最優秀賞

RPGを知らなくてもここまでできる
――ユーザーサイドからのアプリケーション開発

石井 裕昭 様
豊鋼材工業株式会社
製造総括部　部長代行

豊鋼材工業株式会社
http://www.yutaka-steel.co.jp/

「鋼材のことならあらゆるニーズに応える企業」をモットーに、広島から沖縄までをカバーし、鋼材およびその他金属の加工、販売を行っている。伊藤忠丸紅鉄鋼・新日本製鐵系の会社である。

ソリューション導入の経緯と概要

豊鋼材工業株式会社の主体である溶断事業では、従来、生産に関するさまざまな情報が、紙の帳票による事後の実績入力や、あるいは担当者の経験等に基づく繁閑の判断と工程管理といったことに依存していた。したがって営業部門、経営者、さらに製造部門内からも、工場の操業状況や受注製品の進捗・計画状況はブラックボックス化して、お客様対応のレスポンスやさまざまな判断に支障をきたしていた。

このような状況を打開するため、平成17年より工場内の無線LAN強化をはじめ、各設備へのWindows CE端末、無線ハンディターミナル、およびラベルプリンタの配備と帳票へのバーコード出力等の基盤整備を開始した。【図1-1】

Delphi/400は、この基盤整備でリアルタイムに収集される情報を定量的かつ直感的に利用する手段として導入された。本アプリケーションは、生産・出荷計画、進捗、履歴、山積み等の情報をさまざまな角度から提供する参照系と、生産指示や計画作成等の更新系の処理を備えている。

開発の独創性・創意工夫

RPGを使用できるシステム室要員数が少なく、本アプリケーションで実現したい数多くの機能について、データ抽出の条件・項目等の仕様が事前に確定しにくいこともあり、開発はユーザーに近い立場の者が主担当で行った。

そのため、データ抽出はBDEのQueryで、SQL文を動的に生成して抽出する構成とした。だが、初めてのDelphi/400での開発ということもあって、当初は実用に堪えないものであった。

いかに一度に表示できる情報量が多くても、表示の速度が実用レベルになければ、また操作性等で5250での参照に対して大きな優位性がなければ、ユーザーには使われない。

そこで、次の対応を順次実行し、実用レベルに到達させた。

① Query → DatasetProvider → ClientDatasetの組み合わせで、抽出したデータに対して、ローカルでのフィルタリング、並べ替えのボタン、DBGridをマウス操作のみで実現するという機能を実装
② SQLのパフォーマンスについては、i-SeriesナビゲータのVisual Explainにより、推奨索引確認および索引自動生成での改善
③同じくVisual Explainにより、結合キーの型の妥当性確認と調整
④ SQL実行時のSQL文抽出機能とSQL動作検証用アプリにより、デバッグとチューニング
⑤関連データの一括抽出とマスターデータ参照による連携機能で、同時参照（マスター⇔詳細⇔最詳細）【図2-1】
⑥ SQL実行待機の体感時間の軽減（ProgressBar、AVI動画再生等）
⑦ QueryのOnCalcFieldsイベントでの処理により、SQLの速度低下の抑制

図1-1　生産管理システム再構築イメージ

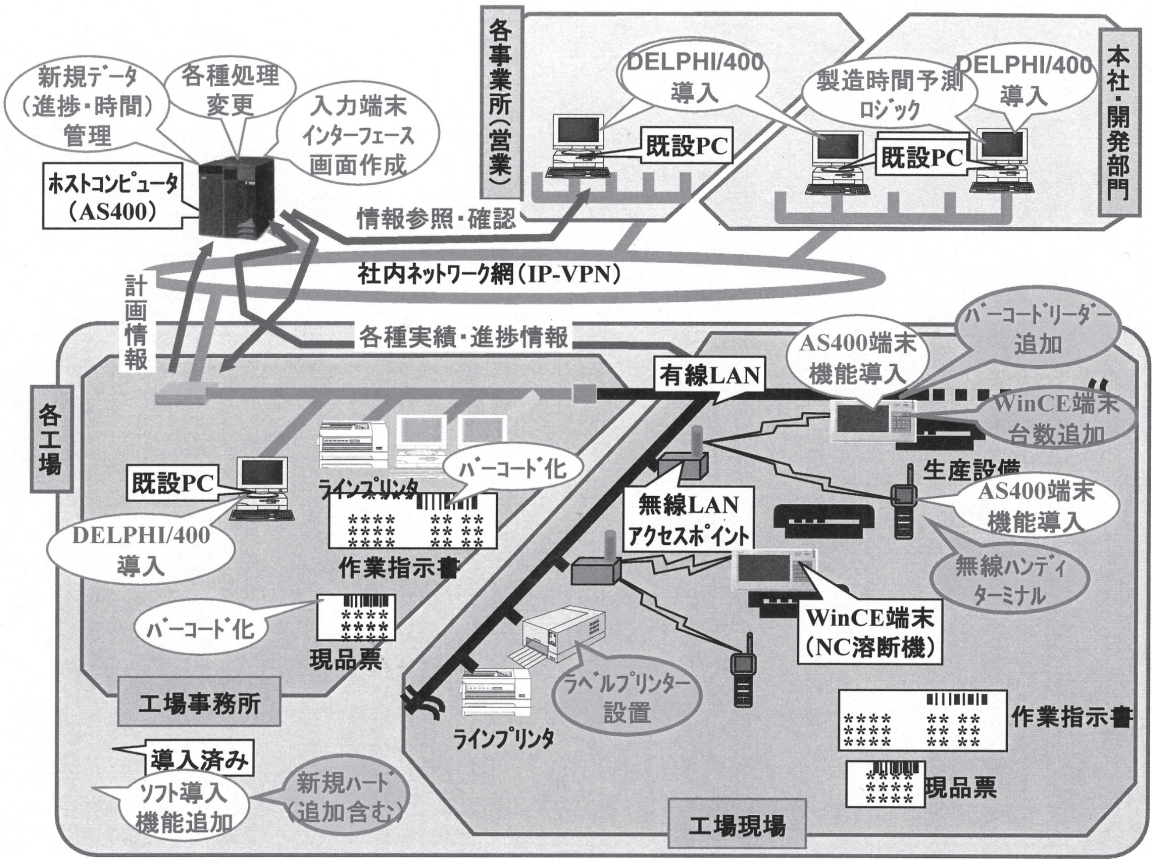
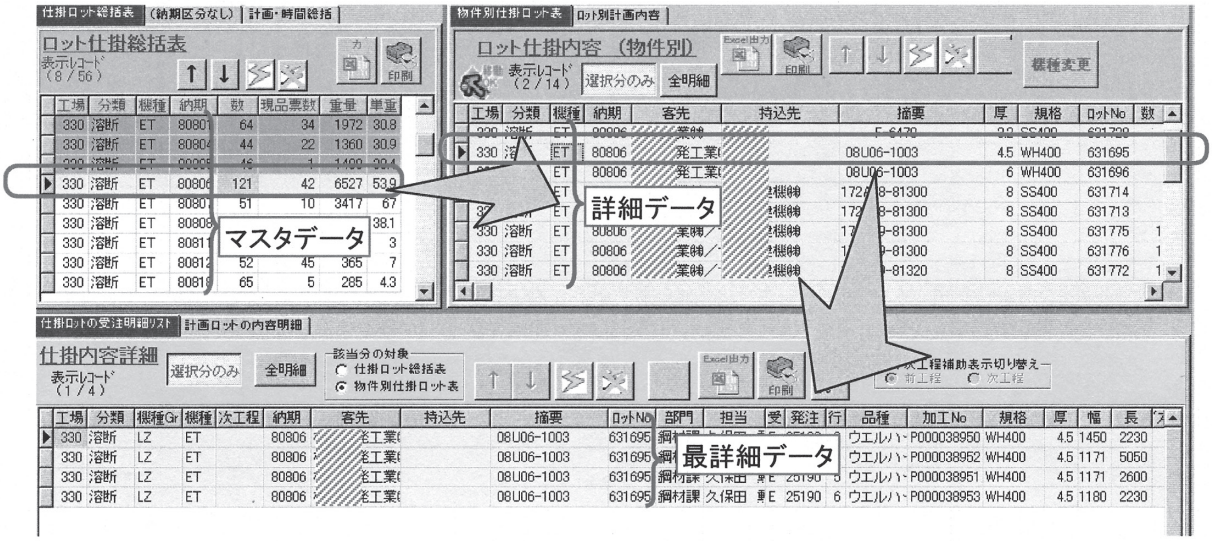


図2-1　マスタ⇔詳細⇔再詳細のデータ連携例



⑧動的に内容を変化させるcomboBox用のデータベースの、ローカル化と定期更新
⑨一定の未操作時間経過で、自動的に表示情報を更新
⑩Gridに表示される不要項目の非表示化､並び順変更の可能化｡その条件は､次回起動時にも保持（Iniファイルに配列情報を書き込み）

実用レベルに向けた改善ポイント

アプリケーション開発の独創性・創意工夫について詳細を述べる。

①毎回、抽出条件を設定しSQL処理を行うよりも、ある程度の範囲で抽出し、ローカルでキャッシュされたデータに対してフィルター、並べ替え、検索等を瞬時に実行可能にすることにした。これにより、思考の中断も少なくできる。そのために、ClientDatasetの使用を基本とした。

抽出した結果の並べ替えは、任意の項目の組み合わせで可能であり、優先順が明示できるようにしている。また、フィルター機能についても、複数条件の組み合わせが可能で、条件とフィルター中であることがわかるようにした。【図2-2】

なお、DBGridの各列のタイトル、巾等については、開発のメンテナンス性を考慮した。Queryの静的項目のみの設定で、その内容が自動的にClientDataSetに引き継がれるように、OnAfterOpenイベントに汎用処理を記述した。【図2-3】【図2-4】

またDBGrid、Query、ClientDataSet、フィルター用ボタン等、関連する一連のコンポーネントを配列化した｡とともに、Tagプロパティ等の活用により、呼び出し元の識別を汎用化し、コードの記述を抑えた。【図2-5】【図2-6】

②③IBMのIBM i講座「SQLパフォーマンス･チューニングの基礎」を受講し、IBM iに同梱されているi-Seriesナビゲータを開発用PCに導入した。【図2-7】

これにより､組み込むSQLのパフォーマンスについて検証を行うことができ、適正な索引の確認と索引の生成が一連の操作で可能となった。結合キーの型不一致のチェック等も可能であり、その際にはiSeries DB2のSQL解説書を参考に型の変換を行った。【図2-8-1】【図2-8-2】

i-Seriesナビゲータでは、GUIの画面で既存ライブラリに新規にテーブル、ビュー（論理ファイル）、索引等を簡単に作ることも可能であるため、情報システム室に依存せず、新たな管理項目を含むデータベースを作れるため有効である。【図2-9】【図2-10】

④開発時のテストにおいて、データが正しく抽出されない、またはエラーが発生する場合、実アプリケーションのコード修正を繰り返すとコンパイルの時間ロスが大きい。また、どのようなSQL文が実行されているかも、コードからは読み取りにくい。そのため、該当SQLの配列番号指定で実行されたSQL文を、テキストで読み出せる仕組みを作った。

また、任意のSQL文での抽出結果を確認するテスト専用のアプリケーションを、別途準備し、SQLのテストが簡単にできるようにした。【図2-11】【図2-12】【図2-13】

⑥1回の操作で複数のQueryをOPENする際には、その変わり目でProgress Barの Positionを変更し、SQL処理が進んでいることをユーザーに示す、ということを可能にした。

一方、単体の場合、個々の処理時間が長いと本当に処理が進んでいるか不安になり、体感時間も長くなる。そのため、動画を指定のPanel上で再生して、ユーザーの手待ち感の軽減を試みた。【図2-14】【図2-15】

⑦SQL文には文字数の制限もあり、CASE文等を多用するとレスポンスが低下するケースも多い。また、複雑な処理の記述が困難となるため、内容に応じてQueryのOnCalcFieldsイベントを用いた。

⑧抽出条件の設定では、ユーザーの好みに応じてコードのEdit入力､またはコードに対応する文字のComboBoxからの選択を行えるようにしている。

ComboBox使用の場合、選択肢を少なくし操作性を高めるために、例えば営業担当者のComboBoxの内容は、部門のComboBoxの選択結果に応じて動的に変更される。【図2-16】【図2-17】【図2-18】

その際、IBM iの情報を参照してComboBoxを設定すると、ネットワーク、SQLの待ち時間が多少発生する。そこで、ComboBoxで多用される社員マスター、客先マスター等の更新頻度が少ないデータベースは、定期的にユーザーのPCにParadox形式でダウンロードされる仕組みを作り、ComboBox操作時の待ちをなくした。【図2-19】【図2-20】【図2-21】

⑨同一の抽出条件であれば、設定に対応して起動中に一定時間アプリケーション操作がない場合は、ClientDataSetのRefreshメソッドを実行するようにした。

これは、ApplicationEventsのOnIdle、OnMessageイベントと、TimerのOnTimerイベントで実装できる。【図2-22】【図2-23】

⑩各機能で抽出される項目は、ある程度汎用性を持たせるために多くなり、ユーザーによって不要だったり、または優先度が異なる。横スクロールの手間、同時参照性により、並べ替え、一部項目の非表示化等のカスタマイズのニーズがあった。【図2-24】【図2-25】【図2-26】

教訓と今後の予定

参照系のアプリケーションについては、RPG等の知識がなくても十分構築できる。ただし､SQLの構文､JOINキーの型の整合性等により、パフォーマンスが大きく変化する。十分な検証が必要である。

関連する各種情報を同時参照可能とすることで、5250アプリケーション以上のTotalパフォーマンスが得られる。今後は､抽出に時間を要する機能について、パフォーマンスを改善する手段をさらに追求したい。

図2-2　フィルタ、並べ替えの状況

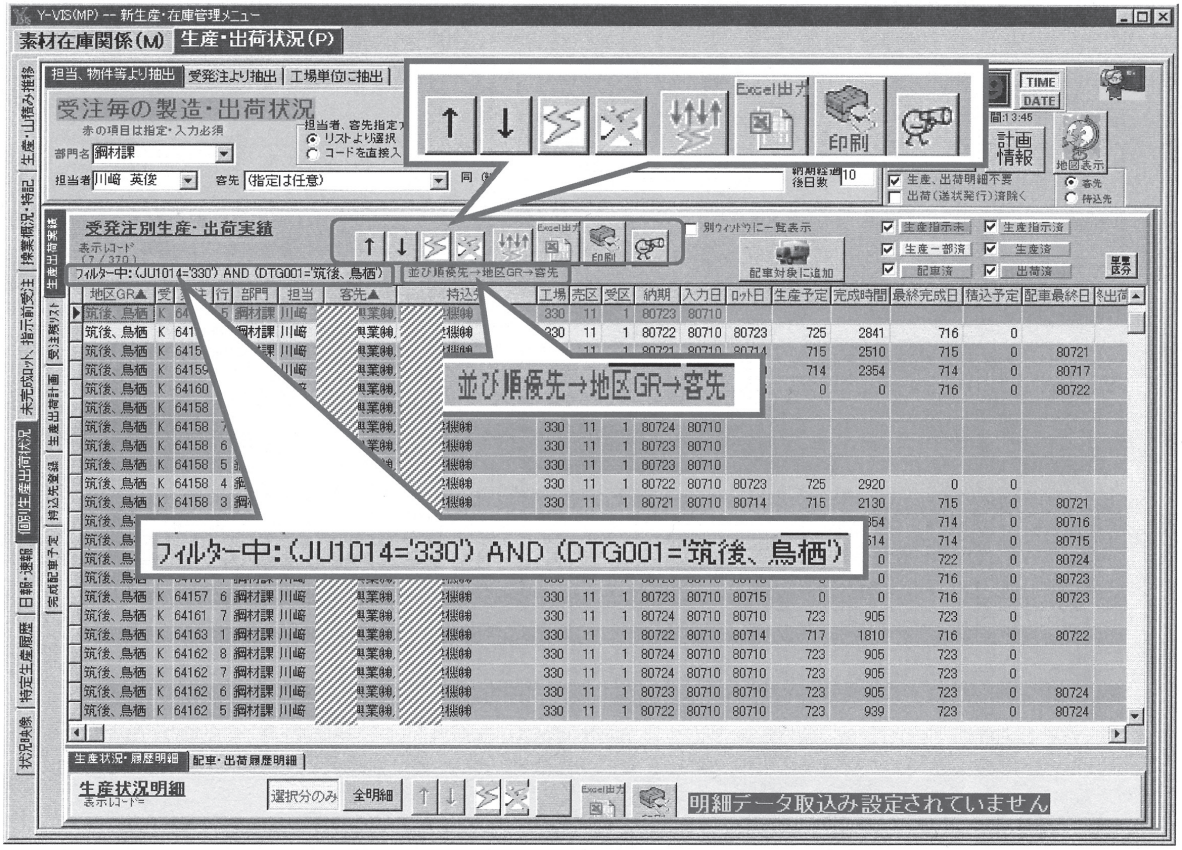
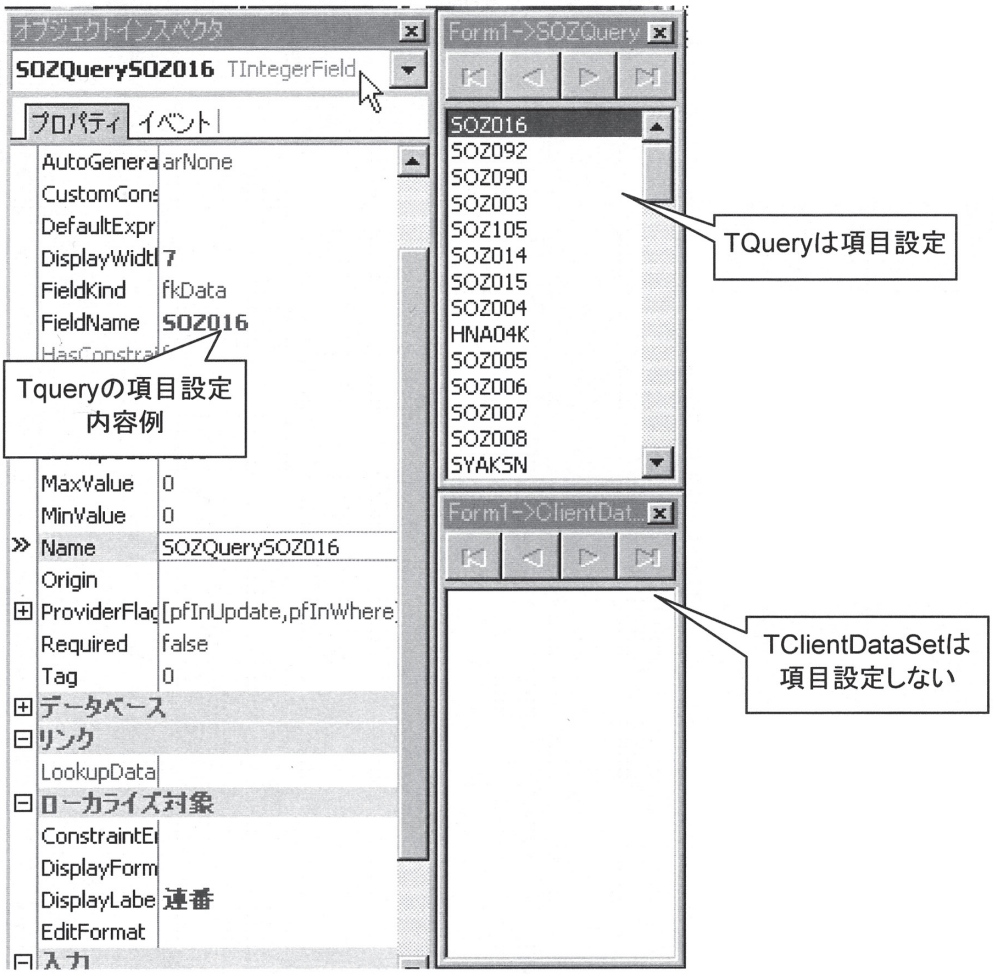


図2-3　Tqueryへの静的項目設定例



エンドユーザー評価

●従来からの使い慣れの点もあり、単純な照会については、レスポンスの点から5250画面が依然として使用される場合もある。だが、Delphi/400のアプリケーションでは、情報がより定量的に確認でき、かつ一度に多くの関連情報が参照できる。この部分に対する評価が高い。

●多目的なアプリケーションのため、使用者は役員～業務スタッフまでと幅広い。

M

図2-4　Tqueryへの静的項目設定内容をTClientDataSetに反映する処理例

```
n:=TClientDataSet(DataSet).Tag;

for i:=0 to mainQUERY[n].FieldCount-1 do
begin
  // TQueryの列名と同じ列がTClientDataSetに存在するか？
  if Assigned(Form1.mainCDS[n].FieldByName(Form1.mainQUERY[n].Fields[i].FieldName)) then
  begin
    // TClientDataSetの列に対してTQueryのDisplayLabelをコピー
    Form1.mainCDS[n].FieldByName(Form1.mainQUERY[n].Fields[i].FieldName).DisplayLabel :=Form1.mainQUERY[n].Fields[i].DisplayLabel;
    Form1.mainCDS[n].FieldByName(Form1.mainQUERY[n].Fields[i].FieldName).DisplayWidth :=Form1.mainQUERY[n].Fields[i].DisplayWidth;
    if (mainQUERY[n].Fields[i].DataType=ftFloat) Or (mainQUERY[n].Fields[i].DataType=ftInteger) then
     (Form1.mainCDS[n].FieldByName(Form1.mainQUERY[n].Fields[i].FieldName)As TNumericField).DisplayFormat
              :=(Form1.mainQUERY[n].Fields[i] As TNumericField).DisplayFormat;
    Form1.mainCDS[n].FieldByName(Form1.mainQUERY[n].Fields[i].FieldName).Visible :=Form1.mainQUERY[n].Fields[i].Visible;
    Form1.mainCDS[n].FieldByName(Form1.mainQUERY[n].Fields[i].FieldName).tag:=Form1.mainQUERY[n].Fields[i].tag;
  end;
end;
```

図2-5　機能毎の共通コンポーネントを配列化

```
mainQUERY: array[1..86] of TQuery;
mainCDS: array[1..86] of TClientDataSet;
mainSource:array[1..86] of TDataSource;
mainDBGrd:array[1..86] of TDBGrid;

OrderJunLBL: array[1..86] of TLabel;
EXCELoutBTN:array[1..86] of TBitBtn;
RecNoLBL:array[1..86] of TLabel;
FiltCondLBL:array[1..86] of TLabel;
```

図2-6　汎用処理での呼び出し元特定容易化例

```
for i:=1 to HighMainN do
  mainCDS[i].tag := i; //操作の呼び出し元の目印とする

for i:=1 to HighMainN do
  mainSource[i].tag := i; //操作の呼び出し元の目印とする

for i:=1 to HighMainN do
  mainDBGrd[i].tag := i; //操作の呼び出し元の目印とする

for i:=1 to HighMainN do
if EXCELoutBtn[i]<>Nil then
  EXCELoutBtn[i].tag := i; //操作の呼び出し元の目印とする
```

配列番号をTagﾌﾟﾛﾊﾟﾃｨに割り付け

（Formの OnCreate ｲﾍﾞﾝﾄで設定）

```
j:=(Sender as TBitBtn).Tag;

 if (mainCDS[j].Active=False) or (mainCDS[j].RecordCount=0) then
 begin
  showmessage('対象データを表示の上、実行しなさい!!');
  exit;
 end;
```

汎用化されたｴﾗｰ処理例

図2-7　iSeriesナビゲータの管理画面例（アクティブ・ジョブでDelphi/400使用状況確認）

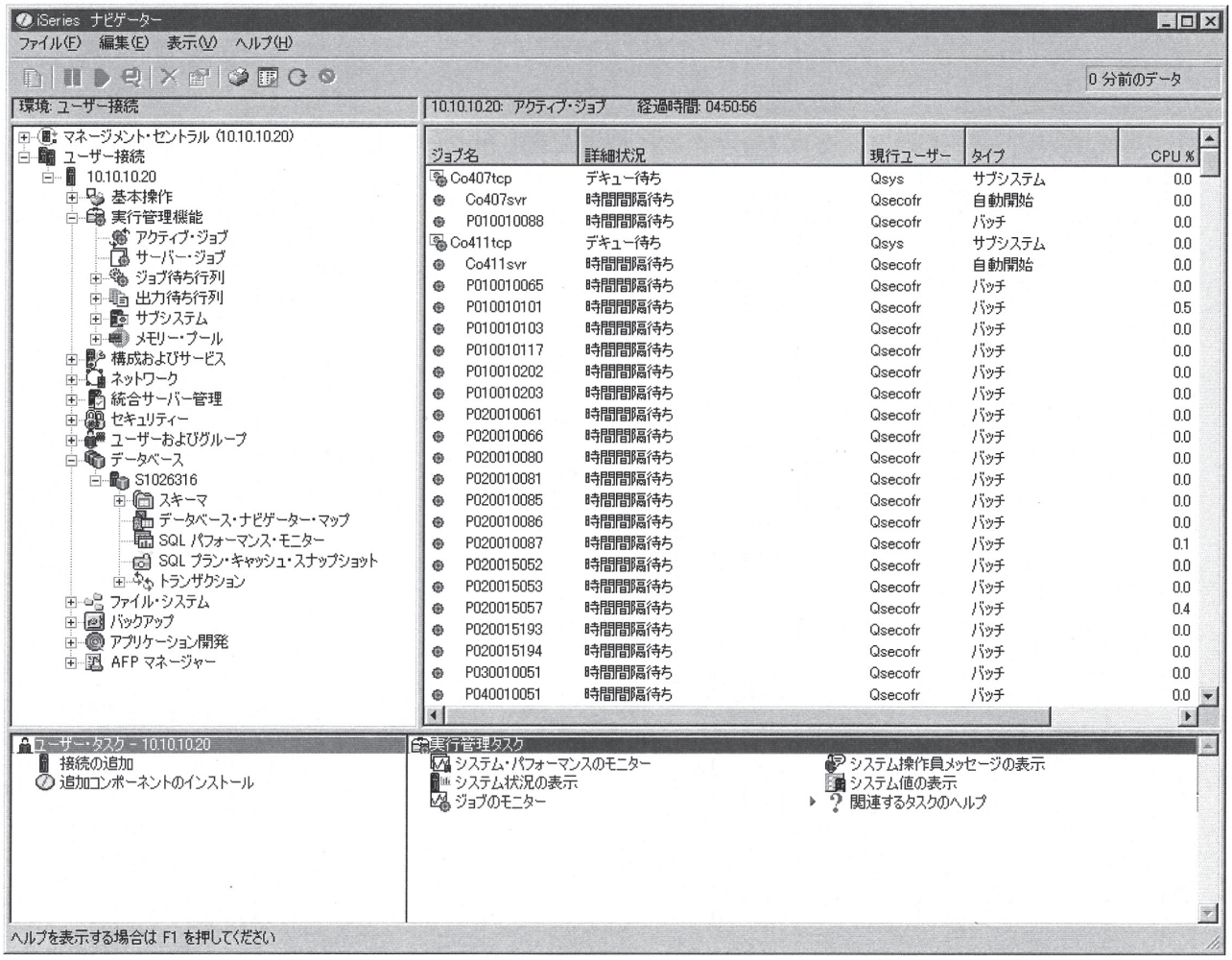


図2-8-1　iSeriesナビゲータでのSQLパフォーマンス検証例

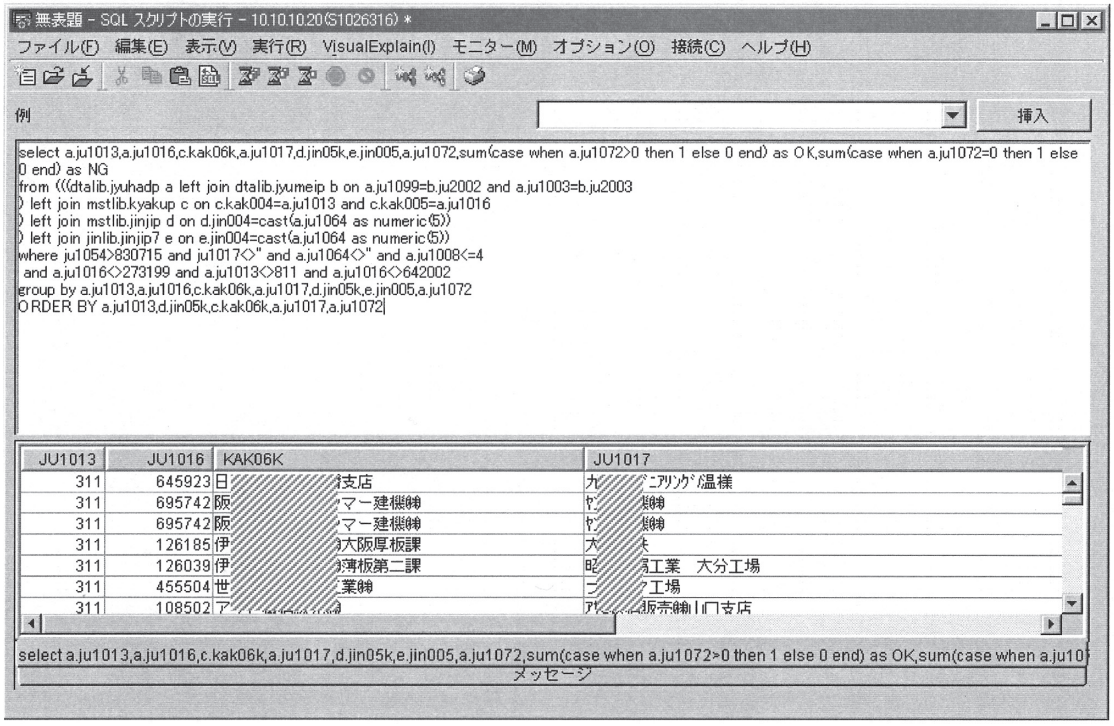


図2-8-2　iSeriesナビゲータ（VisualExplain）でのアクセスプラン、索引処理確認例

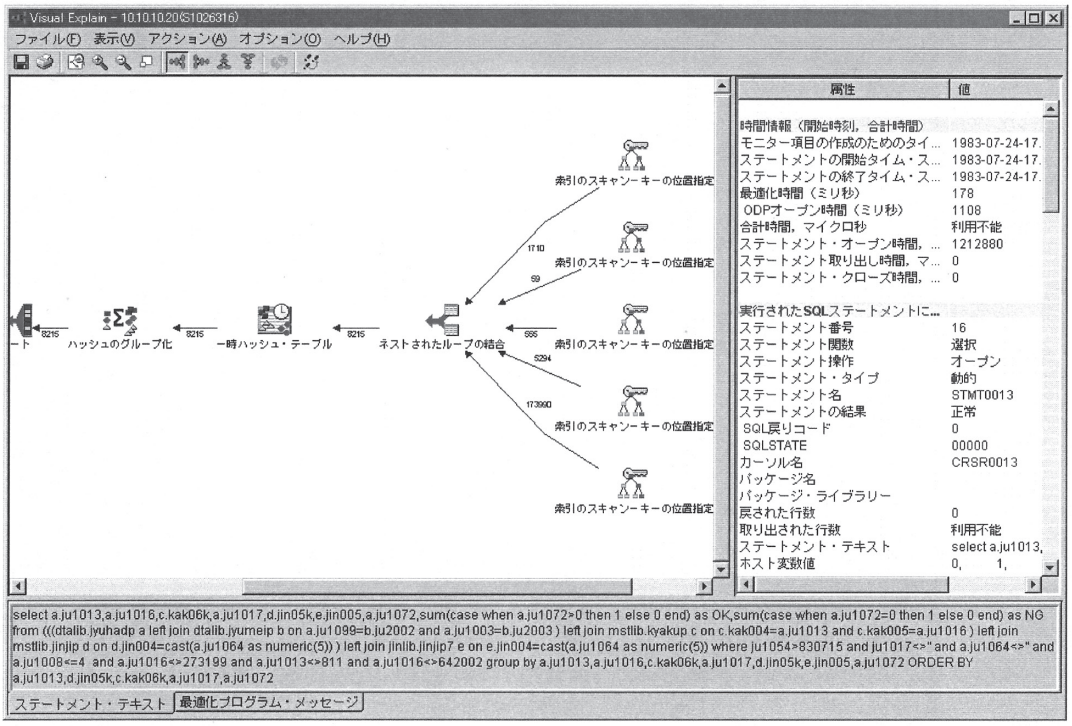


図2-9　iSeriesナビゲータによるGUIでのテーブル作成例

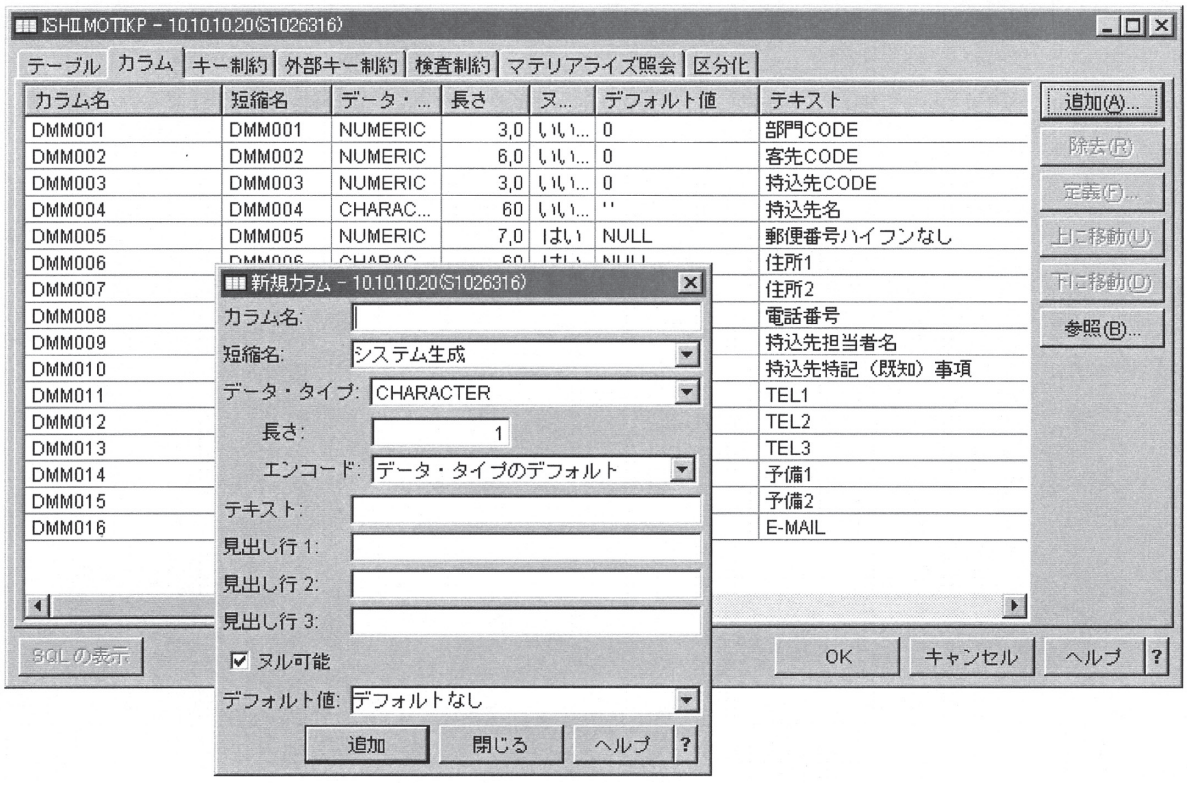


図2-10　iSeriesナビゲータによるGUIでの索引作成例

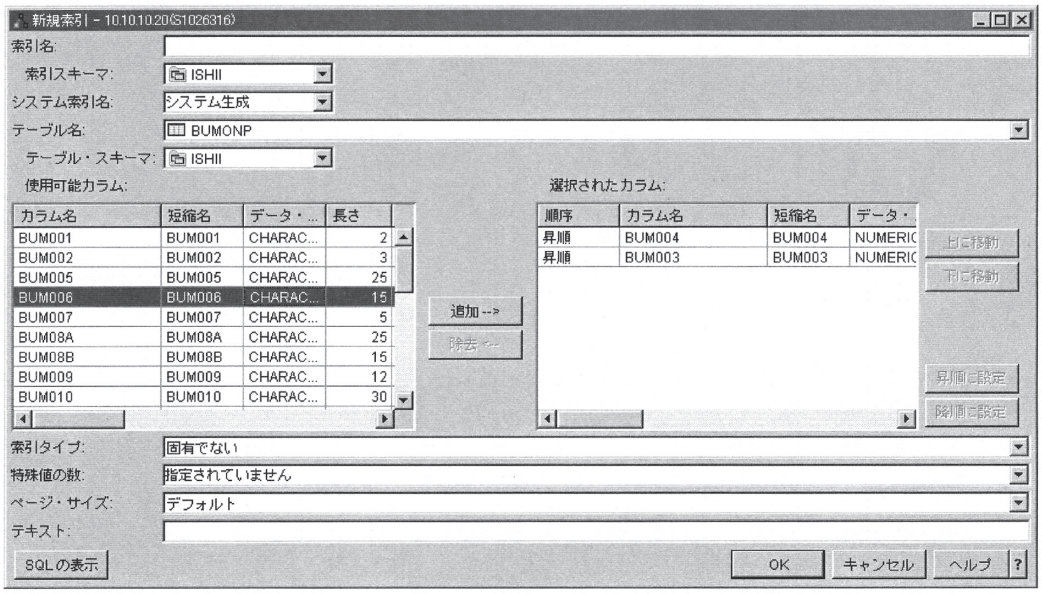


図2-11　Queryの配列番号でSQL文抽出

図2-12　抽出されたSQL文

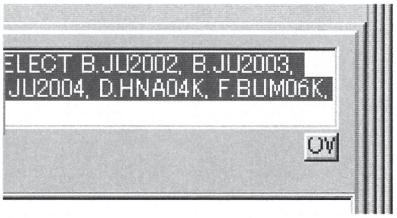


図2-13　単体テスト（SQL検証）用アプリケーション使用例

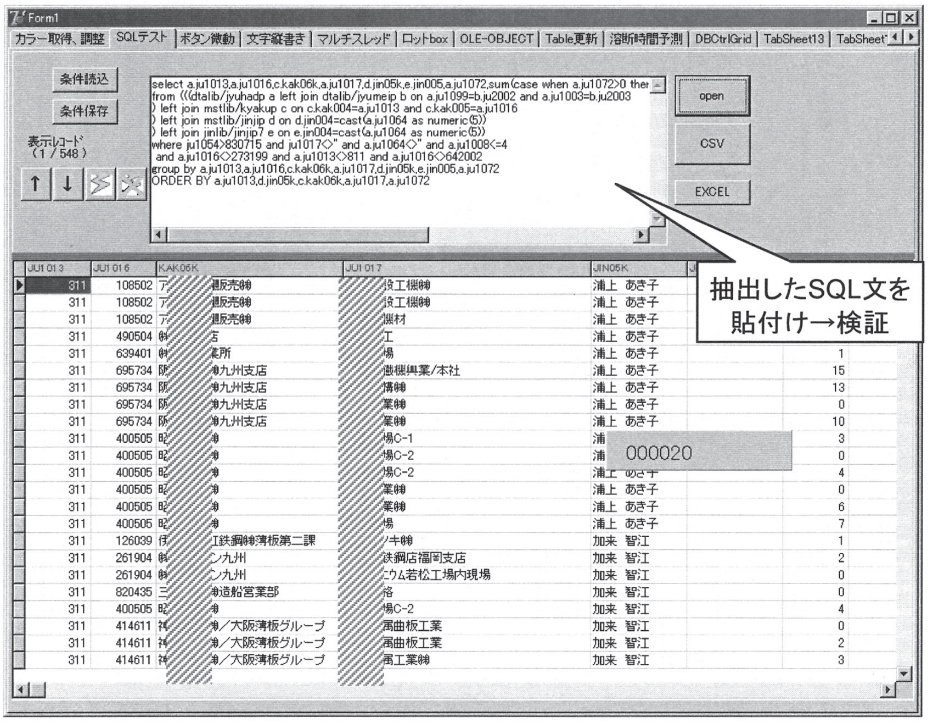


図2-14　SQL実行中の状況例-1

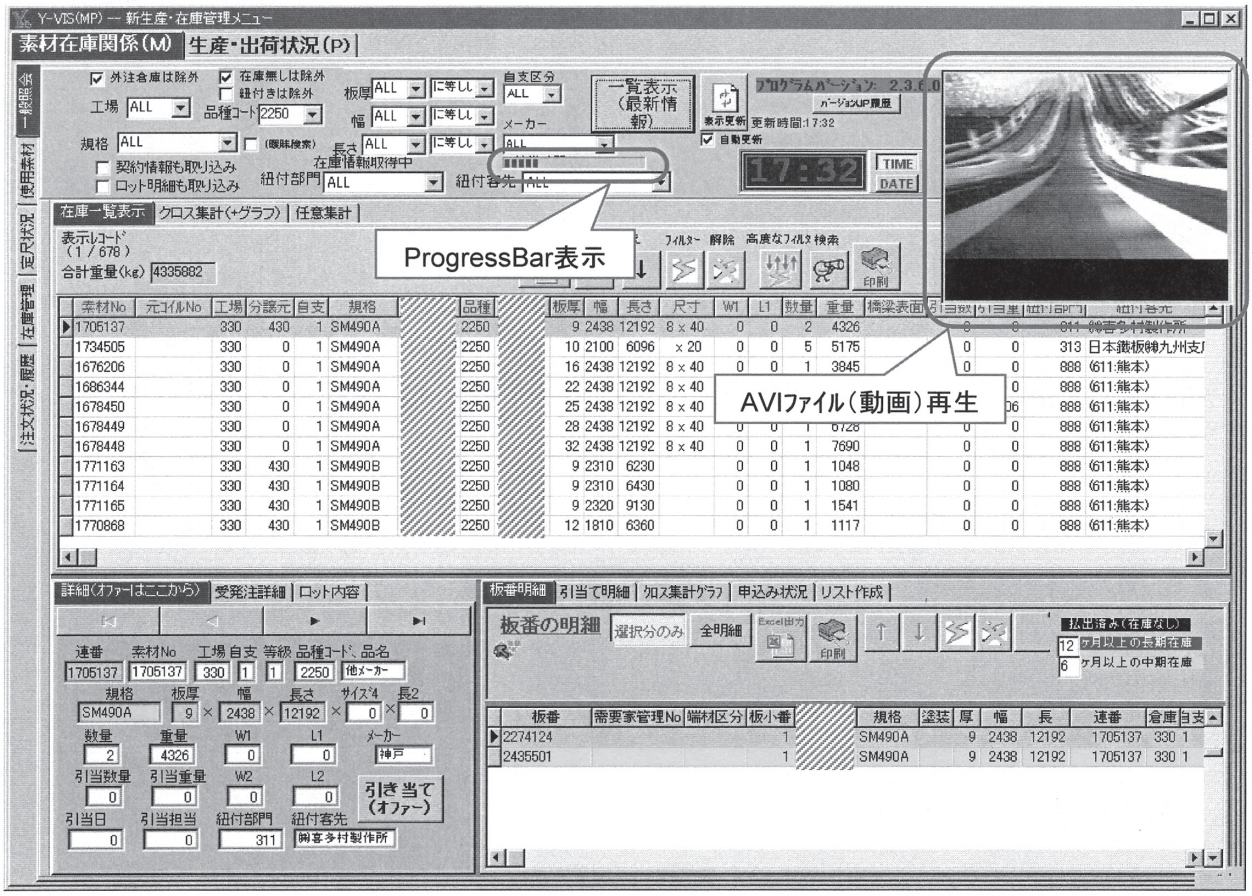


図2-15　SQL実行中の状況例-2

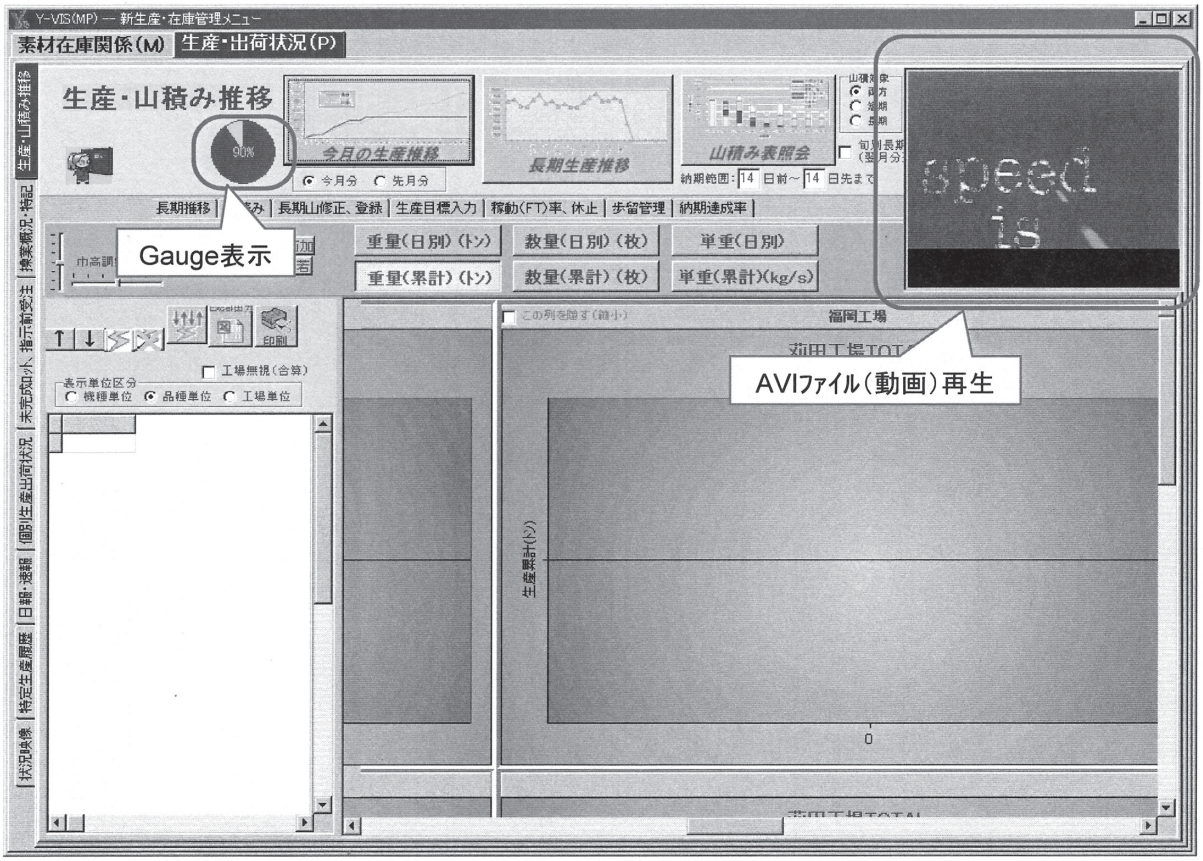


図2-16　ComboBoxのItem内容動的変更例-1

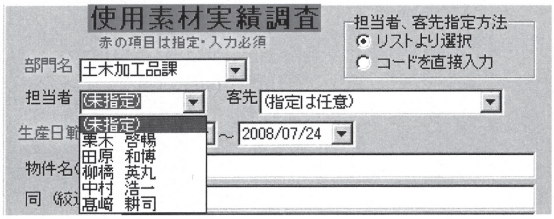


図2-17　ComboBoxのItem内容動的変更例-2

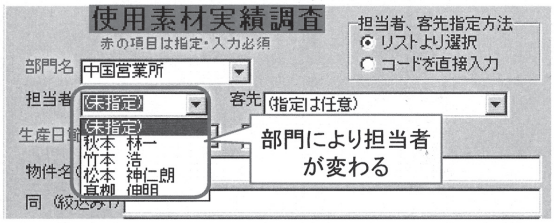


図2-18　ComboBoxのItem内容動的変更例-3

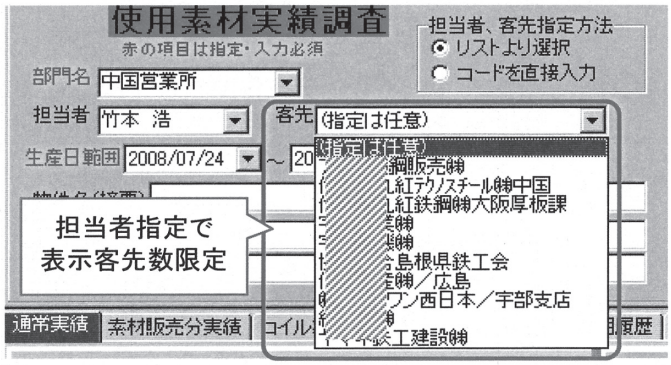


図2-19　各種マスターのローカルPC保存例

図2-20　各種マスターの定期更新例(起動時に実行)

図2-21　各種マスターの更新処理

```
for i:=1 to high(masterTB) do
begin
 BatchMoveMaster.Destination:=masterTB[i];
 BatchMoveMaster.Source:=masterOri[i];
 BatchMoveMaster.Execute;
end;
```

配列化したマスターを
BatchMoveで更新
（batCopyモード）

図2-22　一定時間 アプリケーション操作無しでの表示更新処理

```
procedure TForm1.ApplicationEventsRefIdle(Sender: TObject;
  var Done: Boolean);
begin
  RefreshTimer.Enabled:=true;
  Done:=true;
end;

procedure TForm1.ApplicationEventsRefMessage(var Msg: tagMSG;
  var Handled: Boolean);
begin
 case Msg.message of
    WM_MOUSEMOVE,WM_KEYDOWN,WM_NCLBUTTONDOWN,WM_NCRBUTTONDOWN
    begin
      RefreshTimer.Enabled:=False;
    end;
 end;
end;
```

アプリケーション操作が
無ければTimerを
起動

アプリケーション操作が
有ればTimer停止
→
Intervalのカウントは
一旦リセットされる

```
procedure TForm1.RefreshTimerTimer(Sender: TObject);
var
 t:integer;
begin
 //定時更新処理

 for t:=1 to high(AutoRefreshBTN) do
 begin
  if AutoRefreshYN[t].checked=true then
  AutoRefreshBTN[t].onClick(sender);
 end;

end;
```

Timer のInterval設定時
間に到達すれば
OnTimerイベントで
更新処理起動

図2-23　表示自動更新の設定例

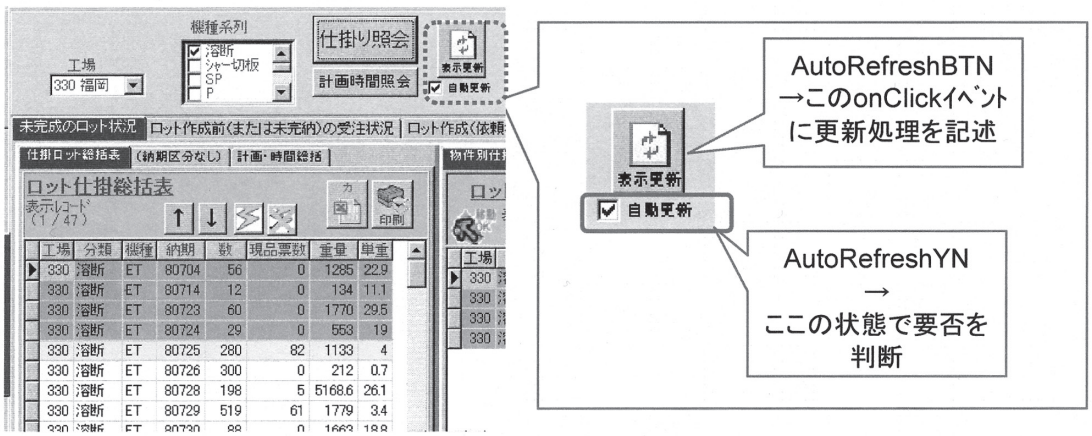


図2-24　項目の並び順カスタマイズの実現方法

```
procedure TForm1.ANYDBGridColumnMoved(Sender: TObject; FromIndex,
  ToIndex: Integer);   //GRIDの列並べ替えあれば並び順定義のStringList を書き直し
var
 n,c:integer;
begin
 n:=(Sender As TDBGrid).Tag;
 mainQUERYorder[n].Clear;  //Public変数の配列

 for c:=0 to mainDBGrd[n].Columns.Count-1 do
  mainQUERYorder[n].Add(mainDBGrd[n].Columns[c].FieldName); //並び順どおりの配列生成
end;
```

並べ替え時にOnColumnMovedイベントで並び順リスト(PublicのStringList)を変更

FormのCloseイベントにIniファイルに並び順等書き込み

```
for n:=1 to HighMainN do      //列の並び順,非表示項目 をPublic変数より書き込み
 begin
 Ini.WriteString('Form',IntToStr(n)+'ColumnORDER',mainQUERYorder[n].CommaText);
 Ini.WriteString('Form',IntToStr(n)+'ColumnNoVisible',mainQNoVisible[n].CommaText);
 end;
```

```
for i:=1 to HighMainN do   //各系列項目並び順のINIファイル定義を使用して表示順を入れ替え
begin
 mainQUERYorder[i]:=TStringList.Create;     //順番用の配列生成
 mainQUERYorder[i].CommaText:=Ini.ReadString('Form',IntToStr(i)+'ColumnORDER',''); //該当INIファイルより取込み
 if mainQUERYorder[i].CommaText<>'' then  //順番定義あれば以下処理
  begin
   for j:=0 to mainQUERYorder[i].Count-1 do
    begin
     for k:=0 to mainQUERY[i].FieldCount-1 do
      begin
       if mainQUERY[i].Fields[k].FieldName=mainQUERYorder[i].Strings[j] then
       begin
         mainQUERY[i].Fields[k].Index:=j;    //INIファイルの順番にIndexを振りなおし
         break;
       end;
      end;
    end;
  end;
end;
```

FormのCreateイベントにIniファイルより並び順を設定

図2-25　項目の並び順のIniファイル記述例

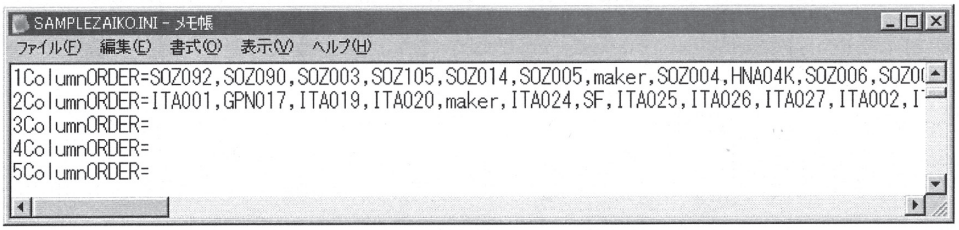


図2-26　項目の非表示化、再表示化操作例