メモ	5 顧客メモ	M=メモ	200

8. 移行直後の動作確認

前節でテーブルの移行は終了しましたので、次にプログラムの変更と動作確認に進みましょう。

ここでは、最初にトランザクションの変更を行います。Pervasive.SQL を使ったペットショップアプリケーションではトランザクションを使っていなかったので、トランザクションの設定に気を使う必要はありませんでしたが、MS-SQL Server などのリレーショナル DBMS を使う場合には、必ずトランザクションの設定が必要になります。この違いに起因して、プログラムもいろいろ変更する必要が出てきます。詳しいことは「11 章 受注入力プログラムの変更」(76 ページ)で説明します。

今の時点でまずしておかなければならないことは、「遅延トランザクション」の設定を「物理トランザクション」に変更しておくことです。遅延トランザクションというのは、Magic が内部で独自に実装しているトランザクション機能で、大変高度で便利な機能なのですが、使いこなすにはトランザクションについてよく理解していなければならないため、本書では使いません。物理トランザクションというのは、MS-SQL Server などの DBMS がサポートしているトランザクションのことで、本書ではこちらのトランザクションのみを使います。設定の変更については、「8.1 トランザクション設定の変更」(57 ページ)に解説します。

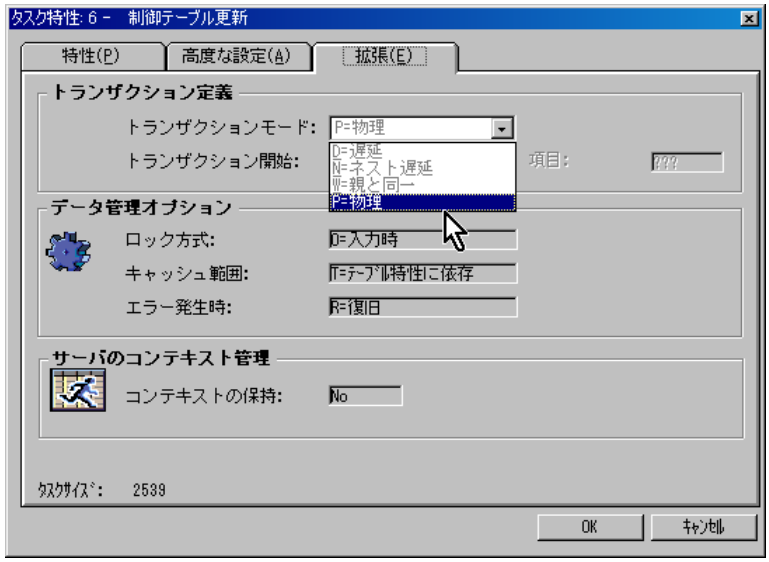
次に、「8.3 マルチユーザ環境での問題点」(59 ページ)で、マルチユーザ環境で移行直後のアプリケーションを実行してみて、どのような問題が起こるかを見ていきます。

8.1. トランザクション設定の変更

オンラインプログラムでは、デフォルトの「トランザクションモード」が「D=遅延」となっています。これは遅延トランザクションを意味します。
本書では遅延トランザクションを使わないので、アプリケーションのオンラインプログラムのトランザクションモードの設定をすべて「P=物理」に変更してください。

トランザクションモードの設定は、次のように行います。

1. プログラムリポジトリを開き、変更したいプログラムにカーソルを置きます。
2. F5 キーでプログラムを開きます。
3. タスク特性ダイアログを開きます。タスク特性ダイアログは、右マウスクリックでポップアップメニューを開き「特性(P)」を選ぶか、あるいは Ctrl+P キーにより開きます。
4. 「拡張(E)」タブを開きます。
5. 「トランザクションモード」が「D=遅延」になっていたら、「P=物理」に変更してください。



アプリケーション中のすべてのオンラインタスクについて、トランザクションモードを確認し、D=遅延になっていたら P=物理に変更してください。

参考：

- 受注入力プログラムのように、サブタスクのある場合には、サブタスクのトランザクションモードは「W=親と同一」にしてください。
- バッチタスクでは、デフォルトでトランザクションモードが「P=物理」になっているので、変更する必要はありません。

参考： V9Plus 9.40JSP3 より、MAGIC.INI ファイルの [MAGIC_SPECIALS] セクションに以下の設定を行うことにより、トランザクションモードのデフォルト設定が「P=物理」になるようになりました。遅延トランザクションを使わない場合には、いちいち「D=遅延」から「P=物理」に変更する必要がなくなります。本書でもこの設定を行った MAGIC.INI を使います。

```
[MAGIC_SPECIALS]
SpecialDefaultTransactionMode = P
```

8.2. マルチユーザ環境での実行

マルチユーザ環境での動作テストを行うには、複数の Magic セッションで同一アプリケーションを開き、同一データベースをアクセスすることが必要です。実際の運用環境では、通常データベースサーバがあって、各ユーザは自分のパソコンに Magic クライアントをインストールし、データベースを共用アクセスする、という形になります。しかし、開発時には自分のパソコン上にデータベースがあり、Magic のクライアント版を二つ起動することにより、マルチユーザ環境を模擬的に作成してテストすることができます。

8.2.1. Magic クライアント版

Magic クライアント版というのは、Magic アプリケーションを実行する機能のみ持っているプログラムであり、アプリケーションを開発・修正する開発機能は持っていません。実際の運用の場では、Magic クライアント版を使うことになります。Magic クライアント版は、Magic をインストールしたディレクトリにある mgrntw.exe というプログラムです。
一方、アプリケーションを開発・修正する開発版は、mggenw.exe という名前のプログラムです。Magic eDeveloper 製品や体験版をインストールすると、デスクトップに Magic のアイコンが登録されますが、これにより起動されるのは開発版です。開発版は、アプリケーションを開発・修正する機能と、実行する機能とを両方備えています。

> **参考：** Magic クライアント版 mgrntw.exe はデスクトップやメニューに登録がされないので、エクスプローラなどから自分でショートカットを作成してください。

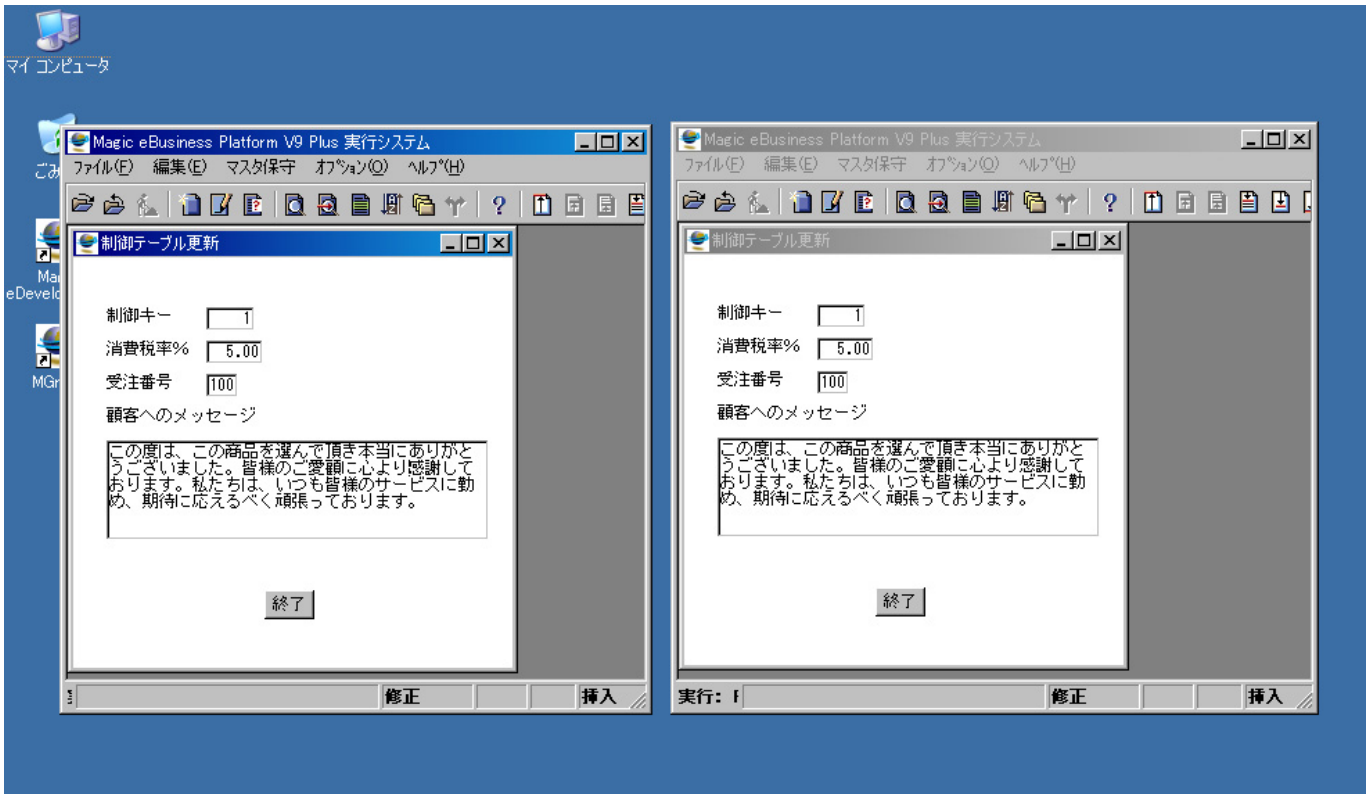
8.2.2. アプリケーションの同時オープン

マルチユーザ環境のテストを行うには、同一アプリケーションを二つ以上の Magic セッションで開いて実行させることが必要です。
同一アプリケーションを同時に開きたい場合には、Magic クライアント版を使います。Magic クライアント版はアプリケーションの修正を行わないので、アプリケーションファイル(MCF ファイル)を共有ロックをかけて、読込専用で開きます。共有ロックなので、複数の Magic セッションが同時にオープンすることが可能です。
一方、開発版では、アプリケーションを修正するためにアプリケーションファイルに排他ロックをかけます。このため、開発版で開いているアプリケーションは、他の開発版、あるいはクライアント版で開くことができません。
従って、マルチユーザのテストを行う場合には、開発版ではアプリケーションを閉じ、実行版で開きます。逆に、プログラムの修正を行いたい場合には、開発版で開く前に、実行版でアプリケーションを閉じる必要があります。
以下、本章では、実行版を二つ同時に起動し、ペットショップアプリケーションを実行していきます。

8.3. マルチユーザ環境での問題点

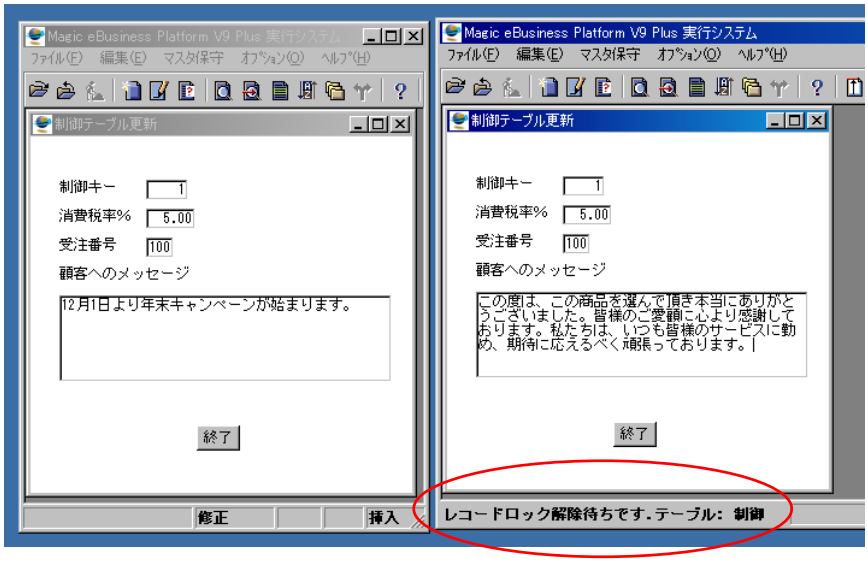
8.3.1. テーブルを一つだけ使うタスクの場合

最初に、テーブルを一つだけ使う単純なタスクについて、動作確認しましょう。
ここでの例としては、「制御テーブル更新」プログラムを使います。
下図は、クライアント版を二つ起動し、それぞれで制御テーブル更新プログラムを開いたところです。



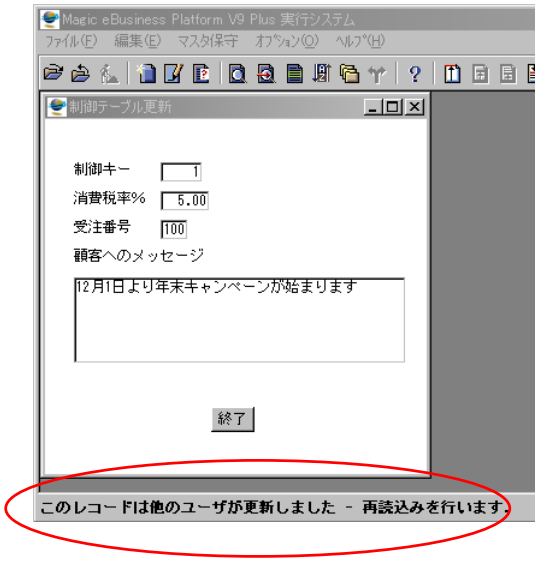
マルチユーザ環境で重要な確認事項は、ロックが適切にかかっているかということと、データの損失が起こらない、ということです。

1. 左側の Magic（二人のユーザになぞらえて、ユーザ A と呼びます）で、「顧客へのメッセージ」を適当に修正します。
2. 右の Magic(ユーザ B と呼びます)で、同じように「顧客へのメッセージ」を変更しようとすると、「レコードロック解除待ち」が出ます。



このことから、ロックは正しくかかっていることがわかります。

3. ユーザ A の方を、ESC キーで終了させます。すると、ロックが解除されるので、ユーザ B が再開されます。しかし、レコードはユーザ A により変更されているので、「このレコードは他のユーザが更新しました – 再読込を行います」というエラーがでて、レコードのデータが最新のものに変更されます。

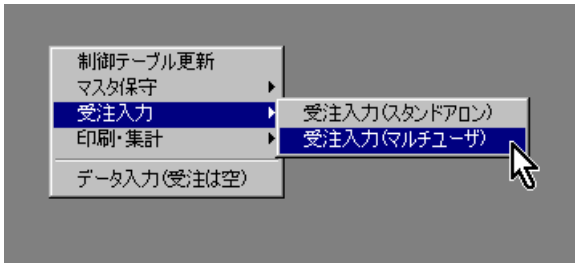


以上の動作は、マルチユーザ環境での排他制御の動作として適切です。従って、テーブル一つだけをアクセスする単純なプログラムでは、テーブルを Pervasive.SQL から MS-SQL Server に移行しても特に問題は見られない、ということになります。

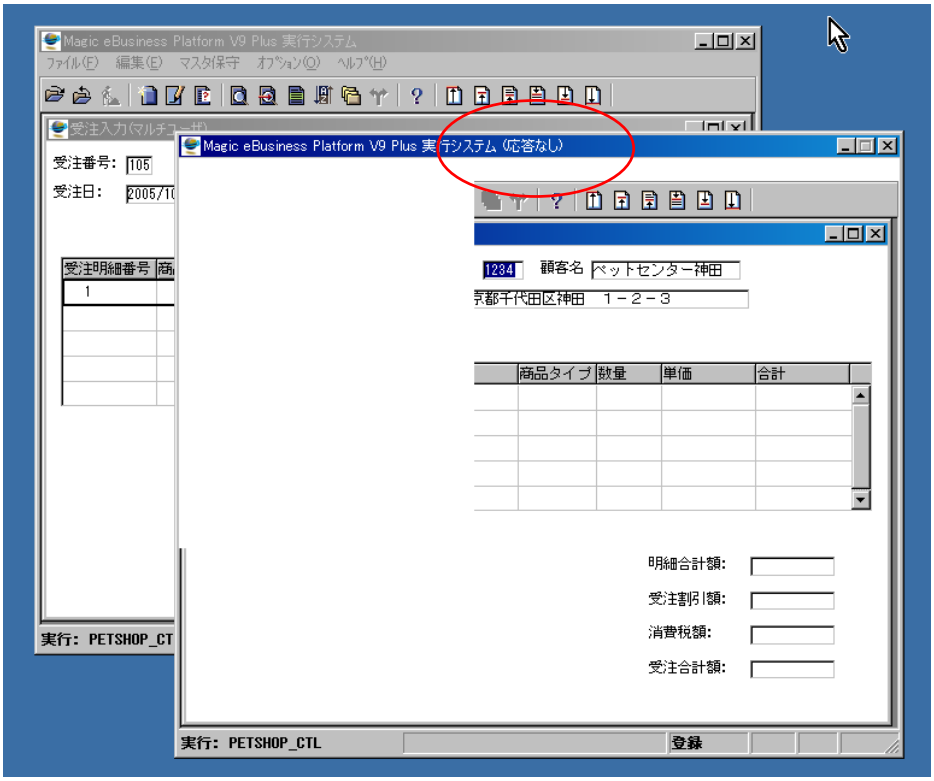
他のマスターメンテナンスプログラム(顧客マスター更新、商品マスター更新)でも同様に、MS-SQL Server に移行しても特に問題はありません。

8.3.2. テーブルを複数使うタスクの場合

次には、もう少し複雑な受注入力プログラムについて確認してみましょう。



1. 二つの Magic クライアント版を起動し、ペットショップアプリケーションを開きます。以下、それぞれを「ユーザ A」、「ユーザ B」と呼びます。
2. それぞれで、ポップアップメニューから、「受注入力」→「受注入力(マルチユーザ)」を選択します。これは、Pervasive.SQL において、マルチユーザ環境に対応した受注入力プログラムで、Pervasive.SQL をデータベースに使っているときにはマルチユーザ環境下でも正しく動作していました。



3. ユーザ A の方で、顧客番号 #1008 を選択します。カーソルは子タスクの最初の行に移動します。
4. ユーザ B の方で、顧客番号 #1234 を選択します。すると、カーソルは子タスクには行かず、しかもユーザ B の方の Magic は「応答なし」になってしまいます。
5. ユーザ A に戻り、商品番号 #1002 (プードル)を一つ注文し、ESC を 2 回押して、受注入力プログラムを終了します。
6. ユーザ A の方でタスクが終了した瞬間、ユーザ B が動き出します。

このように、受注入力プログラムでは、データベースを MS-SQL Server に移行すると、今までは正しく動いていたプログラムが動かなくなってしまいます。

このようなことの起こる原因は、MS-SQL Server のトランザクションのしわざによるものです。これを理解するために、まず次章「トランザクション」でトランザクションを基本から簡単に解説します。その後、第 10 章「受注入力で何が起こっているのか？」(73 ページ)で上のような現象の起こる理由を解析し、第 11 章「受注入力プログラムの変更」(76 ページ)で、トランザクションにも対応させるために受注入力プログラムをどう修正するかを説明します。

9. トランザクション

前章で見たような、マルチユーザ環境での「応答なし」に対応するためにはプログラムを少し変更する必要がありますが、それに先立って、まずは MS-SQL Server で何が起こっているために「応答なし」になるかを正確に理解しておく必要があります。
そのために、まずトランザクションの基本について本章で簡単に説明します。

参考：

- トランザクションとロックの概念についてすでによく理解している読者は、本章をスキップして「10 受注入力で何が起こっているのか？」(73 ページ)に進んでください。
- 本章では、以後の説明に必要な最低限の事柄しか解説しません。トランザクションについてはデータベースの解説書籍などに必ず載っているので、詳しいことは書籍などを読んでください。

9.1. トランザクションとは？

トランザクションというのは、データの整合性を保証するための DBMS のメカニズムで、MS-SQL Server、Oracle、DB2/UDB などのリレーショナルデータベース管理システムでは例外なくサポートされています。

9.1.1. トランザクションの定義

トランザクションとは「全体としてコミットされるかアボートされなければならない、一連のデータ修正処理から成る作業単位」と定義されます。ここで、

- 「一連のデータ修正処理」というのは、リレーショナルデータベースでは、SELECT 文、UPDATE 文、INSERT 文、DELETE 文などの DML です。この DML 文による修正内容は、すぐにはデータベースに反映されません。
- 「コミット」というのは、修正内容ををすべてデータベースに反映してトランザクションを終了することです。
- 「ロールバック」というのは、修正内容をすべて破棄し、データベースは一切変更せずにトランザクションを終了することです。

トランザクションは、処理全体として成功(コミット)するか失敗(ロールバック)するかのどちらかとなります。中途半端はありません。

9.1.2. トランザクションの例

トランザクションについてよく出てくる例は、銀行の振込みの例です。例えば、A さんが B さんに 5000 円振り込みをすることを考えて見ます。データベースのレベルでは、振込みの作業は次のような処理に分解されます。

1. A さんの口座の残金を読み込む。(SELECT 文)
2. A さんの口座の残金が十分ならば、5000 円を引く。(UPDATE 文)
3. B さんの口座の残金お読み込む。(SELECT 文)
4. B さんの口座に 5000 円を加える。(UPDATE 文)

トランザクションを使わない場合、ちょうど処理 2 と 3 の間でマシントラブルが起こり、データベースが停止してしまったとします。この状態では、A さんは 5000 円引かれているが、B さんには 5000 円加えられていません。したがって、A さんは振込みしたつもりだが B さんの口座には入っていない、ということになります。これはデータ整合性が壊れてしまうことになります。

このようなことを防ぐために、上の 1 から 4 までの処理をトランザクションで囲みます。4 まですべて成功したらトランザクションはコミットし、途中で何か問題が起こったらロールバックします。コミットするまでは、データベースには一切の変更が加えられません。データベースがなんらかの理由で途中で停止してしまった場合には、処理途中のトランザクションはすべてロールバックされ破棄されます。従って、上の例のように、処理 2 と 3 の間で障害が起こっても、途中の変更内容(A さんの口座から 5000 円を出したこと)はロールバックされ、障害から回復した時点では振込み前の状態に戻され、データの整合性が保たれることになります。

9.2. OSQL を使ってトランザクションを試してみる

OSQL コマンドを使って、実際にトランザクションを試して見ましょう。例としては、「第 5 章 定義取得」(38 ページ)で作った、簡単な商品テーブルを使って見ます。

最初に、データの初期状態を見てみます。

```
C:¥>osql -E
1> use magic
2> select * from 商品
3> go
 商品番号          商品名                単価      有効
 ----------- ---------------- ----- -------- ----
        5023 ウマカコーヒー                500      1
        5055 山梨観光地割引切符          23500      1
        6034 加湿器 SVKR-703             13200      1
(3 件処理されました)
```

次に、トランザクションを使ってデータを変更してみます。トランザクションの最後はコミットします。

```
1> begin transaction
2> update 商品 set 単価 = 600 where 商品番号 = 5023
3> update 商品 set 単価 = 24000 where 商品番号 = 5055
4> commit work
5> go
(1 件処理されました)
(1 件処理されました)
1> select * from 商品
2> go
 商品番号          商品名                単価      有効
 ----------- -------------------- -------- ----
        5023 ウマカコーヒー                600      1
        5055 山梨観光地割引切符          24000      1
        6034 加湿器 SVKR-703             13200      1
(3 件処理されました)
```

上に示したように、トランザクションは begin transaction で始まり、commit work でコミットします。コミットしたら、update 文により修正した内容はデータベースに反映されます。

次には、コミットせずにロールバックします。

```
1> begin transaction
2> update 商品 set 単価 = 700 where 商品番号 = 5023
3> update 商品 set 単価 = 25000 where 商品番号 = 5055
4> rollback work
```

```
5> go
(1 件処理されました)
(1 件処理されました)
1> select * from 商品
2> go
 商品番号        商品名                 単価      有効
 ----------- -------------------- -------- ----
        5023 ウマカコーヒー              600     1
        5055 山梨観光地割引切符        24000     1
        6034 加湿器 SVKR-703           13200     1
(3 件処理されました)
```

この場合には、二つの update 文で行ったデータ修正の内容は両方とも破棄され、データベースはトランザクションの前の状態に戻っています。

このように begin transaction ･･･ commit/rollback work を使うことにより、複数の DML 文の処理内容をいっぺんに反映あるいは破棄させることができることがわかります。

9.3. 分離レベル

次に知っておく必要があるのが、マルチユーザ環境でのトランザクションの振る舞いについてです。

今、ユーザ A と B とが同一のデータベースを共有しているとします。ユーザ A がトランザクション実行中に UPDATE 文でレコードを更新し、まだコミットあるいはロールバックしていない状態の時に、ユーザ B が同じレコードを SELECT 文で読み込もうとしたらどうなるでしょうか？
これはトランザクションの**分離レベル**の設定により結果が異なってきます。分離レベルというのは、異なるユーザによるトランザクションが、どれだけ分離されているか、という度合いを表すものです。
分離レベルは、DBMS により実装レベルが異なりますが、MS-SQL Server では、分離度の低い順に、次の四つが定義されています。

- 非コミット読込（Read Uncommited）
- コミット済み読み取り(Cursor Stability、あるいは Read Commited)
- 反復可能読み取り（Repeatable Read）
- 直列化（Serializable）

分離レベルが非コミット読込に設定されている場合には、ユーザ A が変更した修正内容が、たとえコミット前であっても、ユーザ B から見ることができます。しかし、コミット前の修正内容は、ロールバックにより取り消されてしまう可能性のある不確定なものなので、ダーティ・リードと呼ばれます。トランザクションがロールバックされる可能性を考えると、非コミット読込のレベルはデータの一貫性で問題があります。

分離レベルがコミット済み読み取りに設定されている場合には、ユーザ A が変更した修正内容は、コミットされるまで、ユーザ B から見ることができません。具体的には、ユーザ A の修正したレコードにはロックがかけられるので、ユーザ B がこのレコードを対象とする SELECT 文を発行するとロック待ちとなります。ロックは、ユーザ A がトランザクションを終了するまで続きます。トランザクションがコミットあるいはロールバックされた時点で、ロックは解除され、変更された値(コミットの場合)、あるいは変更前の値(ロールバック)が読み込まれます。カーソル固定の分離レベルでは、非コミット読込の場合と比べ、ロック待ちが発生するため同時並行性は減りますが、未確定のデータを読み込む危険性がなく、データの一貫性では望ましい設定といえます。

反復可能読み取りと直列化については、ここでは説明を省略します。これらの分離レベルは、より高度なデータ一貫性を保証しますが、ロックの衝突が起こりやすく並列度が減少します。

9.4. OSQL を使って分離レベルを試してみる

OSQL を使って、試してみましょう。ここでも「商品」テーブルを例に使います。

複数ユーザでのアクセスを想定しているので、コマンドプロンプトを二つ開き、それぞれで OSQL を起動します。それぞれを二人のユーザと見立てて、ユーザ A、ユーザ B と呼びます。

最初に商品テーブルの内容を見ておきます。

```
c:¥> osql -E
1> use magic
2> select * from 商品
3> go
 商品番号          商品名                 単価          有効
 ----------- -------------------- ----------- ----
        5023 ウマカコーヒー                   600     1
        5055 山梨観光地割引切符             24000     1
        6034 加湿器 SVKR-703                13200     1
(3 件処理されました)
```

9.4.1. 非コミット読込の場合

ユーザ A、B の双方で、分離レベルを非コミット読込(read uncommitted)に設定します。以下のコマンドを、それぞれの OSQL で実行してください。

```
（ユーザ A、ユーザ B の双方で実行）
1> set transaction isolation level read uncommitted
2> go
```

ユーザ A でトランザクションを開始、商品単価を変更します。

```
(ユーザ A での操作)
1> begin transaction
2> update 商品 set 単価 = 700 where 商品番号 = 5023
3> update 商品 set 単価 = 25000 where 商品番号 = 5055
4> go
(1 件処理されました)
(1 件処理されました)
1> select * from 商品
2> go
 商品番号          商品名                 単価          有効
 ----------- -------------------- ----------- ----
        5023 ウマカコーヒー                   700     1
        5055 山梨観光地割引切符             25000     1
        6034 加湿器 SVKR-703                13200     1
```

```
(3 件処理されました)
```

この時点ではまだコミットしていませんので、データベースにこの修正は反映されていません。
ここでユーザ B が商品の内容を SELECT 文で見てみます。

```
(ユーザ B での操作)
1> select * from 商品
2> go
 商品番号      商品名                  単価          有効
 ----------- -------------------- ----------- ----
        5023 ウマカコーヒー                 700    1
        5055 山梨観光地割引切符             25000    1
        6034 加湿器 SVKR-703                13200    1
(3 件処理されました)
```

ユーザ A がトランザクションをコミットしていないにも関わらず、変更内容が見えてしまっています。これが「非コミット読込」分離レベルの効果です。

では、ユーザ A がトランザクションをロールバックします。

```
(ユーザ A での操作)
1> rollback work
2> go
```

これで、データ変更は取り消されます。ここで再度ユーザ B から商品テーブルを見てみると、どうなるでしょうか？

```
(ユーザ B での操作)
1> select * from 商品
2> go
 商品番号      商品名                  単価          有効
 ----------- -------------------- ----------- ----
        5023 ウマカコーヒー                 600    1
        5055 山梨観光地割引切符             24000    1
        6034 加湿器 SVKR-703                13200    1
(3 件処理されました)
```

当然ながら、データ変更は取り消されて、トランザクション前の値となります。

9.4.2. コミット済み読み取りの場合

コミット済み読み取りの場合、他のユーザがコミットしていない修正内容は、他のユーザから見ることができません。

まず、ユーザ A とユーザ B の双方で、分離レベルをコミット済み読み取り(read committed)に設定します。

```
（ユーザ A、ユーザ B の双方で実行）
1> set transaction isolation level read committed
2> go
```

ユーザ A がトランザクションを開始し、商品の単価を更新します。まだコミットはしません。

```
（ユーザ A での操作）
1> begin transaction
2> update 商品 set 単価 = 700 where 商品番号 = 5023
3> update 商品 set 単価 = 25000 where 商品番号 = 5055
4> select * from 商品
5> go
(1 件処理されました)
(1 件処理されました)
商品番号    商品名                  単価        有効
----------- -------------------- ----------- ----
       5023 ウマカコーヒー                  700    1
       5055 山梨観光地割引切符            25000    1
       6034 加湿器 SVKR-703               13200    1
(3 件処理されました)
```

ここで、ユーザ B で商品テーブルを見てみます。

```
（ユーザ B での操作）
1> select * from 商品
2> go
（・・・反応なし・・・）
```

SELECT 文を実行すると、結果は表示されず、待ち状態になっています。これが分離レベルがコミット済み読み取りの場合の効果です。すなわち、商品テーブルのレコード(#5023 と#5055)が更新され、ロックがかかっているため、他のユーザが見れないようになっています。

ここで、ユーザ A がトランザクションをロールバックします。

```
（ユーザ A での操作）
1> rollback work
2> go
```

するとユーザ B の方のロック待ちが解除され、SELECT 文の結果が表示されます。

```
（ユーザ B での操作）
1> select * from 商品
2> go
```

```
商品番号       商品名                  単価          有効
---------- -------------------- ---------- ----
      5023 ウマカコーヒー                600     1
      5055 山梨観光地割引切符          24000     1
      6034 加湿器 SVKR-703           13200     1
(3 件処理されました)
```

この場合、ユーザ A はロールバックしたので、商品テーブルの内容はトランザクション前の値と同じです。

ユーザ A がロールバックではなく、コミットした場合には、当然のことながら、ユーザ B の方では UPDATE による修正が反映された結果が表示されます。

このように、コミット済み読み取りの場合には、不確定なデータが読み込まれる可能性がないけれども、ロック待ちが起こるので、アプリケーションの設計にあたっては、ユーザがロック待ちで長い間待たされることがないよう、ロックの衝突に注意して設計する必要があります。

9.5. ロックとトランザクション

最後にトランザクションに関連して理解しておかなければならないことは、ロックとトランザクションの関連です。

9.5.1. MS-SQL Server でのロック

MS-SQL Server ではいろいろなレベルのロック機能を備えていますが、Magic でよく利用されるのは、次のレコードロックです。

- UPDATE によるロック
- UPDLOCK 付き SELECT 文によるロック

UPDATE によるロックについては、前節の分離レベルのところで説明しました。UPDLOCK 付き SELECT 文によるロックは、更新することを前提としてレコードを読み込む場合、SELECT 文に(UPDLOCK)というヒントを付けて実行することにより、そのレコードにロックを掛けるものです。これにより、他のユーザが同一レコードをロックしたり更新したりすることができなくなります。
Magic のロックメカニズムでは、オンラインタスクでユーザがキー入力をしたときに、そのレコードがロックされます。例えば、制御テーブルの場合には次のような SELECT 文が発行されます。

```
SELECT 制御キー,消費税率,最終受注番号,顧客へのメッセージ
FROM MAGIC..制御 (UPDLOCK NOWAIT)
WHERE 制御キー = 1
```

9.5.2. ロックとトランザクションの関係

では、このロックとトランザクションとは、どのような関係にあるのでしょうか？
結論から言うと、トランザクションを使わずとも済む Pervasive と対照して、次のようになります。

	Pervasive	MS-SQL Server (他 RDBMS も同じ)
トランザクションの利用	任意（使わない場合が多い）	必須
ロックを掛けるタイミング	ファイルがオープンされていれば、任意の時点でかけられる。	トランザクション中であることが必要。トランザクション中ならば任意の時点でかけられる。

ロックを解除するタイミング	任意の時点でアンロックできる。	トランザクションの終了時(ロールバックあるいはコミット)にのみ、すべてのロックが一括して解除される。任意の時点で解除することはできない。

注意： ここで Pervasive と書いているのは、Magic が利用するトランザクショナルアクセスを使った場合のことです。Pervasive にはこの他にリレーショナルアクセスという SQL ベースのアクセス方式があり、このインターフェースを使った場合には MS-SQL Server と同じになります。Magic からリレーショナルアクセスのインターフェースを使って Pervasive のテーブルをアクセスするには、ODBC ゲートウェイ(現在ベータ版)を使いますが、本書では説明を省略します。

上記の性質は、MS-SQL Server に限らず、トランザクションをサポートするすべての DBMS で共通なもので、トランザクションの本質的な機能に由来する性質です。

ここからわかることは、「トランザクション中のロックは累積される」ということです。すなわち、Pervasive を使っていた場合には、必要な時点でロックをかけ、必要がなくなったらロックを解除することができるので、ロックの期間は必要最低限にすることができます。ところが、MS-SQL Server のようなリレーショナル DBMS の場合には、ロックに先立ってトランザクションを開始しなければならず、しかもロックの解除は任意の時点で行うことができなくて、トランザクションの終了時に一括して行われます。このため、トランザクションの開始から終了までの期間をよく検討して設計しなければ、非常に長い間レコードロックがかかったままになってしまう可能性があります。

「8.3.2 テーブルを複数使うタスクの場合」(61 ページ)で見たように、Pervasive から MS-SQL Server に行こうしたての状態で、受注入力プログラムでロック待ちが発生してユーザ B が操作できなくなってしまいましたが、これもトランザクションによってロックが長引いてしまったためです。

それでは、受注入力プログラムで内部ではどのようなことが起こっていたのかを、次章でトランザクションとロックの観点から分析してみましょう。

10. 受注入力で何が起こっているのか？

「8.3.2 テーブルを複数使うタスクの場合」(61 ページ)で見たように、Pervasive から MS-SQL Server に行こうしたての状態で、受注入力プログラムでロック待ちが発生してユーザ B が操作できなくなってしまいました。これが何故起こったのかを、前章で習ったトランザクションの知識をもとにして解析していきましょう。

受注入力を行った場合には、受注レコード、受注明細レコードのほかに、次のレコードがロックの対象となります。

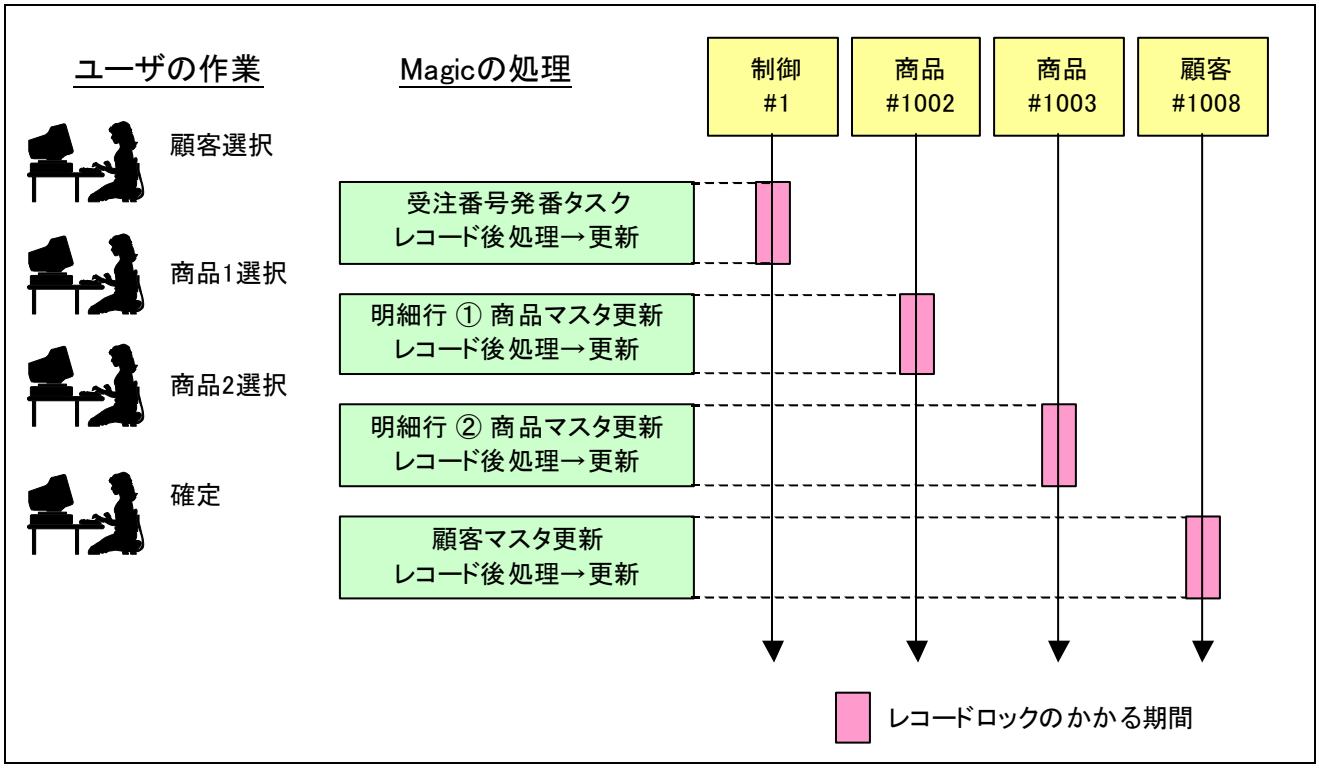
テーブル名	ロック対象レコード	ロックの理由
制御テーブル	「キー」＝ 1 のレコード	新しい受注番号を発番するたびに「最終受注番号」を更新するためにロックします。
商品マスタ	各明細行で選択されている商品のレコード	商品マスタの「受注数」を更新するためにロックします。
顧客マスタ	発注者として選択されている顧客のレコード	受注入力を確定してデータベースに格納するときに、顧客マスタの「累積受注額」と「受注回数」を更新するためにロックします。

まず最初に、Pervasive を DBMS として使ったとき、トランザクションを使わないで処理していましたが、このときのこれらのレコードのロックの状況を「10.1 Pervasive の場合」で見ていきます。次に、MS-SQL Server に移行した場合に同じ操作をするとロックがどうなるのかを「10.2 MS-SQL Server の場合」で見ていきます。

10.1. Pervasive の場合

Pervasive.SQL を DBMS として利用した場合、トランザクションの利用は任意であり、ペットショップアプリケーションではトランザクションを全く利用しませんでした。
トランザクションを利用しなかった場合には、ロックもロック解除も、任意のタイミングで掛けることができました。このため、ロックをかける期間はは必要最小限に留めることができます。
それぞれのレコードのロック期間を図で示したのが図 3-1 です。
図に示したように、Pervasive の場合には、任意のタイミングでロックを解除できるので、更新処理が終わると直ちにロックは解除されます。これらのロックは、すべて極めて短時間に終了するバッチタスクで実行するものなので、ロックの期間は必要最小限に短いものとなり、マルチユーザ環境であってもロックの衝突の可能性は非常に低くなります。

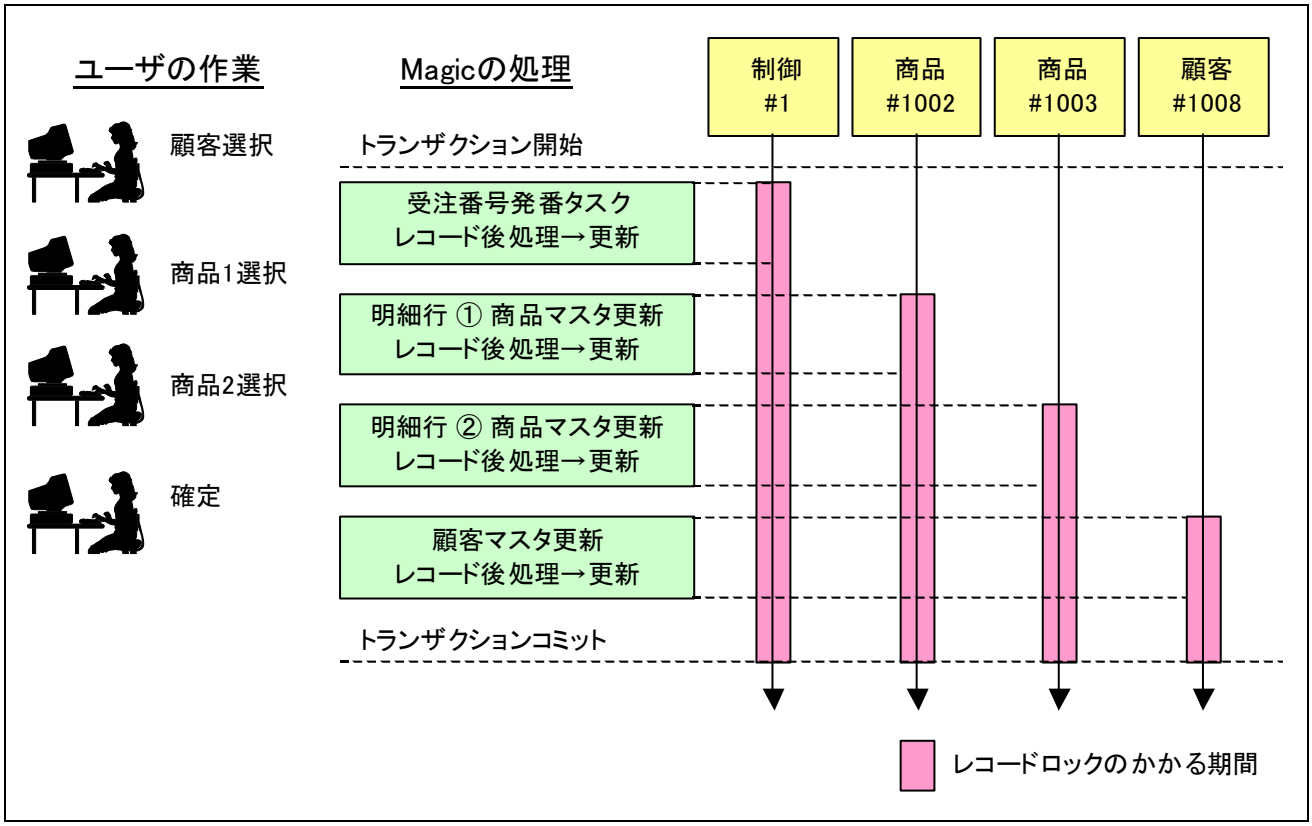
図 10-1 Pervasive.SQL の場合のレコードロックの期間



10.2. MS-SQL Server の場合

MS-SQL Server を DBMS として使う場合には、事情が異なってきます。MS-SQL Server では、トランザクションの使用が必須であり、また、トランザクションの一般的な性質として、ロックされたレコードはトランザクションが終了するまでロック解除できない、という条件があります。
このため、同じように受注入力操作を行った場合には、レコードのロックの状況は図 10-2 のようになります。
このように MS-SQL Server の場合にはロックが累積され、その間にオンラインのユーザ入力待ちが入りますので、レコードは非常に長い間ロックされたままになる可能性があります。
もし、2 人のユーザ(ユーザ A とユーザ B)が同時に受注入力を行うとどうなるでしょうか？最初にユーザ A が入力し、受注番号発番タスクを実行すると、そこで制御テーブルのレコードがロックされてしまいます。このレコードはシステム中に一つだけしかないものなので、これがロックされると、他のユーザが一切先に進めなくなってしまいます。このためユーザ B はユーザ A のロック解除待ち、すなわち、一連の入力が終わってトランザクションがコミットされるまで待たされることとなります。これが、「8.3.2 テーブルを複数使うタスクの場合」(61 ページ)で起こった現象の正体です。

図 10-2 MS-SQL Server の場合のロックの期間



11. 受注入力プログラムの変更

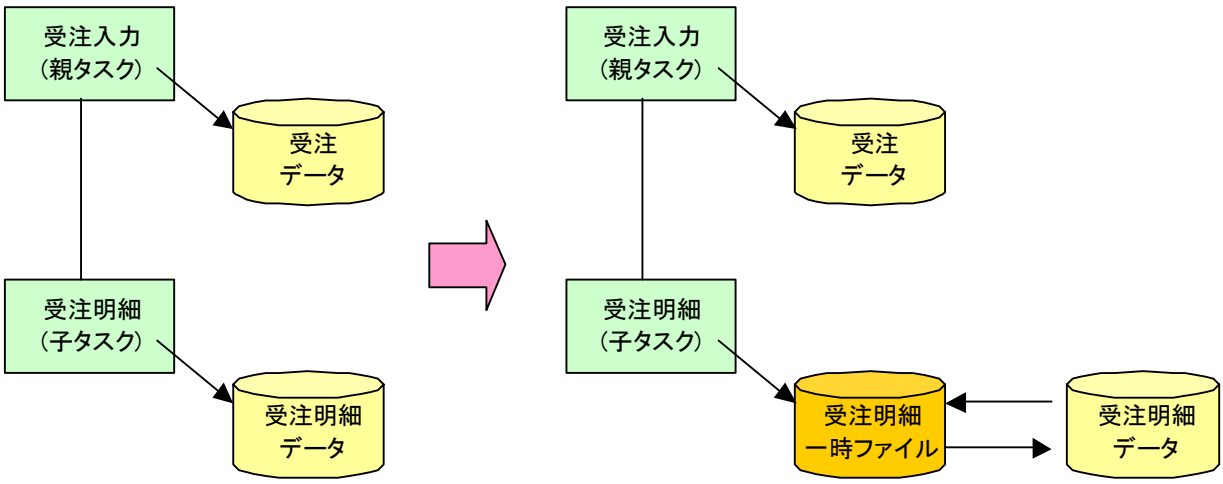
前章で見たように、トランザクションを使うようになるとレコードロックが長い間かけられたままになることがあり、マルチユーザ環境での並列性を下げる原因となります。
この問題に対応するためには、一時テーブルを使って、入力や変更はすべて一時テーブルに溜めて置き、最後に確定した時点でいっぺんにバッチタスクで一時テーブルの内容をデータベースに反映する、という手法がよく用いられています。
本章ではこの手法を使って受注入力プログラムを書き直していきます。

11.1. 考え方

一時テーブルを使ってロック待ちを回避する方法の基本的な考え方は、以下の通りです（図 11-1 参照)。

- 受注明細ファイルにリンクされている商品ファイルのロックを避けるため、明細レコードは一時テーブル(メモリテーブルや、ローカルの Pervasive ファイル)に蓄えておきます。具体的には、
  - 受注レコード(親タスク)のレコード前処理で、登録モードの場合には一時テーブルを空にします。修正モードの時には現在の受注番号の明細レコードを一時テーブルにコピーします。この処理は、一時テーブルの準備用のバッチタスクで行います。
  - ユーザの明細行に関する入力(登録モード)/修正(修正モード)は、一時テーブルに記録しておきます。
  - 一枚の注文票に対するユーザの入力が終了したら、親タスクのレコード後処理で、一時テーブルの追加・変更内容を DBMS の明細レコードに反映させます。このときに同時に、商品ファイルの受注数も更新します。(これは、一時テーブルの修正反映用のバッチタスクで行います)。
- 受注レコードにリンクされている顧客レコードのロックおよび連番発行のための制御ファイルのレコードのロックを避けるための手法は、Pervasive の時にやったのと同じです。すなわち、
  - 顧客レコードについては、受注登録プログラムでは「アクセス」＝「R=読込」でオープンし、データ更新は更新用のバッチタスクを呼び出して行います。
  - 制御ファイルのレコードについては、連番作成のバッチタスクを呼び出して行います。 ただし、制御ファイルのレコードロックがトランザクションコミットまで残るので、連番作成のタイミングを変え、ユーザの入力がすべて済んで一時ファイルに蓄積したデータをデータベースに反映する直前に連番作成を行います。

図 11-1 受注入力タスクの変更



変更前

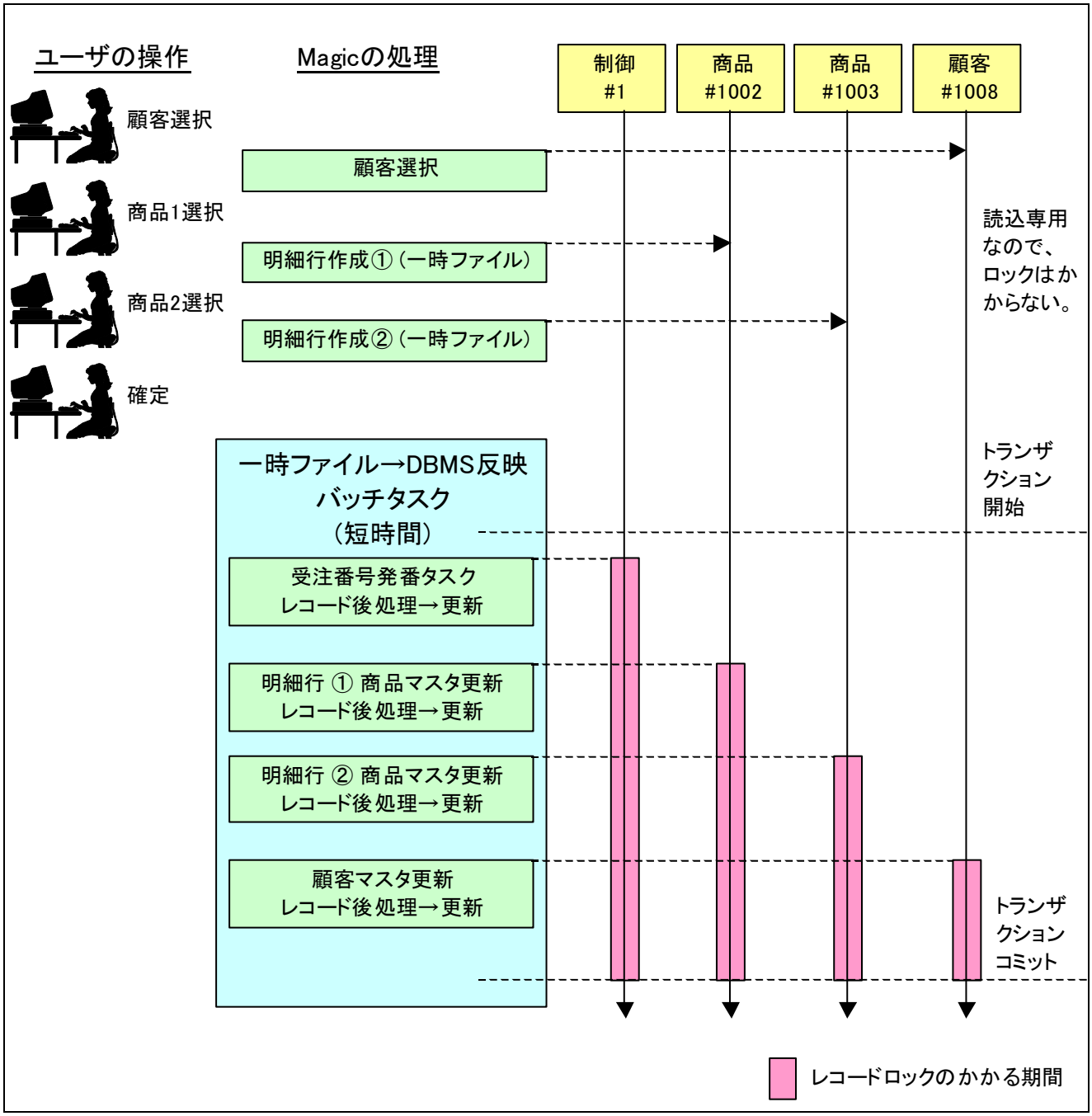
受注データ、受注明細データを直接操作していた。

変更後: 明細レコードは・・・

レコード前処理で一時ファイルにコピーし、レコード後処理で一時ファイルの内容をデータベースに反映する。

この方式を使うと、ロック待ちの問題がどう解決されるかを示したのが、図 11-2 です。

図 11-2 一時テーブルを使う方法でのロックの状況



1. ユーザの入力により、
   - 顧客選択（顧客レコードが読み込まれる）
   - 明細行作成（商品レコードが読み込まれ、明細行データは一時テーブルに書き込まれる）

   が起こりますが、顧客マスタ、商品マスタは読込専用なのでロックは起こらず、明細行データは、一時テーブルへの書き込みなのでやはりロックはかかりません。
2. 入力が終了したら、一時テーブルの内容を MS-SQL Server へ反映するためのバッチタスクが走ります(上図の「一時テーブル→DBMS 反映バッチタスク」と書かれている部分)。この中で、

- 受注番号の発番のため、制御レコードがロック・更新されます。
- 各明細行の受注数を反映するため、商品レコードがロック・更新されます。
- 累積受注額、受注回数を更新するため、顧客レコードがロック・更新されます。

ここでも、レコードロックが発生し、トランザクションの間中累積され、コミットによって始めてロック解除される、という点では前と同じです。しかし、この方法が前の方法と異なるところは、ロックからロック解除までの処理がすべてバッチタスクの中で、極めて短時間のうちに行われることで、ユーザの入力待ちの間にはどのレコードにもロックがかかっていない、ということです。
このため、ロックの期間は非常に短くなりロック衝突の可能性が低くなるし、万一タイミングが悪くてロックが衝突してしまった場合でも、次のリトライのタイミングで書き込みが可能になります。

11.2. 一時テーブルの定義

では、実際に上の考え方にしたがって受注入力プログラムを改造していきましょう。

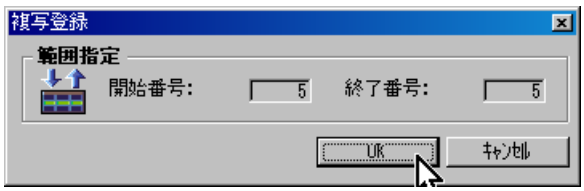
最初にすることは、明細レコードを一時的に格納する一時テーブルをテーブルリポジトリに定義することです。これは**メモリーテーブル**として定義します。
メモリテーブルというのは、Magic 独自のデータベースで、次のような性質を持っています。

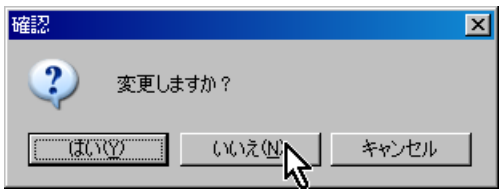
- レコードはすべてメインメモリ上に格納し、ディスク上に保存しません。
- アクセスが非常に高速です。
- ディスクに保存されないため Magic が終了すると同時にメモリテーブルの内容も消えてしまいます。
- 大量のデータを保存するとメモリを圧迫する原因ともなります。
- 本質的に複数ユーザ間で共有することがありません。そのためロックなどを掛ける必要がありません。

メモリテーブルはこのような性質を持っているので、基本的に少量の個人用の一時データを格納するために用います。受注入力プログラムで使う一時テーブルなどとしては最適です。

カラムの定義は、受注明細ファイルと同じですので、Magic の複写登録機能を利用して受注明細ファイルの定義をコピーして、「データベース」を「Memory」に変更するのが、簡単な方法です。

1. テーブルリポジトリを開き、最下行にカーソルを置きます。
2. メニュー「編集(E)」から「複写登録(R)」を選択します。複写登録ダイアログが開きます。
3. 開始番号、終了番号とも、「５」(受注明細テーブルの番号)を指定します。
4. OK ボタンを押すと、受注明細テーブルのコピーが 6 番目のエントリに作成されます。
5. 「データベース」を「Memory」に変更してください。
6. ここでデータ再編成機能が働き、「変更しますか？」と確認ダイアログが表示されます。今はデータの移行ではないので、「いいえ(N)」で答えます。
7. 区別するために、名前を「受注明細データ TMP」に変更します。





最終的には、図 11-3 (次ページ)のようになります。

図 11-3 一時テーブル「受注明細データ TMP」の定義

テーブルリポジトリ

#	名前	カラム	ｲﾝﾃﾞｯｸｽ	外部ｷｰ	DBテーブル	データベース	フォルダ	公開テーブ
1	制御テーブル	4	1	0	制御	MSSQL		
2	顧客マスタ	8	2	0	顧客マスタ	MSSQL		
3	商品マスタ	7	3	0	商品マスタ	MSSQL		
4	受注データ	8	2	0	受注	MSSQL		
5	受注明細データ	7	2	0	受注明細	MSSQL		
6	受注明細データTMP	7	2	0	受注明細	Memory		

カラム:受注明細データTMP

#	名前	モデル	型	書式
1	受注番号	9 受注番号	N=数値	8Z
2	受注明細番号	10 受注明細番号	N=数値	3Z
3	商品番号	6 商品番号	N=数値	5Z
4	商品タイプ	8 商品タイプ	A=文字	UA
5	数量	14 商品個数	N=数値	N5CZ
6	単価	16 金額（8桁）	N=数値	N8CZ
7	合計	16 金額（8桁）	N=数値	N8CZ

11.3. 一時テーブル操作用バッチタスク

一時テーブルに対するいくつかの簡単な操作を行うバッチタスクを作成しましょう。以下、一時テーブルを「受注明細データ TMP」と呼びます。

11.3.1. 一時テーブルのレコード削除

これは、受注明細 TMP のレコードをすべて削除するものです。初期化の一環としてデータをクリアする場合に用います。

> **注意：** 本章以下では、Magic のプログラムロジックを説明するために、下記のような表記法で説明していきますが、Magic にこのようなスクリプト言語があるわけではありません。実際に Magic のプログラムを作成・修正する場合には、Magic 開発版でプログラムリポジトリを開いて行います。

タスク特性
名前： B_受注明細 TMP 削除 タスクタイプ： B=バッチ 初期モード： D=削除 メインテーブル： 6（受注明細データ TMP） インデックス： 1（受注番号） トランザクションモード： P=物理 トランザクション開始： T=タスク前処理の前
レコードメイン
セレクト R=実データ 1（受注番号）

初期モードが D=削除なので、処理対象となるレコードのすべてが削除されます。今の場合、レコードメインのセレクトコマンドなどで範囲指定がなされていないため、テーブルの全レコードが対象になります。従って、このバッチタスクを実行することにより、テーブルのレコードがすべて削除されます。

11.3.2. データベースから一時テーブルへのレコードコピー

受注番号をパラメータとして受け、その受注番号に対応する受注明細レコードを、データベースから一時テーブルにコピーします。このとき、一時テーブルは空であることが前提です。

タスク特性
名前：B_受注明細 本→TMP コピー タスクタイプ： B=バッチ 初期モード： D=修正 メインテーブル： 5（受注明細データ）

インデックス： 1（受注番号）
トランザクションモード： P=物理
トランザクション開始： T=タスク前処理の前

レコードメイン

セレクト P=パラメータ P_受注番号 モデル： 9（受注番号）
セレクト R=実データ 1（受注番号） 範囲小： P_受注番号、範囲大： P_受注番号
セレクト R=実データ 2（受注明細番号）
セレクト R=実データ 3（商品番号）
セレクト R=実データ 4（商品タイプ）
セレクト R=実データ 5（数量）
セレクト R=実データ 6（単価）
セレクト R=実データ 7（合計）

リンク C=登録 6（受注明細データ TMP)、インデックス： 1
セレクト R=実データ 1（受注番号）
セレクト R=実データ 2（受注明細番号）
セレクト R=実データ 3（商品番号）
セレクト R=実データ 4（商品タイプ）
セレクト R=実データ 5（数量）
セレクト R=実データ 6（単価）
セレクト R=実データ 7（合計）
リンク終了

レコード後処理

項目更新 I（受注番号） 式： B(受注番号)
項目更新 J（受注明細番号） 式： C(受注明細番号)
項目更新 K（商品番号） 式： D(商品番号)
項目更新 L（商品タイプ） 式： E（商品タイプ）
項目更新 M（数量） 式： F（数量）
項目更新 N（単価） 式： G（単価）
項目更新 O（合計） 式： H（合計）

コピー元のテーブルをメインテーブルとして設定し、コピー先のテーブルは、「登録」モードのリンクコマンドで参照します。レコードメインでは各テーブルの全カラムをセレクトコマンドで選択し、レコード後処理で全カラムについて値をコピーします。コピーレコードを絞り込むため、パラメータで受注番号を受け取り、セレクトコマンドの「範囲小/大」で範囲指定を行っています。このような形は、レコードをテーブルからテーブルにコピーする場合の常套手段です。

11.3.3. 受注明細レコード削除

これは、受注番号をパラメータとして受け、その受注番号に属する受注明細レコードを削除します。このとき、各明細の商品のレコードについて、受注番号を調整します。

タスク特性
名前：B_受注明細削除 タスクタイプ： B=バッチ 初期モード： D=削除 メインテーブル： 5（受注明細データ） インデックス： 1（受注番号） トランザクションモード： P=物理 トランザクション開始： T=タスク前処理の前
レコードメイン
セレクト P=パラメータ P_受注番号 モデル：9（受注番号） セレクト R=実データ 1（受注番号） 範囲小： P_受注番号、範囲大： P_受注番号 セレクト R=実データ 2（受注明細番号） セレクト R=実データ 3（商品番号） セレクト R=実データ 5（数量） リンク Q=照会 3（商品マスタ） インデックス： 1 セレクト R=実データ 1（商品番号） 位置付小： C（商品番号）、位置付大：C（商品番号） セレクト R=実データ 6（受注数） リンク終了
レコード後処理
項目更新 F（受注数） 式： F-D（受注数 - 数量）

11.3.1 の「一時テーブルのレコード削除」の場合と同じく、初期モードが「D=削除」のバッチタスクとして組みます。メインテーブルは、削除の対象となる「受注明細データ」テーブルです。
11.3.1 の「一時テーブルのレコード削除」の場合には、レコード後処理に何も処理が入りませんでしたが、今回は商品マスタの「受注数」を調整する必要があります。ここでは、受注明細を削除しているので、単純な引き算となります。レコードメインでは、この明細レコードの商品番号カラムで指定されている商品のレコードをリンクし、レコード後処理で受注数を減らしています。

11.3.4. 一時テーブルのデータをデータベースに反映

これは、一時テーブルに格納されている仮の受注明細データをデータベースにコピーするものです。コピーに伴って、各受注明細の商品について商品マスターの受注数を調節します。

タスク特性

名前：B_受注明細 TMP→本コピー

タスクタイプ： B=バッチ

初期モード： M=修正

メインテーブル： 6 (受注明細データ TMP)

インデックス： 1 (受注番号)

トランザクションモード： P=物理

トランザクション開始： T=タスク前処理の前

レコードメイン

セレクト P=パラメータ P_受注番号 モデル： 9 (受注番号)

セレクト R=実データ 1 (受注番号)

セレクト R=実データ 2 (受注明細番号)

セレクト R=実データ 3 (商品番号)

セレクト R=実データ 4 (商品タイプ)

セレクト R=実データ 5 (数量)

セレクト R=実データ 6 (単価)

セレクト R=実データ 7 (合計)

リンク C=登録 5 (受注明細データ)、インデックス： 1

セレクト R=実データ 1 (受注番号)

セレクト R=実データ 2 (受注明細番号)

セレクト R=実データ 3 (商品番号)

セレクト R=実データ 4 (商品タイプ)

セレクト R=実データ 5 (数量)

セレクト R=実データ 6 (単価)

セレクト R=実データ 7 (合計)

リンク終了

リンク Q=照会 3 (商品マスタ) インデックス： 1

セレクト R=実データ 1 (商品番号) 位置付小： D (商品番号)、位置付大：D (商品番号)

セレクト R=実データ 6 (受注数)

リンク終了
レコード後処理
項目更新 I (**受注番号**) **式**: A(**パラメータ**) **項目更新** J (**受注明細番号**) **式**: C(**受注明細番号**) **項目更新** K (**商品番号**) **式**: D(**商品番号**) **項目更新** L (**商品タイプ**) **式**: E (**商品タイプ**) **項目更新** M (**数量**) **式**: F (**数量**) **項目更新** N (**単価**) **式**: G (**単価**) **項目更新** O (**合計**) **式**: H (**合計**) **項目更新** Q (**受注数**) **式**: Q + F (**受注数 ＋ 数量**)

このプログラムは、11.3.2 の「データベースから一時テーブルへのレコードコピー」とは反対方向の処理を行うもので、プログラムの作りは良く似ています。
ただし、「受注明細データ」テーブルにレコードを追加していくので、商品マスターレコードの「受注数」を調整する必要があります。この場合には、レコードの追加なので、単純な加算となります。これをレコード後処理で行っています。

以上、単純なバッチタスクを 4 つ用意しましたが、これを使ってどのようにして一時テーブルを使った受注入力プログラムを作ることができるでしょうか？次節以降に、プログラムの修正点を説明します。

11.4. 親タスクのレコード前処理の修正

参考： ここから受注入力プログラムを変更していきますが、念のために、既存の受注入力プログラムを直接修正するのではなく、プログラムをコピーしてコピーしたものを修正していくことをお勧めします。ここでは、コピー後のタスクを「受注入力(SQL)」という名前にしておきます。

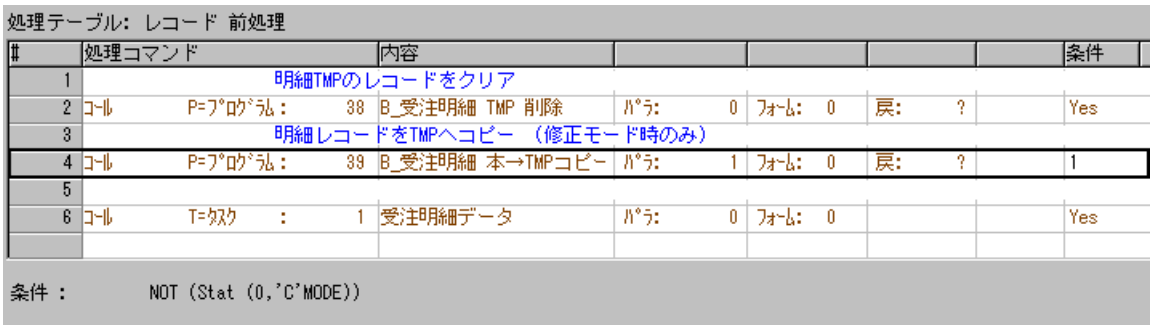
一時テーブルを利用する方法では、各受注レコードについて、明細レコードをデータベースから一時テーブルにコピーしてやらなければなりません。これは、受注入力プログラムの親タスク側のレコード前処理で行います。

一時テーブルは、まず内容をクリアする必要があります。これには「11.3.1 一時テーブルのレコード削除」(82 ページ)で説明したプログラム「B_受注明細 TMP 削除」を呼び出します。
次に、「11.3.2 データベースから一時テーブルへのレコードコピー」（82 ページ）で説明した、コピープログラム「B_受注明細 本 → TMP コピー」を呼び出します。
ただし、タスクが登録モードの場合には、データベースに明細レコードがまだ入力される前ですから、コピープログラムは呼び出す必要はありません。従って、NOT Stat(0,'C'MODE) を条件に設定します。

以上まとめると、親タスクのレコード前処理は次のような内容となります。

レコード前処理
コール プログラム 38(B_受注明細 TMP 削除)
コール プログラム 39(B_受注明細 本 → TMP コピー)　条件： NOT Stat(0,'C'MODE)
パラメータ： A （受注番号）
コール タスク 1 （受注明細データ）

図 11-4 親タスクのレコード前処理



11.5. 親タスクのコントロール後処理の削除

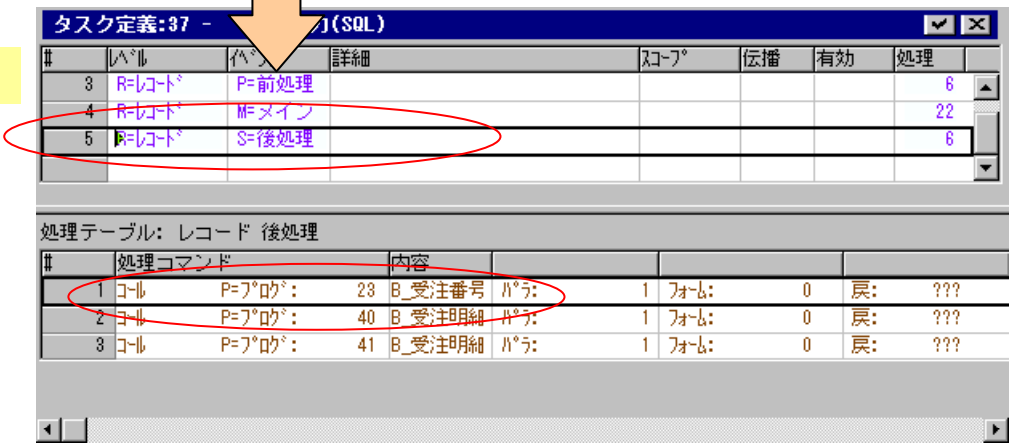
SQL 対応前の受注入力プログラムでは、「顧客番号」のコントロール後処理で受注番号を発番していました。しかし、SQL 対応の基本方針として、受注番号はユーザの入力が終わりレコードを一時ファイルからデータベースに書き込む直前に行うようにしました。

この変更に従い、親タスクの「顧客番号」のコントロール後処理のハンドラは削除します。

連番作成のバッチタスクは、レコード後処理に移動します。(11.7.1「受注番号の発番」(91 ページ)を参照。)

修正前

修正後



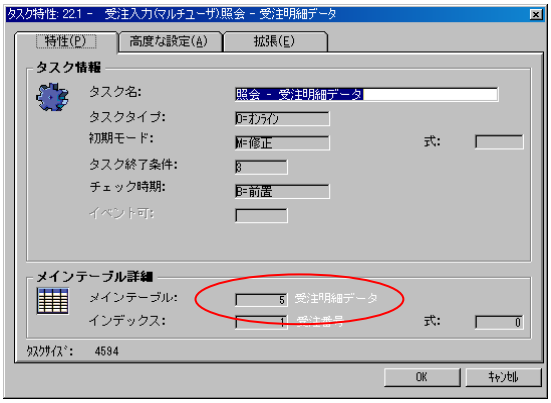
11.6. 受注明細タスク(子タスク)の変更

先に、子タスク(受注明細を扱うタスク)から考えます。子タスクは商品マスターへの更新がなくなるので、すっきりとします。

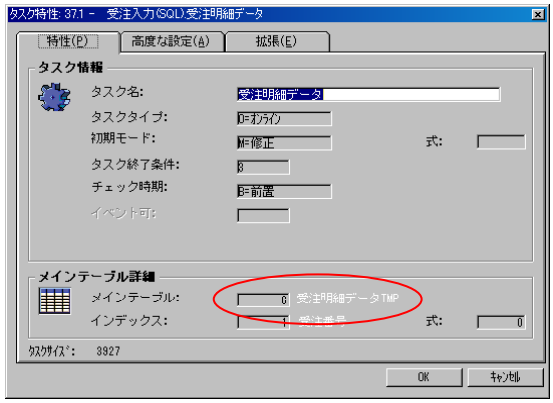
11.6.1. 受注明細タスクのメインテーブルの変更

SQL 化の基本方針は、明細レコードを一時テーブルにコピーして扱う、ということなので、受注明細のタスクのメインテーブルとしては、DBMS の受注明細ファイルを指定するのではなく、一時テーブルを使うようにします。これは、タスク特性の「メインテーブル」を受注明細テーブル(テーブル 5 番)から受注明細テーブル TMP（テーブル 6 番）に変更するだけです。この二つのテーブルは、カラムの定義が全く同じなので、レコードメインやその他の部分には手をつける必要はありません。

修正前



修正後

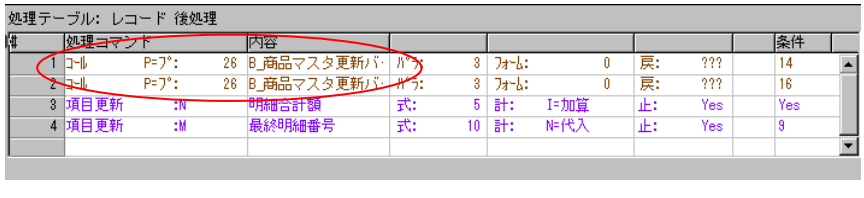


11.6.2. レコード後処理

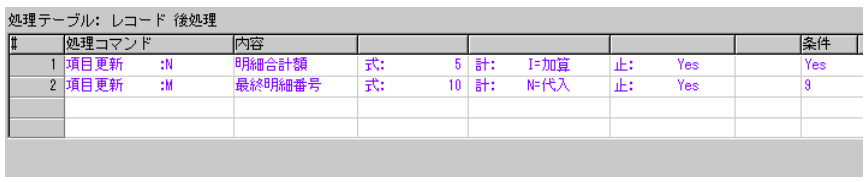
次に、SQL 化のもうひとつの方針として、商品マスターへの更新(受注個数の更新)は、入力が確定したときに、最後にバッチタスクでいっぺんに行う、ということでした。したがって、子タスクの中では商品マスターに変更をかけることをしません。

レコード後処理で商品ファイルの「受注数」を変更するには、バッチプログラム「B_商品マスタ更新バッチ」を呼び出していましたが、このコールコマンドを削除します。

修正前

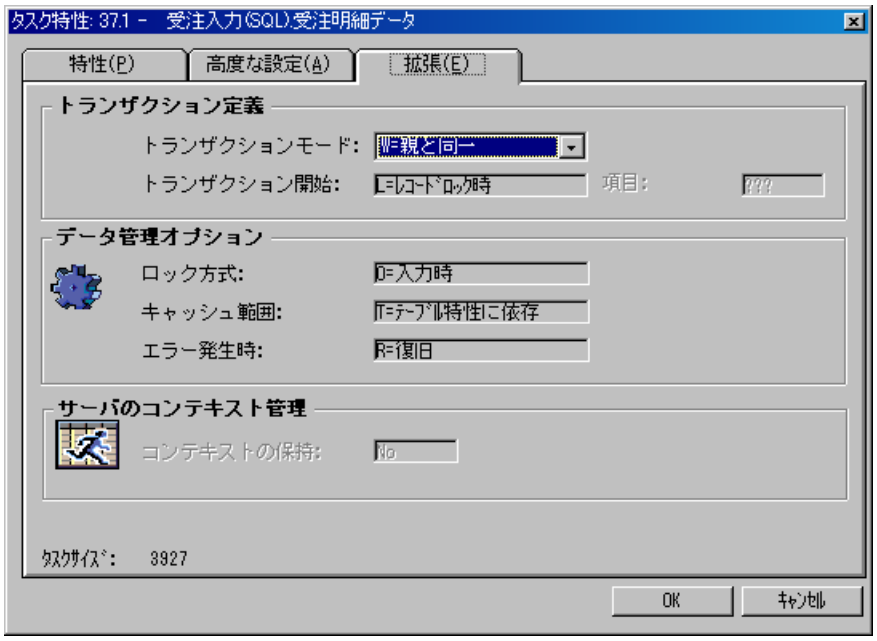


修正後



11.6.3. トランザクションの設定

もう一つの修正事項(確認事項)は、トランザクションの設定です。親タスクと同一のトランザクション内で実行するために、タスク特性の「高度(A)」タブで、トランザクションモードの設定が「W=親と同一」となっていなければなりません。デフォルトで子タスクのトランザクションはこの設定となっているはずですが、確認してください。



11.7. 親タスクのレコード後処理の変更

受注明細の一時テーブルに対する入力・修正内容は、受注入力(登録、修正)が確定したタイミングでデータベースに反映させます。これは、プログラムで言えば、親タスクのレコード後処理になります。ここでは、親タスクのレコード後処理で行うべき処理について説明します。

11.7.1. 受注番号の発番

最初に、登録モードのときにはまだ受注番号が確定されていないので、新しい受注番号を発番する必要があります。これは、以前にマルチユーザ対応のために作成したバッチタスク「B_受注番号発番」をそのまま利用します。これは登録モードのときだけ実行されるものなので、条件としては Stat(0,'C'MODE) となります。

11.7.2. 明細レコードをデータベースに反映

一時テーブルの内容をデータベースに反映する際には、登録モードの場合には単に一時テーブルの内容をデータベースに追加すればよいのですが、修正モードの場合も考えると、データベース中の明細レコードと一時テーブル中の明細レコードとをつき合わせてマージする必要があります。
レコードのマージを行うのに、ひとつひとつ受注明細番号が同じという条件でレコードをつき合わせる、という方法もあるのですが、明細レコードが追加された場合、削除された場合、追加も削除もされなかったが修正された場合、変更のなかった場合、商品番号が変わった場合、変わらなかった場合、というように場合分けするのが面倒で誤りやすいものです。しかもこのときに同時に、明細レコードに指定されている商品について、受注数の調整を行いますので、複雑化します。

ここでは単純化のために、データベース中に格納されている現在の受注に関する既存の明細レコードをいったん全部削除してから、一時テーブルの内容をデータベースに追加コピーするという方法を使って、最終的にマージを行ったのと同じ結果になるようにします。

よって、このステップは次のような二つの操作となります。

1. 修正モード、あるいは削除モードの場合には、データベースにある既存の明細レコードを「0 受注明細レコード削除」(84 ページ)で説明したバッチタスク「B_受注明細 削除」により削除します。
2. 登録モード、あるいは修正モードの場合には、一時テーブルにあるレコードを、データベースに追加します。これは「0 一時テーブルのデータをデータベースに反映」(85 ページ)で説明したバッチタスク「B_受注明細 TMP→本」で行います。

これらのバッチタスクは、同時に商品マスタレコードの受注数の更新も行いますので、商品マスタの「受注数」も正しく維持されます。

11.7.3. 顧客レコードの累計受注額と受注回数の更新

最後に、顧客レコードの累積受注額と受注回数の更新も行います。
この処理については SQL 化する前と同じで、バッチタスク「B_顧客マスタ更新バッチ」を呼び出すことにより行います。

11.7.4. まとめると

以上をまとめると、レコード後処理は次のようになります。

レコード前処理

コールプログラム 23(B_受注番号発番)　**条件**： Stat(0,'C'MODE)
　　パラメータ： A (受注番号)
コールプログラム 40(B_受注明細 削除)　**条件**： Stat(0,'MD'MODE)
　　パラメータ： A (受注番号)
コールプログラム 41(B_受注明細 TMP→本)　**条件**： Stat(0,'CM'MODE)
　　パラメータ： A (受注番号)

コールプログラム 25 (B_顧客マスタ更新バッチ)　**条件**： Stat(0,'MD'MODE)
　　パラメータ： VarPrev('C'VAR), VarPrev('Q'VAR), 'FALSE'LOG
コールプログラム 25 (B_顧客マスタ更新バッチ)　**条件**： Stat(0,'MD'MODE)
　　パラメータ： C(顧客番号), Q(受注合計額), 'TRUE'LOG

図 11-5 親タスクのレコード後処理

処理テーブル: レコード 後処理

#	処理コマンド			内容							条件
1	ｺｰﾙ	P=ﾌﾟﾛｸﾞ:	23	B_受注番号発番	ﾊﾟﾗ:	1	ﾌｫｰﾑ:	0	戻:	???	9
2	ｺｰﾙ	P=ﾌﾟﾛｸﾞ:	40	B_受注明細 削除	ﾊﾟﾗ:	1	ﾌｫｰﾑ:	0	戻:	???	10
3	ｺｰﾙ	P=ﾌﾟﾛｸﾞ:	41	B_受注明細 TMP→本	ﾊﾟﾗ:	1	ﾌｫｰﾑ:	0	戻:	???	11
4											
5	ｺｰﾙ	P=ﾌﾟﾛｸﾞ:	25	B_顧客マスタ更新ﾊﾞｯﾁ	ﾊﾟﾗ:	3	ﾌｫｰﾑ:	0	戻:	???	10
6	ｺｰﾙ	P=ﾌﾟﾛｸﾞ:	25	B_顧客マスタ更新ﾊﾞｯﾁ	ﾊﾟﾗ:	3	ﾌｫｰﾑ:	0	戻:	???	11

11.8. 実行してみる

プログラムができたので、実行して確認しましょう。

11.8.1. メニューへの登録

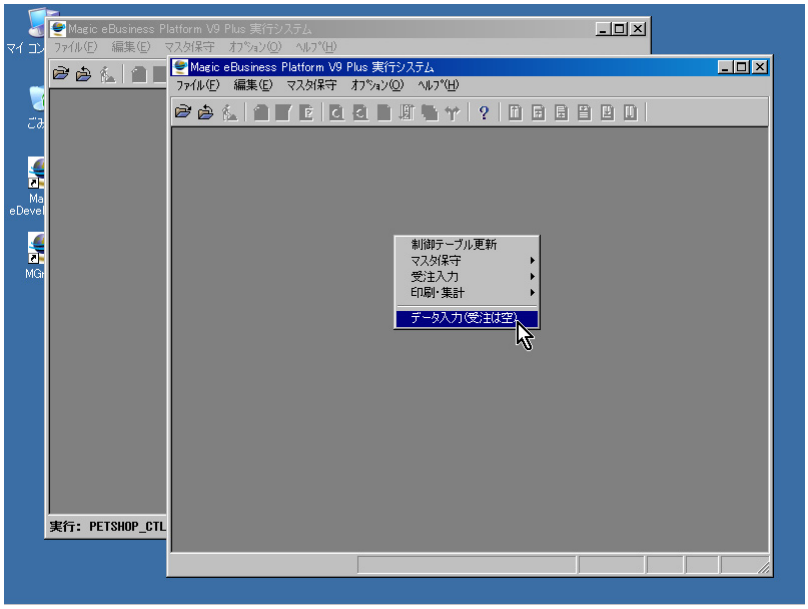
先に作った受注入力プログラムをメニューに登録します。メニューリポジトリを開き、「受注入力」のサブメニューとして、今作った「受注入力(SQL)」を登録してください。

11.8.2. テスト準備

開発版ではアプリケーションを閉じます。マルチユーザのテストをするので、実行版を二つ起動し、それぞれでアプリケーションを開いておきます。複数のユーザになぞらえて、それぞれをユーザ A、ユーザ B と呼びます。

テスト用のデータを初期化するため、ポップアップメニューから「データ入力(受注は空)」を選んで実行してください。

これにより、受注データは空になり、マスターの受注関係データはすべてクリアされます。



11.8.3. テストパターン

テストは、大きくわけて次のようなポイントを確認しましょう。ここではひとつひとつについて細かく書きませんが、実際に行って試してください。

- 一人で使っている場合に、登録・修正・削除を行い、正しくデータが計算・登録されること。
  - 新規で受注データ(明細行を複数含む)を登録する。

- 既存の受注データについて、以下のことを行う。
  - 明細行を 1 行追加する。
  - 既存の明細行の商品番号と個数を変更する。
  - 明細行を 1 行削除する。
  - 明細行の商品番号と個数を変更した後にその行を削除する。
  - 親タスクにカーソルがある状態で、
    - 顧客番号を変更する。
    - 明細行がすべて削除した後、F3 キーで削除する。
- 二人で同時に使っている場合に、利用を妨げるようなロック待ちなどや不正更新などが発生しないこと。
  - 同時登録の場合：
    - ユーザ A が受注の入力を行い、タスクを終了しないでそのまま待っている状態で、ユーザ B が同一顧客番号、同一明細内容で入力を行う。このとき、ロック待ちが発生しないことを確認。
    - ユーザ A がタスクを終了した後、ユーザ B がタスクを終了する。このとき、ロック待ちが発生せず、データが両方とも正しく入力されていることを確認。
  - 同時修正の場合：
    - ユーザ A、ユーザ B が、それぞれ自分が入力したデータを開き、先に一人で使っている場合に行ったような修正を同時に行う。このときにロック待ちが発生しないことを確認。
    - ユーザ A、ユーザ B がそれぞれタスクを終了する。データが正しく更新されていることを確認。

このようなテストがすべてパスすれば、受注入力プログラムの SQL 化は完成です。

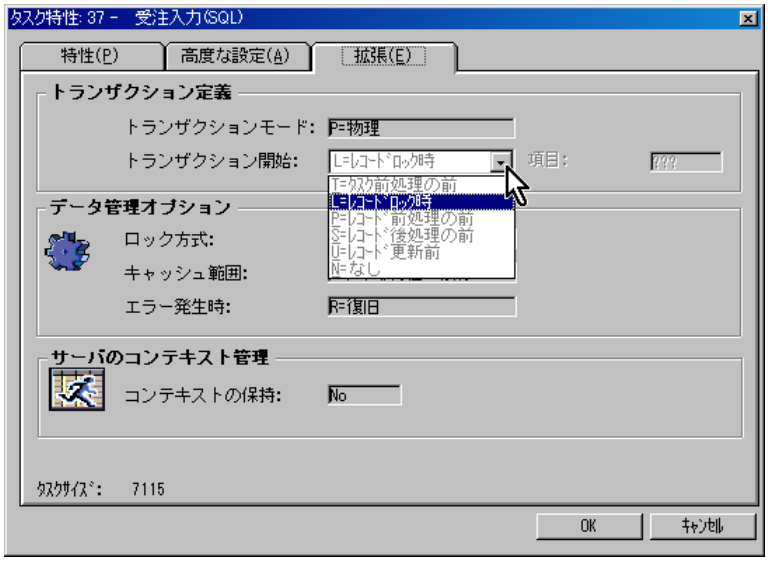
12. トランザクション設定の注意事項

前章では、受注入力プログラムについて、SQL 対応を行いました。
本章では、前章で説明した事項のほかに、トランザクションを使う Magic プログラムを開発・設計する上で注意すべき次のような点について簡単に説明します。

- トランザクションの開始と終了のタイミング
- 物理トランザクションのネストはできない
- トランザクションの対象となるデータベース
- テーブルを使わないタスク

12.1. トランザクションの開始と終了のタイミング

Magic のプログラムでのトランザクションの開始と終了は、タスク特性の「高度」タブの「トランザクション開始」パラメータにより決定されます。



このパラメータの意味は、以下の通りです。

トランザクション開始パラメータ	トランザクション開始	トランザクション終了
T=タスク前処理の前	タスク前処理の前	タスクタスク前処理の後
L=レコードロック時	レコードがロックされる直前	レコードが更新された後
P=レコード前処理の前	レコード前処理の前	レコードが更新された後
S=レコード後処理の前	レコード後処理の前	レコードが更新された後
U=レコード更新前	レコード後処理の後、データベースに対して更新処理が行われる直前	レコードが更新された後
N=なし	トランザクションは開始されません	
G=グループ	グループ前処理の前	グループ後処理の後

参考： 「G=グループ」は、バッチタスクで「グループレベル」が定義されている場合に指定できます。この場合には、グループ項目を［項目］欄に定義します

トランザクションの開始と終了は、タスクのレベルで対称になっていることに注意してください。具体的には、

- 「T=タスク前処理の前」ならば、タスク前処理の前に開始し、タスク後処理の後に終了します。すなわち、タスクレベルで始まり、終わります。
- 「L=レコードロック時」から「U=レコード更新時」は、開始時期はそれぞれ異なりますが、いずれも一つのレコードの処理の中であり、終了時期もそのレコードが更新されたときとなっており、いずれの場合にもレコード単位で閉じています。
- 「G=グループ」の場合には、同一のグループの前処理の前に始まり、後処理の後に終わります。すなわ

ち、グループの中で始まり終わります。
となっています。

逆に言うと、異なるレベルにまたがってトランザクションの開始・終了が対応することはありません。例えば、次のようなトランザクションの設定はすることが**できません**：

- ✖ あるレコードの前処理の前に始まり、タスクの後処理の後で終了する
- ✖ 子タスクのレコード前処理の前で始まり、親タスクのタスク後処理の後で終了する。

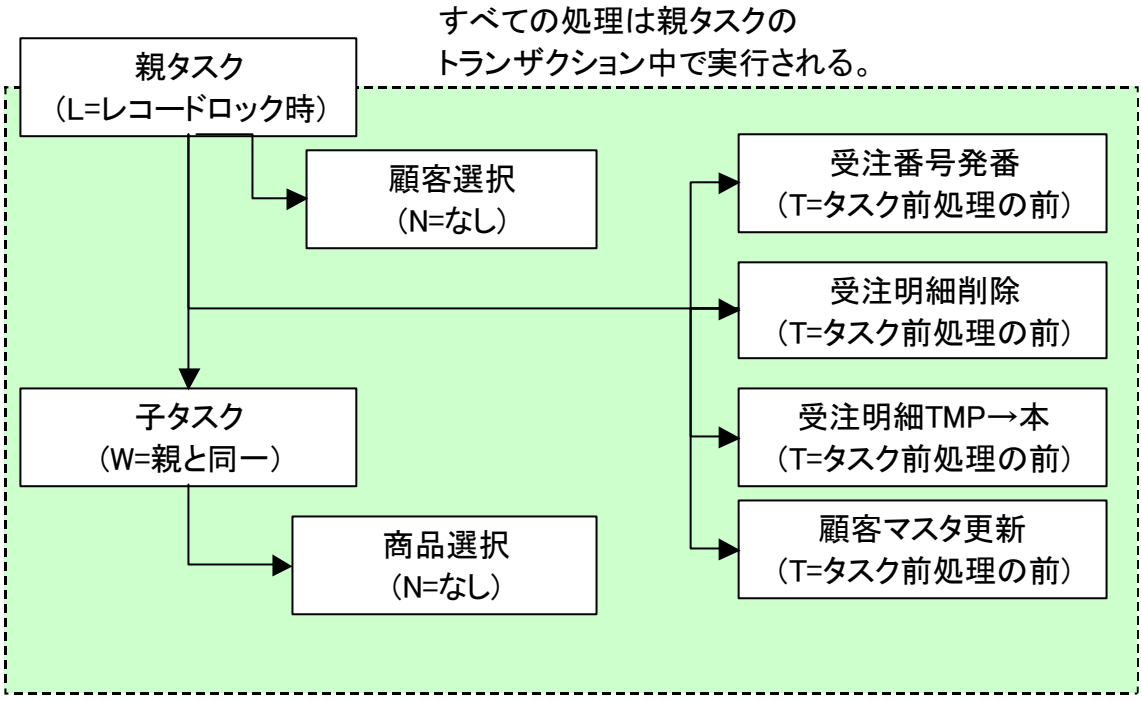
12.2. 物理トランザクションのネストはできない

あるタスクでトランザクションの設定がされており(つまり、「N=なし」以外に設定されている)、そのタスクが別のタスクを呼び出したとき、そのタスクにもトランザクションの設定がされている場合には、トランザクションはどのように扱われるのでしょうか？親タスクでトランザクションが開始され、子タスクでも別のトランザクション(ネストされたトランザクション)が開始されるのでしょうか？

結論から言うと、Magic では、物理トランザクションの場合には、トランザクションのネストを**行いません**。今の例では、親の方でトランザクションが開始されたら、それ以降は、呼び出されるタスクのトランザクションの設定のいかんにかかわらず、すべての処理は親タスクのトランザクションの中で処理されます。

受注入力タスクの例で、具体的に説明しましょう。図 12-1 は、前章で作成した受注入力プログラム(SQL対応版)から呼び出されるタスクと、それぞれのトランザクション設定を表しています。

図 12-1 受注入力(SQL)から呼び出されるタスクのトランザクション設定



受注入力タスクでは、親タスクで「L=レコードロック開始時」にトランザクションが開始されます。これは、顧客選択を行ってデータの変更を行ったりした時点で始まります。
以後、受注入力では、子タスクに移って商品選択・個数の入力をし、親タスクに戻ってからレコード後処理で一連のバッチタスクを実行します。
このとき、子タスクは「W=親タスクの中」のトランザクション設定となっており、受注番号発番などの一連のバッチタスクでは、それぞれで「T=タスク前処理の前」と設定されています。
しかし、これらのタスクのトランザクションの設定にかかわらず、すでに最初のタスク(親タスク)でトラン

ザクションが開始されてしまっているので、すべての処理はこのトランザクションの中で行われます。このトランザクションは親タスクのレコードの更新が終了した時点で終了します。このときに、下位のタスクで行った修正がすべていっぺんにコミットされます。逆に言うと、このときまでは、修正内容はコミットされていないので、他の人からは見ることができません。また、「9.5.2 ロックとトランザクションの関係」(71ページ)で説明したように、トランザクションの中ではロックが累積され、トランザクションをコミットするときにすべてのロックが一度に解除される、というようになっていますので、下位のタスクでロックしたレコードも、親タスクでレコード書き込みが行われてコミットされるまで解除されないことになります。

これは単純な約束事ですが、実際にいろいろなタスクを呼び出す複雑なプログラムになると、どこでトランザクションが開始されるのかをあらかじめよく考えて設定しておかなければ、書き込んだつもりでいるレコードがコミットされずに他から見えなかったり、あるいはロックが解除されないでロック待ちが頻発したりなどの事態が起こります。この手の不具合は、単体テストでは見つかりにくく、複数のユーザが同時に使い始めて初めて気がつく、というケースが多く、また、原因を追究するのも面倒になりがちなものです。

12.3. トランザクションの対象となるデータベース

次に注意すべき点は、トランザクションが開始されるとき、「どのデータベースがトランザクションの対象になるか？」ということです。

あるタスクで異なるデータベース上のテーブルを利用していたとします。受注入力の場合には、MS-SQL Server 上のテーブルと、明細データの一時格納場所としてメモリゲートウェイを利用していました。もう少し複雑な構成では、例えば社員マスタと受注データとを異なるサーバマシンの MS-SQL Server に格納してあるのを同時にアクセスし、一時ファイルとして Pervasive.SQL をローカルに保存したり、小さなデータはメモリゲートウェイに保存する、などといった使い方もするかも知れません。ことによると、Magic の「データベーステーブル」には、もっと多くのデータベースが登録されているかもしれません。
このタスクでトランザクションが開始されるとき、どのデータベースにトランザクション開始を伝達するのでしょうか？

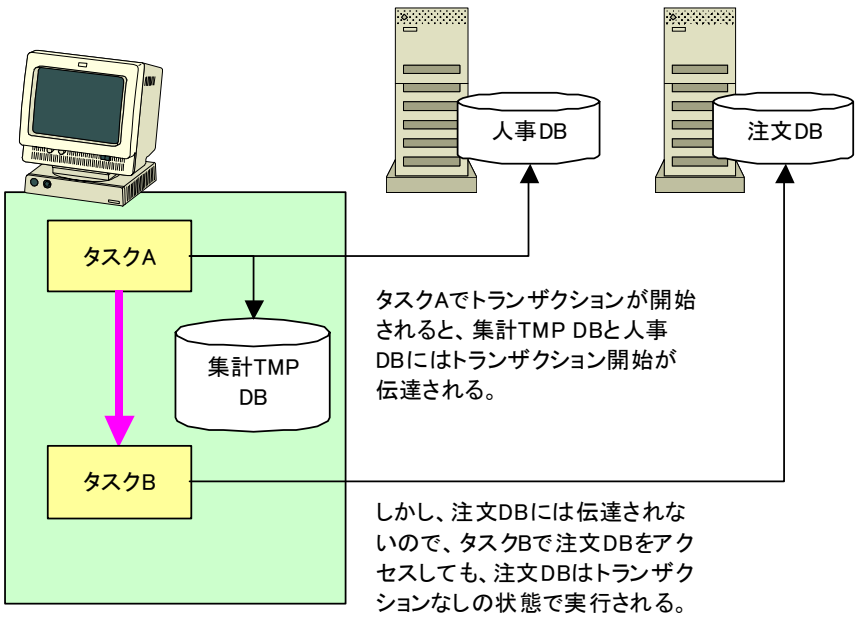
12.3.1. DB テーブルに登録されているテーブルに属するデータベース

答えは、トランザクションを開始する時点で、オープンされているテーブルが格納されているデータベースに対して、トランザクション開始を伝達します。すなわち、

- このタスクの「DB テーブル」に登録されているテーブルの属するデータベース
- このタスクを呼び出ししたタスク(あるいはその先祖のタスク)の「DB テーブル」に登録されているテーブルの属するデータベース

にはトランザクションが伝達され、それ以外のデータベースには伝達されません。
この規則を具体的な例で考えて見ましょう。図 12-2 を見てください。

図 12-2 トランザクション開始の伝達されるデータベース



これは今までのペットショップデモよりも複雑で、二つの MS-SQL Server のデータベース「人事データベース」と「注文データベース」およびローカルの Pervasive.SQL を集計の一時テーブルとして使っているとします。
ここで、タスク A では集計の一時テーブル(Pervasive.SQL)と、社員マスタ(人事データベース)とを利用しています。ここでタスク A がトランザクションを開始したら、Pervasive.SQL と人事データベースにはトランザクション開始(BEGIN TRANSACTION 命令)が伝達され、トランザクションが開始されます。しかし、受注データのある受注データベースは、タスク A が利用していないので、上の規則に従いトランザクション開始は伝達されません。
次に、タスク A がタスク B を呼び出します。タスク B は受注テーブルを使うので、受注データベースをアクセスします。しかしこのような設定では、受注データベースにはトランザクションが開始されないままアクセスされる状態になります。
MS-SQL サーバの場合、トランザクションを使わずに、つまり明示的に BEGIN TRANSACTION を発行っせずに DML 文(SELECT、UPDATE、DELETE、INSERT など)を実行すると、各 DML 文ごとにトランザクションが自動コミットされるものとなります。トランザクションが DML 文ごとにコミットされてしまうので、レコードロックはすぐに解除されてしまい、実質的にはロックがかからずに実行されることになります。
もし、タスク B で受注テーブルにロックをかけて排他制御を行うことを意図して設計されているとするならば、実際には意図に反して、受注テーブルへのロックはかからないことになってしまいます。このような動作の不正は、テストでなかなかひっかかりにくいもので、本番で多くの人が使い始め、データの不正が不定期的に散見されるようになり始めて露呈するようなものです。
このような状況の場合の解決法は、注文 DB に属するテーブルを、タスク A の DB テーブルに追加登録しておくことです。DB テーブルは普通、そのタスクでメインテーブルかリンクテーブルとして使っているテーブルが自動的に登録されていくのですが、それ以外にも、F4 キーで新しくエントリを追加し、任意のテーブルを追加することができます。これは「これから呼び出すタスクでこのテーブルを使います」ということをあらかじめ宣言しておくことになります。
上の例で言えば、タスク A の DB テーブルにエントリを追加し、受注テーブルを登録しておきます。このようにしておけば、タスク A でトランザクションが開始されるときに、Perevasive.SQL、人事データベースのほかに、注文データベースにもトランザクションの開始が伝達されますので、タスク B においてもトランザクションの中で動作することになり、意図していた通り、ロックなどもかかるようになります。
よって、開発時の留意点を大雑把に言えば、

- トランザクションが開始されるタスクを正しく把握する。
- そのタスクの「DB テーブル」に、トランザクション中に利用するテーブルをすべて登録しておく

ということになります。

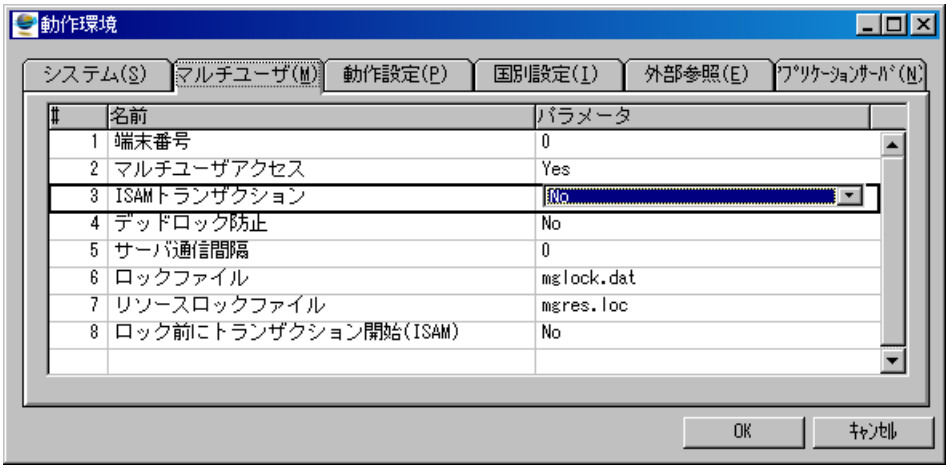
12.3.2. SQL データベースと ISAM データベースの違い

SQL データベースでは、データロックやデータ更新にトランザクションの利用が必須です。このため、SQL データベースに対しては、先に「12.3.1 DB テーブルに登録されているテーブルに属するデータベース」説明した規則に従い、トランザクションの開始が伝達されます。

一方、ISAM データベース (Pervasive.SQL やメモリゲートウェイ)では、データロックやデータ更新にトランザクションの利用が必須ではありません。このため、ISAM データベースではトランザクションをかけないで利用することが多いです。

ISAM データベースでトランザクションを利用するかしないかは、動作環境の「マルチユーザ」タブの「**ISAM トランザクション**」パラメータの設定により制御することができます。(図 12-3)

図 12-3 ISAM トランザクションの設定



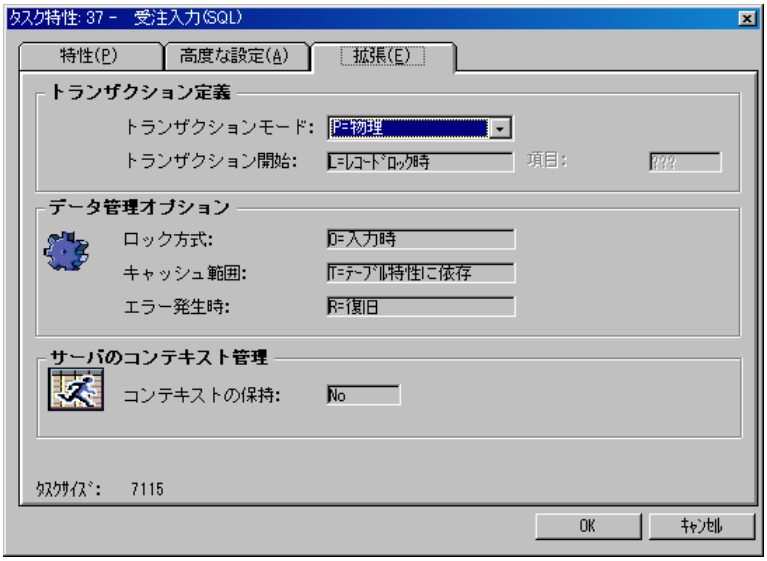
このパラメータが No (デフォルト)の場合には、ISAM データベースに対してトランザクションを利用しません。Yes の場合には、トランザクション開始が伝達されます。

なお、SQL データベースの場合には、トランザクションの利用が必須なので、これを制御するパラメータはありません。

ところで、前にも書いたとおり、トランザクションを利用する場合には、レコードロックはすべてトランザクション内で行われることになります。すなわち、レコードロックに先立ってトランザクションが開始されていなければなりません。

ところが、トランザクション開始のタイミングとレコードロックのタイミングとは、それぞれ独立して設定することができます。

右図は、タスク特性の「拡張(E)」タブで、ここで、



- トランザクション開始（トランザクション開始のタイミングを指定）
- ロック方式（ロックのタイミングを指定）

をそれぞれ指定できます。
これは独立しているので、設定によっては、「ロックはトランザクションの中で」という規則に反する設定をすることもできます。例えば、「トランザクション開始」が「レコード後処理の前」なのに、ロック方式が「入力時」だと、ロックの開始がトランザクションの開始に先立ってしまい、矛盾します。

このような規則違反は、Magic の文法チェック機能（プログラムリポジトリで F8 で実行)により検出することができ、違反している場合には「チェック結果」ウィンドウにエラーが報告されます。
このチェック機能は、うっかりミスを検出する上で役に立つのですが、ISAM データベースでトランザクションを利用しない場合には、この規則違反のチェックは不要になります。このような場合、「ロックとトランザクションの開始の関係のチェックはしなくても良いよ」と Magic に対して指示するのが、同じく図 12-3 動作環境のマルチユーザタブにある「**ロック前にトランザクション開始(ISAM)**」というパラメータです。これが No になっている場合には、ISAM データベースのみを使うタスクについては規則違反のチェックを行いません。Yes の場合には、ISAM データベースのみを使うタスクでもチェックを行います。

簡単には、「ISAM トランザクション」は実行時のパラメータ、「ロック前にトランザクション開始(ISAM)」は開発時のパラメータと覚えておくとよいでしょう。通常はこれらのパラメータの値は同じ(両方 No、あるいは両方 Yes)に設定しておきます。

12.3.3. トランザクションをサポートしない DBMS

Magic が扱う DBMS には、トランザクションをサポートしない DBMS もあります。例えばメモリゲートウェイは、メモリ上での一時データのみを扱い、複数ユーザでデータの共有も行わないものなので、トランザクションの必要性もないので、トランザクションはサポートしていません。当然のことながら、このような DBMS の場合には、「ISAM トランザクション」の設定に関わらず、実行時にトランザクションは伝達されません。

12.3.4. まとめると・・・

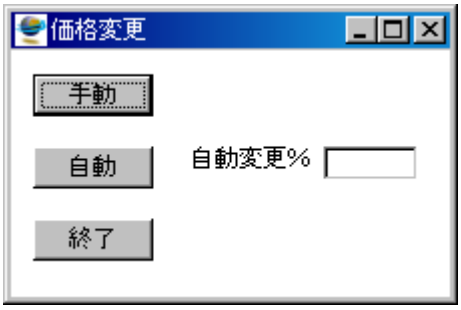
以上のことをまとめると、Magic がトランザクションを開始する場合に DBMS にもトランザクション開始が伝達されるかどうかは、下表のようにまとめられます。

DBMS のタイプ	DBMS のトランザ	ISAM トランザクション設定	トランザクションが DBMS に伝達される？
SQL	サポートする	(いずれでも)	○
ISAM	サポートする	Yes	○
	サポートする	No	×
	サポートしない	(いずれでも)	×

12.4. テーブルを使わないタスク

アプリケーションの中には、まったくテーブルを使わないタスク、というものもあります。このようなタスクの場合には、トランザクションの設定はどのようにすればよいでしょうか？

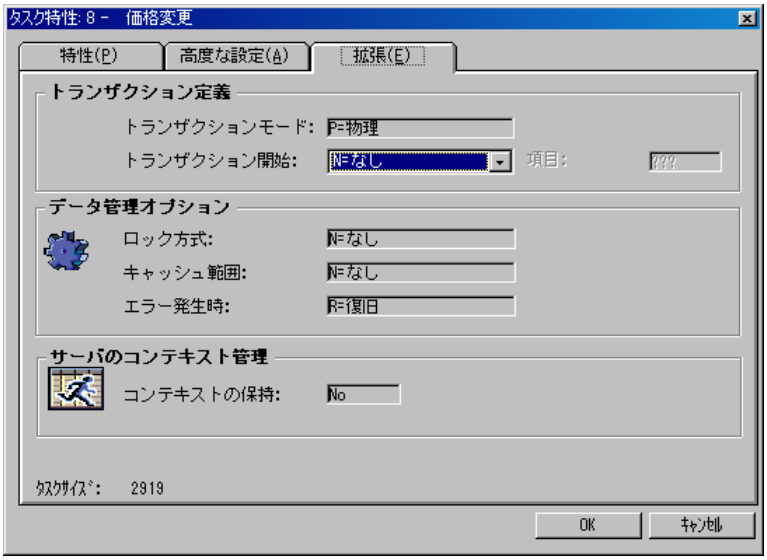
例えば、ペットショップアプリケーションに、「価格変更」のタスクがありました。右図がその初期画面です。このタスクでは、ボタンが三つメニューとして定義されており、自動価格変更の場合には変更のパーセントを指定する数値型の変数項目があります。このタスクでは一切テーブルは使いません。



このような、メニューのような役目で使われているタスクでは、一般にはトランザクションを開始しないように設定します。トランザクションを開始しないようにするには、タスク特性の「拡張(E)」タブでの設定で、

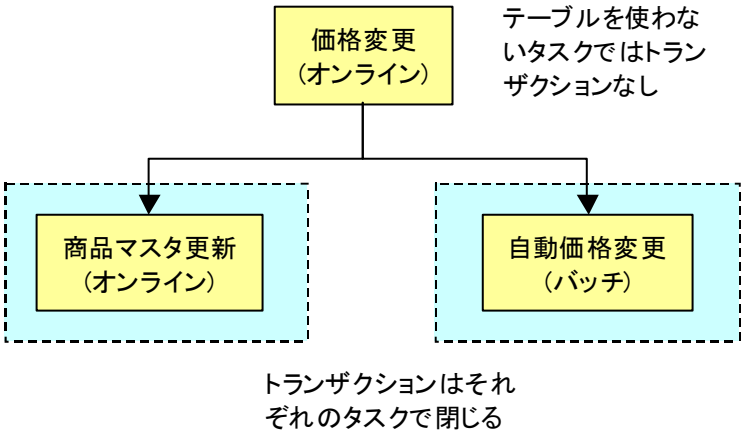
- 「トランザクションモード」を「P=物理」
- 「トランザクション開始」を「N=なし」
- 「ロック方式」を「N=なし」

にします。



このようにしておけば、親のメニューのタスクではトランザクションが開始されません。このタスクから呼び出される「商品マスタ更新」のオンラインタスクや、「自動価格変更」のバッチタスクでは、それぞれにトランザクションの設定がされているので、それぞれのタスクの中でトランザクションが開始され終了します。

このため、親に戻ってきたときには、すでにトランザクションがコミットされた状態なので、処理中のロックなどもすべて解放され、変更はデータベースに反映された状態となっています。



もし、親の「価格変更」タスクでトランザクションを開始するような設定にしてしまったら、どういった不都合が起こるでしょうか？例えば、トランザクション開始を「S=レコード前処理の前」などにしたらどうなるでしょう？

親タスクでは、レコードサイクルの最初(レコード前処理の前)にトランザクションが開始されます。このと

き、このタスクの DB テーブルには何もテーブルが登録されていません。このため、Magic の実行エンジンとしては「トランザクションは開始された」という状態になっているものの、実際にトランザクション開始を伝達されたデータベースはひとつもありません。
ここでユーザが「手動」ボタンを押して、オンラインの「商品マスタ更新」タスクが呼び出されたとします。このタスクでは、オンラインのデフォルトとして、トランザクション開始が「O=入力時」になっていますので、ユーザがキー入力をした瞬間、Magic はレコードロックをかけようとします。本来ならばここで、MS-SQL Server に BEGIN TRANSACTION が送ってトランザクション開始を伝達しなければならないのですが、Magic の実行エンジンとしては、すでに「トランザクションは開始された」という状態になっていますので、「トランザクションのネストはできない」という規則により、Magic はトランザクション開始を伝達しません。
このため、MS-SQL Server はトランザクションなしの状態で動くことになります。この状態では、前にも説明したとおり、各 DML 文ごとにトランザクションが自動コミットされるモードなので、ロックがかからなくなってしまいます。従って、排他制御に問題がでてくることになります。
バッチタスク「自動価格変更」の場合も、トランザクション開始のタイミングが、バッチのデフォルトとして、レコード前処理の前であるところが異なりますが、トランザクションなしで動作するようになることは同じで、同じく排他制御の問題が起こります。

13. おわりに

本書では、Magic の SQL 対応機能について、ペットショップデモを題材に説明してきました。
SQL データベースとしてここでは MS-SQL Server を例にとって説明してきましたが、本書の内容は他の SQL データベース(Oracle や DB2/UDB など)にも応用が利くものです。

SQL データベースを使ったプログラムを設計・開発する上で、一番問題になるのはトランザクションの利用についてのことで、本書では、Pervasvie から移行してきた開発者が一番ひっかかる内容、すなわちトランザクションとロック関係について重点的に説明してきました。
この他にも、トランザクションについては、テーブルレベルのロック、パフォーマンスとの兼ね合い、データベース管理システムでのチューニングに関する問題などいろいろなトピックがあるのですが、本書の範囲を超えてしまうので、ここでは扱いません。
また、Magic の SQL 対応機能として、任意の DML 文を指定して実行することのできる埋め込み SQL の機能や、複雑な条件文を指定できる SQL WHERE 句、Magic 独自の高度なトランザクション機能である遅延トランザクションなどがありますが、これも紙面の都合で割愛いたしました。
これらのより高度なトピックについては、弊社のセミナーコース、自習テキスト、サポートセンターの技術資料などではより広く深い範囲で扱っておりますので、ぜひご利用ください。

SQL データベースは、大規模なエンタープライズ基幹システムなどでは必ずといってよいほど使われるものですので、Magic の持つ SQL 対応機能を十分に生かすことにより、システム開発の範囲が大きくひろがるものと確信しております。今後とも、Magic eBusiness Platform V9Plus をご愛用くださいますようお願い申し上げます。

14. 参考資料： 移行時の注意事項

ここでは参考情報として、Pervasive.SQL(リレーショナルアクセス)対応のアプリケーションから、MS-SQL対応アプリケーションに移行するためのポイントをまとめました。本書では説明していない機能も多数出てきますが、それらについてはリファレンスマニュアルやセミナーなどを通して習得してください。

14.1. テーブルリポジトリに関する変更

14.1.1. デフォルト値の違い

テーブルリポジトリとプログラムの各特性に設定されるデフォルト値は、DBMS ごとに細かな違いがあり、それらは、テーブルリポジトリの各テーブルのデータベース欄の設定時、または変更時に決定されます。これらはアプリケーションの特徴によって、ユーザが変更できるように設計されていますが、一つひとつの仕様を正しく理解せずに変更すると、意図した動作が行われないことがあります。
したがって、最初にデータベースを変更する際は、テーブルのデータベース欄を一つひとつ変更していただき、必要に応じて手動で設定していただくことを推奨いたします。

14.1.2. DB テーブル

Pervasive.SQL のデータベースは、テーブル名を空白にしてもエラーになりませんが、MS-SQL のデータベースを定義すると、テーブル名を必ず定義する必要があります。更にテーブル名は、MS-SQL のテーブル名の規約に沿って命名する必要があります。サイズはデータベース名、スキーマ名を含めて、128 バイトまでです。使用できる文字も決められており、例えば.(ドット)はスキーマ名とテーブル名の区切文字として認識されるため、テーブル名に含めることはできません。すなわち「～.dat」という名前は、すべて.を除く必要があります。記号は@_$#が使用できますが、先頭に@,@@,#,##をつけると特別な意味がありますので、通常は使用しないことをお勧めします。更に MS-SQL で決められた予約語(CHECK, ORDER など)も避ける必要があります。詳しくは、MS-SQL の SQL リファレンス等を参照してください。命名の規約は、DB カラム名、DB インデックス名にもあてはまります。

14.1.3. テーブルの存在チェック

通常プログラムでテーブルアクセス時にテーブルが存在しなかった場合、自動的にテーブルを作成します。MS-SQL データベースにアクセスするときには、これらの処理のために SELECT コマンドと CREATE コマンドを発行しますが、アプリケーション環境でテーブルが存在することがわかっている場合は、これらの処理はパフォーマンスを劣化させる原因になります。
アクセス時に作成を行う必要がないテーブルに対しては、テーブル特性のテーブル存在チェックパラメータを No にし、すべてのテーブルでその必要がない場合、データベース特性のテーブル存在チェックパラメータを No にし、テーブル存在チェックパラメータを[テーブル特性に依存]とすることを推奨します。
逆に存在しないテーブルに対して、この設定のままアクセスすると、MS-SQL からエラーが返りますので注意が必要です。

14.1.4. カラム／DB カラム名

Pervasive の物理的なテーブルにはカラムの名前はありませんが、MS-SQL のテーブルには存在します。テーブルリポジトリの通常のカラムの名前はアプリケーションの中だけで有効な名前であり、MS-SQL のテーブルのカラム名は、[カラム特性]／[DB カラム名]で設定します。つまり、この DB カラム名は、MS-SQL のテーブルのカラム名に対応するため、両者に相違があるか、MS-SQL のカラム名の規約に違反するか、1 つのテーブルで同じ名前が複数設定されていると正しくアクセスできません。
また、開発時には次の 3 つの場合、自動的にカラムの名前が DB カラム名として設定されます。(1) 新規にテーブルを定義したとき (2) リポジトリ入力したとき (3) Pervasive.SQL から MS-SQL に変更したとき。
したがってカラムの名前が規約に違反していた場合、不正な DB カラム名が設定されることがあるので、確認する必要があります。

14.1.5. カラム／カラムタイプ

Pervasive.SQL からの移行時には設定されませんが、あらかじめ作成された MS-SQL テーブルを利用する場合、あるいは項目の型に対応したカラムタイプ以外のタイプを利用する場合、この欄に MS-SQL のカラム

タイプを指定します。

14.1.6. 日付型カラム

Pervasive.SQL では､内部では数値のデータ(0001/01/01 を基点とする日数)ですが､デフォルトでは MS-SQL の DATETIME タイプとして定義されます。この場合、存在しない日付(0000/00/00, 0000/01/01 など)がデータとして格納できないため、システム上 0000/00/00 等を日付データとして利用している場合は、次のような方法で対応できます。

① カラムタイプに CHAR(8)と記述して、アプリケーションでは日付型データ、MS-SQL のカラムでは CHAR 型として格納する。
② NULL 値許可 Yes として、NULL 計算値に 0000/00/00 とする。

ただし②の場合、NULL 値としての細かな仕様を理解しておく必要があります。(14.3.1「NULL 値」(113 ページ)を参照してください)

14.1.7. インデックス／仮想キーの設定

Pervasive.SQL では常にデータファイルの物理的なキーとして定義され、レコードの位置情報を管理していますが、MS-SQL では、テーブルとは別のオブジェクトとしてインデックスが存在します。したがって、テーブルの作成と削除、あるいはレコードの挿入と削除を頻繁に行うテーブルでは、インデックスが多いほどパフォーマンスに影響します。したがって、アプリケーションで使用するものの、必ずしも MS-SQL のインデックスとして必要ではないと思われるものは、[インデックス特性]／[タイプ]で仮想キーに設定します。
それを判断する指針は次の点を参考にしてください。

- ユニークインデックスに設定されていないもの。またはインデックスの重複チェックをアプリケーションで行う必要がないもの。
- データ数が少ないもの。
- そのインデックスを使う処理が高いパフォーマンスを必要としないもの
- これらの条件を満たすインデックスは、[インデックス特性]／[タイプ]で仮想キーに変更し、既にインデックスが作成されている場合は MS-SQL ツール等で削除します。

参考：
仮想キーをプログラムのインデックスに設定した場合、Magic では範囲や位置付処理、または並び順を制御する際に生成される SELECT 文は、実キーと何ら変わることはありません。動作が変更されるとすれば、MS-SQL サーバの中で、その SQL 文を解析して結果を返すプロセスの中でインデックスを使わずに実行されることです。アプリケーションのパフォーマンスを考慮する場合には、MS-SQL のパフォーマンスチューニングの手法を習得した上でインデックスを設計してください。

14.1.8. DB インデックス名

MS-SQL のインデックスは、テーブルとは別のオブジェクトとして管理されているため、テーブルやカラムと同様の注意が必要です。インデックス名の規約はテーブル名と同じです。開発時には次の 3 つの場合、自動的にインデックスの名前が DB インデックス名として設定されます。

- 新規にテーブルを定義したとき
- リポジトリ入力したとき
- Pervasive.SQL から MS-SQL に変更したとき。

したがってインデックスの名前が MS-SQL の規約に反する場合､不正な DB インデックス名が設定されるため、正しくアクセスできませんので、インデックスの名前と DB インデックス名を 1 つひとつ確認する必要があります。また 1 つのテーブルの中で同じインデックス名が重ならないように注意が必要です。

14.1.9. 重複不可インデックスの定義

Pervasive.SQL では、テーブルに重複不可インデックスがなくてもエラーは起こりませんが、MS-SQL データベースでは最低１つ重複不可インデックスが必要です。このため、重複不可になるカラムを組み合わせるか、新しいカラムを追加して新たな重複不可インデックスを作成する必要があります。

参考：
重複不可データを自動作成する方法として、カラムタイプに ... IDENTITY を付加して、MS-SQL の IDENTITY を利用することもできます。

14.1.10. セグメントのサイズ

Pervasive.SQL では、インデックス(キー)はレコードの位置とサイズで指定されるので、文字型項目の場合、セグメントのサイズもテーブルリポジトリで指定することができます。しかし、MS-SQL ではインデックスのセグメントは「カラム」の集まりなので「サイズ」は必ずカラムサイズと同じでなければなりません。もし、Pervasive.SQL のテーブル定義で「サイズ」をカラムのサイズより小さくしている場合、修正する必要があります。

14.1.11. 論理型カラムのインデックス

論理型のカラムをインデックスのセグメント項目として定義されたテーブルはエラーになることがあります。これは、デフォルトの論理型のカラムは、MS-SQL の BIT タイプに割り当てられますが、BIT タイプはインデックス項目に定義できないためです。これを回避する方法の１つは、[カラム特性]／[サイズ]を 2 に変更することです。これにより、論理型カラムが BIT タイプでなく SMALLINT タイプに割り当てられますので、インデックスのセグメント項目に設定することができます。

14.1.12. 重複不可データのチェック

Pervasive.SQL のテーブルでは重複不可インデックスを定義した場合には、実行時に重複したデータを入力した場合、無条件に重複エラーが発生しましたが、MS-SQL ではエラーが発生しなかったり、エラーの発生するタイミングが異なることがあります。

◆ 仮想キーでは重複エラーが発生しない

仮想キーは、アプリケーションの中だけの定義であり、MS-SQL にユニークインデックスが定義されていません。したがって MS-SQL のデータ処理の際には重複エラーが発生しませんので、Magic アプリケーションでもエラーを発生させることができません。レコードの修正時や登録時に重複データのチェックをプログラムで行わず、Magic の機能でチェックする場合は、実キーとして定義する必要があります。

◆ 同一レコードのカラム間の移動時に重複エラーが発生しない

Pervasive.SQL では、インデックスセグメントのカラムに重複した値を入力して、次のカラムに移動した直後に重複エラーが発生しますが、MS-SQL のデフォルトの設定では、すべてのカラムを入力してレコードを格納する直前にエラーが発生します。これは、Pervasieve ではクライアントエンジンがあることを想定して、カラム間の移動時に Pervasive.SQL エンジンに重複データのチェックを行いますが、MS-SQL ではクライアントサーバ環境を想定して、そのパフォーマンスの劣化を考慮し、サーバエンジンへの余計な重複データのチェックを抑止しているためです。
パフォーマンスよりも、Pervasive.SQL と同等の動作を優先する場合は、[設定]／[データベース]の MS-SQL に対するデータベース特性にある[データーベース情報]に CHECK_KEY=Y を設定してください。

14.2. プログラムに関する変更

14.2.1. レコードアクセス

レコードアクセス処理に関して、Pervasive.SQL から MS-SQL にそのまま移行すると、次のようなケースでエラーになることがあります。

◆ レコード登録時にテーブルが存在しないとき

「テーブルの存在チェック」で述べたように、テーブルの作成とレコードの作成は異なるレベルの処理です。存在しないテーブルに対してレコード登録する場合、テーブル作成とレコード作成の２つの処理を実行しなければなりませんが、次の場合、テーブル作成が行うことができませんので、実行環境を確認する必要があります。

- 「テーブルの存在チェック」が No の場合。これは Magic の設定を変更します。
- MS-SQL サーバの権限により、サーバログオンユーザに CREATETABLE 権限がない場合。これは CREATETABLE 権限を付与するか、許可されない場合、あらかじめ別のアカウントで作成しておきます。
- トランザクションが開始されたプログラムからコールした子プログラムでテーブルを作成しようとした場合。この場合、トランザクションを開始するタスクで新規作成します。（トランザクションについては後述します。）

◆ レコード登録時にすべてのカラムがセレクトコマンドに定義されていないとき

テーブルに定義されたすべてのカラムをセレクトコマンドとして定義しないで、レコード登録を行うプログラムでエラーになることがあります。 Magic では、レコード登録を行う Insert 文を生成する際、セレクトコマンドに定義されたカラムのみ付加されるため、それ以外のカラムに対して、MS-SQL が NULL 値、あるいはデータベースデフォルト値を格納しようとします。Pervasive.SQL から移行したテーブル定義のデフォルトの状態では、どちらも許可されていないためにエラーが発生します。これを解決するためには、次のいずれかの変更を行います。

- レコード登録を行うプログラムで、すべてのカラムをセレクトコマンドに定義する。
- [カラム特性]／[NULL 値可]を Yes に変更して、物理テーブルの定義を変更する。（NULL 値可のカラムについて、下記「NULL 値」を参照してください。）
- [カラム特性]／[データベースデフォルト値]に任意の値を設定して、物理テーブルの定義を変更する。

◆ [タスク環境]／[DB テーブル]／[式]でテーブル名を設定しているとき

テーブル名は MS-SQL の規約にあったテーブル名に変更する必要があります。

14.2.2. ロックとトランザクション

◆ ロック

Pervasive.SQL ではトランザクションの設定を行わずにアプリケーションの作成できますが、MS-SQL の場合は、MS-SQL でレコードロックを行う場合、必ずトランザクション処理を開始する必要があるため、トランザクションを常に考慮する必要があります。したがって、[タスク特性]／[ロック方式]が「入力時」の場合、同じダイアログにある[トランザクション開始]が入力時より前の時期に設定することが基本です。ロックトランザクションに関するその他の詳細はリファレンスマニュアルの「13 章 データ管理」を参照してください。

◆ トランザクション

複数のタスクで構成されたプログラムでは、トランザクションの設定によっては、予期しないエラーが発生したり､正しくデータ更新やロールバックが行われないことがあります｡このようなことを未然に防ぐため、トランザクション設定のポイントを以下に挙げます。

- 「物理」と「遅延」のどちらか一方のトランザクションモードにあわせる： 既存のアプリケーションを移行する場合は、同じモードが引き継がれますが、新規作成のデフォルトではオンラインタスクが「遅延」､バッチタスクが「物理」に設定されます。このままオンラインとバッチを組み合わせてプログラムを構成すると、これらのモードが混在してしまうことがあります。この場合､「物理」のタスクから「遅延」のタスクをコールするとエラーになりますので、どちらかのモードにあわせることが必要です。
- トランザクションの開始と終了(コミット)、およびネストを把握する： タスクの構成が複雑な場合は、次のような順序でプログラムを確認して、トランザクションの開始と終了がいつ行われるかを確認します。
- 親のプログラムの DB テーブルの設定とトランザクション開始と終了を確認します。DB テーブルの設定がないと、トランザクションの開始が行われないことも考慮します。
- プログラムの中のコールタスクおよびコールプログラムを確認し、それらがトランザクションの開始前なのか、トランザクションの中なのか、トランザクションの終了後なのかを確認します。例えば、親がレコード前処理に開始した場合、タスク前処理テーブルのコールコマンドでのタスクは独立した別のトランザクションとして処理されますが、レコード前処理のタスクは親と同じトランザクションの中で処理されます。更に、ユーザイベントハンドラの処理テーブルの場合は、デフォルトでは同じトランザクションの中で処理されますが、イベントの[強制終了]パラメータが「レコード」の場合はレコード後処理の後でコールコマンドが実行されます。これによりトランザクションが終了(コミット)してから、コールプログラムが実行されることがありますので、特に注意が必要です。
- トランザクション中にコールされるタスクのトランザクションモードおよび開始パラメータにより、その中でデータベースへの更新が子の終了時か、親の終了時かを確認する。
- トランザクションの開始前、および終了後にコールされるタスクが、親のトランザクションとは独立したトランザクションであることを確認する。

14.2.3. 関数

データベース処理を行う次のような関数の処理は確認が必要です。

- IO 関数(IODEL など)によって、Pervasive.SQL のテーブルを操作している場合、DB 関数に置き換えるなどの変更が必要です。
- DB 関数(DBDEL など)で第 2 引数にファイル名を指定している場合、MS-SQL のテーブル名への変更が必要です。
- OS コマンドや CALL UDF、CALLDLL などによって、Pervasive.SQL のユーティリティを実行している場合は、処理を検討する必要があります。

14.3. その他の違い

14.3.1. NULL 値

Pervasive.SQL ではカラムにスペースや 0 を挿入すると、常に 20h や 0 を格納しますが、MS-SQL では NULL 値としてデータを処理することができます。NULL 値の格納を許可するか否かはテーブル作成時に決められます。Magic でテーブルを作成する場合、カラム特性の NULL 値可パラメータで制御できます。更にこのパラメータのデフォルトは[設定]／[DBMS]の NULL パラメータで制御されます。
NULL 値可 Yes のカラムは、レコード登録時に格納する値がないときにはセレクトコマンドとして定義する必要がない、データベースに余計な領域を確保しない、などのメリットがあります。しかしながら、次のような動作の違いが発生することがありますので、十分な検討が必要です。

- セレクトコマンドの範囲や位置付けに 0 や"(スペース)を設定していた場合、NULL 値のデータがこの条件に満たさないので、今まで行われたレコードの処理が行われないことがあります。
- (数値項目)=0、(文字項目)=" などの定義式の結果が、項目が NULL 値の場合に'FALSE'LOG が返ります。
- (数値項目)+1、(文字項目)&'*' などの定義式の結果が、1 や'*'でなく、NULL 値が返ることがあります。
- NULL 値を含むカラムで構成されたインデックスで絞込みをしたバッチ処理のパフォーマンスが相対的に悪くなります。

14.3.2. ソート順について

MS-SQL のデフォルトのデータベース設定では、照合順序（ソート順）について、大文字と小文字、ひらがなとカタカナの区別がありません。したがって異なる文字種の範囲大小で絞り込むと、Pervasive.SQL とは異なるデータが取得されることがあります。これを回避して Pervasive.SQL と同じ動作にするには、MS-SQL のデータベースの照合順序を「バイナリ順」に変更します。

Magic eDeveloper™ V9 Plus

Magic eBusiness Platform V9Plus

チュートリアル SQL編

第1版　2005年12月31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/