eDeveloper™ V10

Magic eDeveloper V10
チュートリアル
SQL 編

MAGIC
Magic Software Japan K.K.

Enabling Business with Superior Technology

本書に記載の内容は、将来予告なしに変更することがあります。これらの情報について MSE（Magic Software Enterprises Ltd.）および MSJ（Magic Software Japan K.K.）は、いかなる責任も負いません。
本書の内容につきましては、万全を期して作成していますが、万一誤りや不正確な記述があったとしても、MSE および MSJ はいかなる責任、債務も負いません。
MSE および MSJ は、この製品の商業価値や特定の用途に対する適合性の保証を含め、この製品に関する明示的、あるいは黙示的な保証は一切していません。
本書に記載のソフトウェアは、製品の使用許諾契約書に記載の条件に同意をされたライセンス所有者に対してのみ供給されるものです。同ライセンスの許可する条件のもとでのみ、使用または複製することが許されます。当該ライセンスが特に許可している場合を除いては、いかなる媒体へも複製することはできません。
ライセンス所有者自身の個人使用目的で行う場合を除き、MSE または MSJ の書面による事前の許可なしでは、いかなる条件下でも、本書のいかなる部分も、電子的、機械的、撮影、録音、その他のいかなる手段によっても、コピー、検索システムへの記憶、電送を行うことはできません。
サードパーティ各社商標の引用は、MSE および MSJ の製品に対する互換性に関しての情報提供のみを目的としてなされるものです。
本書において、説明のためにサンプルとして引用されている会社名、製品名、住所、人物は、特に断り書きのないかぎり、すべて架空のものであり、実在のものについて言及するものではありません。
Magic は Magic Software Enterprises Ltd. のイスラエルその他の国での商標または登録商標です。
Magic eDeveloper、Magic Client および Magic Application Server は Magic Software Japan K.K. の商標です。
Pervasive.SQL は Pervasive Software, Inc. の商標です。
Microsoft および FrontPage は、Microsoft Corporation の登録商標です。また、Windows, WindowsNT および ActiveX は Microsoft Corporation の商標です。
一般に、会社名、製品名は各社の商標または登録商標です。
MSE および MSJ は、本製品の使用またはその使用によってもたらされる結果に関する保証や告知は一切していません。この製品のもたらす結果およびパフォーマンスに関する危険性は、すべてユーザが責任を負うものとします。
この製品を使用した結果、または使用不可能な結果生じた間接的、偶発的、副次的な損害（営利損失、業務中断、業務情報の損失などの損害も含む）に関し、事前に損害の可能性が勧告されていた場合であっても、MSE および MSJ、その管理者、役員、従業員、代理人は、いかなる場合にも一切責任を負いません。

初版　　2007年7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、SQL データベース管理システム(以下 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、簡単な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。

SQL 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 MSSQL Server 2005 Express 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。

注意: 本書は、Magic Studio V10 (V10.1SP2) に基づいて書かれています。体験版や最新バージョンでは画面や操作性などに多少の違いがある可能性があります。

1.1.目標

本書の目標は、次の通りです。

- MSSQL Server 2005 (Express Edition)を例にとり、SQL データベースの設定方法を理解する。
- ペットショップデモを題材として、マルチユーザ環境や、SQL データベースを使った場合の問題点について、現象とその原因を詳しく理解する。
- SQL データベースを使って、マルチユーザ環境でもちゃんと利用できるようにするには、アプリケーションをどのように変更すべきかを理解し、実際に修正を施す。
- 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ことを確認するために、さまざまなテストパターンでテストを行う。

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 MSSQL 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知識を持っていることを仮定しています。具体的には、チュートリアル Getting Started を勉強し、簡単な受注入力画面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 Studio V10 をインストールしておいてください。製品版 の Magic Studio V10 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 Studio 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、MSSQL Server 2005 Express Edition を使います。もしご利用になる PC に MSSQL Server を利用する環境がなければ、「第 2 章 SQL 利用準備」(10ページ)を参考にして、MSSQL Server 2005 Express Edition を入手し、インストールしてください。

もし SQL データベースとして、MSSQL 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。

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。

本書では MSSQL 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ヘルプにありますので、以下の章も参考にしてください。

機能	リファレンスヘルプの箇所
トランザクション一般に関する話題	データ管理
SQL サポート全体	SQL に関する考慮事項
マルチユーザ環境での考慮事項	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	データソース
埋め込み SQL	データビューエディッタ → 埋め込み SQL
SQL WHERE 句	プログラム → 範囲と位置付ウィンドウ → SQL Where 句
遅延トランザクション	データ管理 → 遅延トランザクション

各 SQL データベースに固有な詳細情報については、Readme.chm の「データベース固有の追加情報」も参照してください。

2. SQL 利用準備

本書では、SQL DBMS として MSSQL Server を利用します。MSSQL Server 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。
MSSQL Server をお持ちでない場合には、以下のいずれかの方法で、MSSQL Server 2005 Express Edition、あるいは期間限定の評価版を入手することができます。

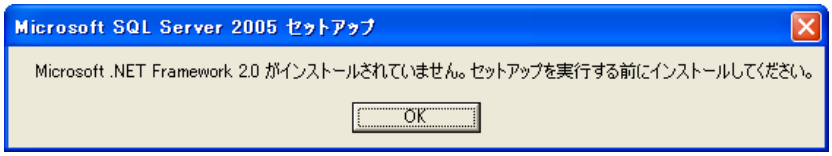
- Magic 製品 CD から MSSQL Server 2005 をインストール
- Microsoft サイトからダウンロード

それぞれについて、以下に説明します。

2.1.Magic 製品 CD から MSSQL Server 2005 をインストール

Magic Studio V10 の製品 CD-ROM をお持ちであれば、MSSQL Server 2005 Express Edition がバンドルされているので、CD-ROM からインストールすることができます。

MSSQL Server 2005 Express Edition をインストールするには、Microsoft .NET Framework 2.0 が必要です。PC に.NET Framework 2.0 がインストールされていない場合、右図のようなエラーメッセージが表示されます。



製品 CDROM の CD2 にある MSDE2 ディレクトリに、.NET Framework インストーラ dotnetfx.exe がありますので、これを実行するか、あるいは Windows Update を実行するなどして、.NET Framework 2.0 をインストールしておいてください。

本書では、デフォルトの設定のままインストールを行います。この場合、設定は次のようになります。

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

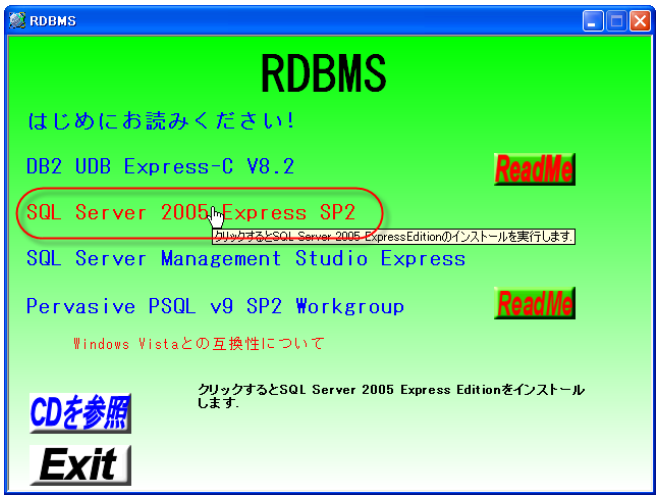
これ以外の設定にしたい場合には、インストールの途中「登録情報」画面で「詳細構成オプションを非表示にする」のチェックをはずして、詳細パラメータ入力ができるようにしてください。また、接続に関する設定は、インストール後、SQL Server 2005 セキュリティ構成 ユーティリティで行えます。

2.1.1.MSSQL Server 2005 Express のインストール

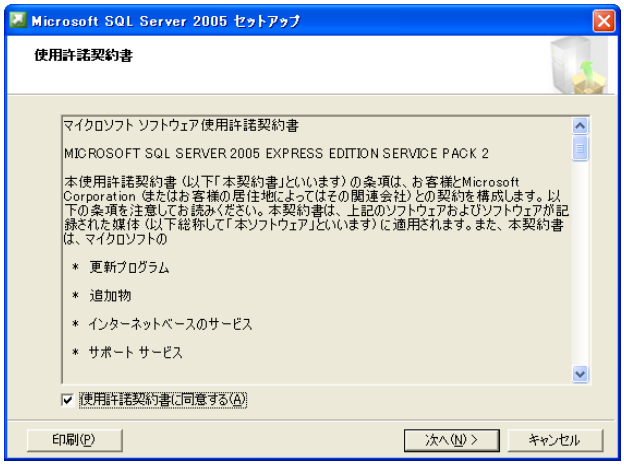
1. Magic 製品の CD-ROM の CD2 を CD-ROM ドライブにセットします。自動的にインストーラが開始されます。

自動的に開始されない場合には、CDROM ドライブのルートにある setup.exe を起動してください。

2. 「SQL Server 2005 Express SP2」をクリックしてください。インストーラが開始します。



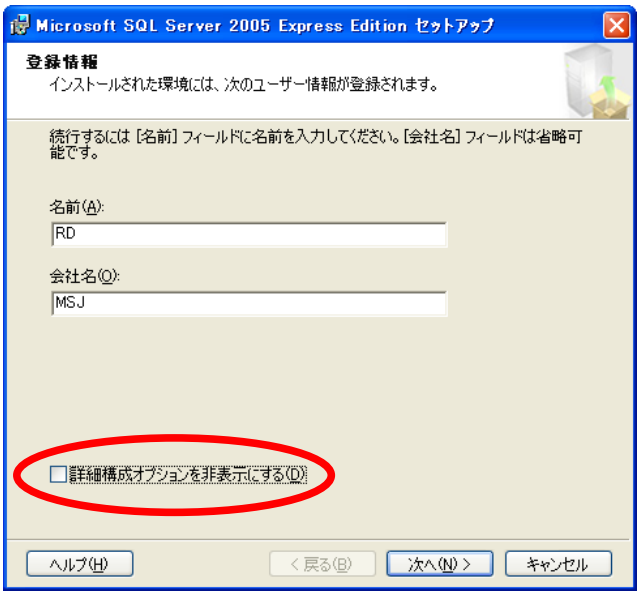
3. インストーラが開始します。オプションはすべてデフォルトのままでインストールしてください。



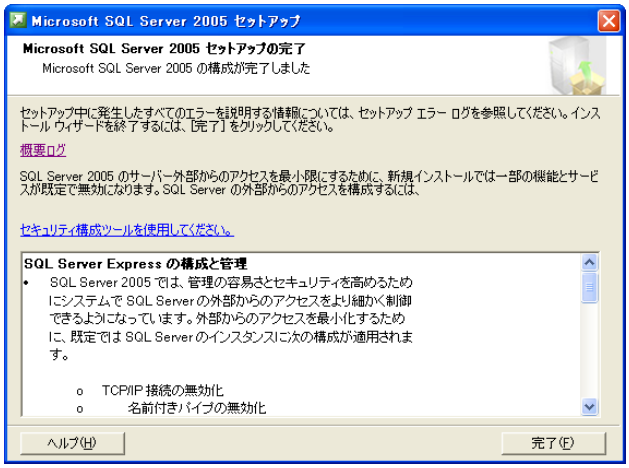
4. しばらく進むと、「登録情報」画面が出てきます。デフォルトのインストールでよい場合には、そのまま「次へ(N)」ボタンを押します。

インストールパラメータを変更したい場合には、「詳細構成オプションを非表示にする」のチェックをはずして、「次へ(N)」ボタンを押してください。詳細画面が表示されるようになり、適宜パラメータを変更できます。

本書では、インストールパラメータについての説明は省略します。マイクロソフト社の技術資料などを参照してください。



5. インストールが完了したら、「完了(F)」ボタンを押してください。

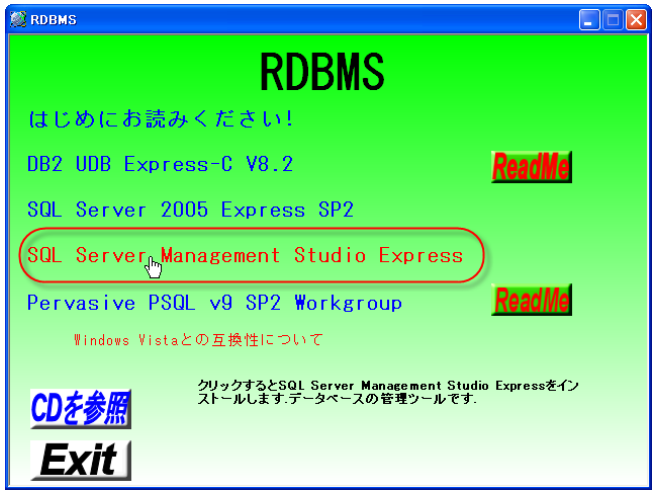


以上で MSSQL Server 2005 Express Edition のインストールが完了です。

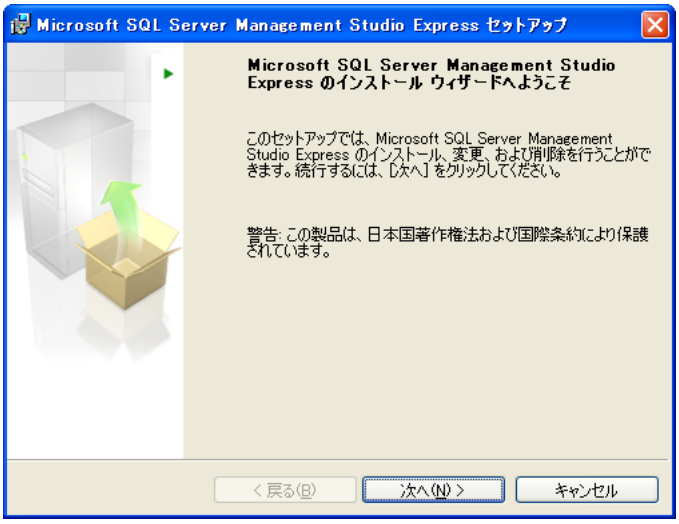
2.1.2.SQL Server Management Studio Express のインストール

MSSQL Server 2005 Express Edition には管理ツールが添付されていませんので、続けて SQL Server Management Studio Express もインストールすることをお勧めします。SQL Server Management Studio Express のインストーラも Magic Studio V10 の CD-ROM CD2 に収納されています。

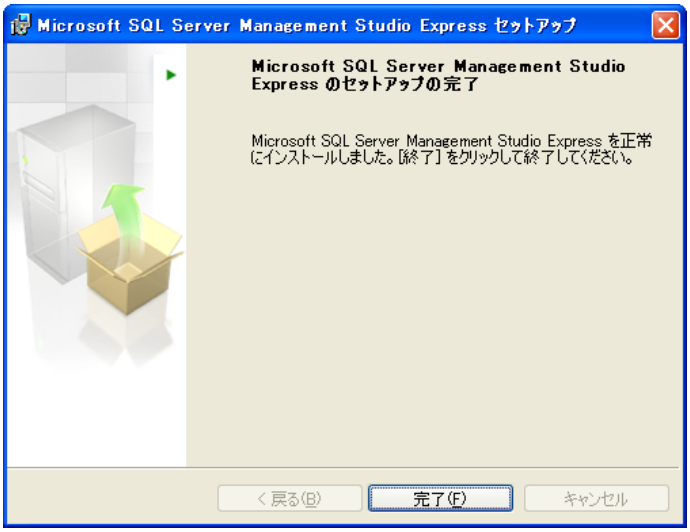
1. SQL Server Management Studio Express のインストーラを起動します。



2. あとはウィザードに従ってインストールを進めます。オプションはすべてデフォルトのままで構いません。



3. すべて完了したら、「完了(F)」ボタンを押します。



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、MSSQL Server 2005 Express Edition あるいは期限限定の評価版 SQL Server 2005 Enterprise Edition を Microsoft 社のサイトから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

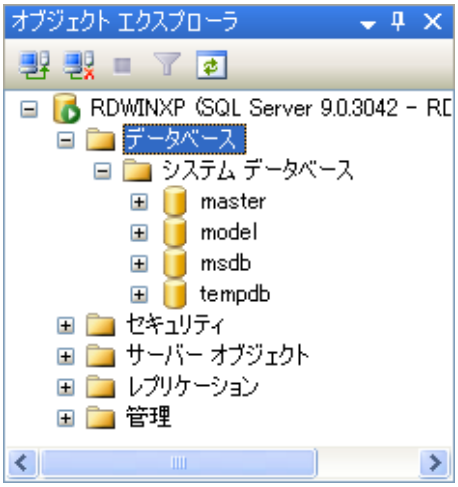
MSSQL Server をインストールした直後では、システムが使うデータベースしか登録されていませんので、テスト用に別途データベースを作成します。

2.3.1.SQL Server Management Studio Express で作成する方法

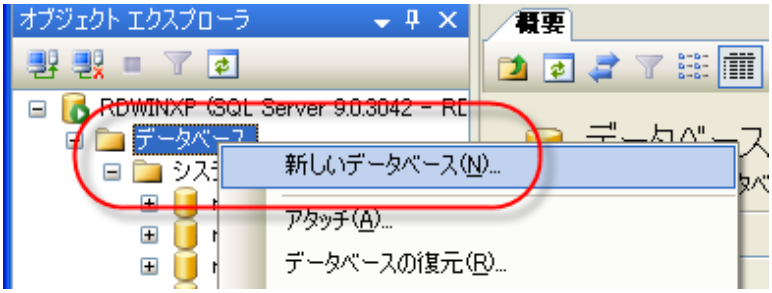
ここでは以下のようにして SQL Server Management Studio Express を使い、MAGIC という名前のデータベースを作成します。

1. SQL Server Management Studio Express を起動します。
2. 最初に「サーバへの接続」画面が出てきます。「サーバ名」には、「PC のホスト名¥SQLEXPRESS」がデフォルトで出ているはずですので、「接続(C)」ボタンを押します。

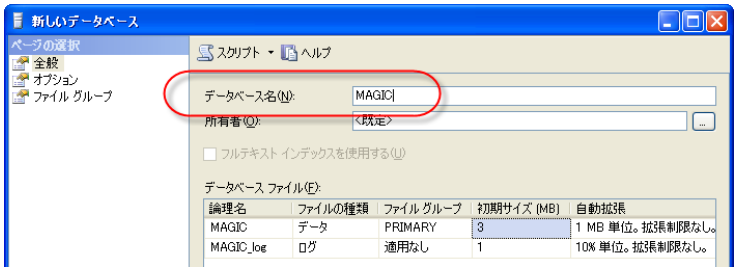
3. 接続できたら、「オブジェクトエクスプローラ」上で、データベースを開きます。初期状態では、システムデータベースのみが登録されています。



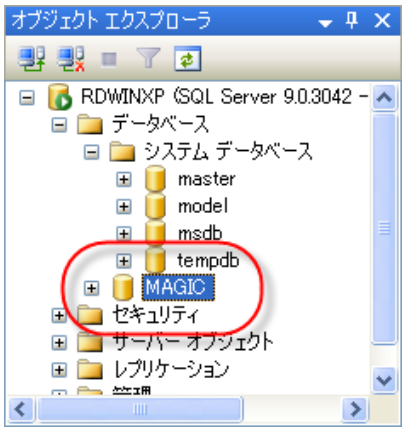
4. 「データベース」ノードで右クリックし、メニューから「新しいデータベース」を選択します。



5. 「新しいデータベース」画面が出てきます。「データベース名」として「MAGIC」と指定します。その他のパラメータは変更する必要はありません。



6. 「OK」ボタンを押すと、データベース MAGIC が作成され登録されます。



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

2.3.2.コマンドツール OSQL で作成する方法

SQL Server Management Studio Express を使わずに、コマンドラインツール OSQL を使ってデータベースを作成することもできます。

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。ここで、RDWINXP（青字）は、この PC のホスト名です。ご利用の PC の名前に合わせて置き換えてください。

```
C:¥>osql  -E  -S RDWINXP¥SQLEXPRESS
1> create database MAGIC
2> go
```

- サーバ名の指定「-S RDWINXP¥SQLEXPRESS」は、「-S (local)¥SQLEXPRESS」とすることも可能です。
- 上の方法で作成したデータベースのサイズは、データファイルが 3MB（拡張制限なし）、ログファイルが 1MB となります。本書で出てくるような小さなテーブルのみを使う場合にはこれでも十分ですが、予め大きなサイズを確保しておきたい場合には、次のように alter database コマンドを使います。

```
alter database MAGIC modify file (name='MAGIC', size=10MB)
alter database MAGIC modify file (name='MAGIC_log', size=10MB)
```

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。ここで、RDWINXP というのは、この PC のホスト名です。ご利用の PC の名前に合わせて置き換えてください。

```
C:¥> osql  -E  -S  RDWINXP¥SQLEXPRESS
```

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 MSSQL Server 設定

MSSQL Server のインストールが済んだので、次には Magic Studio で MSSQL Server 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

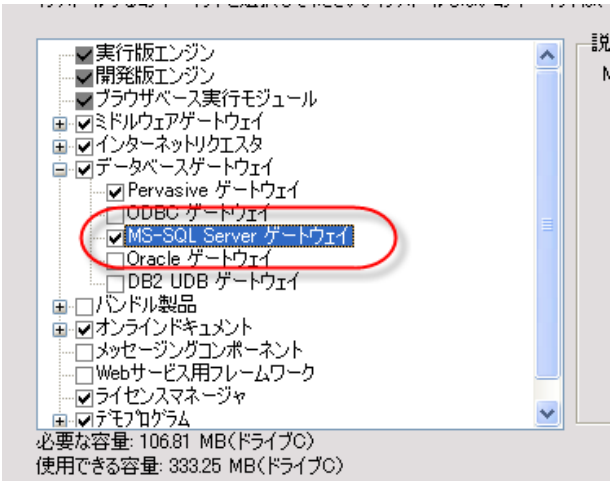
Magic から MS SQL Server にアクセスするには、Magic の MS SQL Server 用ゲートウェイをインストールし、Magic エンジンが実行時にロードできるようにしておく必要があります。

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 MSSQL Server ゲートウェイを選択します。
3. その他のオプションは、デフォルトのままで通常通りインストールを続けてください。

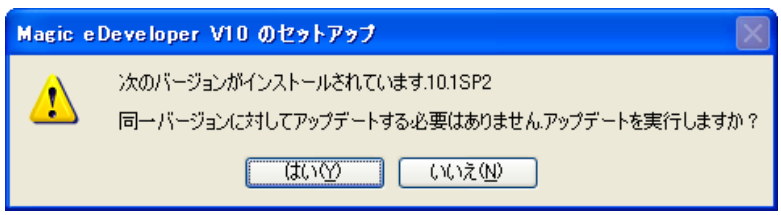


3.1.2.アップデートインストールの場合

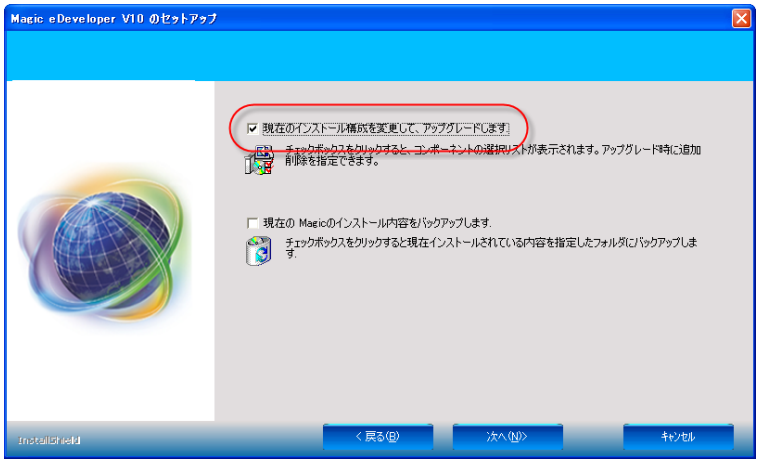
Magic を標準インストールでインストールした環境がすでにある場合には、MSSQL のゲートウェイがインストールされていないので、アップデートインストールでインストールする必要があります。

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。

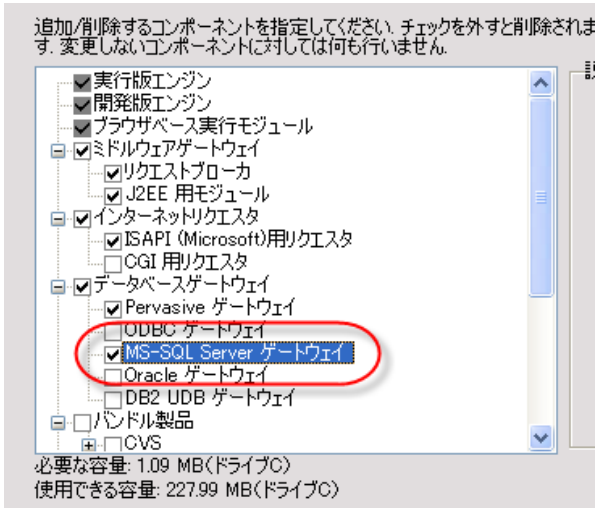
1. インストールしたときと同じようにインストーラを起動します。
   右図のような警告メッセージが出ますので、「はい(Y)」を押します。



2. 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。

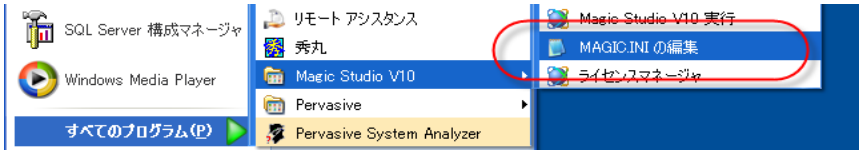


3. 「アップデート時のコンポーネント指定」画面で、データベースゲートウェイから MSSQL Server ゲートウェイを選択します。
4. 後は、通常通りインストールを続けてください。



アップデートインストールが完了すると、MSSQL Server ゲートウェイがインストールされています。次には、インストールされたゲートウェイモジュールを有効にするために、MAGIC.INI を編集します。

5. スタートメニュー → Magic Studio V10 → MAGIC.INI の編集 を選択します。



6. [MAGIC_GATEWAYS] セクションの MGDB20 のコメントを削除します。
7. ファイルを保存します。

```
[MAGIC_GATEWAYS]
;MGCOMM01=mgwsock.dll
MGDB00=Gateways¥MGbtrieve.dll
;MGDB06=Gateways¥MGdb2400.DLL
;MGDB13=Gateways¥MGoracle.dll
;MGDB16=Gateways¥MGeac32.dll
;MGDB18=Gateways¥MGdb2.DLL
;MGDB19=Gateways¥MGobc.dll
MGDB20=Gateways¥MGmssql.dll
MGDB21=Gateways¥MGmemory.dll
```

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。

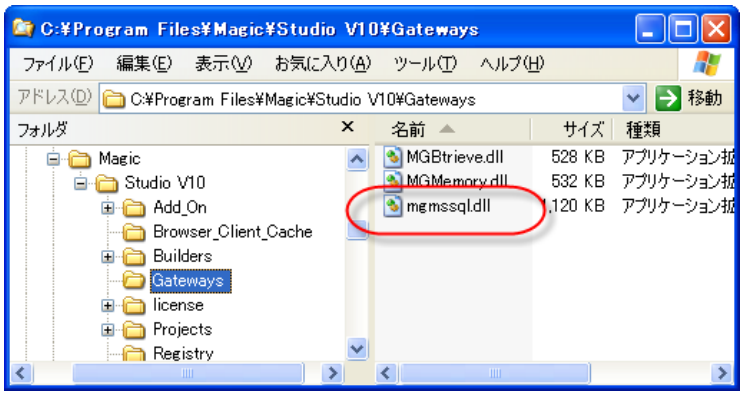
1. Magic を起動します。
2. メニュー「ヘルプ(H)」から「Magic 情報(M)」を選びます。
3. バージョン情報のダイアログ(右図)が表示されますが、ロードモジュール一覧の中に「MicrosoftSQLServer」があることを確認してください。あれば MSSQL Server ゲートウェイが正常にロードされています。

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、MGmssql.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。MSSQL 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¥MGbtrieve.dll
;MGDB06=Gateways¥MGdb2400.DLL
;MGDB13=Gateways¥MGoracle.dll
;MGDB16=Gateways¥MGeac32.dll
;MGDB18=Gateways¥MGdb2.DLL
;MGDB19=Gateways¥MGobc.dll
MGDB20=Gateways¥MGmssql.dll
MGDB21=Gateways¥MGmemory.dll
```

3.2.DBMS テーブル

MSSQL 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 (MSSQL Server、Oracle、Pervasive.SQL、DB2UDB、など)ごとに、共通な設定を行うものです。MSSQL 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 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「オプション(O) → 設定(S) → DBMS(B)」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、Alt+Enter 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。

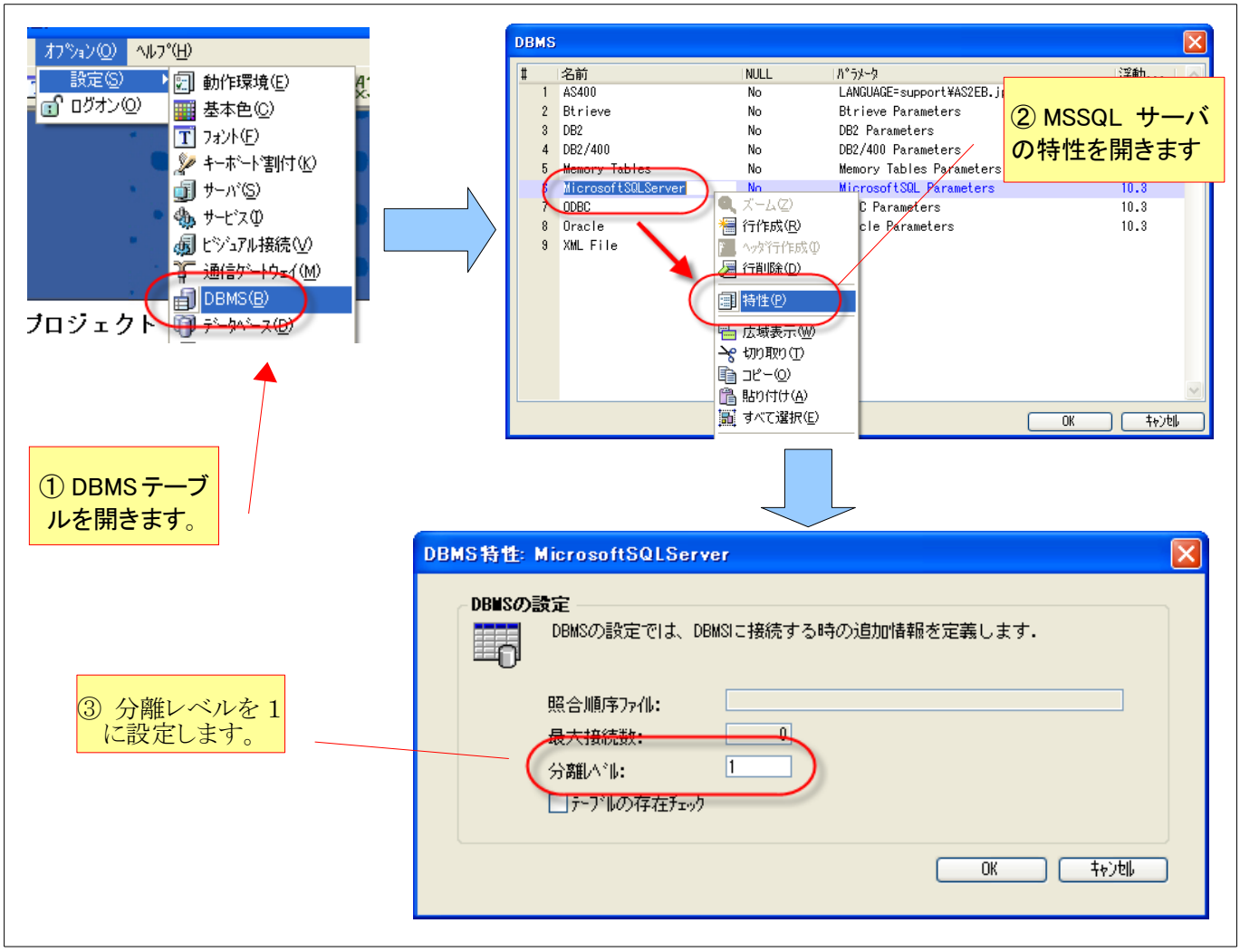


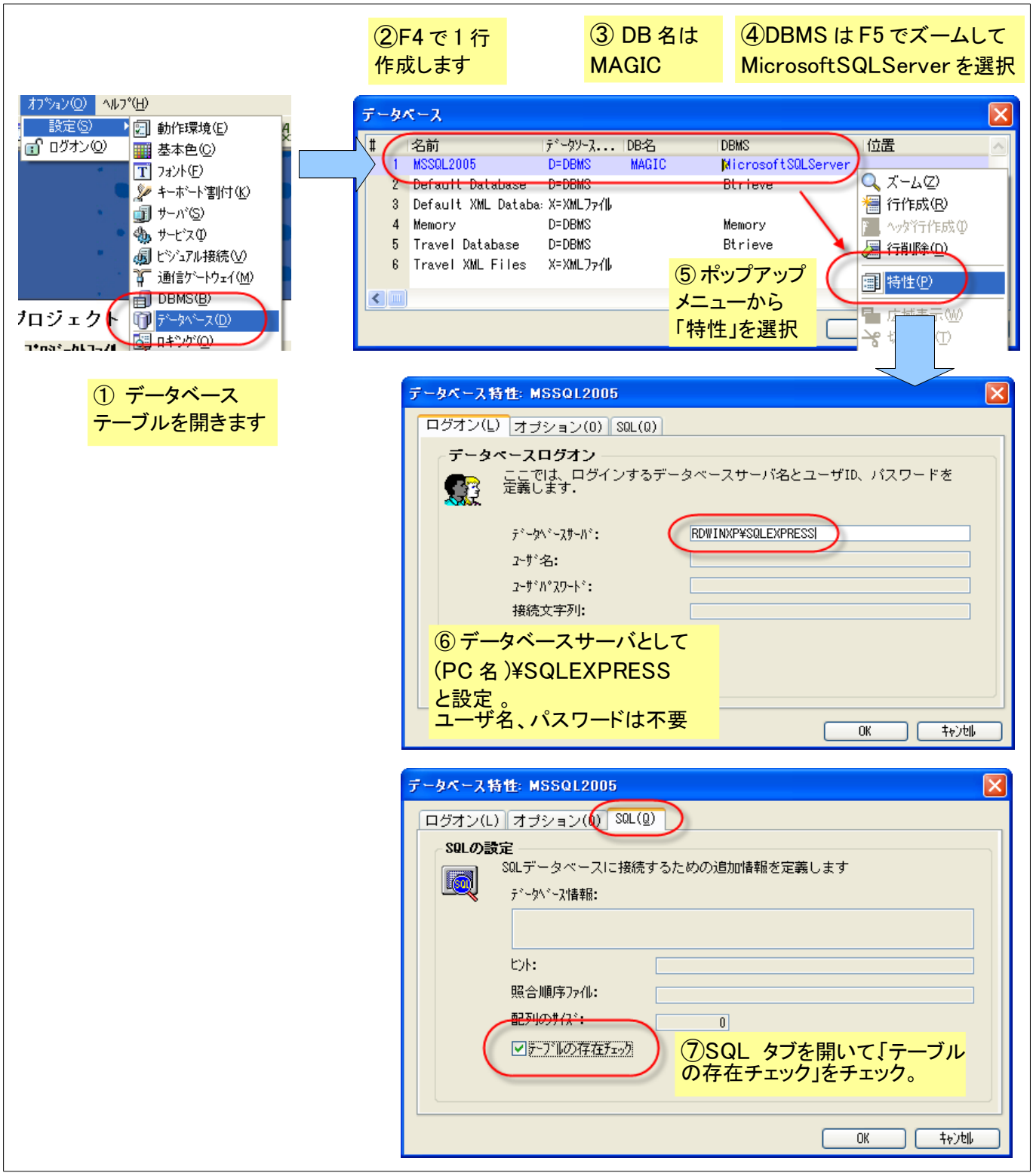
図 1 DBMS テーブルの設定

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」(73 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出る可能性があるので、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 MSSQL 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 2)。

図 2 データベーステーブルの設定



1. メニュー「オプション(O) → 設定(S) → 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。
   「名前」欄には「MSSQL2005」などとします。この名前は Magic のデータ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をつけます。
3. 「DB 名」欄には、作成したデータベース名 MAGIC とします。
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 MSSQL Server をアクセスするので、MicrosoftSQLServer を選択します。
5. 右マウスクリックでポップアップメニューを出し、「特性(P)」を選択します。データベース特性ダイアログが開きます。
6. 「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、「データベースサーバ」欄は 「(PC 名)¥(インスタンス名)」を指定します。
   第 2 章 「SQL 利用準備」 に従ってインストールした場合には、インスタンス名は「SQLEXPRESS」とします。
7. 「SQL(Q)」をタブを開き、「テーブルの存在チェック」にチェックを入れます。

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 からアクセスする

本書の最終目標は、ペットショップデモを MSSQL 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

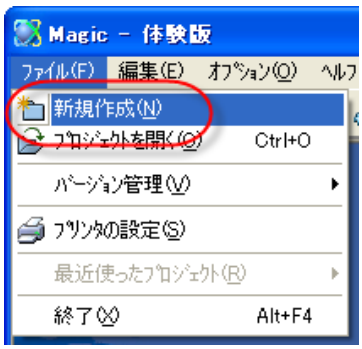
また、MSSQL Server を利用する場合の注意事項として、以下の点について説明します。

- 命名規則
- 名前の対応
- 一意インデックスの定義

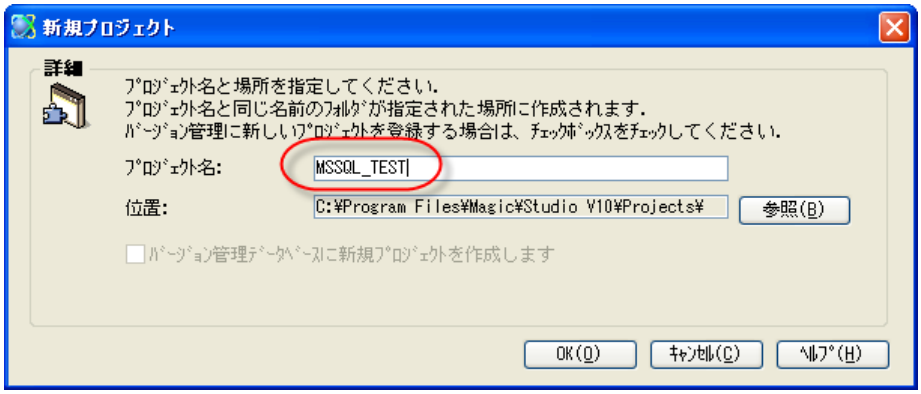
4.1.新規アプリケーションの作成

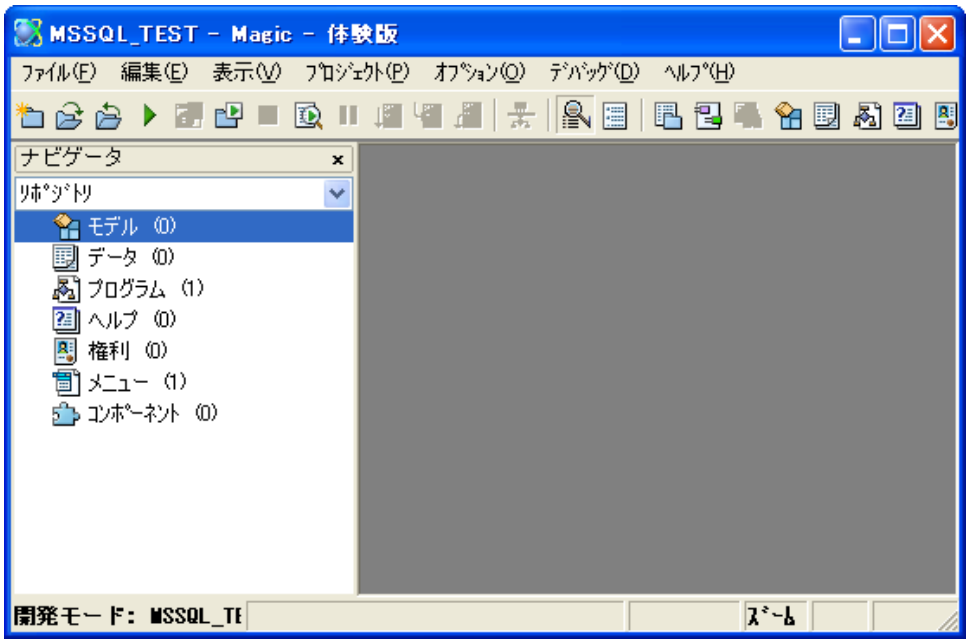
Magic で新規アプリケーションを作成するには、以下のようにします。

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、データリポジトリで、簡単なテーブルを新規定義しましょう。

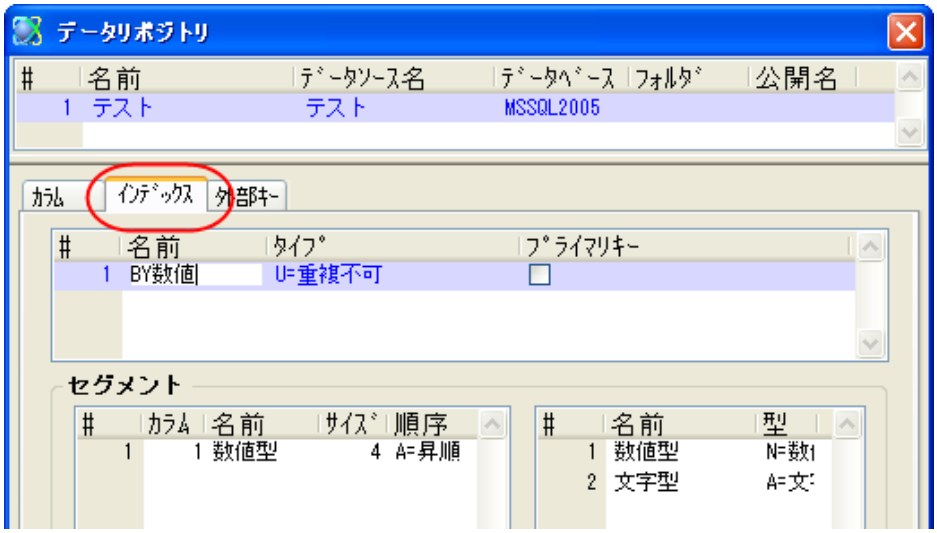
1. データリポジトリを開きます。メニュー「プロジェクト(P)」から「データ(B)」を選ぶか、Shift+F2 で開きます。
2. 次のようなテーブルを定義してください。

名前: テスト、 データソース名: テスト、 データベース: MSSQL2005			
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 のデータリポジトリでは、下図のように定義します。

- カラム定義

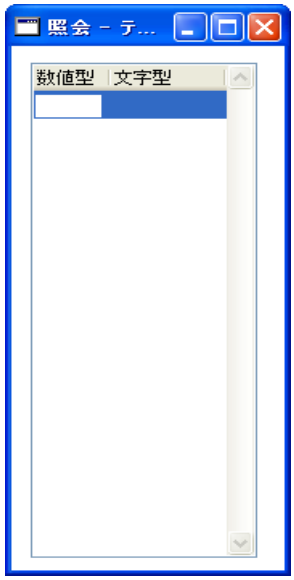
- インデックス定義



4.3.テーブル作成とデータ登録

データ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 MSSQL 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。

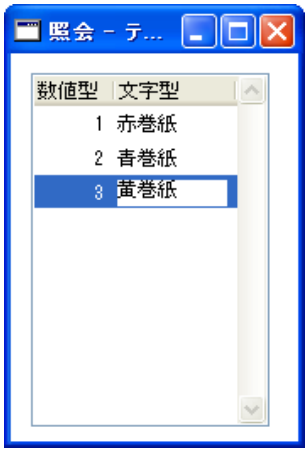


データリポジトリで、APG でテーブルを開いてみてください。最初はレコードが登録されていないので、空のテーブルが表示されます(右図)。この時点で、すでに MSSQL Server 上にテーブルは作成されています。

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

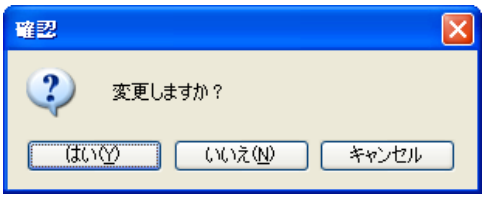
Magic のデータリポジトリの定義を変更すると、MSSQL Server 上のテーブルも自動的に変更されます。これは Magic の**自動データ再編成機能**です。

4.4.1.テーブルを変更してみる

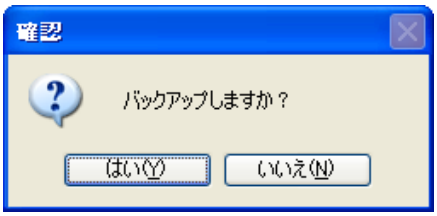
今の例で、文字型カラムを 10 文字から 20 文字に増やし、さらに、第 3 のカラムとして「英語」というカラムを追加しましょう。データ型は A=文字、書式は 5 です。

Enter キーを 2 回押してデータ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。

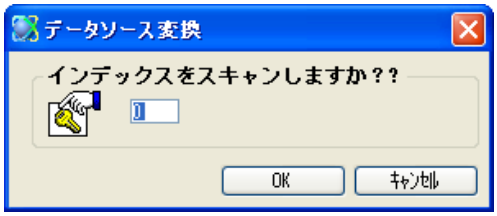
1. 最初に確認画面が出ますので、「はい(Y)」で答えてください。



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。



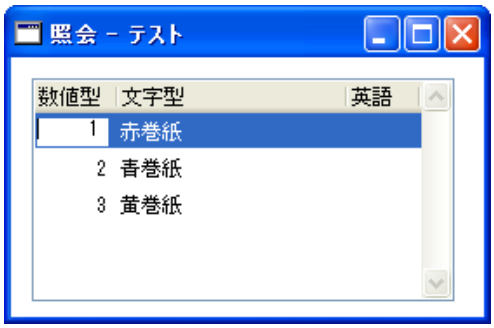
3. テーブル変換オプションが出てきます。今はこのままで OK ボタンを押してください。今回はデータ件数が少ないので、すぐに終了するはずです。



これで、MSSQL Server 上のテーブルも変更されています。

4.4.2.テーブル変更の確認

確認のために、Magic から APG を実行し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。

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。

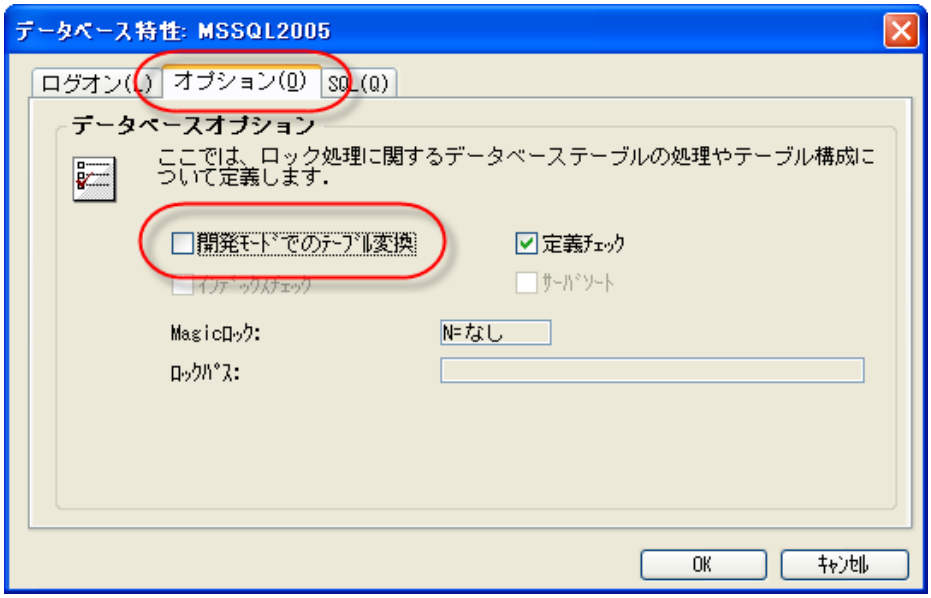
```
1> select * from テスト
2> go
 数値型      文字型              英語
 ----------- -------------------- -----
           1 赤巻紙               Red
           2 青巻紙               Blue
           3 黄巻紙               Yello
(3 件処理されました)
```

4.4.3.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 MSSQL 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 のデータ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、MSSQL 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 MSSQL 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「オプション(O) → 設定(S) → データベース(B)」を選びます)。

3. 「MSSQL2005」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。(ポップアップメニューを開き「特性」を選ぶか、あるいは Alt+Enter 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。

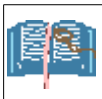
自動データ再編成機能を無効にした場合には、Magic のデータリポジトリでテーブル定義を変更しても MSSQL Server 上の実際のテーブルに反映されません。これによりテーブル定義の不整合が発生し、アプリケーション実行中にエラーが発生したり、意図したとおりに動作しなくなる可能性もあります。自動データ再編成機能の有効/無効は、作業の目的に合わせて適切に選んでください。

4.5.命名規則

Pervasive.SQL の場合には、データソース名は OS 上のファイル名となり、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。
MSSQL Server などの SQL 系の DBMS を利用する場合には、データソース名、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 のデータ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。

Magic Studio V10 のデータリポジトリでは、データソース名欄には Shift-JIS の名前のみを指定できます。また、MSSQL Server 2005 の標準識別子に関する規則（下記）に則った名前でなければなりません。

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。



MSSQL Server 2005 の標準識別子に関する規則

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 3.2 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。

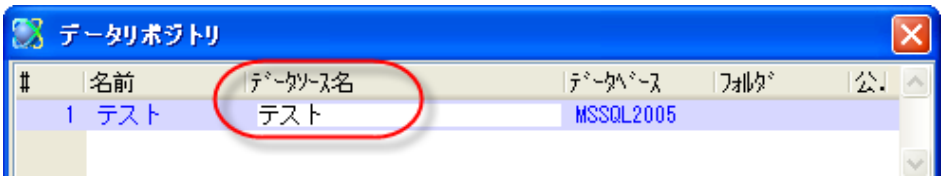
テーブル名、カラム名、インデックス名などには、それぞれまた細かな制限があります。詳細は、MSSQL Server のリファレンスマニュアルを参照してください。

4.6.名前の対応

Magic の場合、データ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 のデータ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。

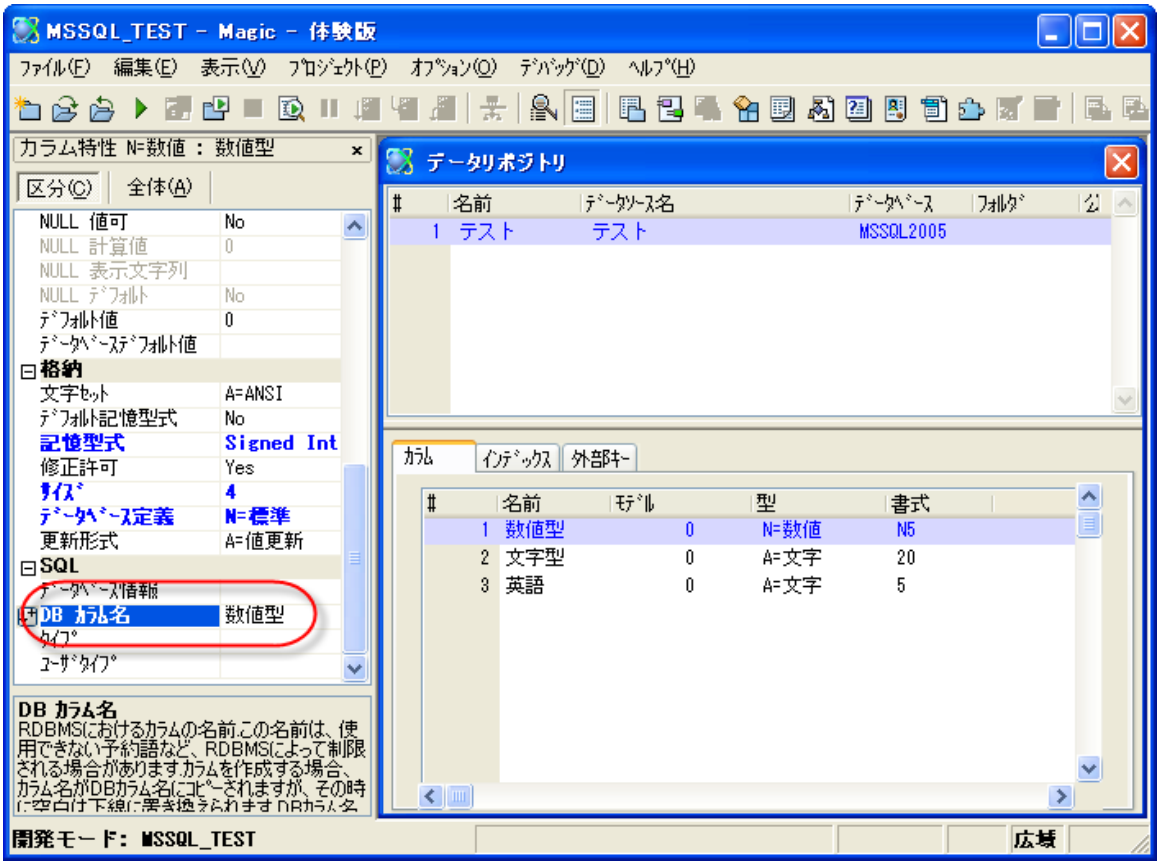
4.6.1.テーブル名

データリポジトリの「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、「データソース名」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「データソース名」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。



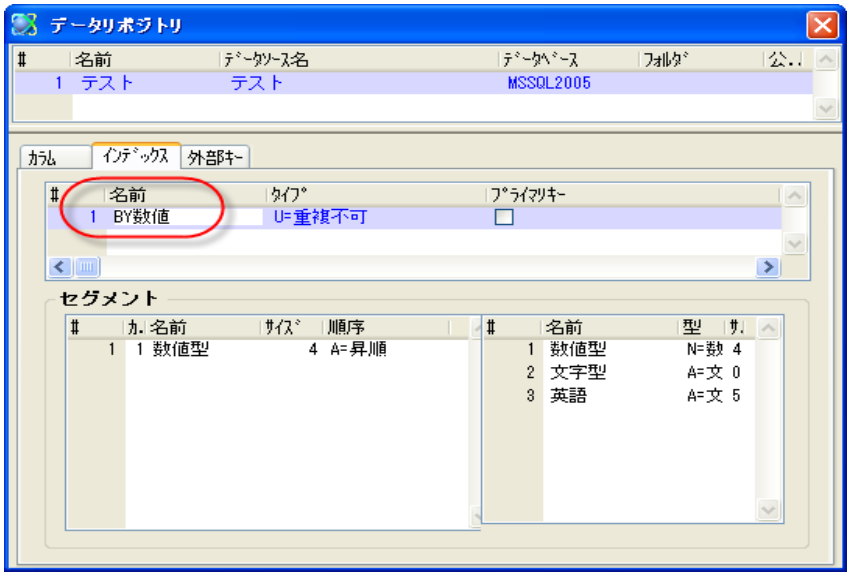
4.6.2.カラム名

データ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

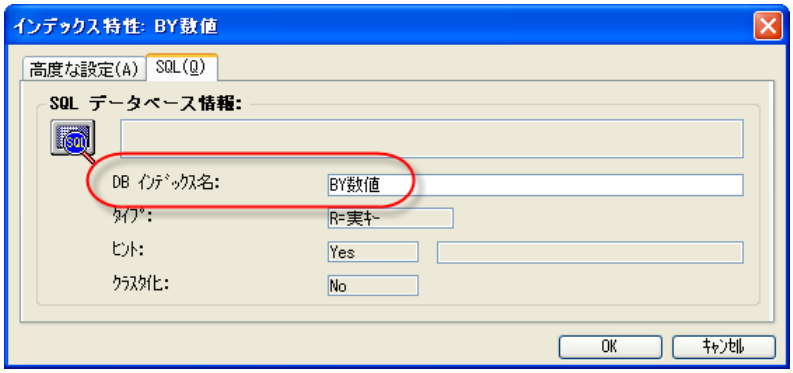
データリポジトリで、「インデックス」タブをクリックすると、そのテーブルに定義されているインデックスが表示されます。



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、Alt+Enter 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。



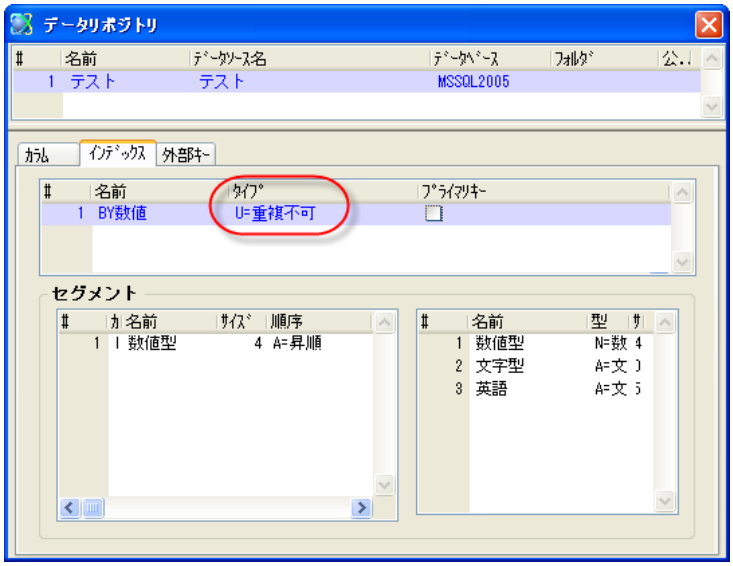
デフォルトで、インデックスの「名前」欄と同じ名前が設定されますが、あとから変更することもできます。

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

MSSQL 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、MSSQL Server 上でユニークインデックスが作成されます。
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インデックス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 1 に、DBMS ごとの必要性をまとめました。

表 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
MSSQL 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 MSSQL 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。

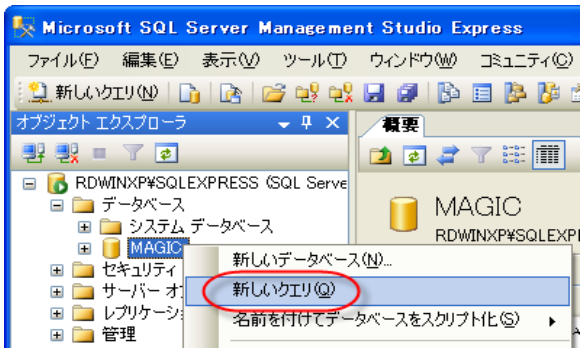

この制限は Magic の実行方式と DBMS の機能との兼ね合いから来るもので、MSSQL Server 自身の制限ではありません。

5. 定義取得

前章では、Magic のデータリポジトリで定義したテーブルを MSSQL 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、MSSQL 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、SQL Server Management Studio Express 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. SQL Server Management Studio Express を開き、SQL Server に接続します。

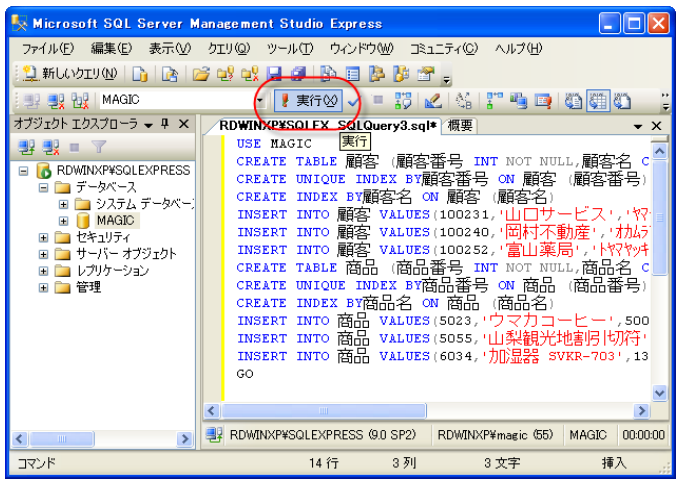
2. データベース MAGIC でポップアップメニューを開き、「新しいクエリ」を選びます。



3. 以下の内容をクエリウィンドウにコピーし、「実行(X)」ボタンを押します。(下図)

```
USE MAGIC
GO
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
```

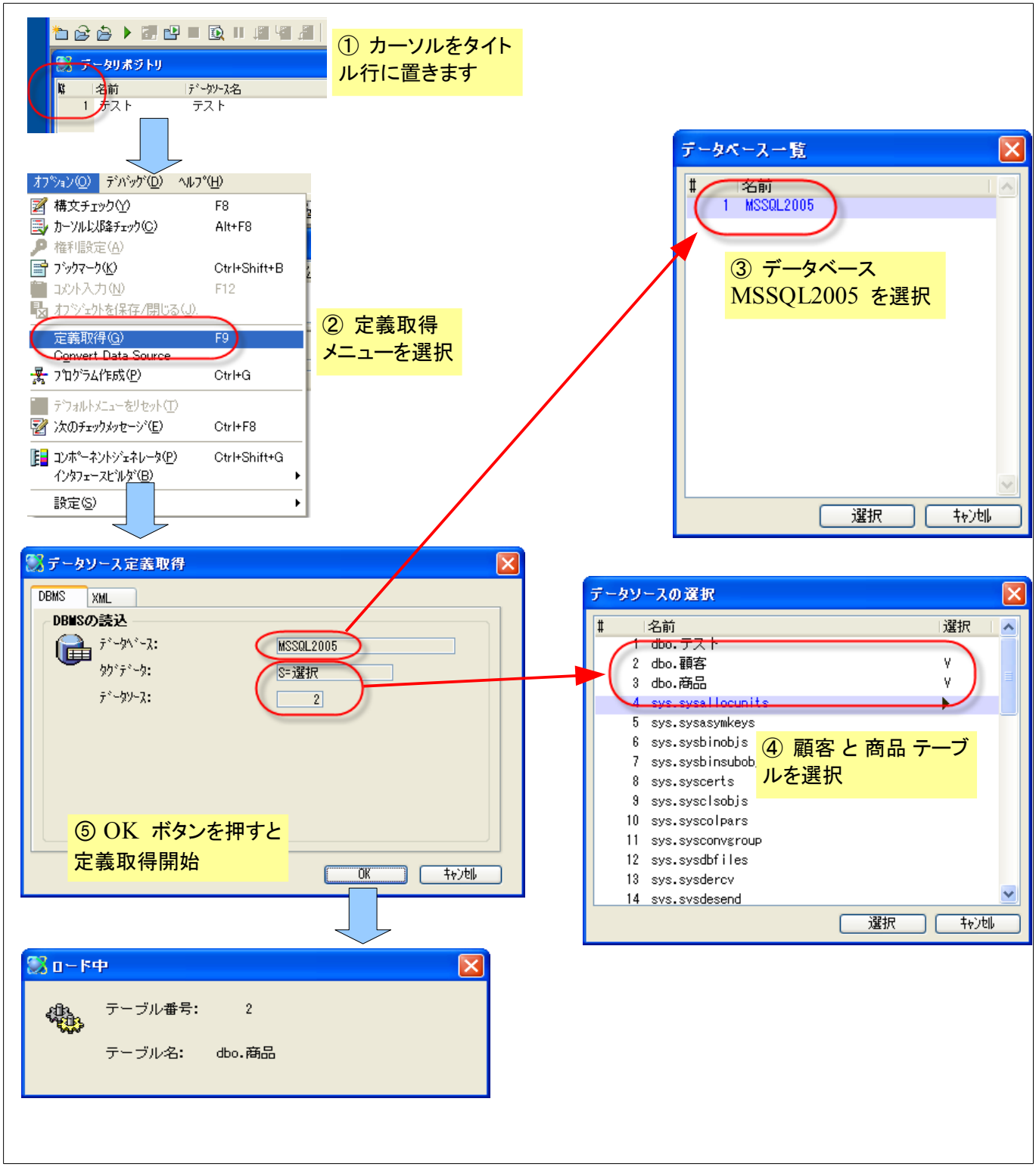
これで、MSSQL Server 上に「顧客」と「商品」というテーブルが作成され、それぞれ 3 レコードづつデータが挿入されました。



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、MSSQL 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。

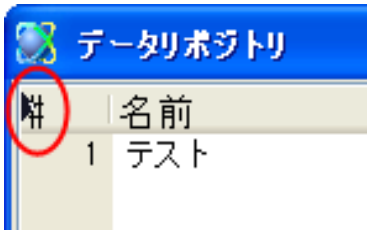
図 3 定義取得の手順



予め、アプリケーション「MSSQL_TEST」を開き、データリポジトリを開いておきます。

1. カーソルをタイトル行に置いておきます。

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、三角形のカーソルがタイトル行(「＃」のすぐ左)で点滅している状態になっていることを確認してください。



2. この状態で、メニュー「オプション(O)」から「定義取得(G)」を選択します。定義取得ダイアログが開きます。
3. 「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、「MSSQL2005」のみですので、これを選択します。
4. 「タグデータ」欄を「S=選択」にすると、データソース選択画面が表示されるので、「dbo.顧客」と「dbo.商品」とを選択します。

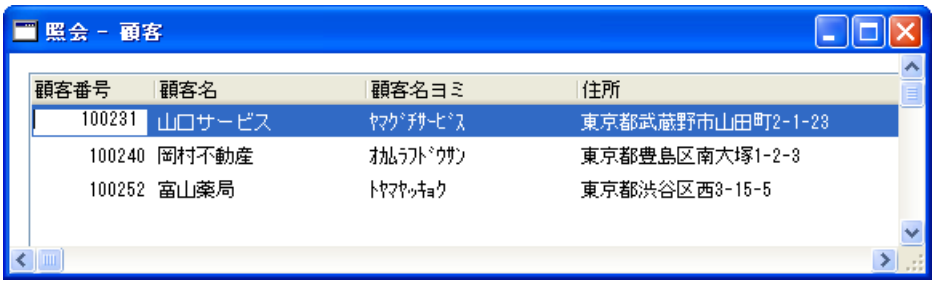
- テーブルの選択をするには、選択するテーブルの行にカーソルを合わせ、スペースキーを押すと、選択を示すチェック記号「V」がトグルします。
- テーブルの選択を変更するためにデータソース選択画面を再度開くには、「データソース」欄からズームします。

「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。

5. OK ボタンを押すと、定義取得が実行されます。

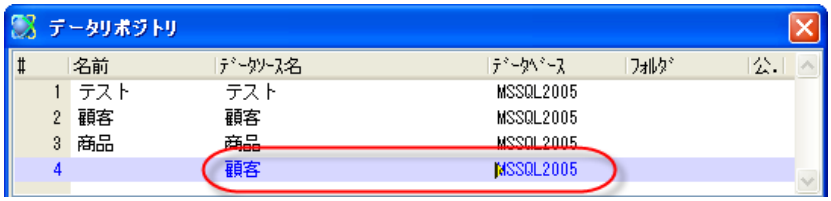
定義取得が完了したら、データリポジトリに「顧客」と「商品」のエントリが追加されているはずです。

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。



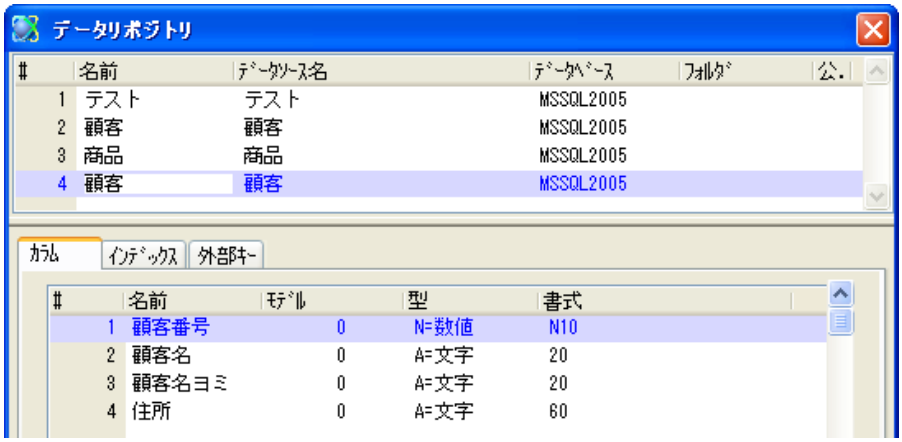
5.2.特定のテーブルの定義取得

前節での定義取得の方法は、MSSQL Server に定義されているテーブルの一覧をいったん取得し、その中から必要な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、MSSQL 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. データリポジトリで、F4 キーにより新規の行を作成します。
2. 「データソース名」欄に、定義取得したいテーブルの名前「顧客」を指定します。
3. 「データベース」欄からズームし、「MSSQL2005」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。

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」(34ページ)で説明したように、MSSQL 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 2 のようになっています。

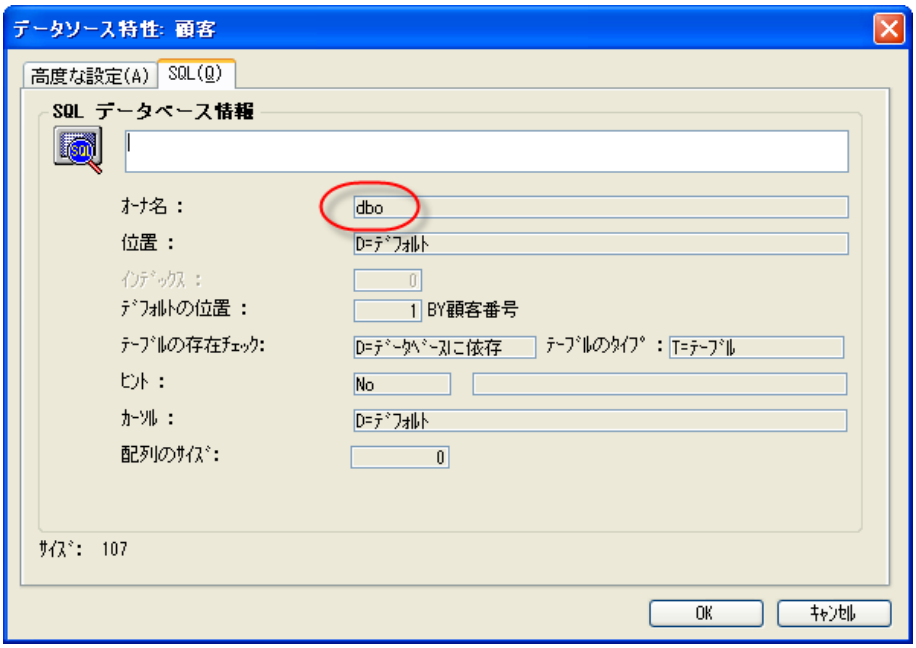
表 2 DBMS での名前と Magic での名前の対応表

MSSQL Server 上での定義	Magic での定義
テーブル名	「データソース名」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

データリポジトリで、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、「データソース特性」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. データソース特性ダイアログを開きます。データソース特性ダイアログは、ポップアップメニューで「特性(P)」を選択するか、あるいは Alt+Enter 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。

5.4.1.準備： ビューの定義

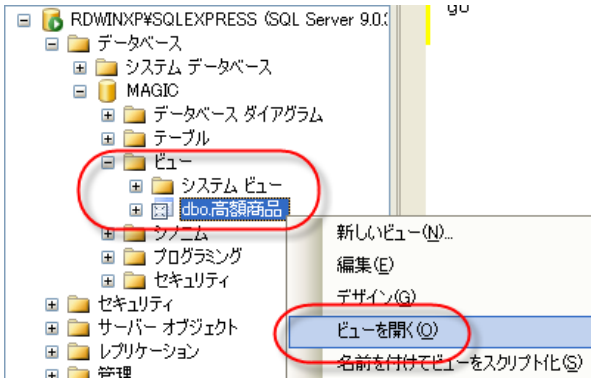
実験をするにあたり、SQL Server Management Studio Express を使って、簡単なビュー**高額商品**を定義しておきましょう。

1. SQL Server Management Studio Express を起動し、データベースに接続します。
2. データベース MAGIC 上でポップアップメニューを開き、「新しいクエリ」を選択します。
3. クエリウィンドウで、次のビューを定義し、「実行(X)」ボタンを押します。

```
create view 高額商品 as select 商品番号,商品名 from 商品 where 単価 > 10000
go
```

これでビューが定義されました。
確認のために、ビューを開いてみましょう。

1. SQL Server Management Studio Express で、同様にして新しいクエリウィンドウを開きます。
2. 「ビュー」ノードを開き、「dbo.高額商品」にカーソルを当てます。
3. ポップアップメニューから「ビューを開く(O)」を選びます。



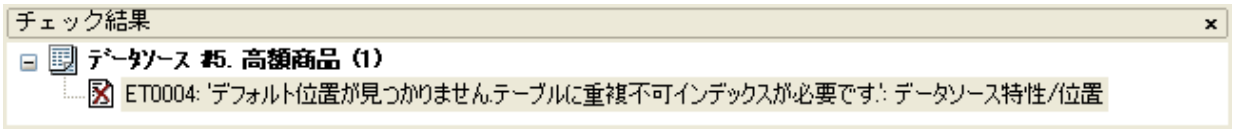
4. 新しいウィンドウが開き、ビューの内容が表示されます。

5.4.2.ビューの定義取得

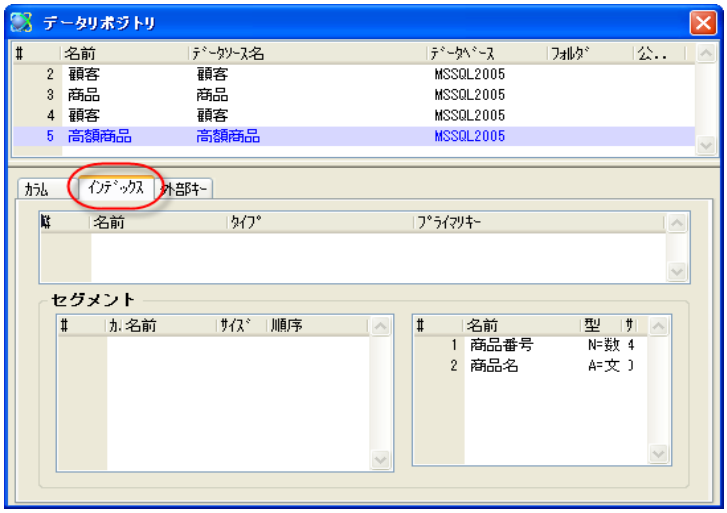
ここで高額商品ビューを定義取得してみましょう。やりかたは前節で説明したのと同じです。

1. データリポジトリに、F4 で新規行を作成する。
2. 「データソース名」欄を「高額商品」とし、「データベース」は「MSSQL2005」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。

定義取得直後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、次の図のようにエラーがでます。



定義取得した結果を下図に示します。これを見ると、カラム情報は取得されていますが、インデックスタブを開いてみると空であり、インデックスが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。



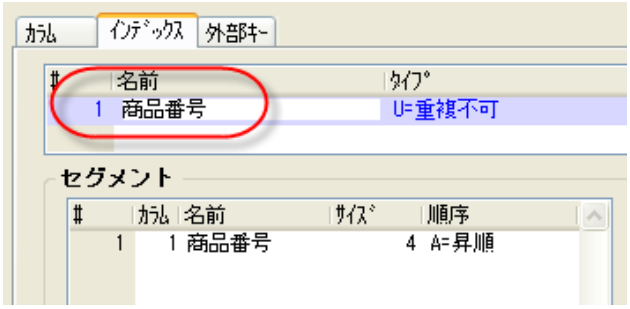
しかし、「4.7 一意インデックスの定義」(36ページ)で説明したとおり、MSSQL Server のデータを Magic から扱う場合には、ユニークインデックスが最低一つが必要です。

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。

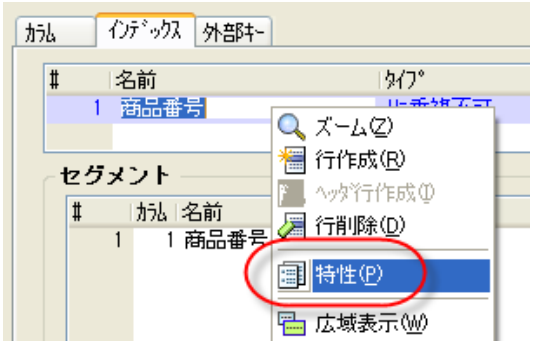
5.4.3.仮想インデックスの定義

仮想インデックスというのは、Magic のデータリポジトリ上で、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。

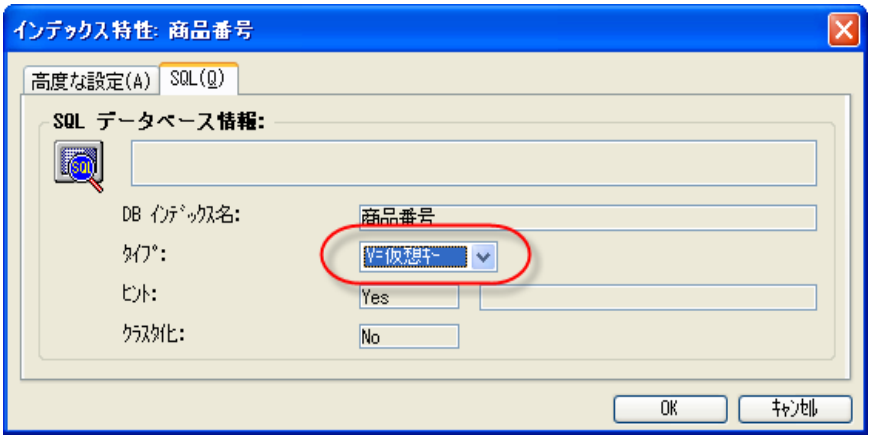
1. 「インデックス」タブを開き、通常のインデックスを定義する場合とまったく同じようにインデックスとインデックスセグメントとを定義します。ここでは名前を「商品番号」とします。



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、Alt+Enter キーでもできます。



3. 「SQL(Q)」タブを開き、「タイプ」欄を「V=仮想キー」に選びます。



このように仮想インデックスを定義しておけば、ビューであっても、あたかもユニークインデックス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。

5.4.4.仮想インデックス利用上の注意

仮想インデックスは、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、実インデックス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、仮想インデックス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。
- 重複不可の仮想インデックス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 には仮想インデックスの一意性をチェックするメカニズムはありません。
  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。

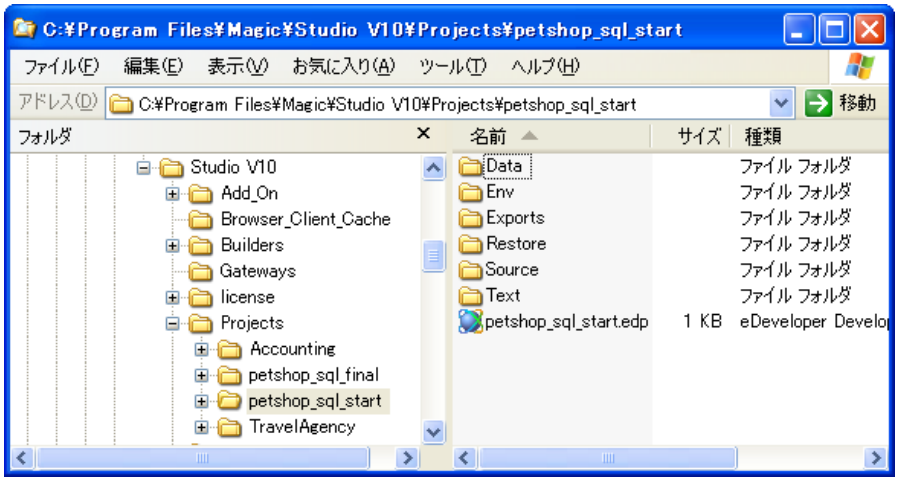
なお、仮想インデックスは、ビューだけでなく実テーブルに対しても定義することが可能です。この場合には、Magic が内部的に MSSQL Server に発行する SELECT 文に、仮想インデックスに定義されているセグメントを指定して ORDER BY ・・・ 句がつけられて実行されます。実行時になんらかの実インデックスが使われるか否かなどは、DBMS のオプティマイザが決定します。
実テーブルに対する仮想インデックスの利用の用途としては、あるテーブルが特定のレコード順に処理されるケースが多いのに、実インデックスが定義されていない場合などに、仮想インデックスを定義して、タスク特性の「インデックス」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 MSSQL Server に移植する方法を説明します。
本章では、ペットショップデモの簡単な説明と、設定の方法について説明します。

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 ディレクトリの下の Projects ディレクトリで解凍してください。サブディレクトリとして petshop_sql_start と petshop_sql_final が作成され、その下にそれぞれ、プロジェクトファイル、その他のファイル、サブディレクトリがあります。



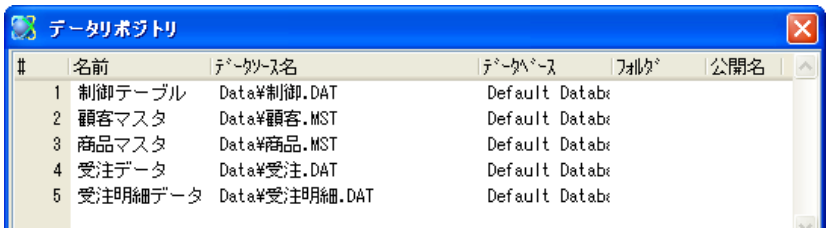
petshop_sql_start というのは、Pervasive を使って作成したペットショップデモアプリケーションで、単体で動かすことを前提とし、マルチユーザや SQL の機能についての考慮をしていないアプリケーションです。これを本書のスタートポイントとして、SQL 対応するするために修正を加えていきます。
petshop_sql_final というのは、本書に従って、SQL 対応のために修正を行ったアプリケーションの最終形です。読んでいる途中でわからないところなどがあれば、参考にしてください。
それぞれのディレクトリには、次のようなサブディレクトリがあります。

ファイル/ディレクトリ名	意味
petshop_sql_*.edp	Magic Studio V10 のプロジェクトファイルです。
Env ディレクトリ	ペットショップデモ用のフォント定義ファイル、色定義ファイルが格納されています。
Data ディレクトリ	Pervasive 形式のデータファイルが格納されています。
Exports ディレクトリ	プロジェクトのリポジトリ出力形式が格納されています。このファイルをリポジトリ入力しなおすことにより、プロジェクトをいつでも初期状態に戻すことができます。
Source ディレクトリ	プロジェクトのソースファイルが格納されています。
Text ディレクトリ	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。
Restore ディレクトリ	petshop_sql_start にだけあります。初期状態の Data、Exports、Text ディレクトリの内容がコピーしてあります。作業中にデータなどを復元するために使います。

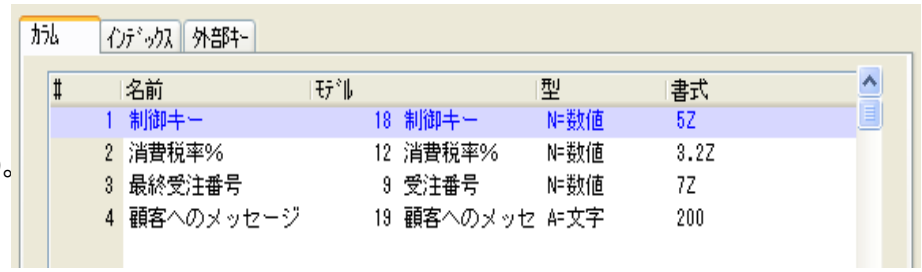
参考: プロジェクトファイルは、V10.1SP2 形式です。互換性のないバージョン/SP の Magic をお使いの場合には、Export ディレクトリにあるリポジトリ出力形式より、リポジトリ入力してください。

6.1.2.データリポジトリ

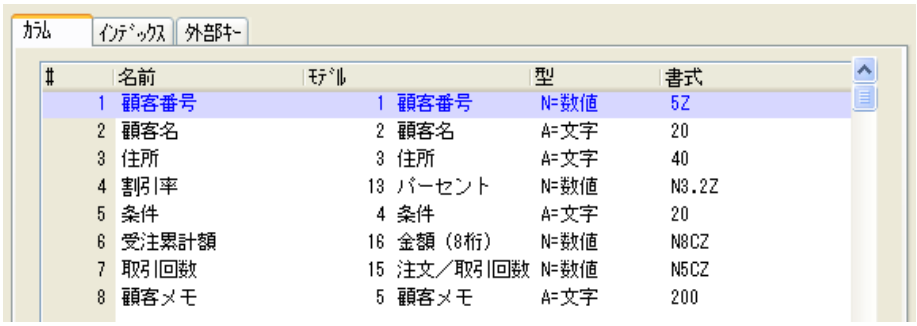
プロジェクトを開き、データリポジトリを開いてください。右図のようなテーブルが定義されています。
これらのテーブルは、いずれも Pervasive (Btrieve) ファイルであり、プロジェクトの直下にある Data ディレクトリに格納されています。



制御テーブル
制御テーブルには、消費税率、メッセージ、最終受注番号という、アプリケーションにグローバルなデータが格納されています(右図)。最終受注番号は、新しく受注レコードが作成されるたびに、1づつ増加されていきます。



顧客マスタ
顧客マスタには、顧客に関する情報が格納されます。名前・住所等、比較的変更の少ない情報のほか、受注累計額、取引回数など受注があるたびに変更される集計情報も一緒に格納されています。

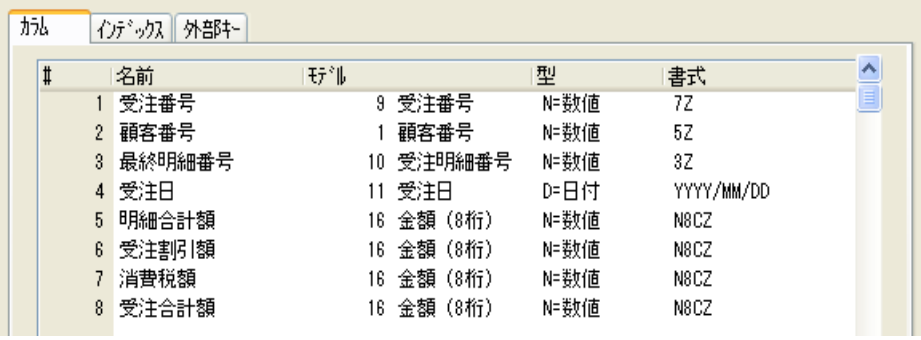


商品マスタ
商品マスタには、商品に関する情報が格納されます。
商品名、タイプ、単価などのほかに、在庫数、受注数など、受注のあるたびに変更される集計情報も一緒に格納されています。

受注データおよび受注明細データ

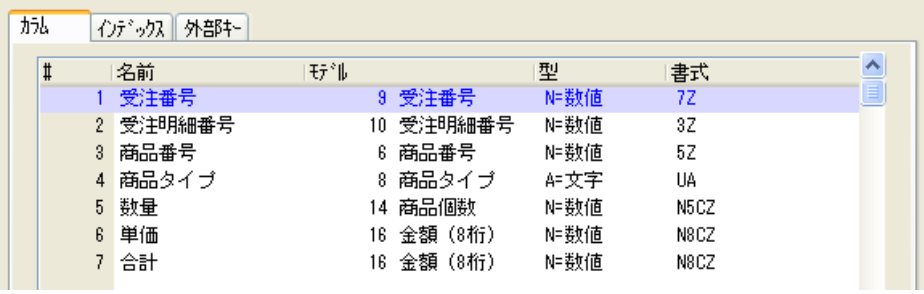
受注情報を格納します。これは1対Nの関係になったもので、受注データには1件の受注に対し1レコード、受注明細には1件のレコードに対して、複数件の明細行レコードがあります。

受注データ



#	名前	モデル		型	書式
1	受注番号	9	受注番号	N=数値	7Z
2	顧客番号	1	顧客番号	N=数値	5Z
3	最終明細番号	10	受注明細番号	N=数値	3Z
4	受注日	11	受注日	D=日付	YYYY/MM/DD
5	明細合計額	16	金額（8桁）	N=数値	N8CZ
6	受注割引額	16	金額（8桁）	N=数値	N8CZ
7	消費税額	16	金額（8桁）	N=数値	N8CZ
8	受注合計額	16	金額（8桁）	N=数値	N8CZ

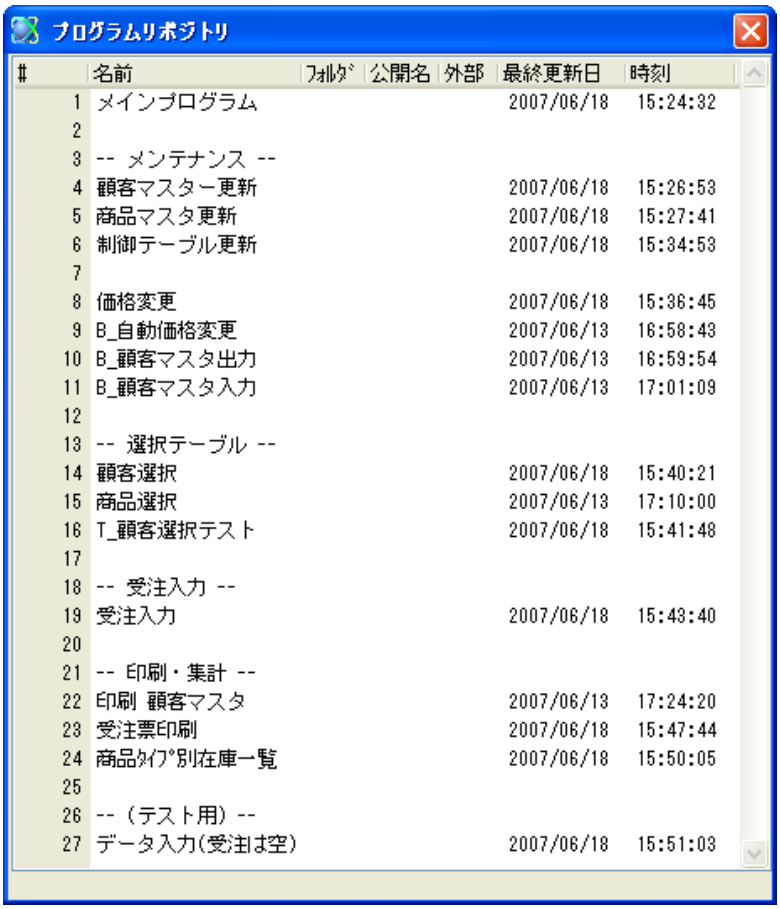
受注明細データ



#	名前	モデル		型	書式
1	受注番号	9	受注番号	N=数値	7Z
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

6.1.3.プログラムリポジトリ

プログラムリポジトリ(右図)には、大別して、マスタメンテナンス、選択テーブル、受注入力、印刷・集計、テスト用のプログラムがあります。



プログラムリポジトリ

#	名前	フォルダ	公開名	外部	最終更新日	時刻
1	メインプログラム				2007/06/18	15:24:32
2						
3	-- メンテナンス --					
4	顧客マスター更新				2007/06/18	15:26:53
5	商品マスタ更新				2007/06/18	15:27:41
6	制御テーブル更新				2007/06/18	15:34:53
7						
8	価格変更				2007/06/18	15:36:45
9	B_自動価格変更				2007/06/13	16:58:43
10	B_顧客マスタ出力				2007/06/13	16:59:54
11	B_顧客マスタ入力				2007/06/13	17:01:09
12						
13	-- 選択テーブル --					
14	顧客選択				2007/06/18	15:40:21
15	商品選択				2007/06/13	17:10:00
16	T_顧客選択テスト				2007/06/18	15:41:48
17						
18	-- 受注入力 --					
19	受注入力				2007/06/18	15:43:40
20						
21	-- 印刷・集計 --					
22	印刷 顧客マスタ				2007/06/13	17:24:20
23	受注票印刷				2007/06/18	15:47:44
24	商品ﾀｲﾌﾟ別在庫一覧				2007/06/18	15:50:05
25						
26	--（テスト用）--					
27	データ入力(受注は空)				2007/06/18	15:51:03

マスタメンテナンス

マスタメンテナンスプログラムは、顧客マスタ、商品マスタ、および制御テーブルの追加、修正、削除を行います。
これらのプログラムは、照会 APG で作成したものとほとんど同じです。ただし、トランザクションは、デフォルトの遅延トランザクションではなく、物理トランザクションになっています。

選択テーブル

ここでは、顧客マスタおよび商品マスタのデータを選択する選択プログラムがあります。

受注入力

受注入力タスクは、受注ヘッダをメインデータソースとする親タスクと、受注明細をメインデータソースとする子タスクの親子タスクです。子タスクを呼び出すのに、V10 で新しく導入されたサブフォームを使っています。
親タスクのレコード後処理で、顧客マスタの集計項目（受注累積額、取引回数)を、加算モードの項目更新コマンドで更新しています。同様に、子タスクのレコード後処理で、商品マスタの集計項目（受注数）および親タスクのレコードの受注合計額を、加算モードの項目更新コマンドで更新しています。

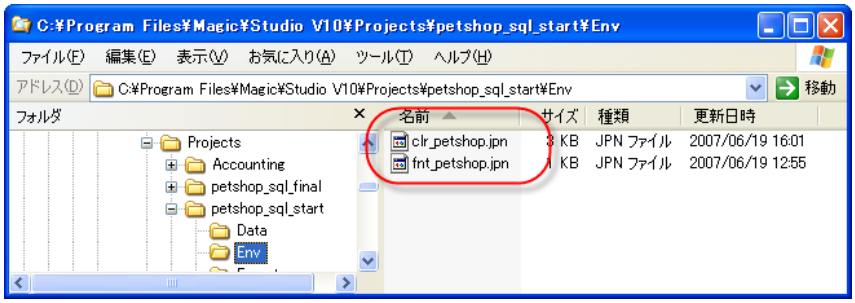
印刷・集計

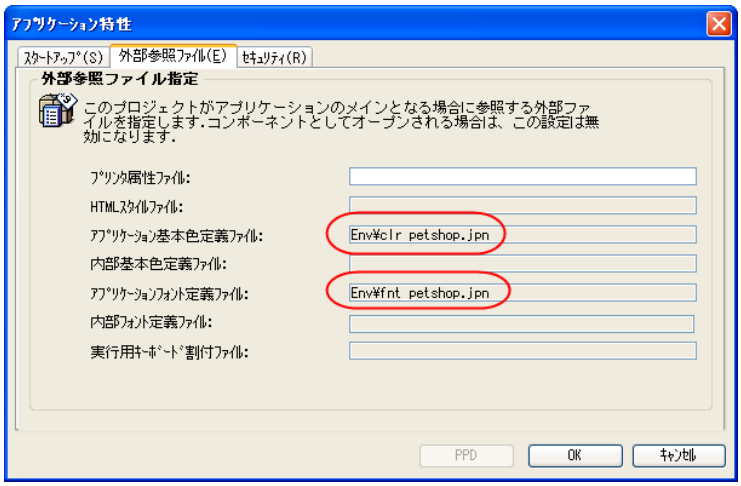
顧客マスタ印刷プログラムは、印刷 APG で作成したプログラムに、ページヘッダとページフッタとを加えています。
受注票印刷プログラムは、受注入力プログラムに似て、受注データと受注明細データの親子タスクで作成されています。ただし印刷プログラムなので、親子ともバッチタスクです。
商品タイプ別在庫一覧は、商品マスタをメインデータソースとするバッチタスクで、商品タイプでグループ化し、グループごとに在庫数を集計して印刷します。

プログラムはいずれも入門編で勉強した機能を使ったものばかりですので、詳細な説明はここでは省略します。

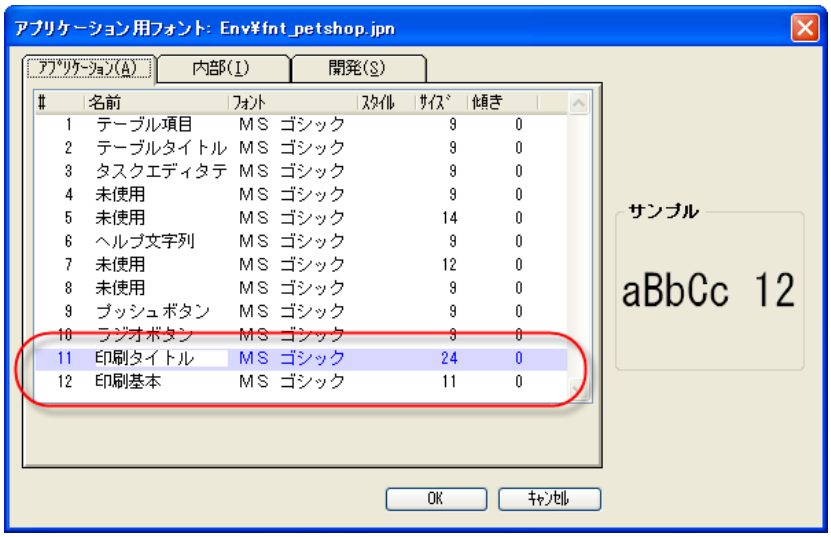
6.1.4. フォントテーブルと色テーブル

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 11 番と 12 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 10 番までしか定義されていないので、サンプルでは Env ディレクトリにアプリケーション用のフォントファイル fnt_petshop.jpn を参照しています。
同様に、アプリケーションでは色番号 101 ～ 103 も使っています。この色定義ファイルも、Env ディレクトリに clr_petshop.jpn として格納されています。
このフォントテーブルと色テーブルとは、ペットショップアプリケーションに特有のものなので、MAGIC.INI ではなく、プロジェクトのアプリケーション特性 → 外部参照ファイル(E) タブで設定しています。

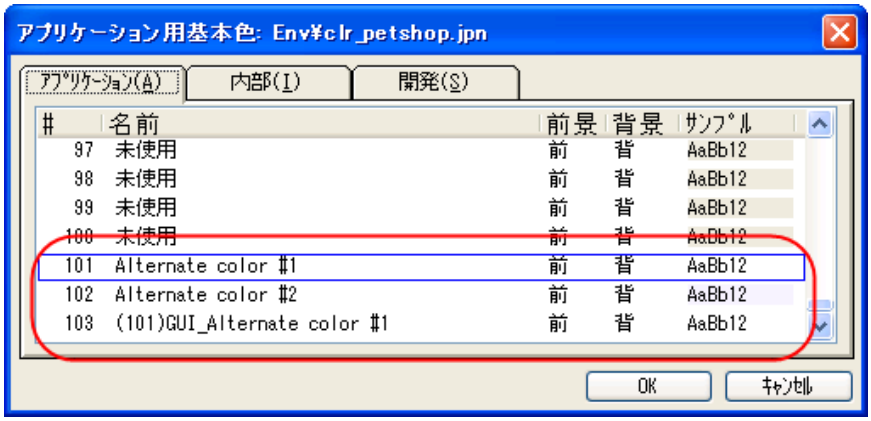




ここからズームしてフォントファイルを開いてみると、フォント 11 番と 12 番とが登録されています（下図参照）。11 番は印刷のタイトルですので、少し大きめのフォント(24 ポイント)で、12 番は通常印刷フォントなので、11 ポイントとなっています。



また、色定義ファイルをズームしてみると、101 番 ～ 103 番が定義されています。これは GUI テーブルの交互色を指定するものです。



以上でアプリケーションの簡単な説明を終えます。

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 MSSQL Server に移行しましょう。Magic は各 DBMS ごとに設計された **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 のデータ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 MSSQL 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。

以下の説明に先立って、「petshop_sql_start」プロジェクトを開いておいてください。

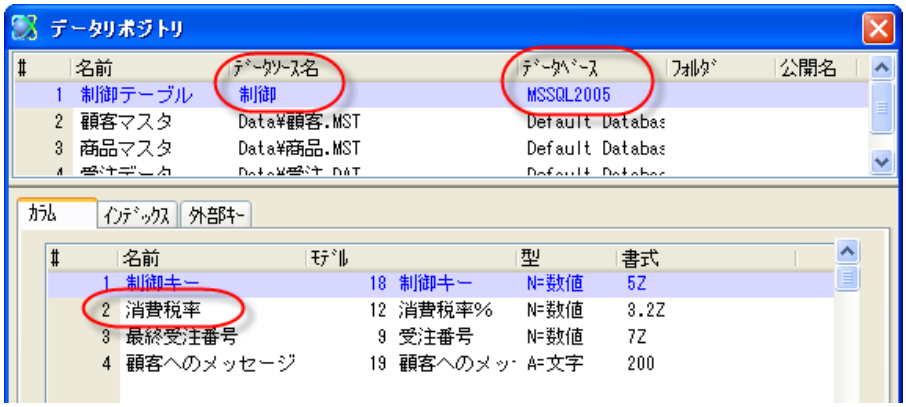
7.1.Magic のデータ再編成機能を使う

Magic のデータ再編成機能については、「4.4 データ再編成」(30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、データリポジトリで「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。

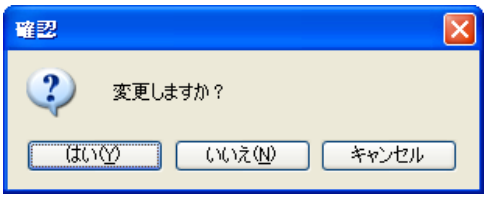
7.1.1.データ再編成によるデータベース移行の手順

次に、ペットショップデモのテーブルを Pervasive.SQL から MSSQL 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。

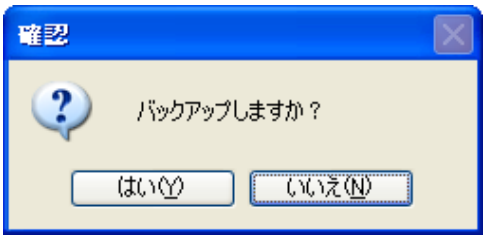
1. データリポジトリを開き、「制御テーブル」の行にカーソルをおきます。
2. 「カラム」タブを開き、各カラムについて、「名前」が MSSQL Server の命名規則に反していないかを確認します。
   「制御テーブル」の場合には、「消費税%」カラムの「%」はダメなので、名前を「消費税率」に変更します。他は OK です。



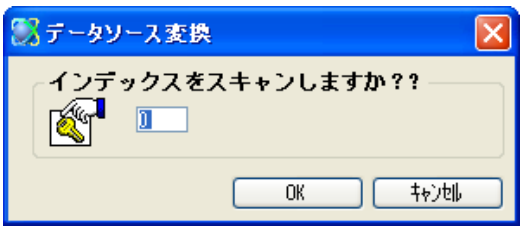
3. 「データソース名」の名前は「Data¥制御.DAT」ですが、MSSQL Server の規約にあわせ、単に「制御」とします。
4. 「データベース」欄からズーム (F5 キー)して、データベースを「Default Database」から「MSSQL」に変更します。
5. ここで APG を実行してみます。(あるいは、別のテーブルにカーソルを移動します)。このタイミングで自動データ再編成機能が働きます。
6. 「4.4 データ再編成」に書いたのと同様に、右図のような確認ダイアログが出てきます。
   (1) 「変換しますか？」には「はい(Y)」で答えます。



   (2) 「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。)



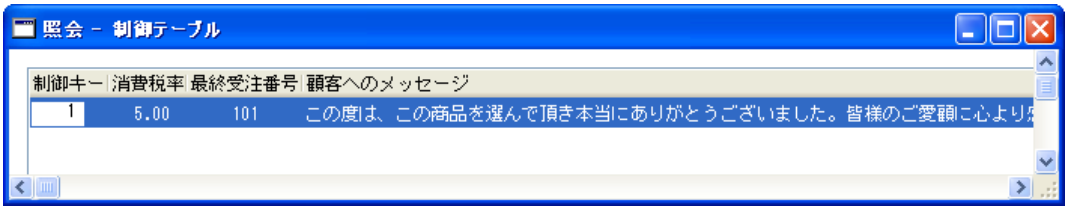
   (3) 「インデックスをスキャンしますか？」には、そのまま OK ボタンを押します。



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 -S rdwinxp¥SQLExpress
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             5.0         101     この度は、この商品を選んで頂き・・・
(1 行処理されました)
1>
```

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.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・入力で移行する

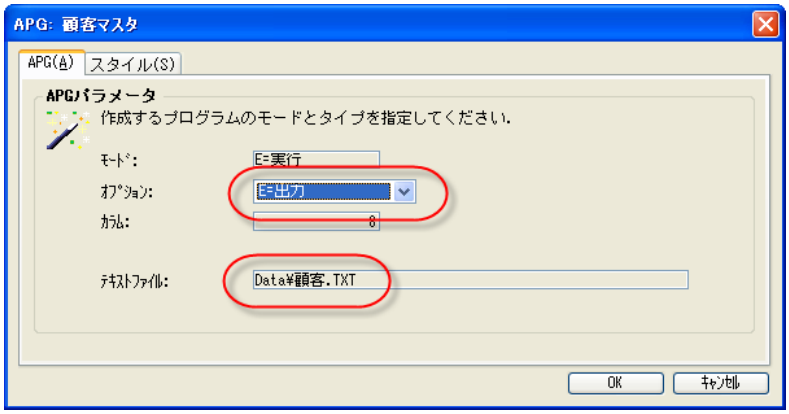
この方法は、データ再編成機能によるデータベース移行を行うのではなく、
1. 変更前、もとのデータベース(Pervasive.SQL)でデータをテキストファイルで出力する。
2. データベースを変更する。この際、データ再編成機能は実行しない。
3. 変更後、テキストファイルからデータを入力する。

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

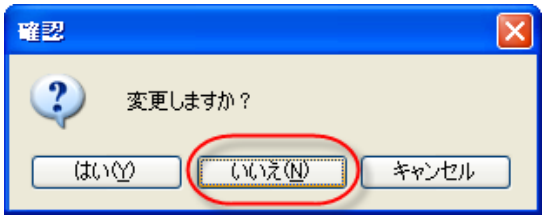
ここでは、顧客マスタをテキスト出力・入力方式によりデータベース移行してみましょう。

1. データ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「Data¥顧客.TXT」というファイルにテキスト出力します。

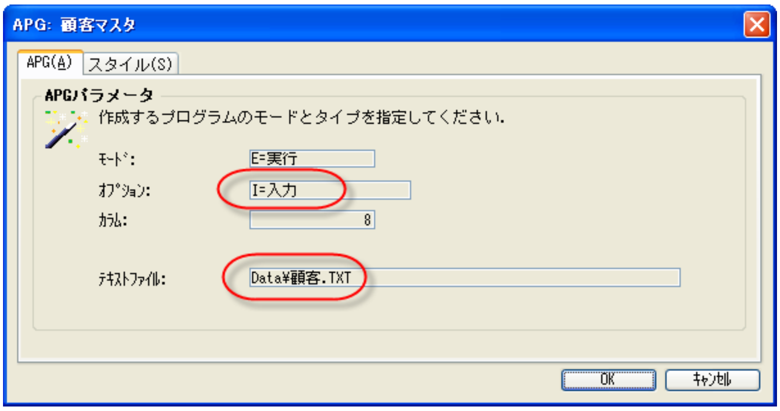


4. 前の例と同様に、カラム名が MSSQL Server の命名規約に違反していないかをチェックします。顧客マスタの場合にはいずれのカラムも OK です。
5. 「データソース」欄を、Pervasive.SQL での名前「Data¥顧客.MST」から、MSSQL Server の命名規約に適合したテーブル名「顧客マスタ」に変更します。
6. 「データベース」欄を「Defautl Database」から「MSSQL 2005」に変更します。

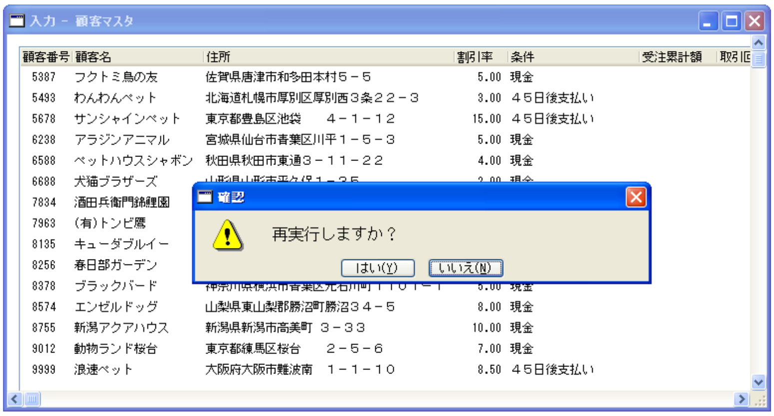
7. APG を再度起動します。ここで、データ再編成機能が働き、「変更しますか？」の確認ダイアログが出ますが、今回はデータ再編成機能を使わないので「いいえ(N)」で答えます。



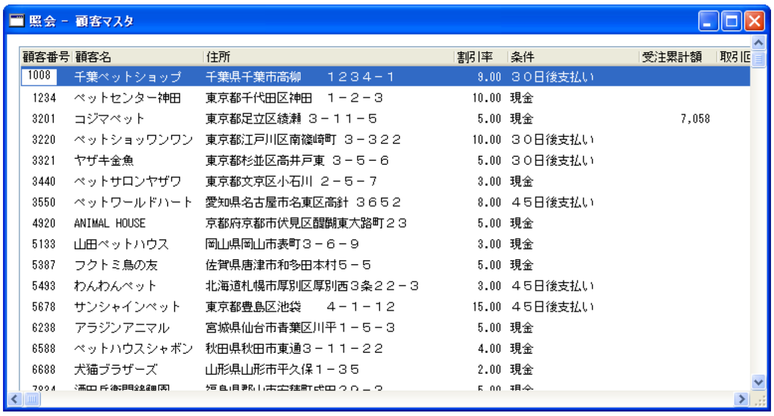
8. データ再編成を行わずに、APG 画面が表示されますので、今度は「オプション」として「I=入力」とし、「テキストファイル」には、先ほどデータ出力したファイル名「Data¥顧客.TXT」を設定します。



9. OK ボタンを押すと、データが入力されます。



10. 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。



11. 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 -S rdwinxp¥SQLExpress
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 ペットショップワンワン 東京都江戸川区南篠崎町 ３－３２２         10.0 ３０日後支払い
   (省略)
(24 件処理されました)
```

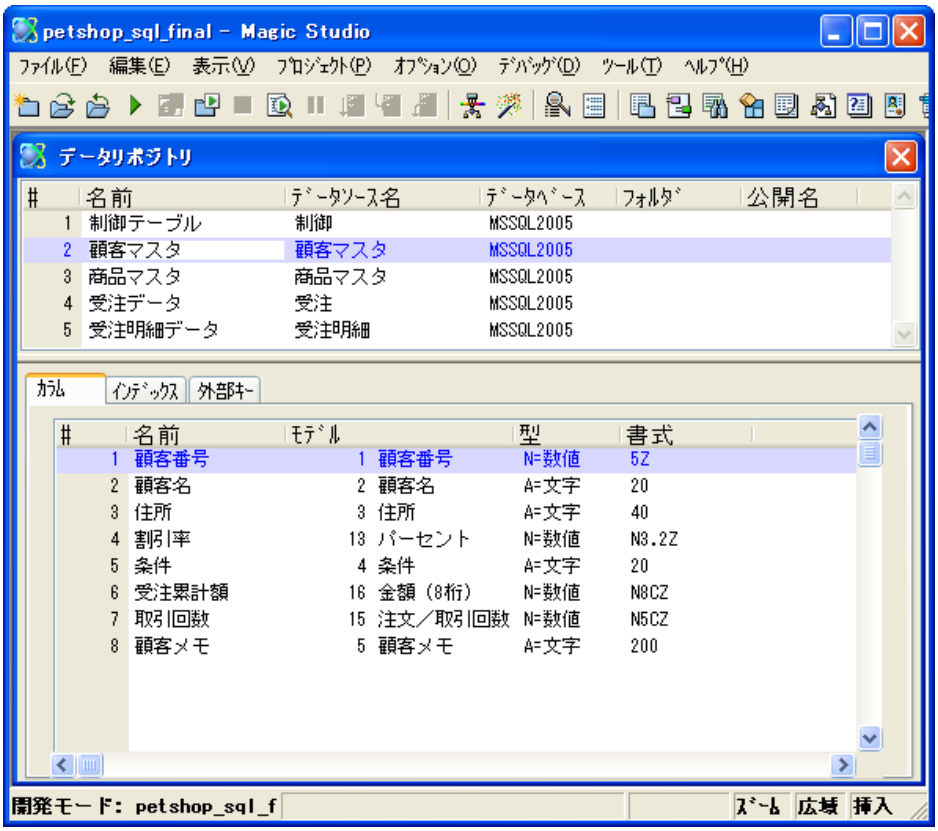
以上で、テキスト出力・入力によるデータ移行ができました。

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 27 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データリポジトリをすべて MSSQL 用に移行してください。



8. マルチユーザ環境

ペットショップアプリケーションを SQL 対応するために修正する前に、基礎知識として、本章でマルチユーザ環境での並行制御(ロック)について、次章で SQL データベースでのトランザクションについて、一般的な概念について簡単に解説します。

今日の情報システムでは、複数のユーザが同時に同一のリソース(データベース中のレコードなど)をアクセスするのが普通ですが、このような環境では、複数ユーザによる同時アクセスの制御を正しく設計しておかないと、データ更新の不正、ロック待ちなどの問題が起こります。これは Magic を使ったシステムに限らず、どの情報システムでも同じことで、マルチユーザ環境でアクセスを正しくコントロールすることを**並行制御**と呼びます。
Magic では、並行制御のためにさまざまな機能が備わっており、これを上手に使うことにより、データ更新の不正を防ぎつつ不要なロック待ちを減らすことができます。
本章では、最初にマルチユーザ環境での一般的な問題 (更新の喪失とレコードロックの問題)について説明し、次に Magic の並行制御機能について説明します。その後、具体的な例として、ペットショップデモを取り上げ、並行制御を考慮せずに作ったプログラムを複数ユーザ環境で実行した場合に起こる問題をとりあげます。
なお、並行制御の問題は、アプリケーションで使っている DBMS がサポートする並行制御の機能に依存する部分がありますが、本章では MSSQL Server を DBMS として利用することにします。


本章の内容については、リファレンスマニュアルに詳しい情報がありますので参照してください。

マルチユーザ対応全般	マルチユーザ環境
動作環境設定	設定 → 設定／動作環境 → [マルチユーザ]タブ
ロック方式パラメータ	プログラム → [タスク特性] → [データ]タブ

8.1.更新の喪失

マルチユーザ環境での一番根本的な問題は、**更新の喪失**と呼ばれる問題です。マルチユーザ環境では同一のレコードを複数のユーザが同時にアクセスし更新する可能性があるので、あるユーザが更新したレコードを、他のユーザが上書きしてしまって、結果として、最初のユーザが行った更新が失われてしまう可能性があります。これは当然、データの不正となってしまいますので、常に正確な情報を維持しておくべき情報システムからは排除しなければならない問題です。

簡単な例で見てみましょう。これは、同一の商品レコードを、ユーザ A と B とが同時に更新した例です。
最初には、商品番号#1002 の発注数は 0 であったとします。ユーザ A が発注数を +3 し、ユーザ B が発注数を +2 したとすると、最終的な発注数は 0 + 3 + 2 = 5 でなければなりません。
図 4 は両者の更新が正しく行われた例です。この図は上から下に時間が流れていく時系列的に描いたもので、次のような順番で処理が進みます。

1. ユーザ A が商品番号#1002 のレコードを読み込みます。このとき、発注数は 0 です。
2. ユーザ A が＋3し、発注数は 3 となります。
3. ユーザ A がレコードを書き込みます。DBMS 中のレコードに、更新が反映されます。
4. 次にユーザ B が同じレコードを読み込みます。発注数は 3 です。
5. ユーザ B が＋2を計算し、発注数は 5 となります。
6. ユーザ B がレコードを書き込みます。DBMS 中のレコードの発注数は 5 となります。

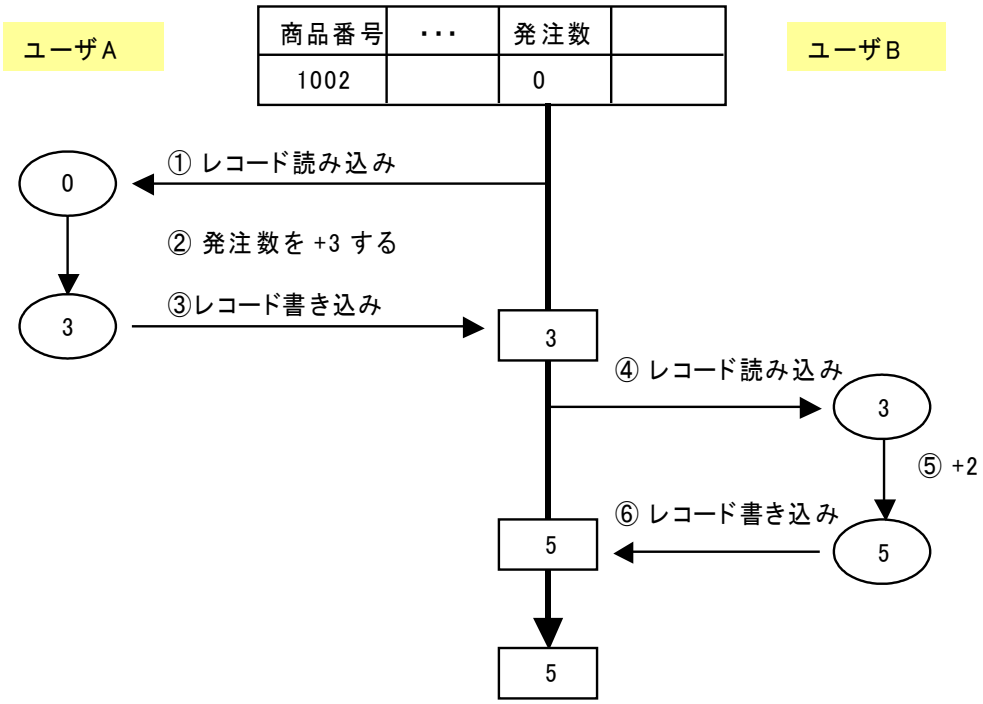


図 4 ユーザ A とユーザ B が発注数を更新

ところが、図 5 のようなタイミングでレコードが更新された場合には、ユーザ A が行った +3 の更新は、ユーザ B が直後に行った +2 の更新によって、上書きされてしまいます。その結果、最終的な発注数は、5 ではなく 2 になってしまいます。これは明らかに問題です。

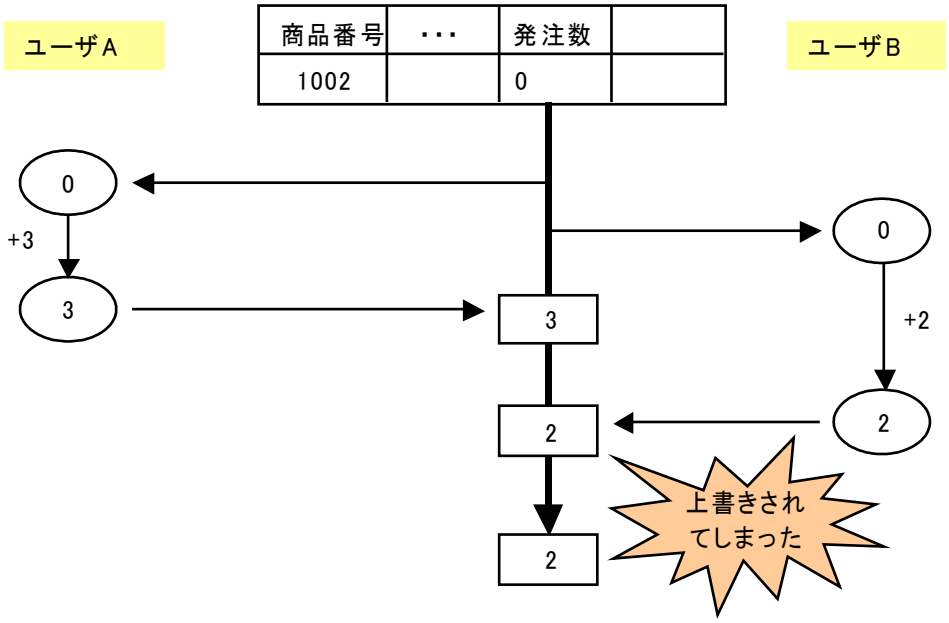


図 5 更新の喪失の例

このような問題が起こる原因は、図 5 を見ればすぐにわかるように、ユーザ A が結果 3 の書き込みをする前に、ユーザ B がレコードを読み込んでしまい、現在の発注数を 0 として計算してしまうからです。
このように、マルチユーザ環境では、レコードの読み込みと書き込みのタイミングが重要であることがわかります。

8.2. レコードロック

更新の喪失の問題を解決するために、DBMS に**ロック**という機構が導入されました。ロックというのにはいろいろなバリエーションがありますが、基本的な**共有ロック**では、あるレコードに対して、ロックをかけたユーザが独占的な更新の権利を持つことができ、他のユーザはこのレコードを読み込むことはできるけれど、ロックをかけたり更新をしたりすることができない、というしくみです。
先ほどの例でロックを使うとどのように更新の喪失を防ぐことができるかを、図 6 に示します。

1. 最初にユーザ A がレコードを読み込みます。このとき、レコードの読み込みと同時に、このレコードにロックをかけます。図中では、レコードにロックがかかっている期間を赤色の線で表しています。
2. 次に、ユーザ B が同じレコードを読み込むと同時にロックをかけようと試みます。しかし、このときにはユーザ A がすでにこのレコードにロックをかけてしまっているので、**ロックの衝突**によりユーザ B は失敗します。ユーザ B はこのようなときには、しばらく時間をおいてから再度読み込むようにします。
3. ユーザ A は、＋3の更新が終わり、レコードに書き込みます。このとき同時に、レコードのロックを解除します。
4. しばらくして、ユーザ B が再度レコード読み込みとロックを試みます。このときにはすでにユーザ A によるレコードロックは解除されているので、読み込みとロックが成功します。
5. ユーザ B は＋2の更新を行い、レコードを書き込み、ロックを解除します。

結果として、受注数は正しく 5 となります。

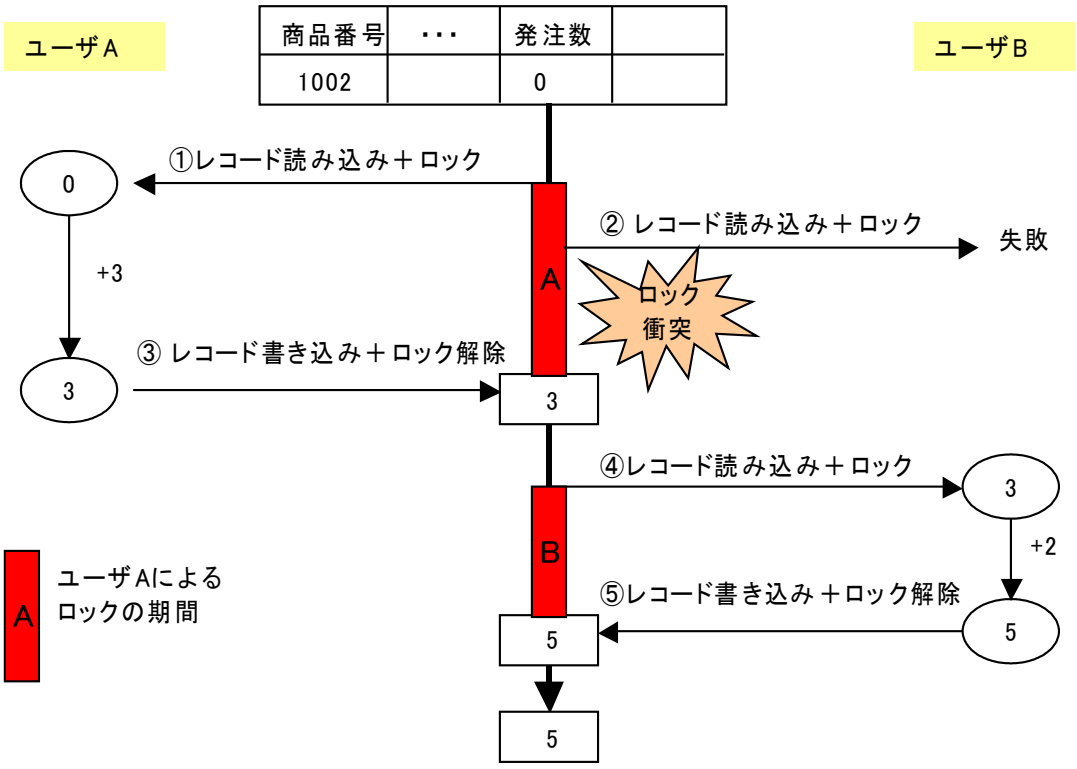


図 6 ロックによる更新の喪失の防止

このように、ロックを使った場合には、ユーザ A が書き込む以前にユーザ B がレコードを読み込んで、その値を元に計算を行う、ということがなくなるので、不正な更新を防止することができるようになります。

8.3.レコードロック利用時の問題

このように、ロックは平行制御の基本メカニズムですが、次に示すような問題もあります。

8.3.1.ロック待ち

まず、ロックは一人のユーザにだけ独占的なアクセス権を与えるものであり、他のユーザはロックの解除を待たせられ、その間は仕事を進められなくなる、という問題があります。
例えば、前に図 6 に出した例をとれば、ユーザ A がレコードを読み込みロックをかけたら、ユーザ B など他のユーザはこのレコードに対する処理を行えず、ユーザ A がロックを解除するまで待たされることになります。
ユーザ A が手際よく処理をしてくれればユーザ B の待ち時間は短くて済みますが、ユーザ A がレコードのロックをしたまま帰宅してしまったりしたら、その間ユーザ B は作業を進められなくなってしまいます。この問題の影響は、ユーザ数が多くてロックの衝突の可能性が高くなるほど深刻になってきます。

8.3.2.デッドロック

ロック待ちのもっとも深刻な形として、**デッドロック**という状態に陥ってしまうことがあります。デッドロックというのは**すくみ**とも呼ばれますが、二人以上のユーザが、お互いがお互いのロックの解除を待ったままにらみ合いになり、どちらも先に進むことができなくなってしまう状態を言います。
簡単な例として図 7 に、ユーザ A とユーザ B とが、商品番号#1002 と#1003 の二つのレコードを同時に更新しようとしてデッドロックになってしまうシナリオを見てみます。このシナリオでは、ユーザ A は #1002 → #1003 という順序で更新し、ユーザ B はその逆に #1003 → #1002 という順序で更新しようとしています。

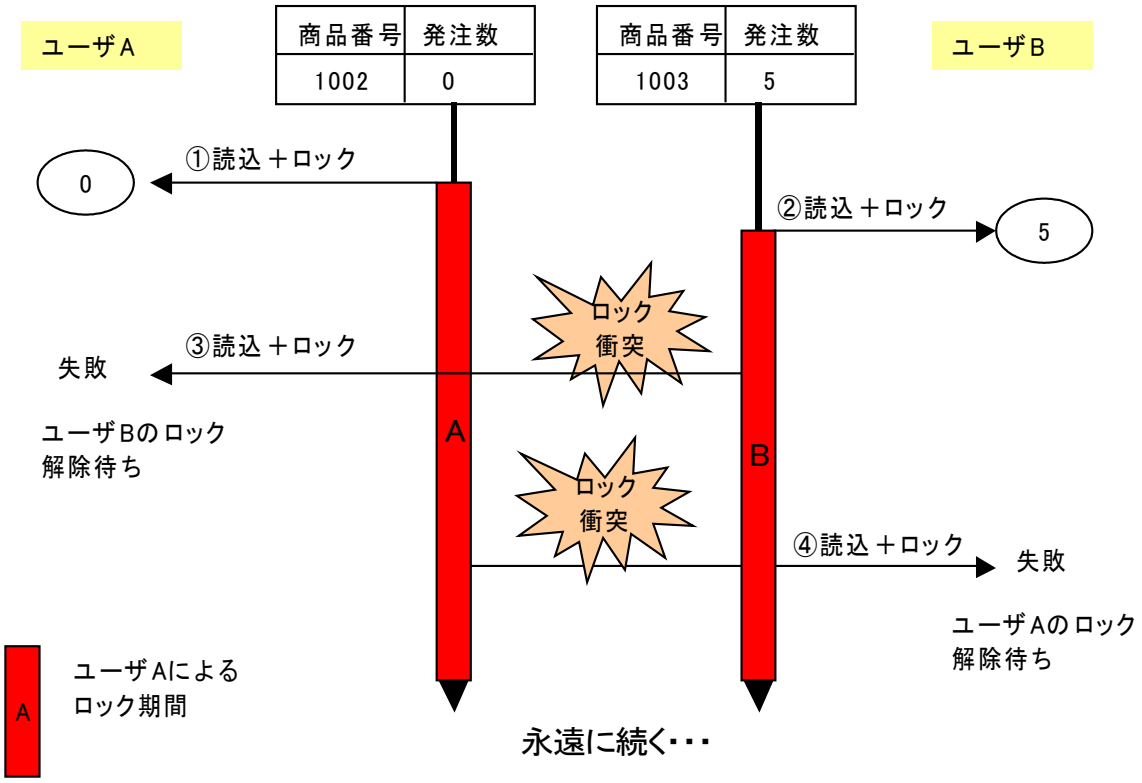


図 7 デッドロック

1. ユーザ A が#1002 のレコードを読み込み、ロックをかけます。
2. ユーザ B が#1003 のレコードを読み込み、ロックをかけます。ここまではいずれも成功します。
3. ユーザ A が今度は#1003 のレコードを読み込みロックをかけようとしますが、#1003 はすでにユーザ B がロックをかけているので、ロックの衝突が起こり、失敗します。このとき、ユーザ A は、ユーザ B が#1003 のロックを解除するまで待たなければなりません。
4. そこにユーザ B が#1002 を読み込み・ロックをしようとしますが、#1002 はユーザ A がロックをかけているの

で、ロックの衝突により失敗します。ユーザ B はユーザ A のロック解除を待たなければなりません。

この状態では、ユーザ A と B はお互いに相手が終わるのを待っているばかりで、これ以上先に進むことができなくなります。これがデッドロックです。ここでの問題は、ユーザ A が#1002→#1003 の順にロックをかけようとするのに対し、ユーザ B が逆に#1003→#1002 の順でロックをかけようとするところにあります。すなわち、デッドロックを防止するためには、設計時にレコードがロックされる順序も良く把握しておく必要があります。

8.3.3.ロックの問題を軽減するために

以上見てきたようなロックの問題は、データベースが複雑になりタイミングがクリティカルになると完全に防止することは難しいものがありますが、できるだけ問題を軽減するために、ロックを使うにあたっては、

- ロックの対象となるレコード数を極力少なくする。
- ロックをかける期間を極力短くする。

という大原則に従って、システム設計と運用を行う必要があります。
一般に、データベースのデータは相互に複雑な関連を持っているものであり、不正更新を確実に防止しようとすると多くのレコードにロックをかけなければならなくなる傾向があり、不正更新の防止とロックの最小化とは相矛盾する要求となりますので、両者を実用上問題ないように最適な設計を行うのが重要です。
以上見てきた並行制御の問題は、Magic に限らず、複数ユーザが同時に同一データをアクセスする可能性のあるすべての情報システムにおいて必ず考慮しなければならない問題です。

8.4.Magic のロック機構の基本

Magic では並行制御を簡単に行えるよう、さまざまな機能が用意されています。その中で基本となる、オンラインタスクでのロック機構についてここで説明します。
図 8 は、Magic のオンラインタスクの基本的なレコードアクセスとロックの流れを、Magic の処理レベルと対比して示したものです。

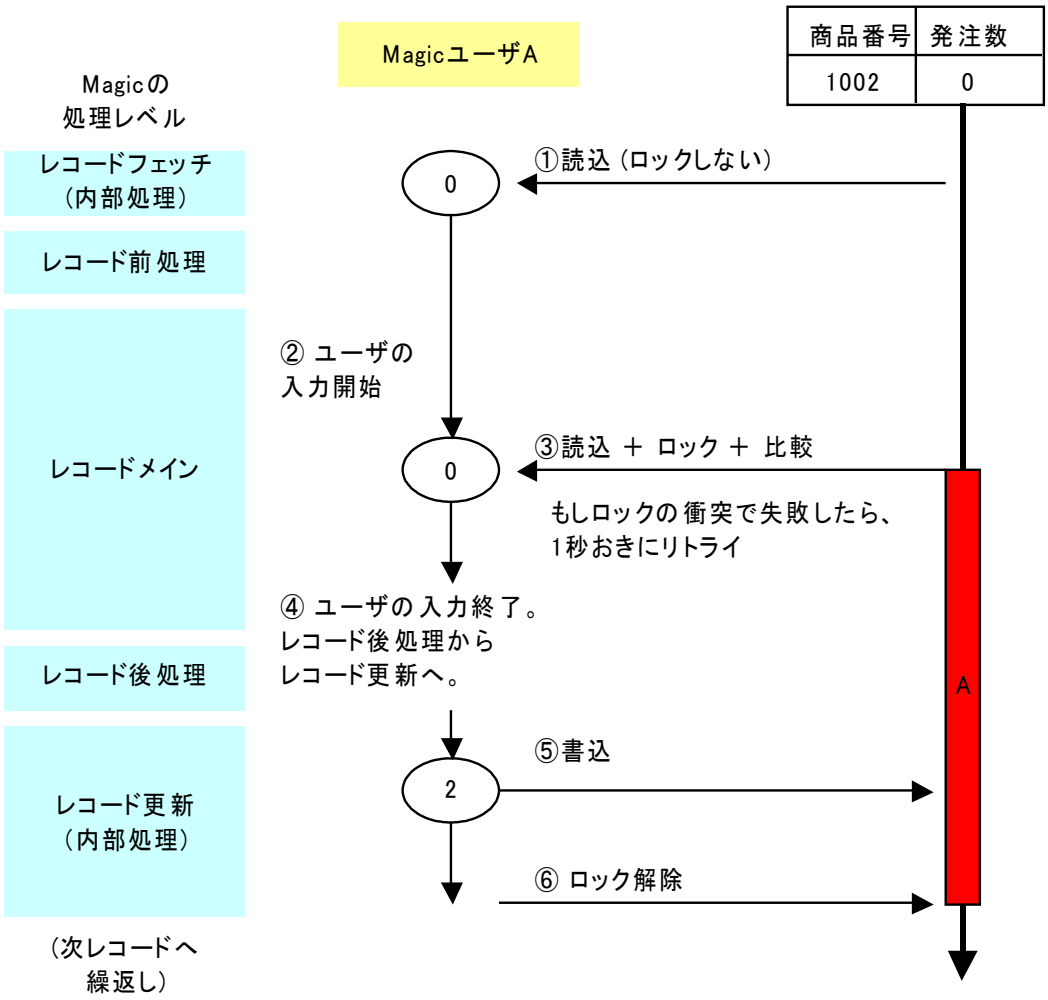


図 8 Magic の基本ロック機構

1. Magic は最初、DBMS からレコードを読み込みます。このときには、「ロックをかける期間を極力短くする」という原則に従い、レコードにロックをかけません。
2. レコード前処理を経て、レコードメインに進むと、Magic はユーザのキー入力を待ちます。
3. ここでユーザがキー入力を始めた瞬間、このレコードにロックをかけます。このとき同時に、レコードが他のユーザにより更新されていないかを確認するため、Magic はこのレコードを再度読み込み、最初読み込んだときと値の比較を行います。
4. ユーザの入力が終了したら、Magic はレコードメインからレコード後処理、更にレコード更新のレベルに進みます。
5. 修正されたレコードを DBMS に書き込みます。
6. 最後に、レコードのロックを解除します。

この流れが基本ですが、他のユーザが同時利用しているので、ユーザが①でロックなしで読み込んだ後、ユーザ B がロックをかけ、データを更新してしまう、ということが起こりえます。このときに起こる流れを示したのが図 9 です。

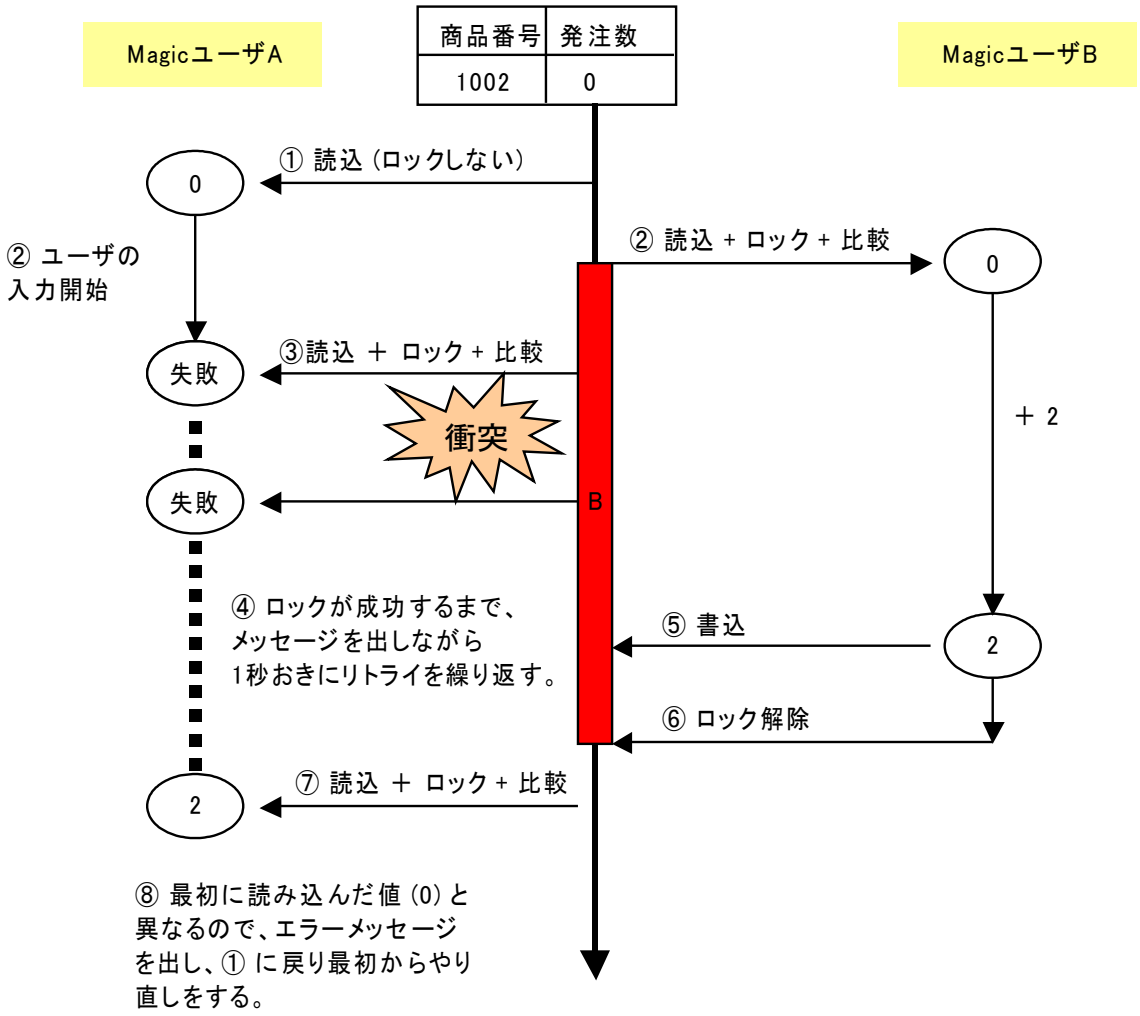
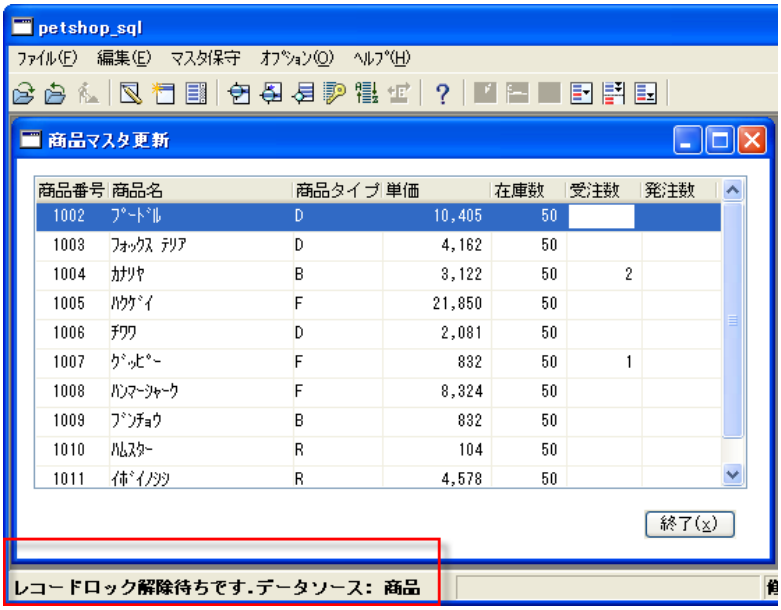


図 9 Magic でロック衝突が起きた場合の処理の流れ

ここでは、次のような流れとなっています。

1 ユーザ A がレコードを読み込みます。最初はロックなしで読み込みます。
2 次に、ユーザ B がデータ入力を始めたので、レコードを読み込むと同時にロックをかけたとします。
3 ユーザ A の方もデータを入力したため、Magic はレコードを再度読み込むと同時にロックをかけようとしますが、B がすでにロックしているのでロックの衝突により失敗します。
4 このような場合には、Magic はステータス行に「レコードロックの解除待ちです」というメッセージを出し、1 秒おきにリトライを繰り返します。(次図)

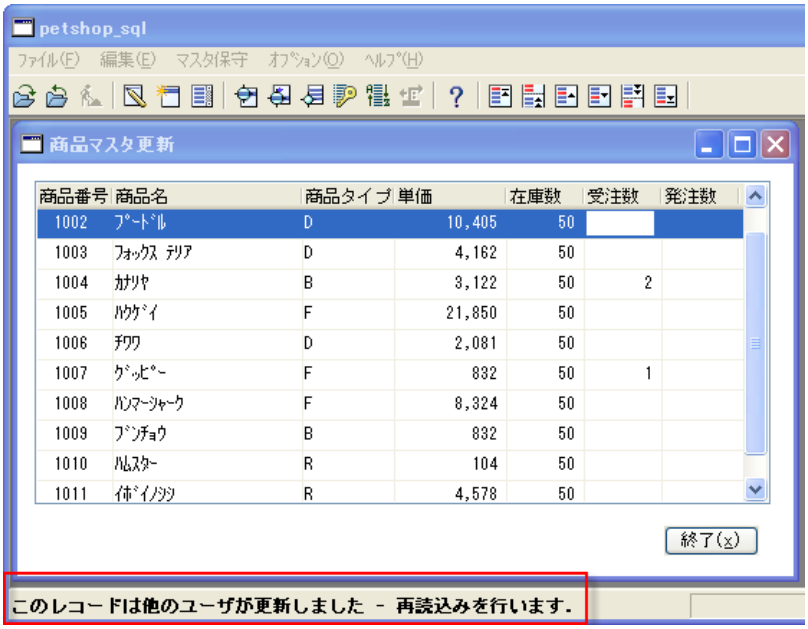


5 ユーザBは発注数を2とし、DBMS に書き込みます。

6　次いで、ロックを解除します。

7　ユーザ A がその後リトライすると、今度はロックが解除されているので読み込みが成功します。

8　ここで、Magic は値の比較をします。この例では、最新の値 (2) が最初に読み込んだ値 (0) と異なるので、Magic は、レコードが他のユーザ(この場合ユーザ B)によって変更されたことを検出しますので、このまま続けることはできず、ステータス行に「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.」というエラーメッセージを出して、新しいレコードについて ① からやり直しをします。



このように、Magic エンジンがレコードのロックの制御を自動的にやってくれるので、開発者はこまごまとしたロックの制御をプログラムに作りこむ必要がなくなり、プログラム開発が非常に簡単になります。

8.5.Magic のロック機構の補足事項

これまでは、Magic のロック機構の基本として、オンラインタスクで典型的な(デフォルトの)ロックの設定で説明しましたが、Magic のロック機構はその他にもさまざまな設定があります。ここではロックのバリエーションについて説明するとともに、いくつかの補足事項も説明します。

8.5.1.ロックのタイミング

先の例のオンラインプログラムでは、ユーザがキー入力した瞬間にレコードロックがかかるようになっていましたが、それ以外のタイミングでロックをかけたり、あるいは全くロックをかけないようにすることもできます。
ロックのタイミングの制御は、タスク特性の「データ(D)」タブにある「ロック方式」パラメータにより設定されます（図 10）。

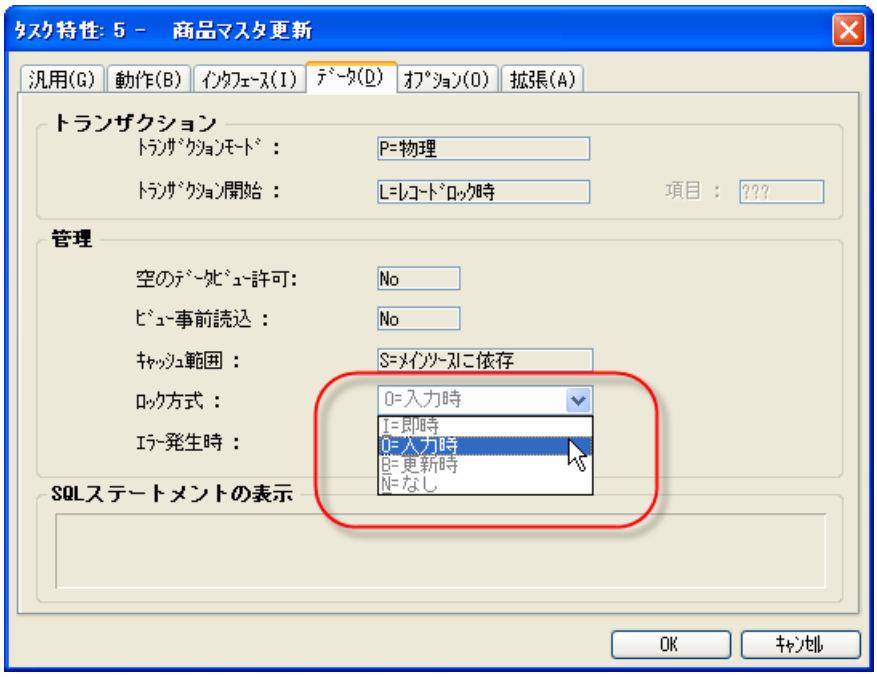


図 10 ロック方式パラメータ

このパラメータでは、次のような設定を行うことができます。

ロック方式	意味
I=即時	カーソルがレコードに移動するとすぐ、ユーザの入力に関わりなくロックをかける。
O=入力時	(デフォルト) ユーザがキー入力を始めた瞬間にロックをかける。
B=更新時	レコード後処理の後、DBMS にレコードを書き込むときに至ってからロックをかける。
N=なし	ロックは一切かけない。

表 3 ロック方式パラメータの意味

オンラインプログラムでは、通常デフォルトの「O=入力時」が最適ですが、特殊な要件のためにタイミングを変える場合には、このパラメータを変更します。

8.5.2.バッチタスクの場合

バッチタスクの場合には、ユーザのキー入力がなく、各レコードごとの処理時間も通常は極めて短いので、デフォルトのロック方式は「I=即時」となります。オンラインプログラムを開発版で開き、「タスクタイプ」を「O=オンライン」から「B=バッチ」に変更すると、Magic は自動的にロック方式を I=即時に変更します。

8.5.3.リンクレコードへのロック

メインテーブルのレコードにロックがかかるタイミングで、リンクコマンドによりリンクされているレコードに対してもロック

がかかります。
例えば、受注入力の親タスクでは、メインソースが受注テーブルですが、これに制御テーブル、顧客テーブルがリンクされています。図 11 では、受注番号#101 のレコードに対し、顧客番号#1008 の顧客レコードがリンクされています。また、画面では見えませんが、キーの値が 1 の制御テーブルのレコードもリンクされています。受注#101 のレコードがロックされるのと同時に、これらのレコードもロックされます。

図 11 受注入力画面

また、同様に、受注入力の子タスクでは、メインソースが受注明細テーブルで、それに商品レコードがリンクされています。上の例では、受注番号#101 の明細#1 には商品#1002 がリンクされているので、子タスクでロックがかけられる場合には、受注明細のレコード#101 とともに、商品#1002 のレコードもロックされます。

8.5.4.リンクレコードへのロックの制御

場合によっては、リンクされたレコードにロックをかけないようにしたい場合があります。例えば、読み込み専用のレコードであればロックする必要はありません。不必要なロックを減らすのは、マルチユーザ環境で円滑な運営をするための大原則なので、ロックの対象とするレコードはきめ細かく制御する必要があります。
特定のテーブルへのリンクレコードのロックを行うか否かはリンクコマンドの「アクセス」特性により指定できます。

1. プログラムリポジトリからタスクを開きます。
2. データビュー を開き、アクセスパラメータを設定したいリンクコマンド（下図の例では L=照会リンク）にカーソルを合わせます。

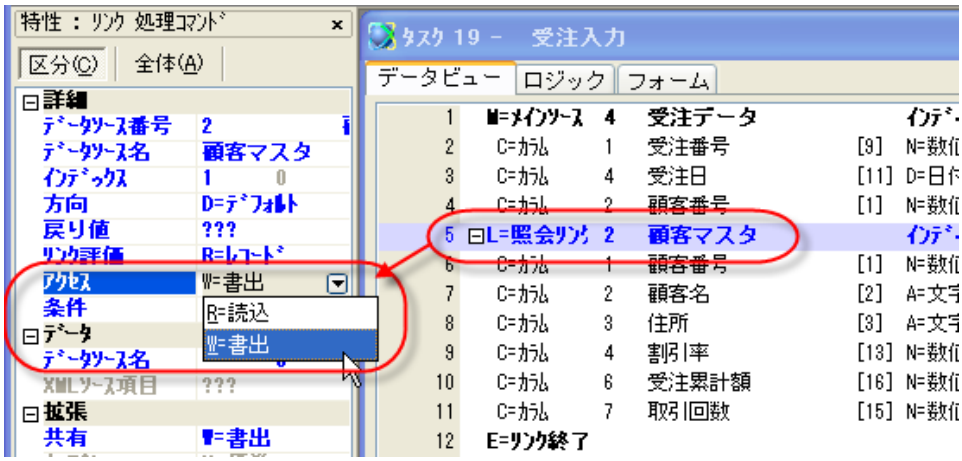
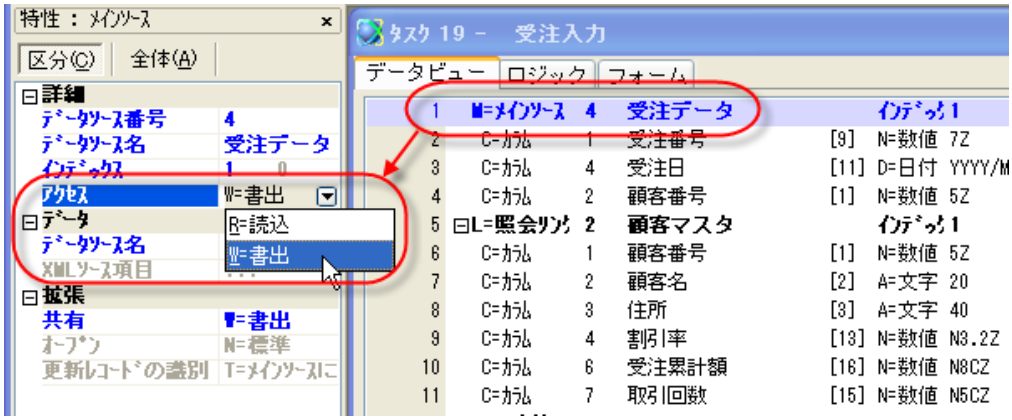


図 12 リンク処理コマンド特性の「アクセス」パラメータ

3. 特性シートの「アクセス」特性で、アクセスの方法を設定します。

「アクセス」特性は、すべてのテーブルに対してデフォルトで「W=書出」となっています。これは、レコードの更新を行う意図がある、という意味です。
特定のテーブルに対してレコードロックを行わないようにしたい場合には、アクセス特性を「R=読込」に設定します。これは、このタスクでは、このテーブルのレコードの更新を行う意図はない、つまり読み込み専用でこのテーブルを使う、という意味で、R=読込になっているテーブルに対しては、Magic はレコードロックを行いません。
マルチユーザ環境で起こるレコードロックの衝突を回避するために、R=読込の設定にすることが良く行われます。例えば、制御テーブルのロックの衝突を避けるために、制御テーブルのアクセスパラメータを R=読込 にします。
もちろん、アクセスパラメータを変更した場合には、いろいろとプログラムを修正する必要があります。例えば、制御テーブルはレコード後処理で項目更新コマンドで更新しているので、R=読込にしただけだと、実行時にエラーとなります。読込専用なのにデータを更新しようとしたからです。
エラーを避けるためには、このタスクではレコードを更新しないようにしなければなりません。通常は項目更新コマンドで更新する代わりに、制御テーブルのレコードの更新を行うバッチタスクを別途作成しておき、このバッチプログラムを呼び出すことにより更新を行うようにする、という手法をとります。リンク処理コマンド特性の設定の有効範囲はタスク単位なので、別タスクを立ててやれば更新することが可能となります。この手法については、11 章で説明します。

ここではリンクレコードについてだけ説明しましたが、メインデータソースについても、アクセス特性を設定し、ロックの有無を制御することができます。アクセス特性を設定するには、データビューの「M=メインソース」行にカーソルを位置付け、特性シートで「アクセス」特性を設定します。



8.5.5.タスクのモードとの関係

レコードロックは、レコードを変更する可能性のある場合にだけ必要となります。従って、タスクが修正モードの場合には、今まで述べてきた条件に従ってレコードロックがかかります。その他のモードの時には、次のようになります。

- **照会**: タスクのモードが「照会」の場合には、すべてのレコードは読込専用で、データの修正が起こらないため、メインソース特性やリンク処理コマンド特性の設定にかかわらずレコードロックはかかりません。
- **登録**: タスクが登録モードの場合には、メインデータソースのレコードにはロックがかかりません。登録モードの場合には、まだレコードが DBMS に作成されていないので、ロックすべきレコードがないからです。ただし、登録モードであっても、リンクされたレコードにはロックがかかります。
- **削除**: バッチタスクでは、初期モードとして「D=削除」というものがあります。これは、メインテーブルのレコードのうち、範囲指定などの条件に合致するレコードを 1 件づつ削除するモードです。削除モードでは、修正モードと同様にレコードロックがかかります。

9. トランザクション

前章ではロックに関して説明しましたが、MSSQL Server のような SQL データベースを利用する場合には、ロックと不可分な関係にあるトランザクションについてもよく理解しておく必要があります。本章ではトランザクションの基本について MSSQL Server を使って簡単に説明します。

Magic では Pervasive を使い、トランザクションを使わずにアプリケーションを開発されている開発者も多いので、本章では MSSQL Server のトランザクション機能の説明に加えて、Pervasive.SQL でトランザクションを使わない場合の動作との比較も説明します。

- トランザクションとロックの機能については、DBMS ごとに細かな違いがありますが、本章で説明する内容は、他の SQL データベース（Oracle、DB2/UDB など）でも基本的に同じです。
- 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構いません。
- 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、MSSQL Server、 Oracle、 DB2/UDB などの SQL 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、SQL 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みます。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」(37ページ)で作った、簡単な商品テーブルを使ってみます。

最初に、データの初期状態を見てみます。

```
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 文の処理内容を一度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、MSSQL Server では、分離度の低い順に、次の四つが定義されています。[1]

- 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、より高度なデーター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。

[1]MSSQL Server 2005 では、これ以外に分離レベルが拡張されていますが、本書では省略します。

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。

```
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。(以下の説明では、ユーザ A のアクションを左側に、ユーザ B のアクションを右側の枠で表示します)

```
（ユーザ A で実行）
1> set transaction isolation level read uncommitted
2> go
```

```
（ユーザ B 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 で実行)
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 で実行）
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 で実行）
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 で実行）
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 で実行）
1> set transaction isolation level read committed
2> go
```

```
（ユーザ B 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 で実行）
```

```
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 で実行)
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 で実行)
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 で実行)
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.MSSQL Server でのロック

MSSQL 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に使い、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	MSSQL 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 MSSQL 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、MSSQL 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、MSSQL Server のような SQL データベースの場合には、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

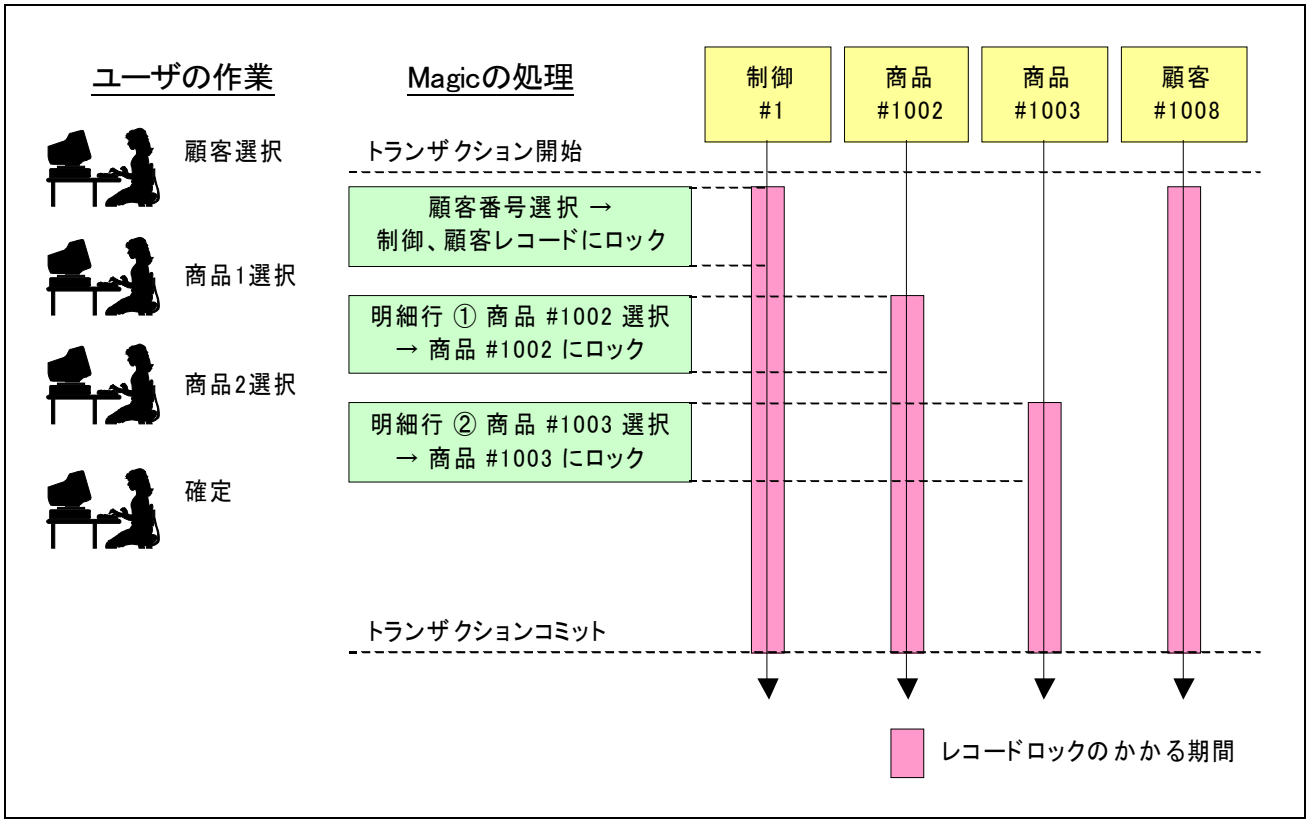
時に一括して行われます。このため、トランザクションの開始から終了までの期間をよく検討して設計しなければ、非常に長い間レコードロックがかかったままになってしまう可能性があります。
Magic を始めて使う開発者はもちろん、今まで Pervasive だけで Magic アプリケーションを開発してきた開発者も、MSSQL Server などの SQL データベースを使う場合には、ロックとトランザクションの関係を良く理解して設計してください。

9.5.3. トランザクション中にロックが累積する例

具体的な例を挙げて、トランザクション中にロックが累積するようすを見てみましょう。次のような受注入力操作を行った場合には、Magic のロック機構によるレコードのロックの状況は図 13 のようになります。

1. ユーザが受注入力タスクを開始します。
2. 顧客 #1008 を選択します。このときに、トランザクションが開始され、制御レコードと顧客#1008 のレコードにロックがかかります。
3. 明細行に入り、商品 #1002 を選択します。このとき、商品 #1002 にロックがかかります。
4. 明細行の 2 行目に移ります。このとき、まだトランザクション中なので商品#1002 のロックは解除されません。
5. 明細行の 2 行目で、商品 #1003 を選択します。このとき、商品 #1003 にロックがかかります。
6. 以下同様に、明細行を追加していくたびに、選択された商品のレコードにロックがかかります。この間、商品レコードのロックは解除されないので、選択されたレコードのロックが蓄積されていきます。この間、入力は手作業であるため、長い間多くのレコードがロックされている状態が続きます。
7. 最後に受注入力を終了すると、トランザクションがコミットされるので、修正内容が DBMS に反映されるとともに、レコードロックがすべて解除されます。

図 13 MSSQL Server の場合のロックの期間



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、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。この制御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。

10.移行直後の動作確認

本章では、マルチユーザ環境やトランザクションについてよく考慮してこなかったペットショップデモを、MSSQL Server に単純移行した直後の状態で、マルチユーザ環境で利用した場合にどのような現象が起こるのかについて、実際に操作しながら説明していきます。

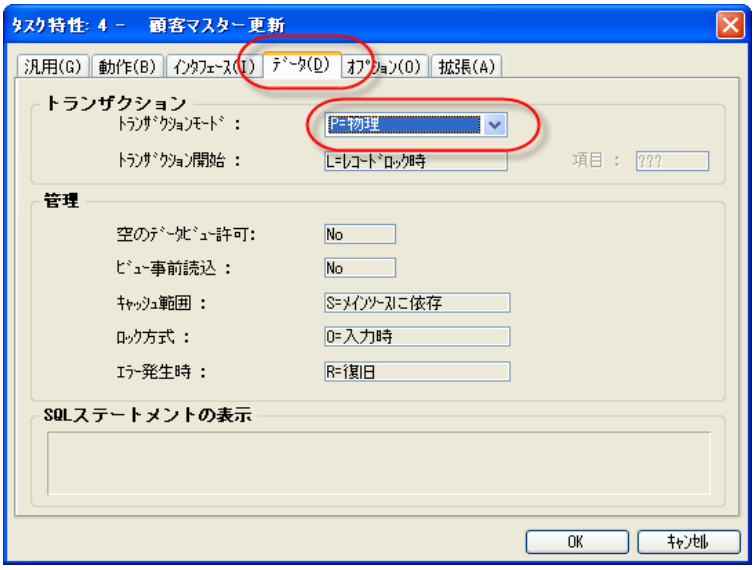
ここでは、最初にトランザクションの設定の確認を行います。Pervasive.SQL を使ったペットショップアプリケーションではトランザクションを使っていなかったので、トランザクションの設定に気を使う必要はありませんでしたが、MSSQL Server などの SQL データベースを使う場合には、必ずトランザクションの設定が必要になります。この違いに起因して、プログラムもいろいろ変更する必要が出てきます。詳しいことは「11 章 受注入力プログラムの変更」(89ページ)で説明します。

10.1. トランザクション設定の確認

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」としています。

トランザクションモードの確認は、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、メニューで「タスク環境 (K) → タスク特性(T)」を選ぶか、Ctrl+P キーにより開きます。
4. 「データ(D)」タブを開きます。
5. 「トランザクションモード」が「P=物理」になっていることを確認してください。



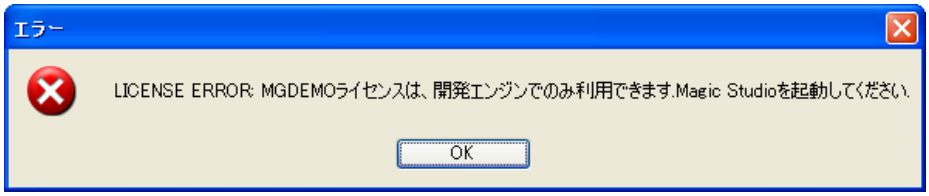
受注入力プログラムのように、サブタスクのある場合には、サブタスクのトランザクションモードは「W=親と同一」にしてください。バッチタスクでは、デフォルトでトランザクションモードが「P=物理」になっているので、変更する必要はありません。

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す。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
```

10.2.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。
開発版には、実行エンジン eDevRTE.exe が添付されているので、これをクライアントとして起動すればよいのですが、体験版の場合にはライセンスの関係で eDevRTE.EXE を直接起動することができません。起動しようとすると、プロジェクトをオープンした時点で右図のようなエラーが出てしまいます。

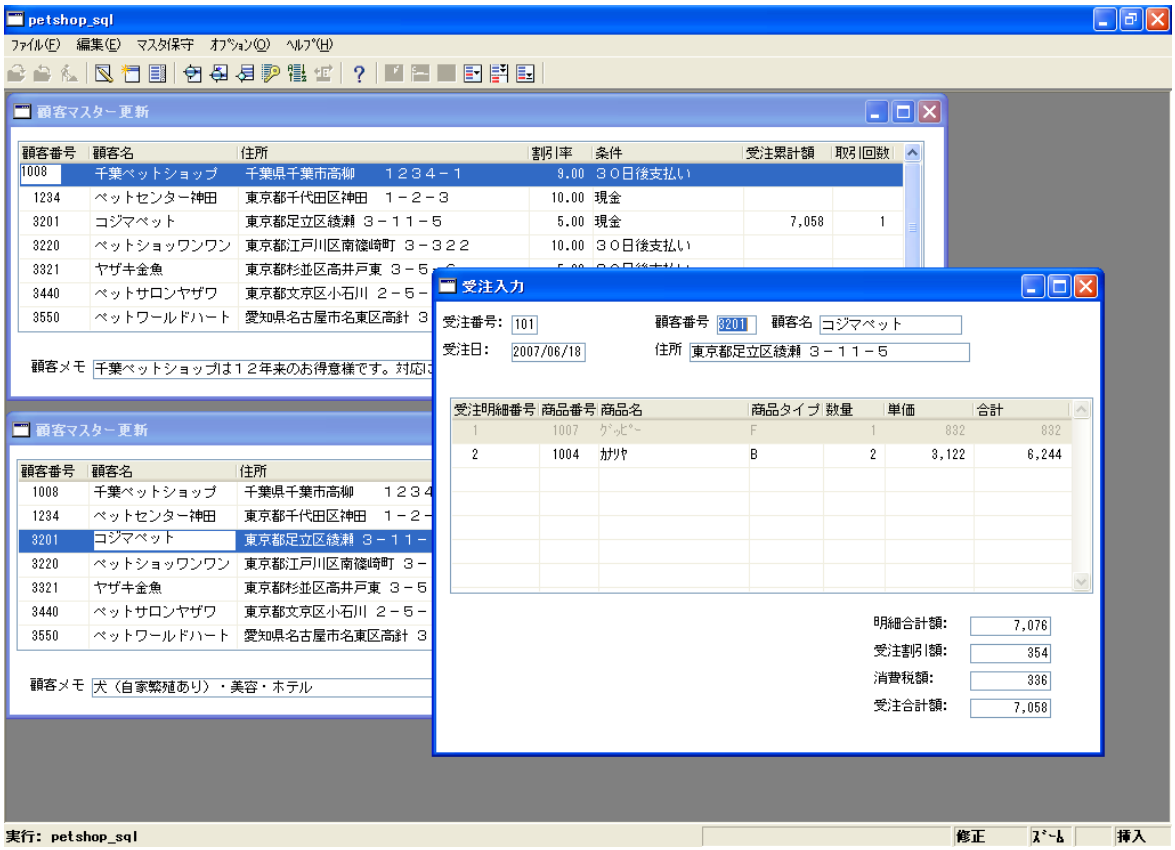


本書では、体験版で使う読者もいることを想定して、体験版でもマルチユーザ環境の実験を行えるように、V10 の新機能である「並行実行」を利用して、マルチユーザ環境を模擬的に作成してテストすることにします。

Magic Studio V10 正規製品版を利用している場合には、クライアントプログラム eDevRTE.EXE を実行しても、上図のようなエラーは出ませんので、そのままキャビネットファイルを作成してクライアントプログラムを二つ起動してテストをすることができます。こちらの方がより現実に近い実験を行うことができます。
以下に説明する並行実行の設定は、体験版を使っている場合にだけ行ってください。

10.2.1.並行実行とは？

並行実行とは、Magic をひとつだけ起動しておいて、その中で複数のプログラムを同時並行に実行することのできる機能です。V9 のサーバ版で導入されましたが、V9 および V9Plus では、バックグラウンドのアプリケーションサーバとして、バッチタスクあるいはブラウザクライアントタスクでだけ利用することができました。
V10 では、オンラインタスクでも、並行実行の機能を利用することができます。下図は、顧客マスタプログラムを二つと受注入力プログラムを並行実行している画面です。

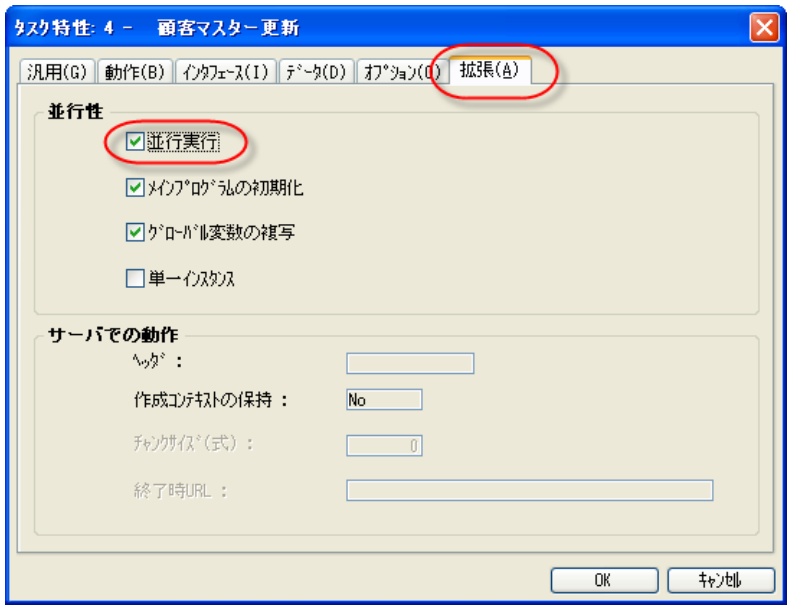


ここで、各プログラムはそれぞれ独立したコンテキストで実行しています。コンテキストというのは、メモリテーブルやグローバル変数、コールスタックなどの内部データの集まりで、レコードロックやトランザクションの情報も含まれます。各タスクが別コンテキストなので、それぞれのタスク間では、あたかも異なるクライアントが別 PC から実行しているのと同じように、排他制御がかかります。これによって、Studio で擬似的にマルチユーザ環境をテストすることができます。

10.2.2.並行実行の設定

Magic のプログラムのデフォルトの設定では並行実行がされません。並行実行を可能にするには、タスク特性を開き、並行実行パラメータをセットします。

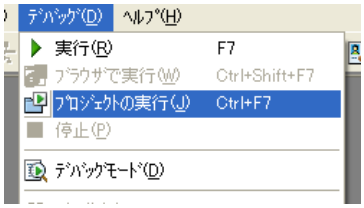
1. プログラムリポジトリから、並行実行させたいプログラムを F5 で開きます。
2. Ctrl+P で、タスク特性を開きます。
3. 拡張(A) タブを開きます。
4. 「並行実行」をチェックします。
   これに伴い、「メインプログラムの初期化」「グローバル変数の複写」にもチェックが入りますが、これらはそのままにしておきます。
5. Enter キーを 2 回押して、プログラムを閉じます。



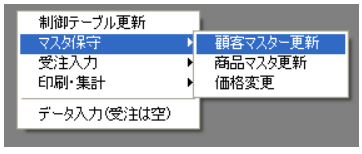
これで並行実行が可能になります。

試しに、プログラムを二つ並行実行させてみましょう。

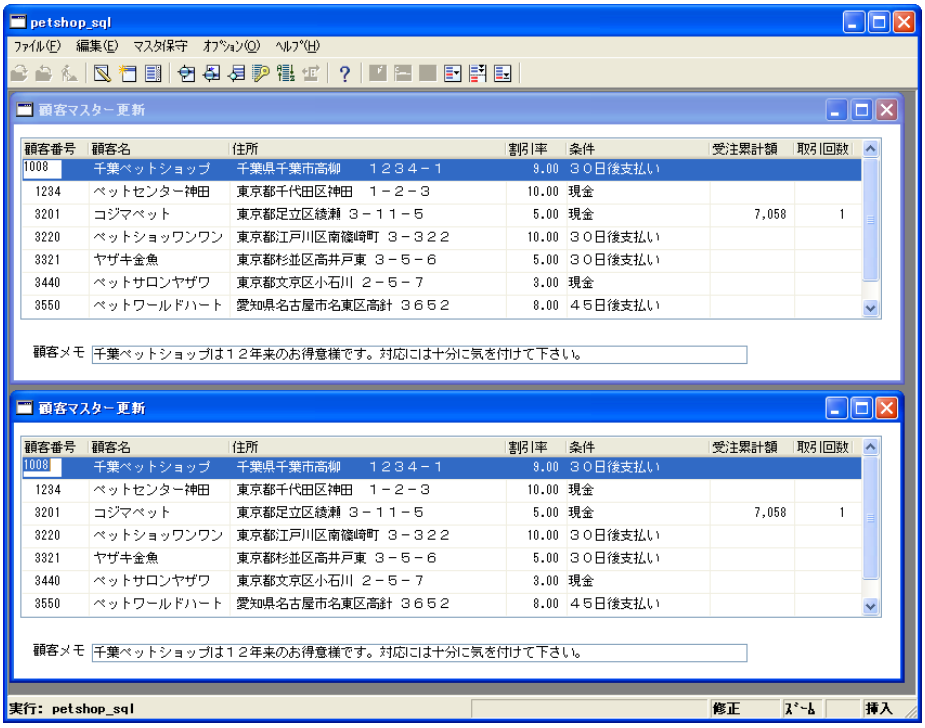
1. メニュー「デバッグ(D) → プロジェクトの実行 (J)」(あるいは Ctrl+F7) で、実行モードになります。



2. ポップアップメニューから、「マスタ保守 → 顧客マスター更新」を選択します。
   顧客マスター画面が表示されます。



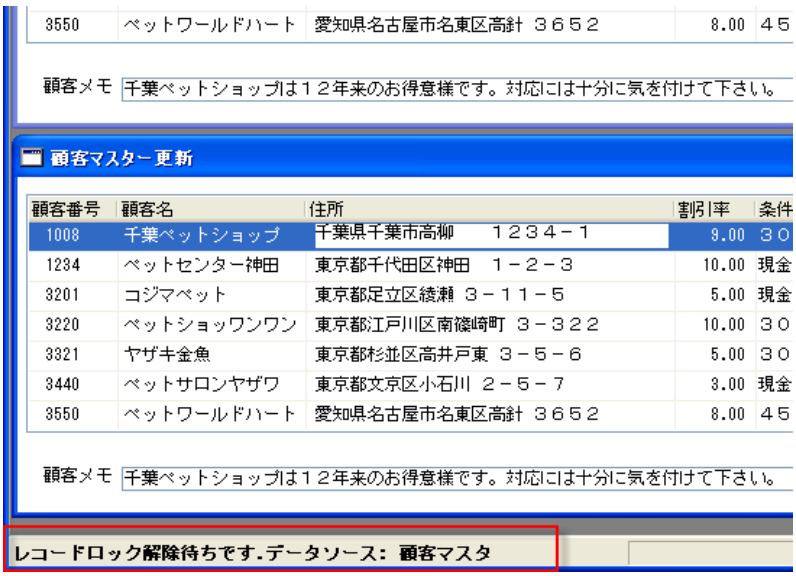
3. もう一度、「マスタ保守 → 顧客マスター更新」を選択します。
顧客マスター画面がもうひとつ表示されます。
（画面が重なっているので、マウスでずらしてください）



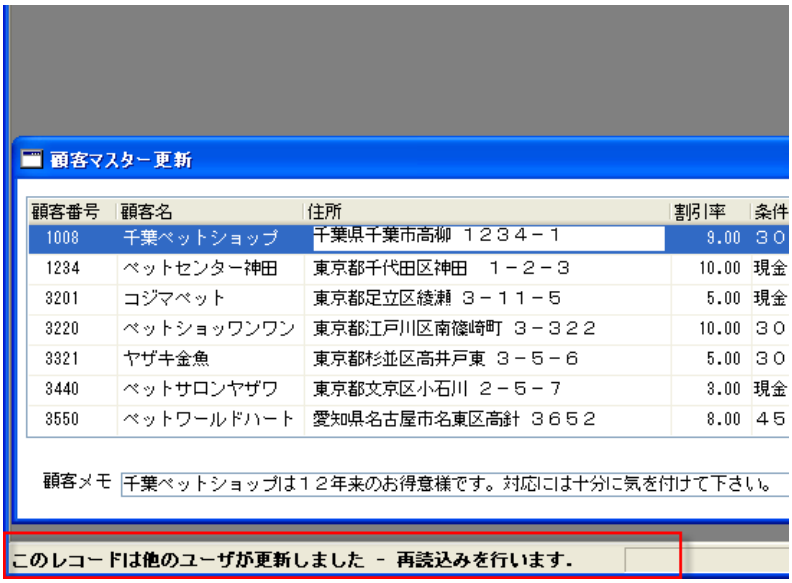
10.3.テーブルを一つだけ使うタスクの場合

前節では、並行実行の機能を使って、顧客マスタのプログラムを二つ開きました。これは、2 人のユーザが同時に顧客マスタをアクセスしている状態を想定しています。
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。
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. 上側の顧客マスター更新プログラム(二人のユーザになぞらえて、ユーザ A と呼びます)で、「顧客名」を適当に修正します。
2. 下の 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 MSSQL 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。

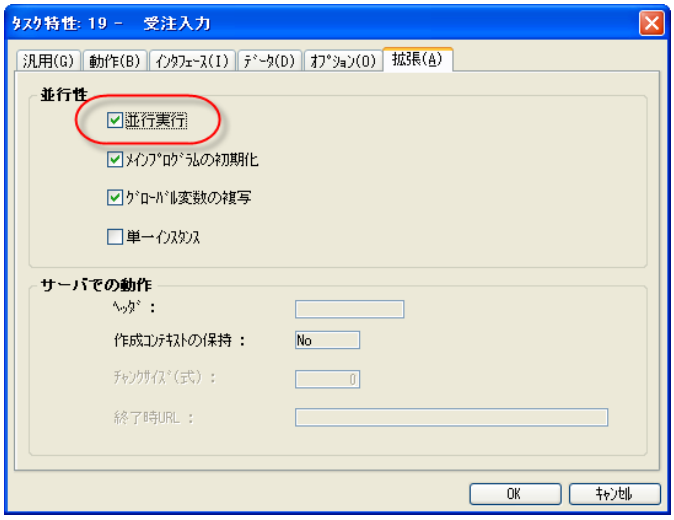
他のマスターメンテナンスプログラム(制御テーブル更新、商品マスター更新)でも同様に、MSSQL Server に移行しても特に問題はありません。

10.4.テーブルを複数使うタスクの場合

前節では、テーブルを一つ使っただけのごく簡単なオンラインプログラムについて調べてみましたが、このような簡単なマスターテーブル更新プログラムなどでは、プログラムに手を加えなくとも、Magic のロック機能のみに頼って十分な排他制御ができました。
しかし、複数のテーブルをリンクしたり、受注入力のように親子のタスクで 1:N 関係のレコードを扱う場合などでは、**並行性**（複数のユーザが、ロック待ちなどにより中断させられてしまうことが極力なく、同時並行してデータをアクセスできる度合い）を確保するために、プログラムを変更する必要が出てきます。
本章では、マルチユーザ環境を意識せずに作成した受注入力プログラムを二人のユーザが同時に実行した場合に起こるロックの問題を具体的に見ていきます。

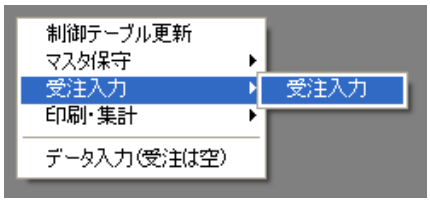
10.4.1.受注入力画面のマルチユーザ環境下での実行

1. 前の顧客マスター更新プログラムと同様に、受注入力プログラムを開き、タスク特性 → 拡張(A) タブで、「並行実行」をチェックします。



2. プログラムを閉じます。

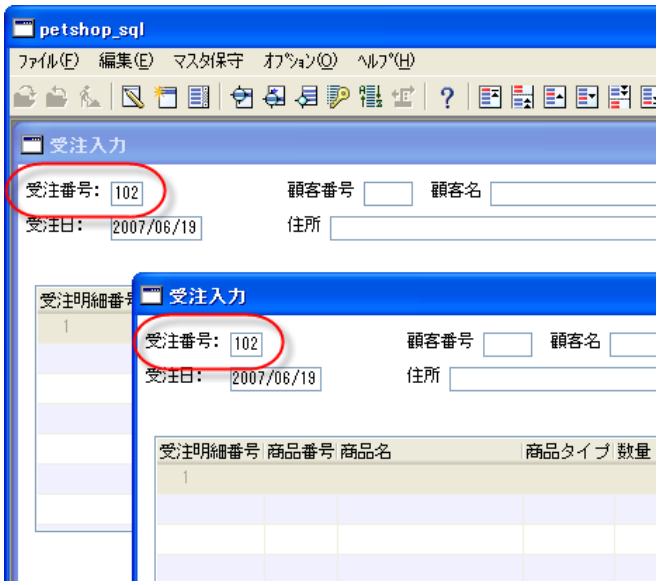
3. メニュー「デバッグ(D) → プロジェクトの実行(J) 」を選んで、実行モードになります。
4. ポップアップメニューで「受注入力 → 受注入力」を選びます。受注入力プログラムが開始します。



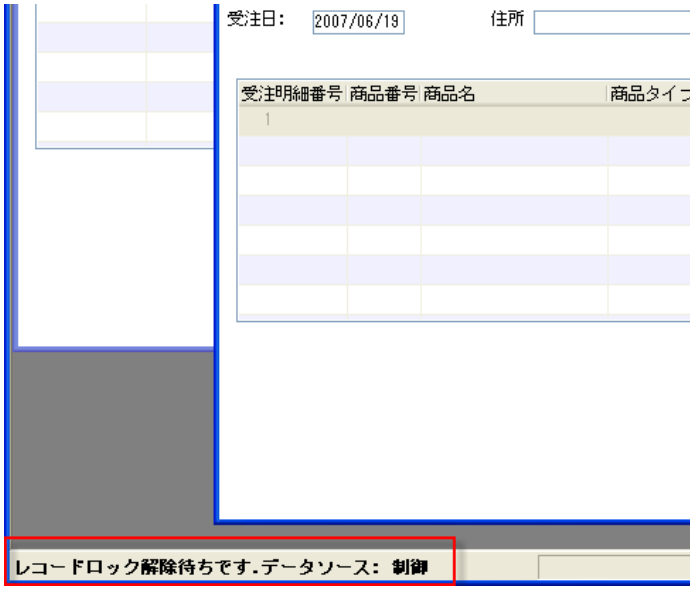
5. これを 2 回繰り返し、受注入力プログラムを二つ開きます。
   これは、二人のユーザが同時に受注入力プログラムを実行している状況を想定しています。

   ここでよく見てみると、両方の画面で、受注番号が 同じ (ここでは 102) になってしまっています。このへんから問題がありそうですが、とりあえず先に進みましょう。

   以下の説明では、先に起動した方を「ユーザ A」と呼び、後から起動した方を「ユーザ B」と呼びます。

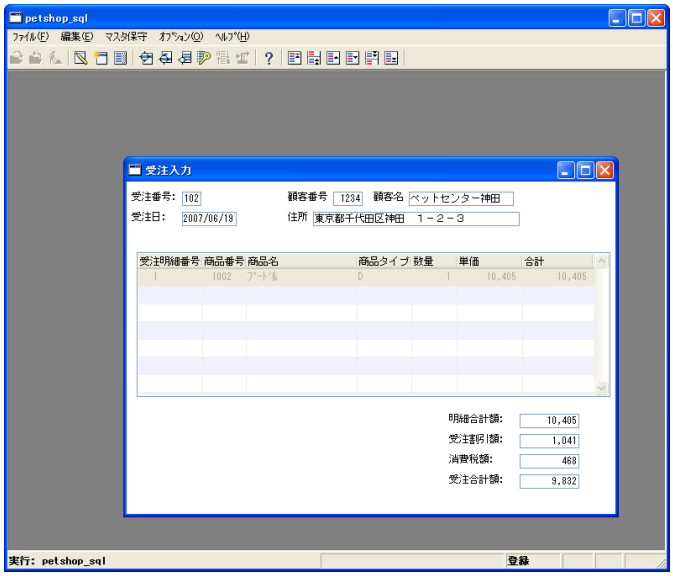


6. ユーザ A の方で、顧客番号 #1008 を選択し、Tab を押します。カーソルは子タスクの最初の行に移動します。
7. ユーザ B の方で、顧客番号 #1234 を選択します。すると、「レコードロック解除待ちです. データソース: 制御」が出ます。



8. 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。
   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。

9. 同様に、明細行で商品番号#1002(プードル)などを一つ注文し、ESC を押します。すると、一瞬「インデックスが重複しています. データソース: 受注明細」というエラーが出て、表示がリセットされてしまいます。



このように、受注入力プログラムでは、データの排他制御が適切に行われていないことがわかります。
このようなことの起こる原因は、MSSQL Server のロックと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、次に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」(89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。

10.4.2.制御テーブルのロック衝突

レコードロックの衝突が起こっているのは、ステータス行のエラーメッセージにあるように、「制御」テーブルです。これは、レコードメインでリンクコマンドによりリンクされているテーブルです。
ユーザ A で顧客番号が入力されると、レコードロックがかかりますが、このときにロックのかかるレコードは、以下のようになります。

テーブル名	ロック対象レコード	備考
受注データ	現在表示中のレコード	修正モードのときのみロックされる。登録モードの場合には、対象レコードがまだ存在していないのでロックはかからない。
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めに「最終受注番号」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。

この状態でユーザ B が顧客番号を入力しようとすると、まず最初に、制御データテーブルの中のキー値が 1 のレコードがリンクされるので、このレコードがロックされます。ところが、このレコードはすでにユーザ A がロックしているので、ロックの衝突が起こります。このために、ユーザ B はレコードロック解除待ちとなります。

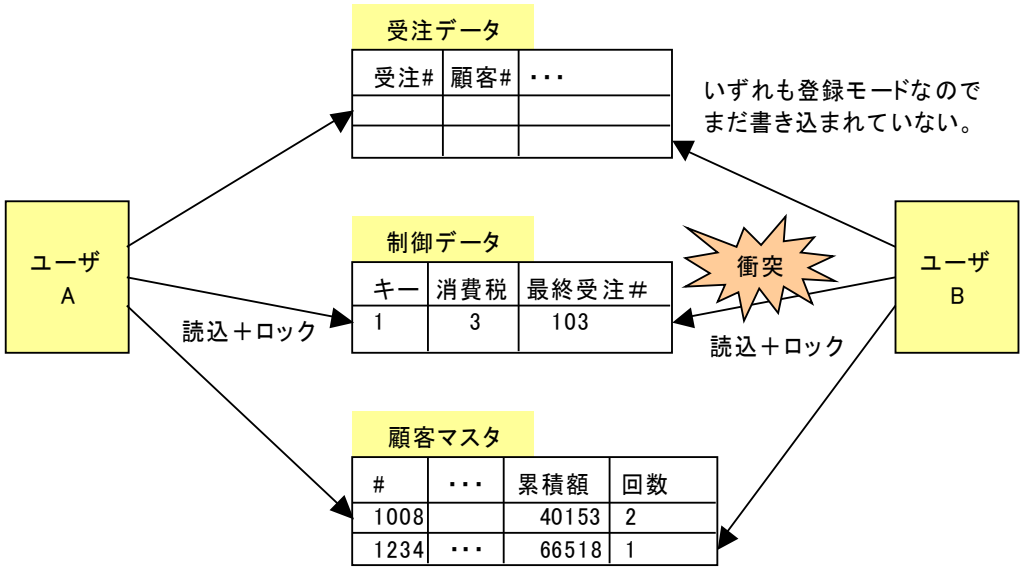


図 14 制御データのロック衝突

10.4.3.顧客マスタのロック衝突

問題は制御テーブルのレコードロックだけではありません。制御テーブルのロック衝突をなんらかの方法で回避したとしても、次には顧客レコードがロックの衝突を起こしてしまいます。この問題は、ユーザ A、B ともに、同じ顧客番号を選択した場合に起こります。例えば #1008 を選択したら、何が起こるでしょうか？

1. ユーザ A、ユーザ B ともに、受注入力2プログラムを起動します。
2. ユーザ A で、顧客 1008 を選択します。
3. ユーザ B でも、顧客 1008 を選択します。

このようにすると、ユーザ B の方でロックの衝突が起こります。なぜかというと、受注入力プログラムでは、制御テーブルのほかに、顧客レコードもリンクしていますので、顧客レコードにレコードロックがかかります。ユーザ A が顧客 1008 を選択したときにこのレコードがロックされますから、ユーザ B も顧客 1008 を選択しようとするとロックの衝突が起こります（図 15）。

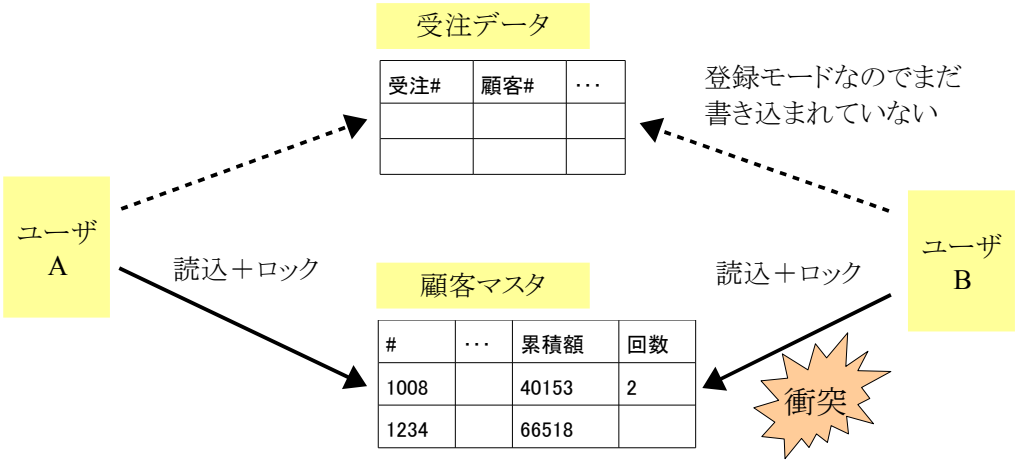


図 15 顧客レコードのロックの衝突

10.4.4.商品マスタのロック衝突

親タスクで顧客マスタのロック衝突が起こるのと同じ理由で、子タスクで商品マスタのロック衝突が起こります。これは、ユーザ A と B が同じ商品を選択した場合に起こります。

下図は、ユーザ A とユーザ B が同じ商品 #1002 を選択したときの状態を表したものです。商品#1002 に両方からロックがかけられ、後からロックしようとしたユーザ B にロックの衝突が起こっています。

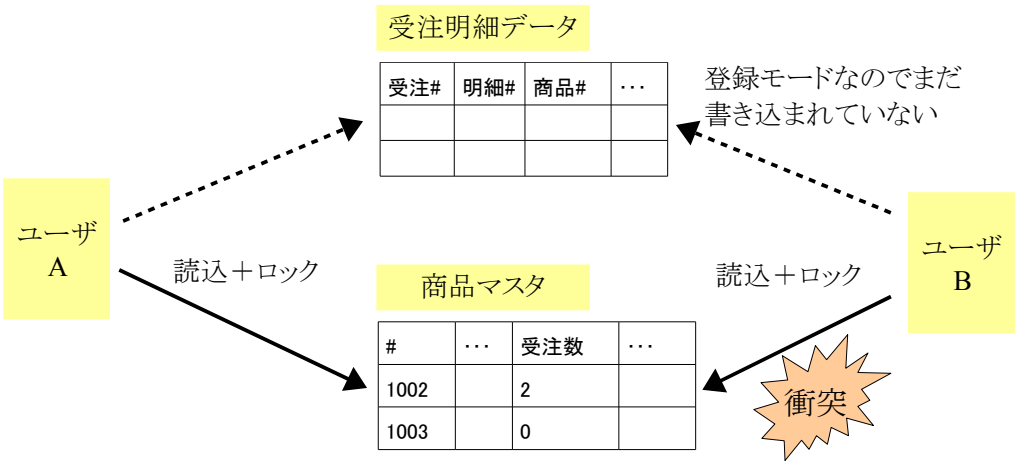


図 16 商品レコードのロックの衝突

11.受注入力プログラムの変更

前章で見たように、マルチユーザ環境ではロックの衝突を考慮する必要があり、特にトランザクションの利用が必須であるSQLデータベースの場合には影響が大きくなりがちです。
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一度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

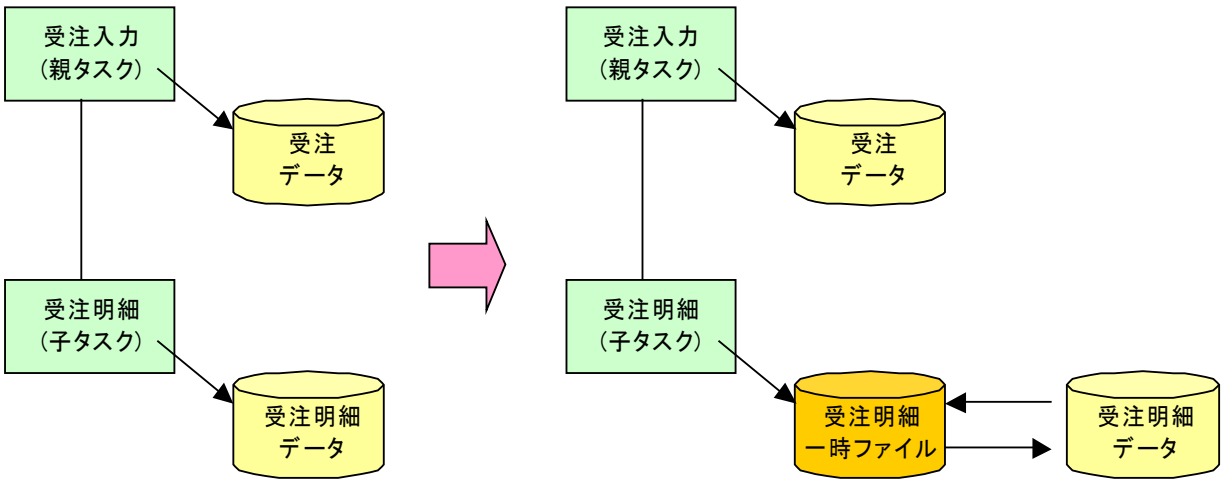
一時テーブルを使ってロック待ちを回避する方法の基本的な考え方は、以下の通りです（図 17 参照）。

- 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)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、照会・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。
    この処理のために、受注明細テーブルから一時テーブルへコピーを行うバッチタスクを作成します。
  - 明細行に対するユーザの入力(登録モード)、あるいは修正(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。
    この処理のために、一時テーブルから受注明細テーブルへコピーするバッチタスクを作成します。

- 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を避けるために、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。

- 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に、制御ファイルは「アクセス」＝「R=読込」でオープンし、新しい受注番号は、連番作成のバッチタスクを呼び出して行います。この連番作成は、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のタイミングで行います。

図 17 受注入力タスクの変更



変更前

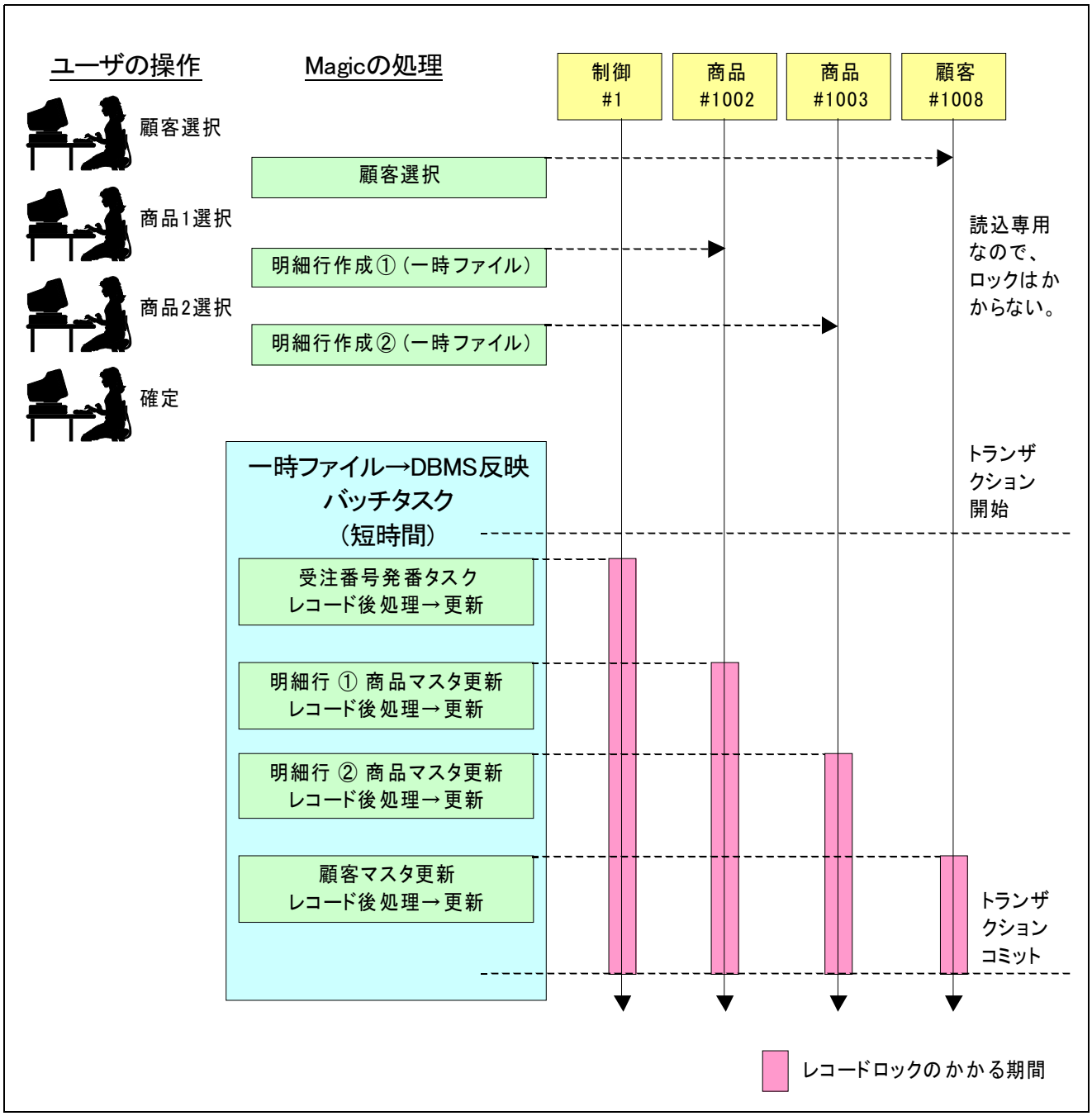
受注データ、受注明細データを直接操作していた。

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 18 です。

図 18 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 MSSQL 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、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り、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。

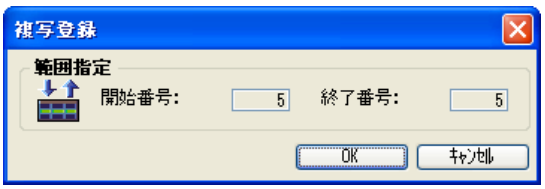
最初に行うことは、明細レコードを一時的に格納する一時テーブルをデータリポジトリに定義することです。これは**メモリテーブル**として定義します。
メモリテーブルというのは、Magic 独自のデータベースで、次のような性質を持っています。

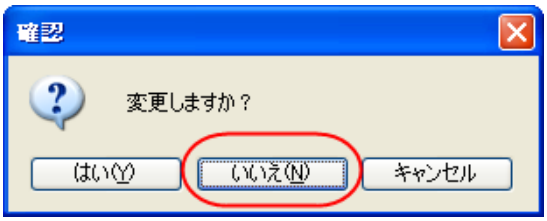
- レコードはすべてメインメモリ上に格納し、ディスク上に保存しません。
- オーバーヘッドが少なく、アクセスが非常に高速です。
- ディスクに保存されないため Magic が終了すると同時にメモリテーブルの内容も消えてしまいます。
- 大量のデータを保存するとメモリを圧迫する原因ともなります。
- 本質的に複数ユーザ間で共有することがありません。そのためロックなどを掛ける必要がありません。また、トランザクションもサポートされていません。

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. データリポジトリを開き、最下行にカーソルを置きます。
2. メニュー「編集(E) → 登録(N) → 複写登録(Y)」(あるいは、Shift+Ctrl+R キー) 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. コピーしたテーブルをもとのテーブルと区別するために、名前を「受注明細データ TMP」に変更します。
6. 「データベース」を「Memory」に変更してください。
7. Enter キーを押して、データリポジトリを閉じます。
   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。





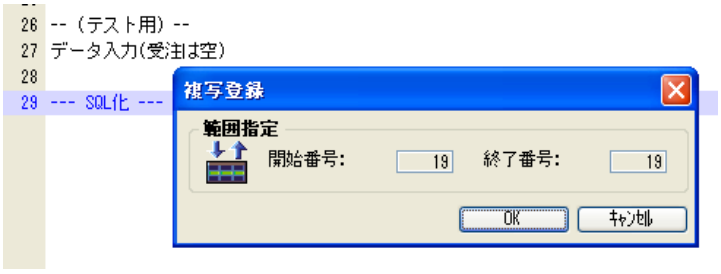
最終的には、図 19 (次ページ) のようになります。

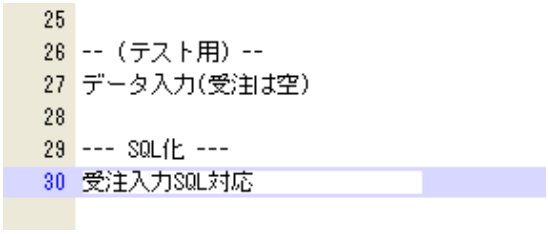
図 19 一時テーブル「受注明細データTMP」の定義

11.3.受注入力プログラムのコピー

以下、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、以下のようにして、コピー後のタスクを「受注入力 SQL 対応」という名前にしておきます。

1. プログラムリポジトリを開き、最後の行にカーソルを置きます。
2. F4 キーを 2 回押します。空のプログラムが二つ作成されます。
3. 二つ目のプログラムの名前欄を「--- SQL 化 ---」としてください。
4. メニュー 編集(E) → 登録(N) → 複写登録(Y) (あるいは、Ctrl+Shift+R) を選びます。→ 複写登録ダイアログが出ます。
5. 開始番号、終了番号として 19 (「受注入力」プログラム)を指定し、OK ボタンを押します。
6. もとのプログラムと区別するため、コピーされたプログラムの名前を「受注入力 SQL 対応」としてください。





11.4.一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:

- 一時テーブルのレコードを削除するバッチタスク
- 受注明細レコードを一時テーブルにコピーするバッチタスク
- 受注明細レコードを削除するバッチタスク
- 一時テーブルのレコードを受注明細にコピーするバッチタスク

その他に、簡単な更新を行うバッチタスクを作成します。

- 新しい受注番号を発番するバッチタスク
- 顧客マスタの受注累積額と取引回数を増減するバッチタスク

これはそれぞれ、制御テーブルのレコード、顧客マスタのレコードをリンクする際に起こるレコードロックを回避するために使うものです。

11.4.1.一時テーブルのレコードを削除するバッチタスク

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。

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
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=タスク前の前
データビュー	
メインデータソース: 6 (受注明細データ TMP)　インデックス: 1 (受注番号) C=カラム　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、データビュー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、一時テーブルのレコードがすべて削除されます。

11.4.2.受注明細レコードを一時テーブルにコピーするバッチタスク

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_明細コピー(元→TMP)
タスクタイプ:	B=バッチ
初期モード:	M=修正
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=タスク前の前

データビュー	
M=メインソース: 5 (受注明細データ)	インデックス: 1 (受注番号)
P=パラメータ PI_受注番号	モデル: 9 (受注番号)
C=カラム 1 (受注番号)	範囲小: PI_受注番号、範囲大: PI_受注番号
C=カラム 2 (受注明細番号)	
C=カラム 3 (商品番号)	
C=カラム 4 (商品タイプ)	
C=カラム 5 (数量)	
C=カラム 6 (単価)	
C=カラム 7 (合計)	
C=登録リンク 6 (受注明細データ TMP)	
C=カラム 1 (受注番号)	代入: B (受注番号)
C=カラム 2 (受注明細番号)	代入: C (受注明細番号)
C=カラム 3 (商品番号)	代入: D (商品番号)
C=カラム 4 (商品タイプ)	代入: E (商品タイプ)
C=カラム 5 (数量)	代入: F (数量)
C=カラム 6 (単価)	代入: G (単価)
C=カラム 7 (合計)	代入: H (合計)
リンク終了	

コピー元のテーブルをメインソースとして設定し、コピー先のテーブルは、登録リンクコマンドで参照します。データビューでは各テーブルの全カラムを選択し、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、「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。

11.4.3.受注明細レコードを削除するバッチタスク

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_明細(元)削除
タスクタイプ:	B=バッチ
初期モード:	D=削除
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=タスク前の前

データビュー	
メインソース: 5 (受注明細データ)	インデックス: 1 (受注番号)
P=パラメータ PI_受注番号	モデル: 9 (受注番号)
C=カラム 1 (受注番号)	範囲小: PI_受注番号、範囲大: PI_受注番号
C=カラム 3 (商品番号)	
C=カラム 5 (数量)	
L=照会リンク 3 (商品マスタ)	インデックス: 1
C=カラム 1 (商品番号)	位置付小: C (商品番号)、位置付大:C (商品番号)
C=カラム 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)

11.4.1 の「一時テーブルのレコードを削除するバッチタスク」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。
11.4.1 の「一時テーブルのレコードを削除するバッチタスク」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。

11.4.4.一時テーブルのレコードを受注明細にコピーするバッチタスク

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_明細コピー(TMP→元)
タスクタイプ:	B=バッチ
初期モード:	M=修正
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=タスク前の前
データビュー	
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)
P=パラメータ PI_受注番号	モデル: 9 (受注番号)
C=カラム 1 (受注番号)	
C=カラム 2 (受注明細番号)	
C=カラム 3 (商品番号)	
C=カラム 4 (商品タイプ)	
C=カラム 5 (数量)	
C=カラム 6 (単価)	
C=カラム 7 (合計)	
C=登録リンク 5 (受注明細データ)	インデックス: 1
C=カラム 1 (受注番号)	代入式: A (PI_受注番号)
C=カラム 2 (受注明細番号)	代入式: C (受注明細番号)
C=カラム 3 (商品番号)	代入式: D (商品番号)
C=カラム 4 (商品タイプ)	代入式: E (商品タイプ)
C=カラム 5 (数量)	代入式: F (数量)
C=カラム 6 (単価)	代入式: G (単価)
C=カラム 7 (合計)	代入式: H (合計)
リンク終了	
L=照会リンク 3 (商品マスタ)	インデックス: 1
C=カラム 1 (商品番号)	位置付小: D (商品番号)、位置付大:D (商品番号)
C=カラム 6 (受注数)	
リンク終了	
レコード後処理	
項目更新 Q (受注数) 式: Q + F (受注数 ＋ 数量)	

このプログラムは、11.4.2 の「受注明細レコードを一時テーブルにコピーするバッチタスク」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。

11.4.5.新しい受注番号を発番するバッチタスク

これは新しく受注レコードを作成するときに、連番を新たに発番するものです。制御テーブルのレコードを読み込み、最終受注番号に1加え、新しい番号としてパラメータで戻します。

タスク特性	
名前:	B_新受注番号発番
タスクタイプ:	B=バッチ
初期モード:	M=修正
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=タスク前の前
データビュー	
メインソース: 1 (制御テーブル)	インデックス: 1 (制御キー)
P=パラメータ PO_受注番号	モデル: 9 (受注番号)
C=カラム 1 (制御キー)	範囲小: 1、範囲大: 1
C=カラム 3 (最終受注番号)	
レコード後処理	
項目更新 C (最終受注番号)	式: C + 1 (最終受注番号 + 1)
項目更新 A (PO_受注番号)	式: C (最終受注番号)

11.4.6.顧客マスタの受注累積額と取引回数を増減するバッチタスク

このバッチタスクは、顧客番号、受注合計額の増減値、加算か減算かのフラグの 3 つのパラメータをとります。
顧客マスタで、指定された顧客番号のレコードをフェッチし、受注合計額を加算/減算し、取引回数を ＋1/－1 します。

タスク特性	
名前:	B_顧客マスタ更新
タスクタイプ:	B=バッチ
初期モード:	M=修正
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=タスク前の前
データビュー	
メインソース: 2 (顧客マスタ)	インデックス: 1 (顧客番号)
P=パラメータ PI_顧客番号	モデル: 1 (顧客番号)
P=パラメータ PI_受注合計額増分	モデル: 17 (合計(10 桁))
P=パラメータ PI_加算減算？	L=論理、書式: 5
C=カラム 1 (顧客番号)	範囲小: A (PI_顧客番号)、範囲大: A (PI_顧客番号)
C=カラム 6 (受注累計額)	
C=カラム 7 (取引回数)	
レコード後処理	
項目更新 E (受注累計額)	式: E + IF (C,B,-B) (受注累計額+IF(PI_加算減算？, PI_受注合計額増分, -PI_受注合計額増分)
項目更新 F (取引回数)	式: F + IF (C,1,-1) (取引回数+IF(PI_加算減算？, 1, -1)

11.5.タスクの修正

以上、単純なバッチタスクを 6 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。

11.5.1.修正の概要

受注明細データの扱い

修正前:
● 子タスクで受注明細データをメインソースとし、直接更新していた。
問題点:
● 受注入力全体をトランザクションで囲まないといけないので、ロックが累積して並列性が著しく損なわれてしまった。
修正後:
● 子タスクでは、受注明細データではなく一時テーブルをメインソースとして使う。 ● 親タスクのレコード前処理で、受注明細データを一時テーブルにコピーする。 ● ユーザの入力中は、明細データへの修正は一時テーブルのレコードに対して行う。 ● 受注入力の確定時(親タスクのレコード後処理)に、一時テーブルの修正をバッチタスクで受注明細データに反映させる。

登録モードの時に、新しい受注番号を発番するには

修正前:
● 親タスクで制御テーブルをリンクし、最終受注番号に＋1して求めていた。
問題点:
● 制御テーブルのレコードがロックされるので、他ユーザが受注入力を行えなくなってしまった。
修正後:
● 受注入力の確定時に、新しい受注番号はバッチタスクで発番するようにする。

顧客マスタの受注累計額と取引回数の更新

修正前:
● 親タスクで顧客レコードをリンクし、レコード後処理で更新していた。
問題点:
● 顧客レコードにロックがかかり、他ユーザが同一の顧客レコードを選択できなかった。
修正後:
● リンクされた顧客レコードのロックを回避するため、リンクコマンドの「アクセス」特性は「R=読込」にする。 ● 受注入力の確定時（レコード後処理）では、項目更新コマンドによって直接更新することはやめる。 ● その代わり、顧客マスタ更新のバッチタスクを呼び出して更新する。

商品マスタの受注個数の更新

修正前:
● 子タスクで商品レコードをリンクし、レコード後処理で更新していた。
問題点:
● 商品レコードがロックされ、他ユーザが同一商品を選択できなかった。
修正後:
● 商品レコードのロックを回避するため、リンクコマンドの「アクセス」特性は「R=読込」とする。 ● レコード後処理での更新はやめる。 ● 受注入力の確定時、一時テーブルの修正を受注明細データに反映させるのと同時に、商品マスタの更新を行う。

以下に、各タスクの各レベルごとに詳しく見ていきます。

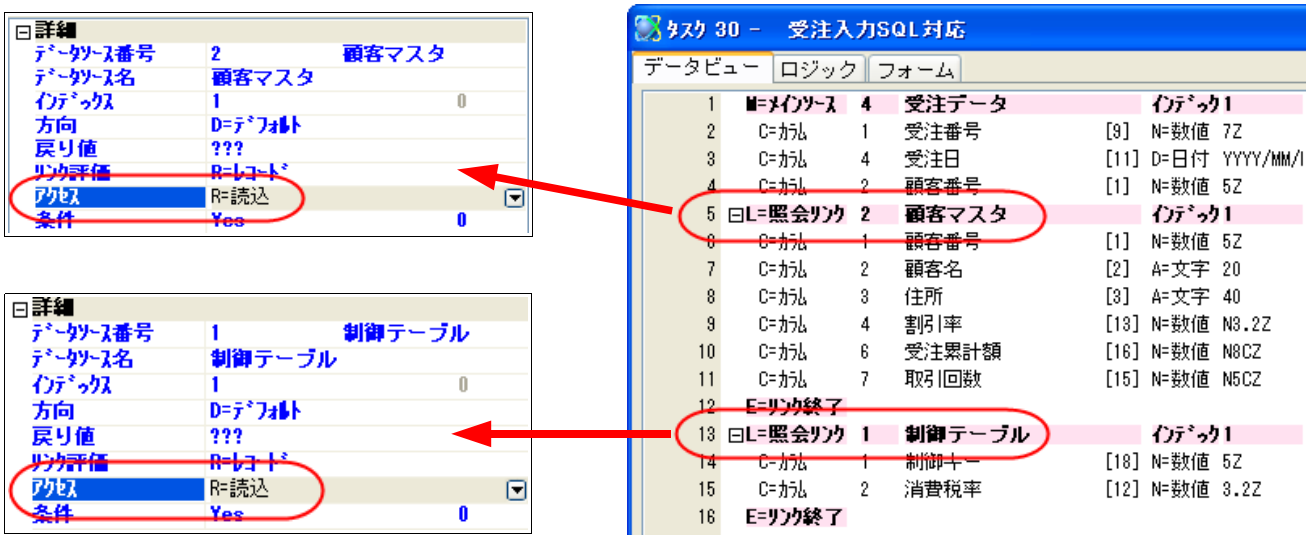
11.5.2.リンクコマンドのアクセス特性

受注入力プログラムでは、次のようなレコードをリンクしています。

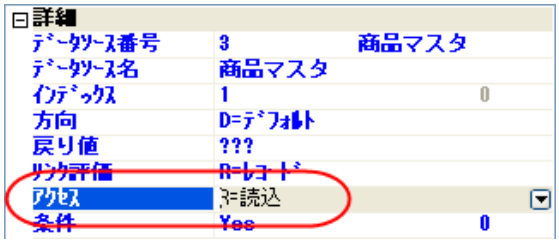
- 親タスク
  - 制御テーブル
  - 顧客マスタ
- 子タスク
  - 商品マスタ

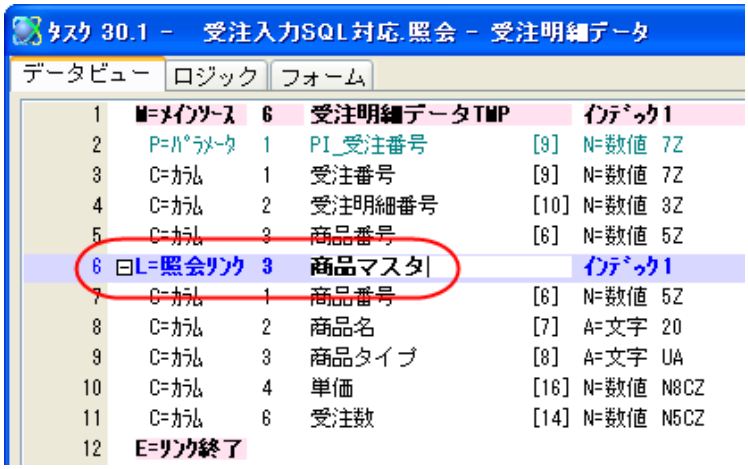
これらのテーブルのレコードロックを回避するために、リンクコマンドの「アクセス」特性は、すべて「R=読込」にします。

親タスクでのリンクコマンドのアクセス特性を R=読込にする



子タスクでのリンクコマンドのアクセス特性を R=読込にする





11.5.3.親タスクのデータビューとレコード前処理

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「11.4.1 一時テーブルのレコードを削除するバッチタスク」(96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、「11.4.2 受注明細レコードを一時テーブルにコピーするバッチタスク」(96 ページ)で説明した、コピープログラム「B_明細コピー (元→TMP)」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存在しないので、コピープログラムは呼び出す必要はありません。従って、Stat(0,' QM' MODE) を条件に設定します。

また、顧客マスターでのレコード後処理で顧客マスタの受注合計額と取引回数を調整するために、顧客番号と受注合計額の初期値を変数に退避しておきます。

- データビューで、退避するための変数を定義しておきます。
- レコード前処理で、初期値を変数に設定しておきます。

さらに、修正前のプログラムでは、新規受注番号を発番するために制御テーブルの「最終受注番号」に1を加えていましたが、修正後はバッチタスク「B_新受注番号発番」で発番するので、この項目は不要になります。そのため、リンク項目からはずします。

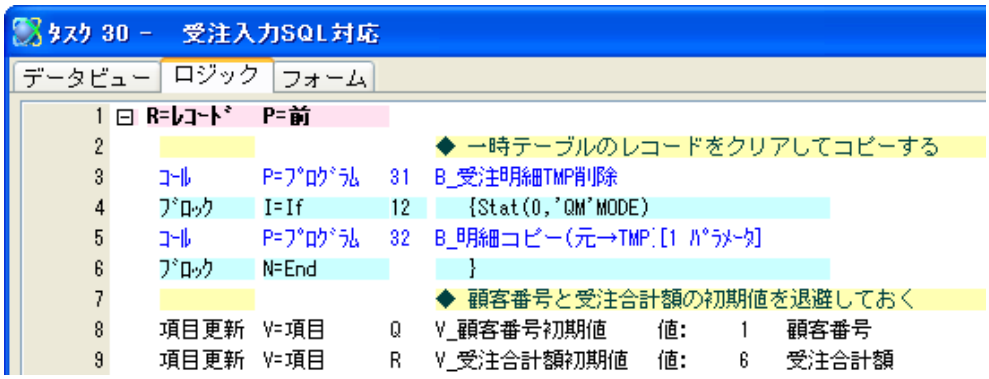
以上まとめると、親タスクのレコード前処理は次のような内容となります。

データビュー
制御テーブルのリンク項目から C= カラム 3 (最終受注番号) の行を削 除する また、最後に以下の 2 行を追加 V=変数 1 (V_顧客番号初期値)　　モデル: 1 (顧客番号) V=変数 2 (V_受注合計額初期値)　モデル: 16 (金額(8 桁))
レコード前処理
コール P=プログラム 31 (B_受注明細 TMP 削除) ブロック If 条件: Stat(0,'QM'MODE) コール プログラム 32 (B_明細コピー(元→TMP)　パラメータ: A (受注番号) ブロック End

項目更新　Q（V_顧客番号初期値）　式：C（顧客番号）
項目更新　R（V_受注合計額初期値）式：P（受注合計額）

親タスクのデータビュー

親タスクのレコード前処理

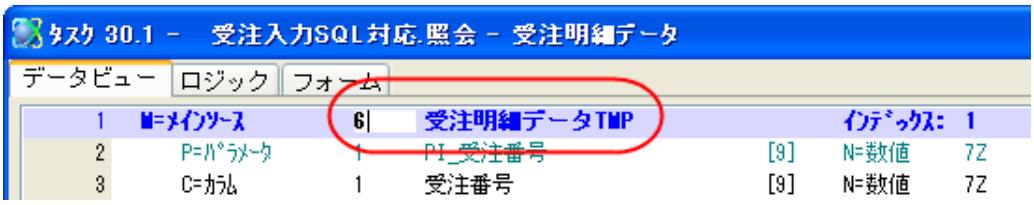


11.5.4.子タスクのメインソースの変更

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。

これには、子タスクのデータビューを開き、メインソース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、その他の部分には手をつける必要はありません。

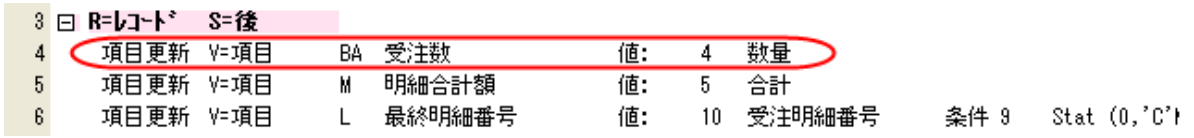
子タスクのメインソースの変更



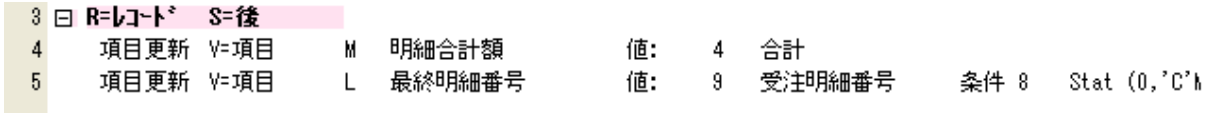
11.5.5.子タスクのレコード後処理

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一度に行う、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。
従って、レコード後処理で商品ファイルの「受注数」を変更していた項目更新コマンドを削除します。同時に、ここで使っている式 4 番 (数量) も不要になるので削除します。

子タスクのレコード後処理 (修正前)



子タスクのレコード後処理(修正後)



11.5.6.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。

受注番号の発番

登録モードのときには新しい受注番号を発番する必要があります。これには、バッチタスク「B_受注番号発番」を呼び出して行います。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。

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。

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います。

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「11.4.2 」(97ページ)で

説明したバッチタスク「B_明細(元) 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「11.4.3 一時テーブルのレコードを受注明細にコピーするバッチタスク」(98ページ)で説明したバッチタスク「B_明細コピー(TMP→元)」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。

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。
この処理では、基本的に、受注レコードの「受注合計額」の差分(最終値と初期値の差)を、顧客レコードの「累積受注額」に加えることになりますが、タスクのモード(登録、修正、削除)での場合分け、さらには、顧客番号が変更になる可能性もあることを考慮すると、一度に行おうとすると意外と複雑になります。

ここでは、簡単化のために、バッチタスク「B_顧客マスタ更新バッチ」を次のように 2 回呼び出すことにより行います。

回数	対象顧客レコードの顧客番号	累積受注額への変更	受注回数への変更	条件
1	初期状態での顧客番号	受注合計額の初期値	－1	修正/削除モード
2	最終状態での顧客番号	受注合計額の最終値	＋1	登録/修正モード

例えば、修正モードのとき、顧客番号は変わらなかったとして、受注合計額が 1000 円から 2000 円になったとします。すると、1 回目の呼び出しで、顧客レコードの累積受注額が － 1000 円 となり、受注回数も －1 となります。次に 2 回目の呼び出しで、累積受注額が +2000 円となり、受注回数が ＋1 となります。最終的に、累積受注額は

－ 1000 + 2000 = ＋ 1000 円

となり、受注回数は

－1 ＋ 1 ＝ ± 0

となります。

その他のパターンでも、正しくデータが更新されることを確認してみてください。

まとめると

以上をまとめると、親タスクのレコード後処理は次のようになります。

親タスクのレコード後処理	
ブロック If　Stat(0,'C'MODE) 　コールプログラム 35 (B_新受注番号発番) ブロック End	パラメータ: A (受注番号)
ブロック If　Stat (0,'MD'MODE) 　コールプログラム 33 (B_明細(元)削除) 　コールプログラム 36 (B_顧客マスタ更新) ブロック End	パラメータ: A (受注番号) パラメータ: 　Q (V_顧客番号初期値)、 　R (V_受注合計額初期値)、 　'FALSE' LOG
ブロック If　Stat (0,'CM'MODE) 　コールプログラム 34 (B_明細コピー(TMP→元))	パラメータ: A (受注番号)

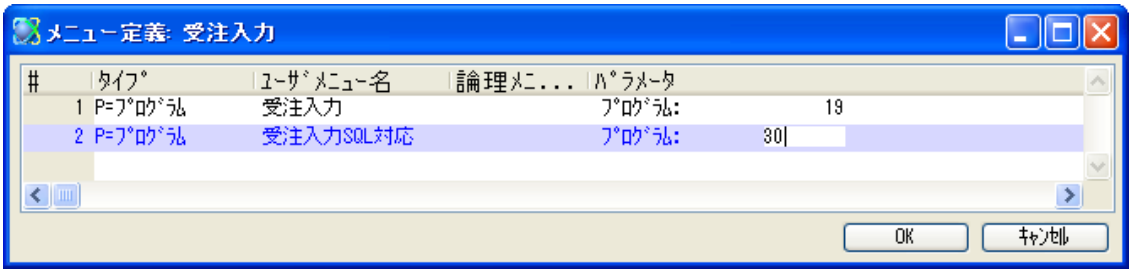
コールプログラム 36 (B_顧客マスタ更新) ブロック End	パラメータ: C (顧客番号), P (受注合計額), 'TRUE' LOG

11.6.実行して確認

プログラムができたので、実行して確認しましょう。

11.6.1.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。

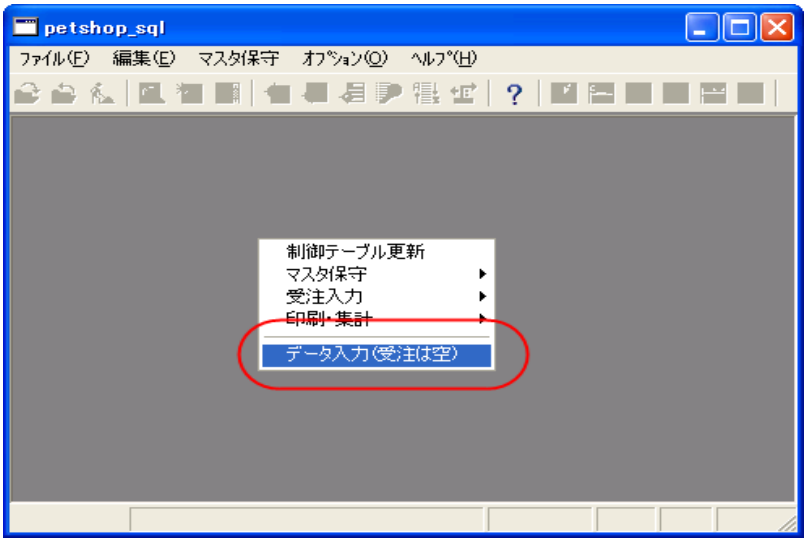


11.6.2.テスト準備

テストを行うにあたって、確認を簡単にするために、データを掃除しておきましょう。

1. Ctrl+F7 キーにより、実行モードになります。
2. ポップアップメニューから、「データ入力(受注は空)」を選択します。

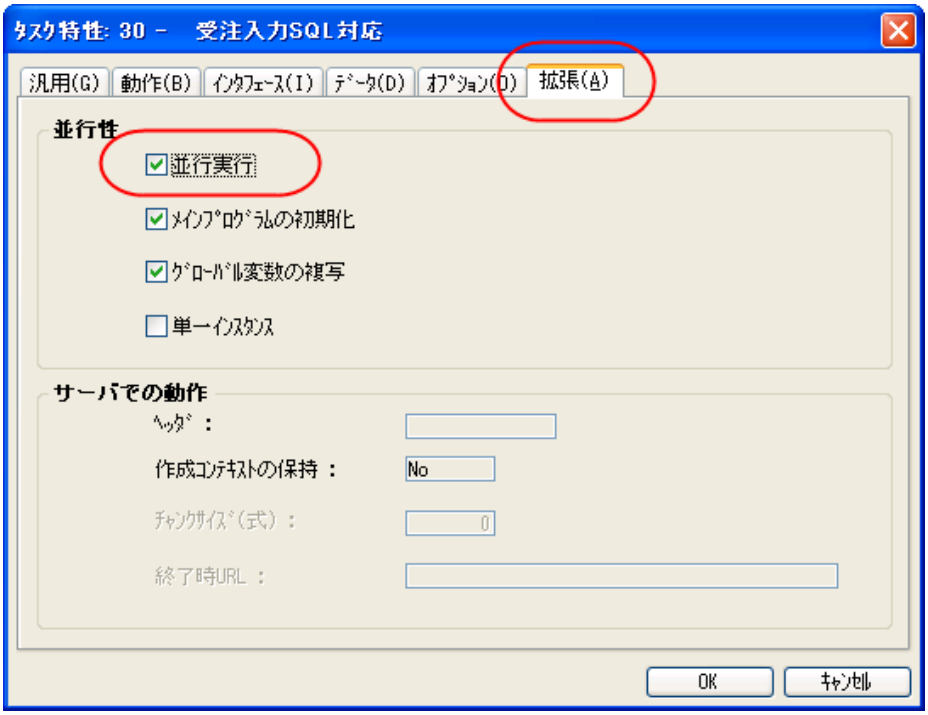
これにより、マスタファイルはすべて初期化され、受注データは空になります。



11.6.3.テストの実施方法

本来は、実行版を二つ動かして動作確認をするのがベストですが、本書では読者が実行版のライセンスを持っていない場合を想定し、前に行ったのと同じく、開発版で並行実行させて確認することにします。

本章で修正した「受注入力 SQL 対応」プログラムは、「受注入力」プログラムをコピーして作ったものなので、「並行実行」の設定がオンになっているはずです。実行に先立って、タスク特性 → 拡張(A) タブの「並列実行」がオンになっていることを確認しておいてください。

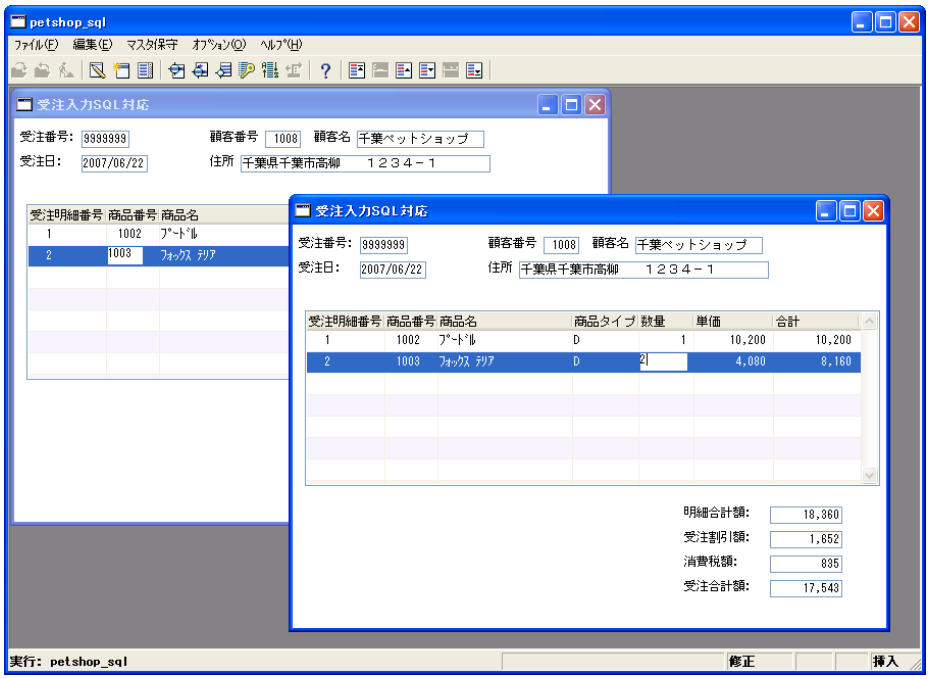


テストは次のように行います。

1. Ctrl+F7 キーにより、実行モードになります。
2. ポップアップメニューから「受注入力 → 受注入力 SQL 対応」を選択します。本章で作成したプログラムが起動します。
3. このプログラムを閉じずに、もう一度ポップアップメニューから「受注入力 → 受注入力 SQL 対応」を選択します。受注入力プログラムがもうひとつ起動します。
   このニつのプログラムを、二人のユーザになぞらえて、ユーザ A、ユーザ B と呼びます。

まずは簡単なテストを行ってみましょう。

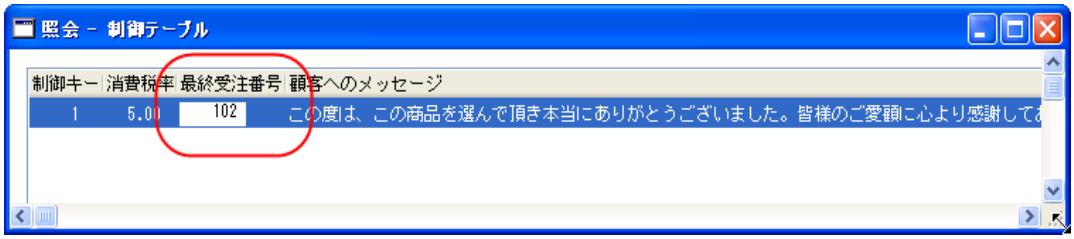
1. ユーザ A で、顧客 #1008 (千葉ペットショップ)を選択し、商品として、#1002 (プードル) 1 匹と、#1003 (フォックステリア) 2 匹とを選択しましょう。
   受注合計額が 17,543 円になっているはずです。
2. この状態で、ユーザ B の方でも、全く同じデータを入力します。
   ここで、どちらでもロック待ちが発生しないことを確認してください。
3. ユーザ A で、ESC キーを押して終了します。
4. 同じく、ユーザ B の方でも、ESC キーを押して終了します。



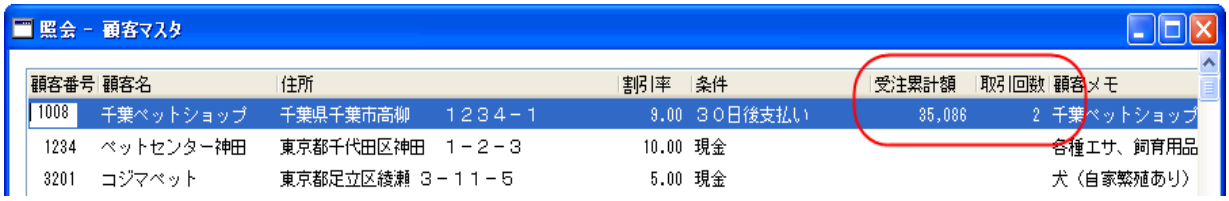
ロックの衝突が起こらないことを確認したら、次にはデータが正しく更新されているかを確認します。

1. 実行画面を閉じて、開発画面に戻ります。
2. データリポジトリを開きます。
3. 次の値を確認してください。

- 制御テーブル：最終受注番号は 102 になっている。(ユーザ A とユーザ B とから、それぞれ 1 回づつ受注を受けたので)。

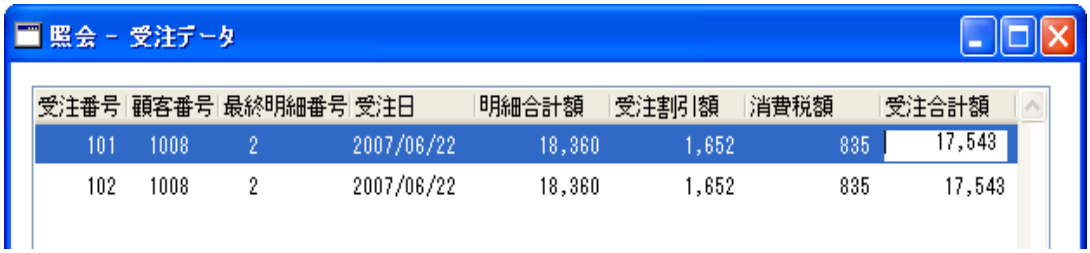


- 顧客マスタ：#1008 (千葉ペットショップ) で、受注累積額が 35,086 円、取引回数が 2 になっている。(ユーザ A と、ユーザ B とから、それぞれ 17,543 円の受注を受けたので)。



- 商品マスタ：#1002 (プードル) の受注数が2、#1003 (フォックステリア)の受注数が4となっている。

- 受注データ：受注番号 #101 と #102 とが登録されていて、それぞれ、受注合計額が 17,543 円となっている。



- 受注明細データ：受注番号 #101 と #102 とに、それぞれ明細行が二つづつ作成されている。それぞれは商品番号 #1002 (プードル) 1 匹と、商品番号 #1003 (フォックステリア)2 匹である。

以上が合っていれば、データの更新も正しく行われたことが確認できました。

11.6.4.テストパターン

テストはこれだけではありません。もっといろんなテストパターンを使って、漏れがないことを確認しましょう。

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。
      - 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。

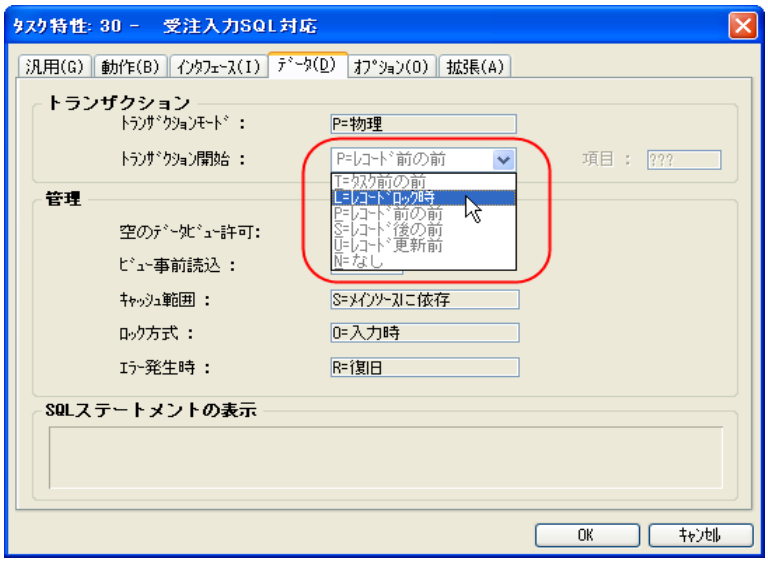
12.トランザクション設定の注意事項

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「データ(D)」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=タスク前の前	タスク前処理の前	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。

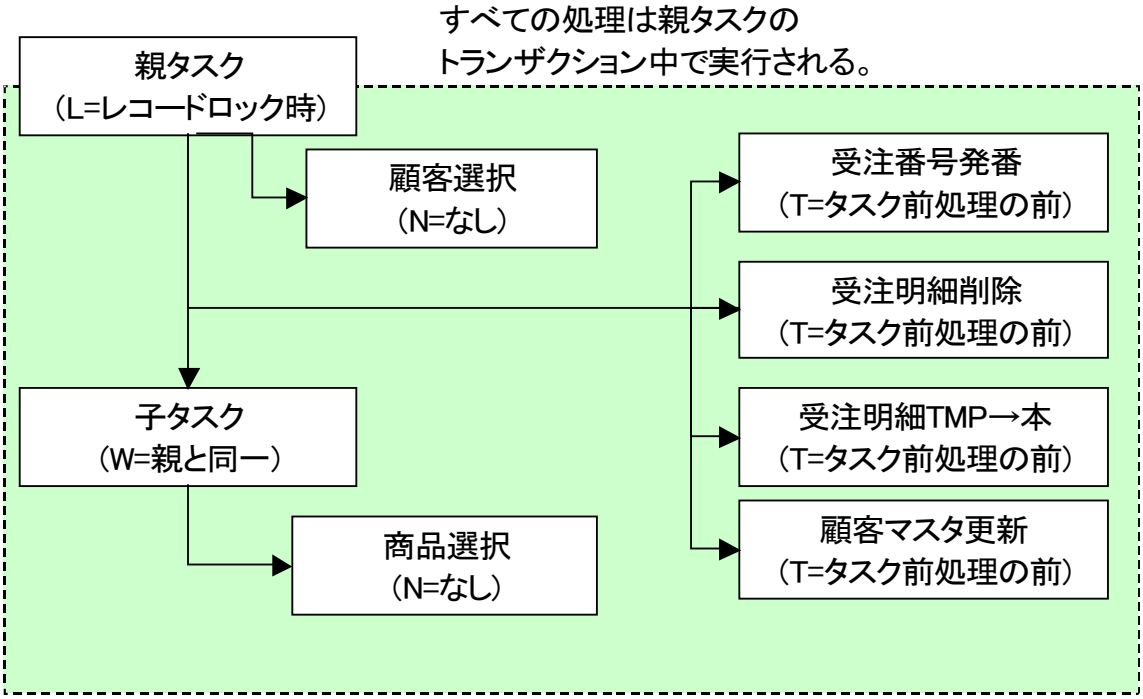
12.2.物理トランザクションのネストはできない

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 20 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 20 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=親と同一」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一度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」(77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、MSSQL 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 MSSQL Server に格納してあるも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？

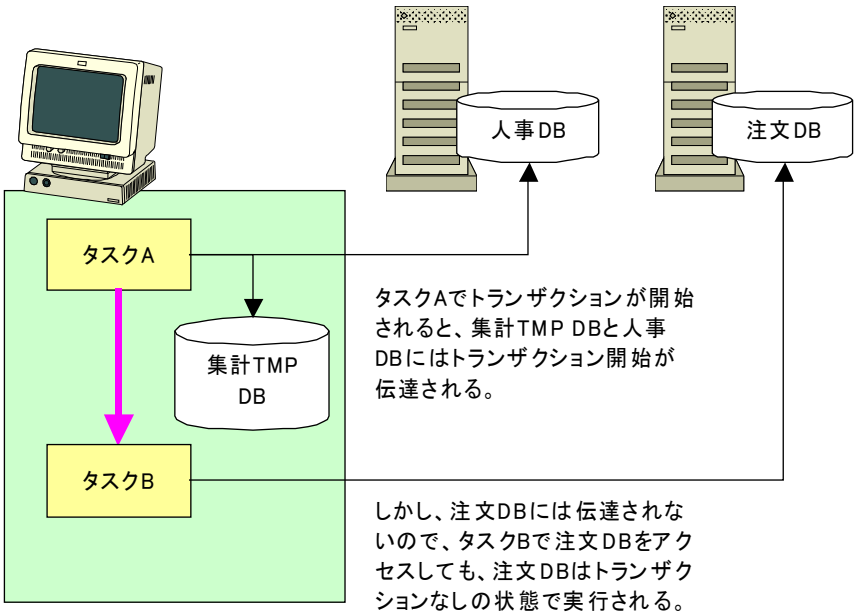
12.3.1.データビューに登録されているテーブルに属するデータベース

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「データビュー」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(あるいはその上位のタスク)の「データビュー」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 21 を見てください。

図 21 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 MSSQL 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。MSSQL Server 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のデータビューに「D=宣言」コマンドで追加登録しておくことです。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のデータビューに「D=宣言」行を追加し、受注および受注明細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、

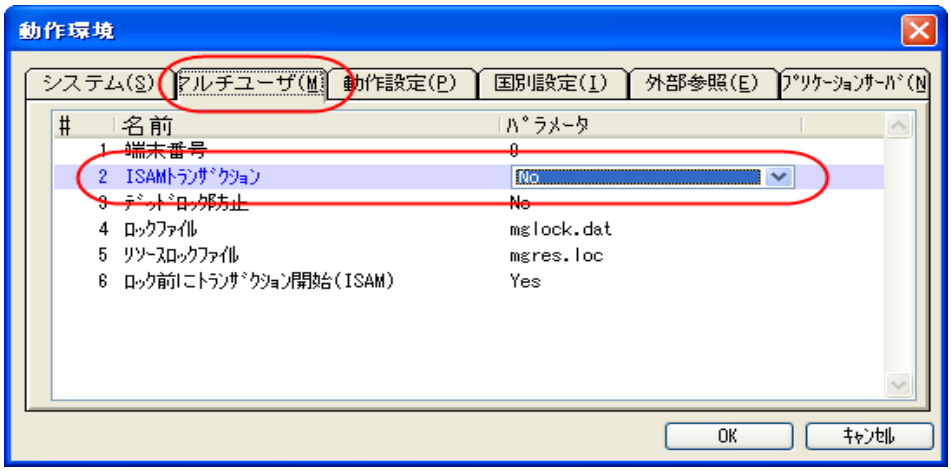
- トランザクションが開始されるタスクを正しく把握する。
- そのタスクのデータビューに、トランザクション中に利用するテーブルをすべて登録しておく。(必要ならば D=宣言 コマンドを用いて、後に利用する意図を宣言しておく。)

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 22)

図 22 ISAM トランザクションの設定



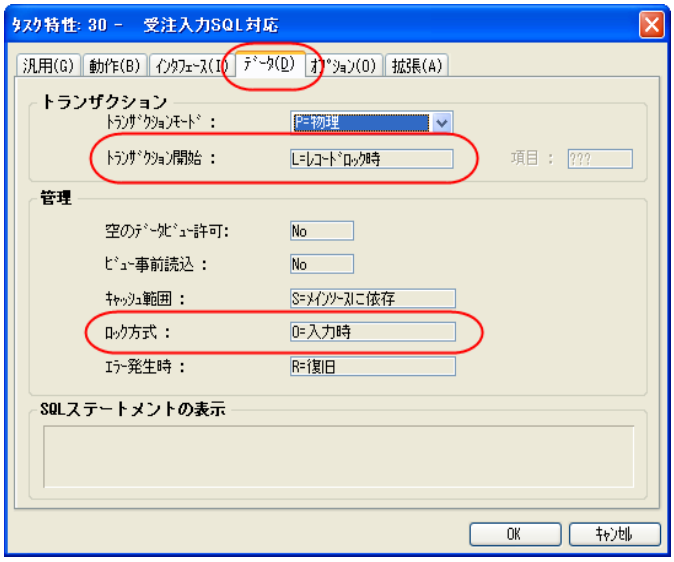
このパラメータが No (デフォルト)の場合には、ISAM データベースに対してトランザクションを利用しません。Yes の場合には、トランザクション開始が伝達されます。
なお、SQL データベースの場合には、トランザクションの利用が必須なので、これを制御するパラメータはありません。

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「データ(D)」タブで、ここで、

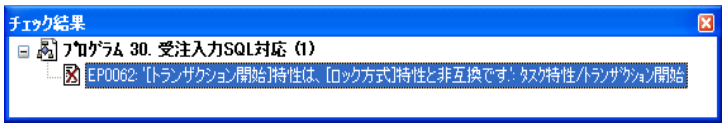
- トランザクション開始 (トランザクション開始のタイミングを指定)
- ロック方式 (ロックのタイミングを指定)

をそれぞれ指定できます。

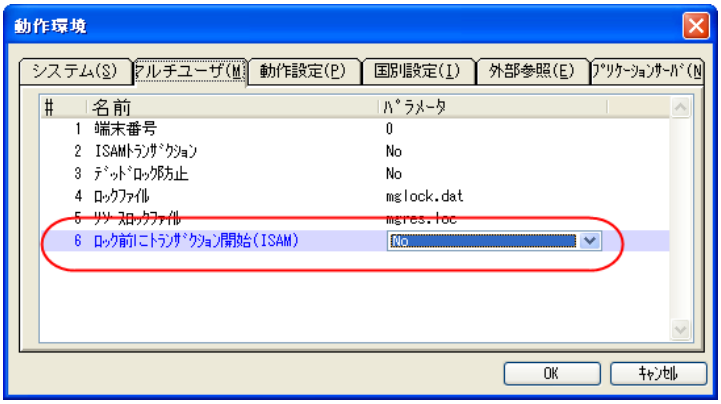


この二つのパラメータは独立して設定できるので、設定によっては、「ロックはトランザクションの中で」という規則に反する設定をすることもできます。例えば、「トランザクション開始」が「レコード後処理の前」なのに、ロック方式が「入力時」だと、ロックの開始がトランザクションの開始に先立ってしまい、矛盾します。

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 22 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・・・

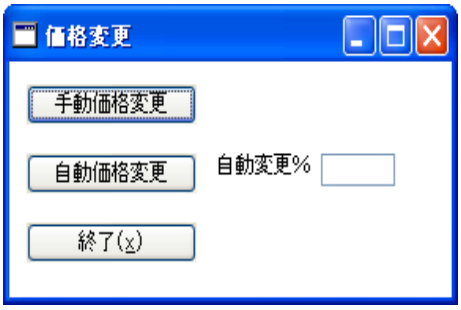
以上のことをまとめると、Magic がトランザクションを開始する場合に DBMS にもトランザクション開始が伝達されるかどうかは、下表のようにまとめられます。

DBMS のタイプ	DBMS のトランザクションサポート	ISAM トランザクション設定	トランザクションが DBMS に伝達されるか？
SQL	サポートする	(いずれでも)	○
ISAM	サポートする	Yes	○
		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？

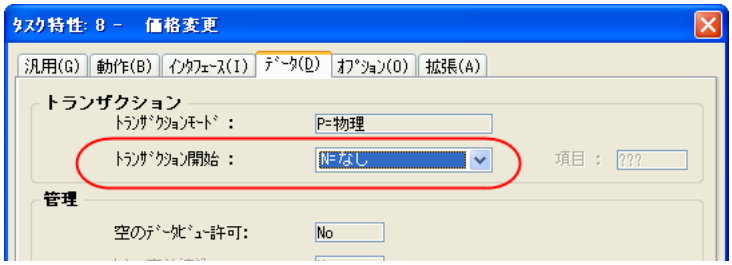
例えば、ペットショップアプリケーションに、「価格変更」のタスクがありました。右図がその初期画面です。このタスクでは、ボタンが三つメニューとして定義されており、自動価格変更の場合には変更のパーセントを指定する数値型の変数項目があります。このタスクでは一切テーブルは使いません。



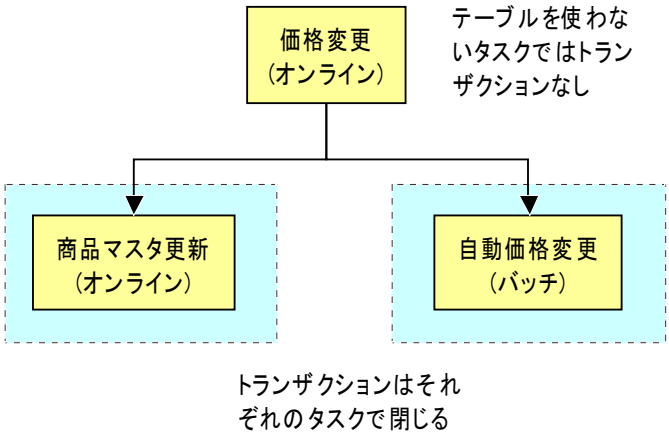
このような、メニューのような役目で使われているタスクでは、一般にはトランザクションを開始しないように設定します。トランザクションを開始しないようにするには、タスク特性の「データ(D)」タブでの設定で、

- 「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、このタスクのデータビュー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、MSSQL 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、MSSQL 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 MSSQL 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 eDeveloper V10 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：SQL への移行時の注意事項

ここでは参考情報として、Pervasive 対応のアプリケーションから、MSSQL Server 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.データリポジトリに関する変更

14.1.1.デフォルト値の違い

データ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、データ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.データソース名

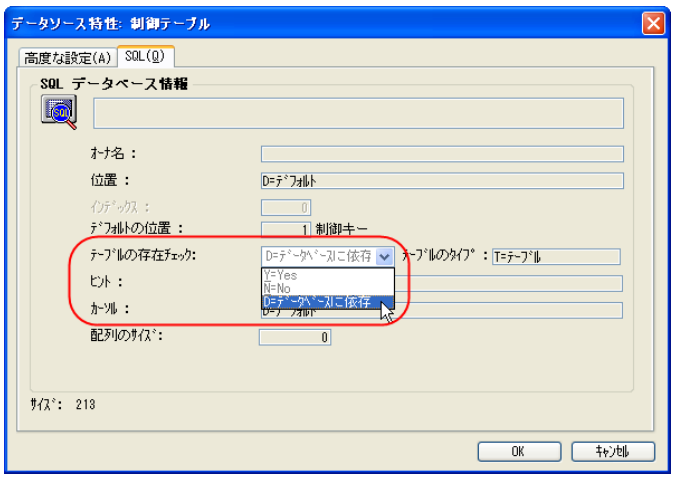
Pervasive.SQL の場合、「データソース名」欄には Pervasive.SQL のファイル名を、Windows のファイル名規約に従って指定します。
MSSQL Server の場合には、「データソース名」欄にはテーブル名を、MSSQL Server のテーブル名の規約に沿って命名する必要があります。

- 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、例えば拡張子を示す .(ドット) 文字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」などといった拡張子は、すべて削除します。
- 先頭に @, @@, #, ## をつけると、特別な性質をもつテーブルとなりますので、注意してください。
- 更に MSSQL Server で決められた予約語(CHECK, ORDER など)も避ける必要があります。

命名規約の詳細は、MSSQL Server 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

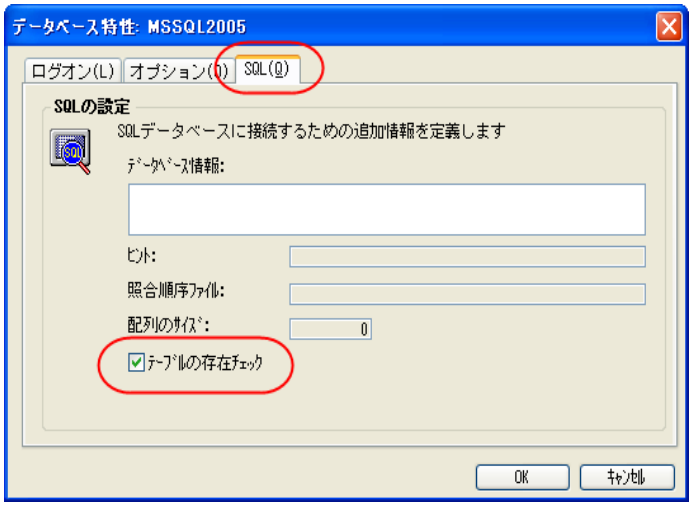
Magic では、プログラムでテーブルをオープンする時にテーブルが存在しなかった場合、自動的にテーブルを作成する機能があります。
MSSQL Server データベースにアクセスするときには、これらの処理のために SELECT コマンドと CREATE コマンドを発行しますが、アプリケーション環境でテーブルが存在することがわかっている場合は、これらの処理は余計なオーバーヘッドとなり、パフォーマンスを劣化させる原因になります。
予め存在することがわかっているテーブルに対しては、テーブル特性のテーブル存在チェックパラメータを No にすると、この存在チェックの処理を省略するので、パフォーマンスを改善できます。



このパラメータは、

- Yes（存在チェックを毎回行う）
- No（存在チェックを行わない）
- D=データベースに依存

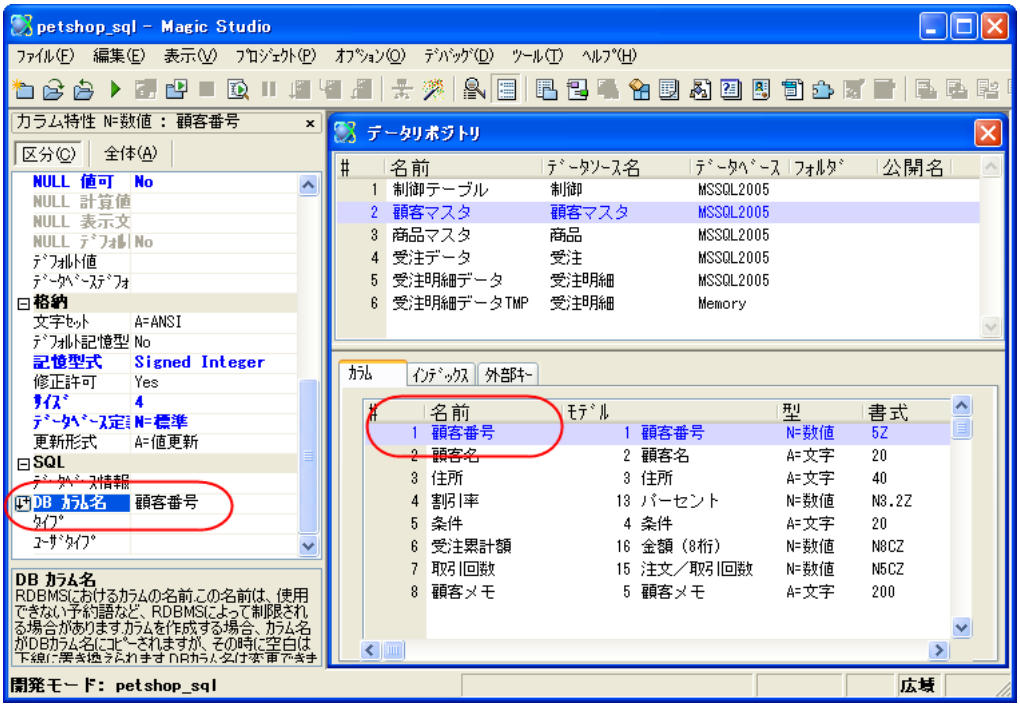
で、デフォルト値は、「D=データベースに依存」です。この設定では、データベーステーブルの「SQL(Q)」タブでの設定を見て判断します。



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、MSSQL Server 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、MSSQL Server のテーブルには存在します。データリポジトリのカラムの「名前」欄の名前は Magic の中だけで有効な名前であり、MSSQL Server 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。



この DB カラム名は、MSSQL Server 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、MSSQL Server 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 カラム名としてコピーされます。

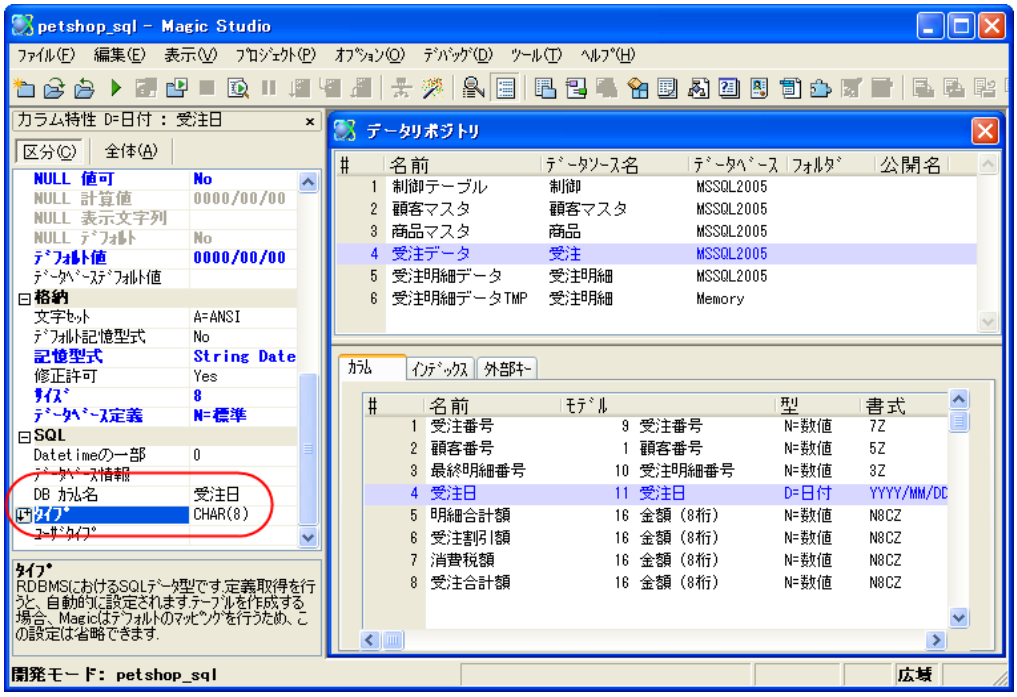
1. 新規にテーブルを定義したとき
2. ISAM 系のデータベース(Pervasive.SQL や Memory)から MSSQL Server に変更したとき。

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 MSSQL Server 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 MSSQL Server のカラムタイプを指定します。

下図は、日付型のデータを CHAR(8) で格納した例です。「受注日」のカラム特性「タイプ」に、「CHAR(8)」と定義されています。



ここでの設定は、Magic がテーブルを作成する時にだけ有効です。すなわち、テーブルが存在しない場合、Magic の存在チェック機能で存在しないことが検出されたときに、CREATE TABLE が発行されますが、この CREATE TABLE 文に「タイプ」特性の設定が反映されます。
すでに存在しているテーブルについて、データリポジトリで「タイプ」特性を変更すると、データリポジトリのファイル自動再編成機能が起動されます。

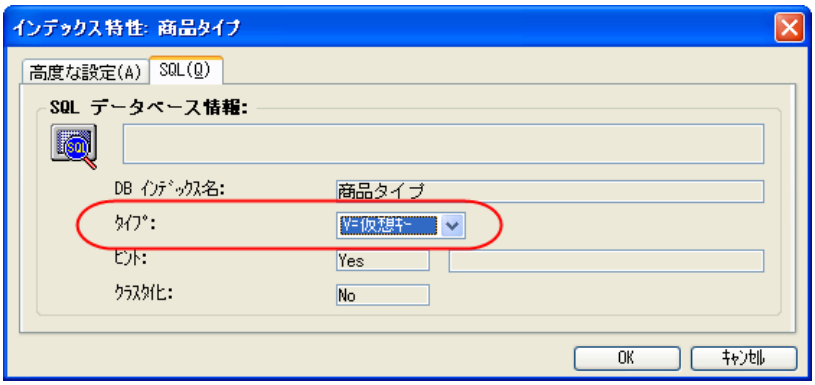
14.1.6.日付型カラム

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 MSSQL Server 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、MSSQL Server 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」(130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



インデックスは、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、MSSQL Server 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 MSSQL Server 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 MSSQL Server ツール等で削除します。

仮想インデックス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、MSSQL Server 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、MSSQL Server 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

MSSQL Server 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 MSSQL Server に変更したとき。

したがってインデックスの名前が MSSQL Server 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。

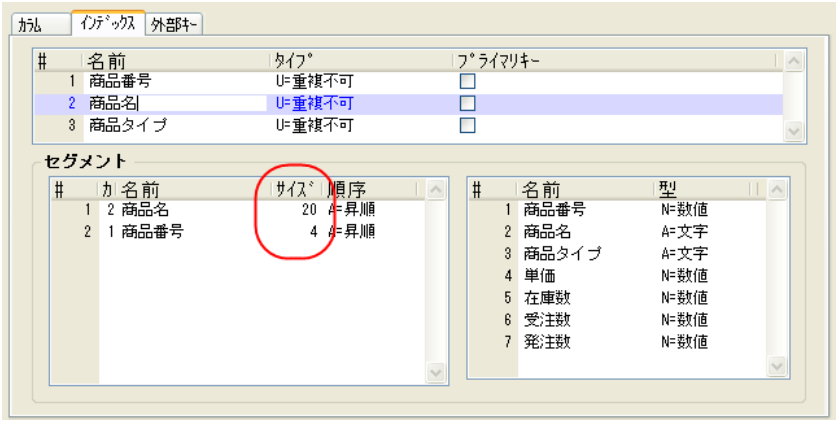
14.1.9.重複不可インデックスの定義

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、MSSQL Server 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。

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 INTEGER IDENTITY を付加して、MSSQL Server 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データリポジトリで指定することができます。しかし、MSSQL Server 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、MSSQL Server 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、MSSQL Server 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。

仮想インデックスでは重複エラーが発生しない

仮想インデックスは、アプリケーションの中だけの定義であり、MSSQL Server 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 MSSQL Server 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、実インデックス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、MSSQL Server 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、MSSQL Server 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 MSSQL Server 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。

Pervasive.SQL の重複データ入力時の動作は、データベース特性の「インデックスチェック」の設定により変わります。Pervasive.SQL のデフォルトの設定では「インデックスチェック」がオンとなっているので、カラム移動時にチェックが入りますが、「インデックスチェック」をオフにすると、MSSQL Server のデフォルトの動作と同じく、レコード書込み時になってからエラーとなります。

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 MSSQL Server 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。テーブルが存在しない可能性がある場合には、「テーブルの存在チェック」を Yes にしてください。
- MSSQL Server サーバの権限により、サーバログオンユーザに CREATE TABLE 権限がない場合。CREATE TABLE 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データビューに定義されていないとき

テーブルに定義されているカラムの中に、データビューで定義されていないものがあるとき、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、データビューで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、MSSQL Server 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、すべてのカラムをデータビューで定義する。
- [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。

データソース名を式で設定しているとき

テーブル名は MSSQL Server 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ができますが、MSSQL Server の場合は、MSSQL Server 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。
- 親のプログラムのデータビューで参照されている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。データビューで参照されていないと、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。

- File 関数(FileDelete 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、MSSQL Server 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、常に空白文字(文字型の場合)や 0 (数値型の場合)を格納しますが、MSSQL Server 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データビューで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。

- 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

MSSQL Server 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、MSSQL Server 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 V10

Magic eDeveloper V10
チュートリアル SQL編

第1版　2007年7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/