【セッションNo. 4】

Delphi/400 テクニック公開

Delphi/400開発
～ パフォーマンス向上テクニック　～

株式会社ミガロ.
RAD事業部　営業推進課
尾崎　　浩司

【アジェンダ】

1. はじめに
2. データ抽出パフォーマンス向上テクニック
3. Excel出力パフォーマンス向上テクニック
4. 体感パフォーマンス向上テクニック
5. まとめ

1. はじめに

ユーザーに評価される業務アプリケーションとは

- 要件をきちんと満たしており、業務効率が向上している。
- 機能が優れていて、使い勝手が良い。

＜もう一つの大きな評価ポイント＞

- 使用していて満足のいくパフォーマンスが得られること。

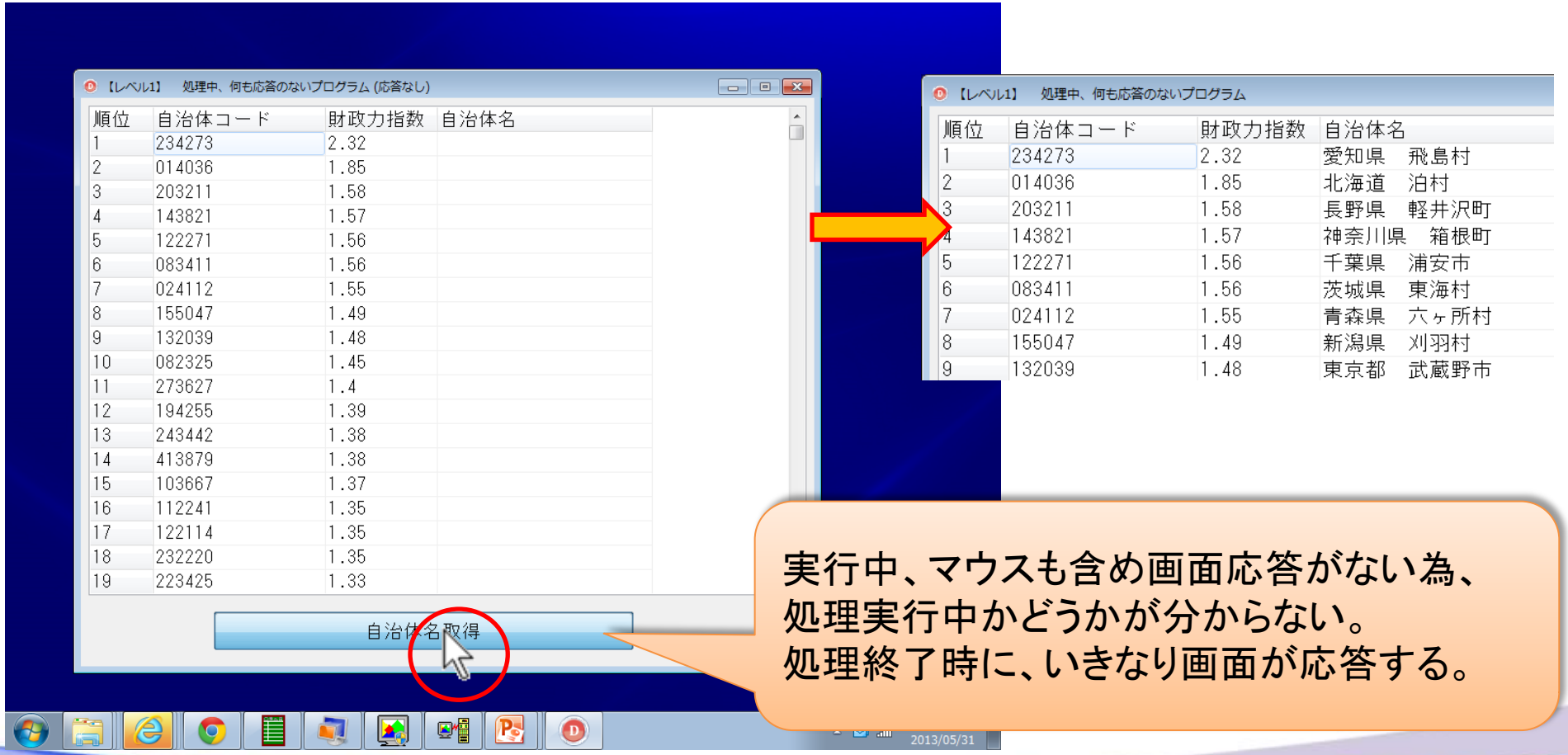
満足のいくパフォーマンスを得るために…

- 物理的な処理時間が短くなるような改善
  - 同じ処理でも、ちょっとした工夫やテクニックで処理時間が大きく変わる。

- 処理の待ち時間を感じさせない体感的な改善
  - バッチ処理など時間がかかるものは、経過状況がわかるようにする等の工夫が大事。

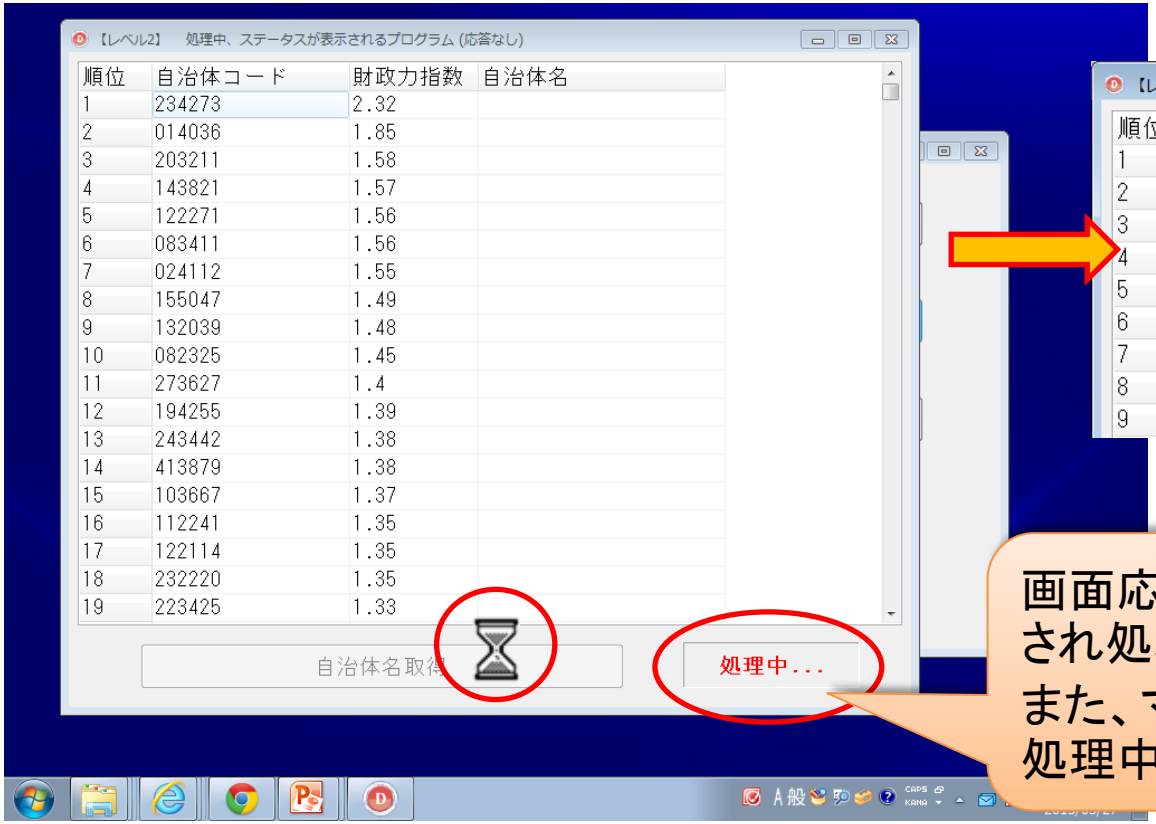
処理に時間がかかるアプリケーション

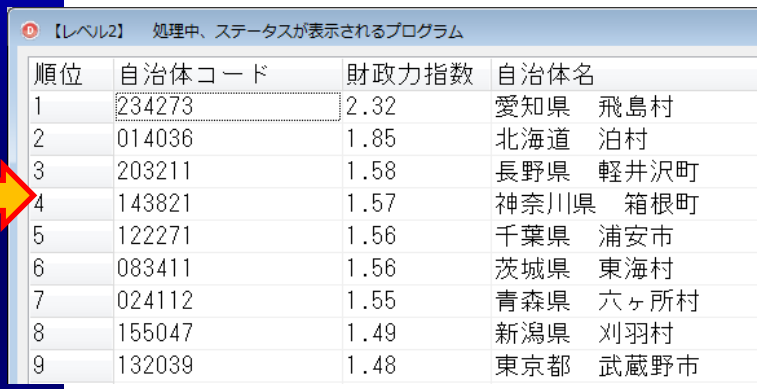
- 悪い例
  - 処理のみが記述されていて、画面制御が無い。



処理に時間がかかるアプリケーション

- 画面制御を改善
  - 「処理中」を知らせるメッセージを表示。
  - 実行中を表すマウスカーソル（砂時計）に変更。

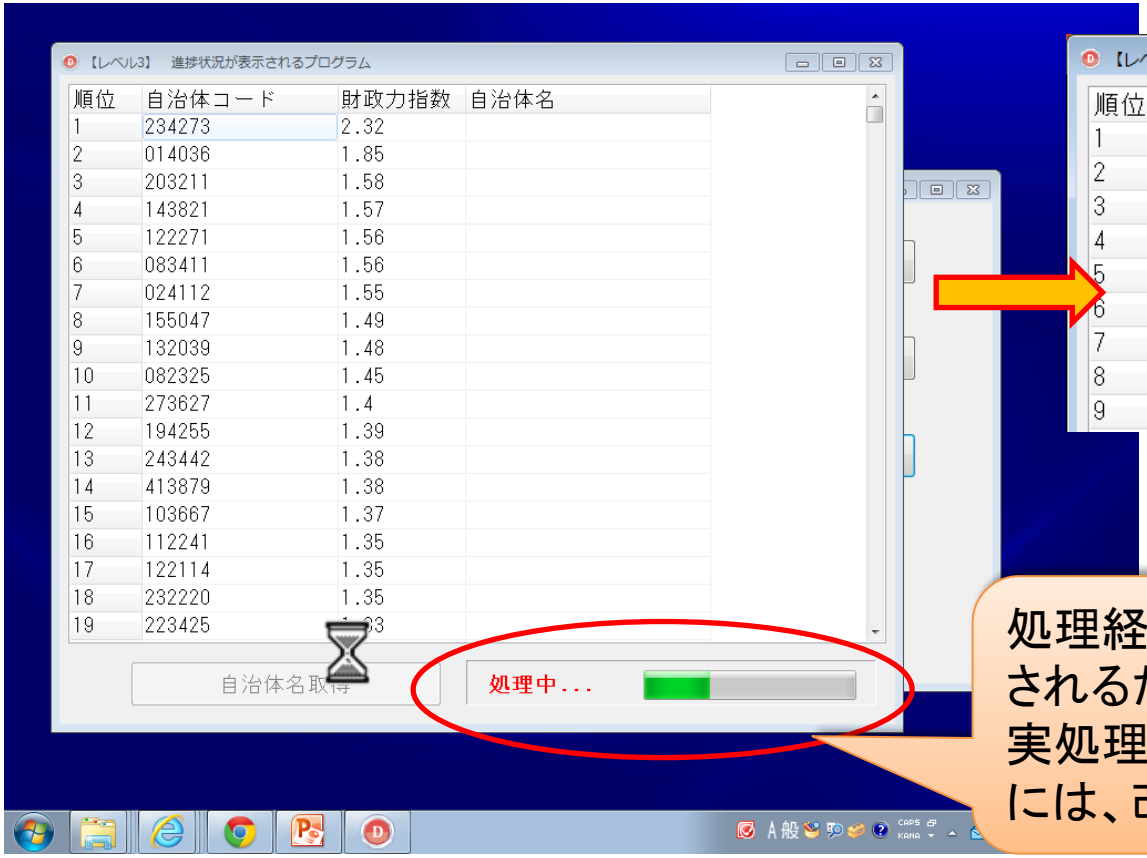




画面応答は無いが、メッセージが表示され処理中であることがわかる。
また、マウスカーソルも砂時計のため、処理中であることを意識できる。

処理に時間がかかるアプリケーション

- さらなる改良を実施
  - プログレスバーを追加し、処理件数に応じた進捗状況を出力。



処理経過がプログレスバーに表示されるため、進捗状況が把握できる。
実処理時間は変わらないが、体感的には、改善されたように見える！

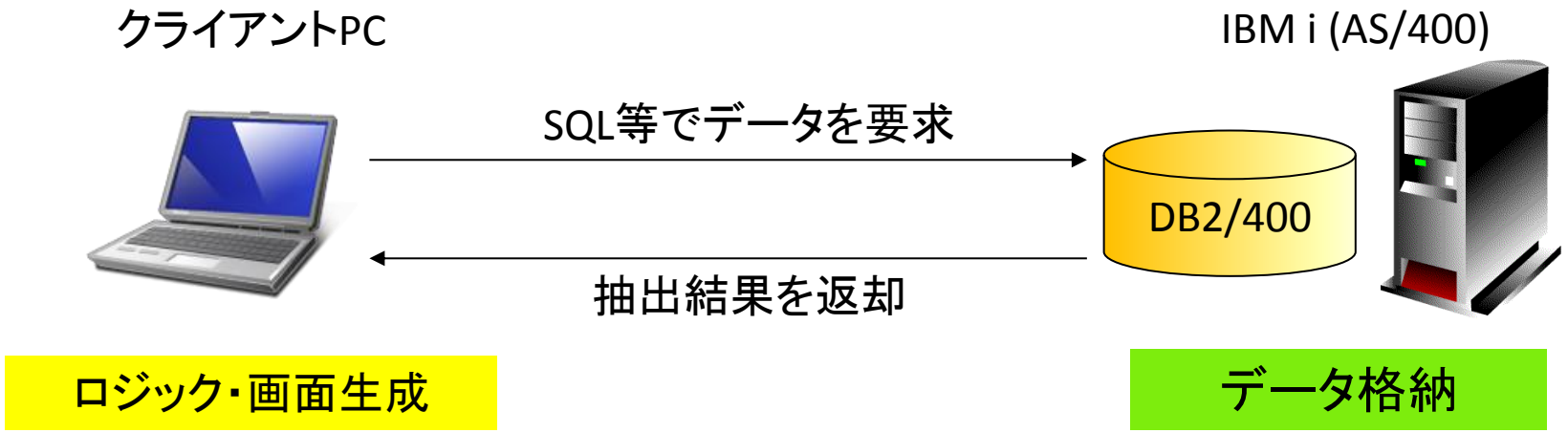
今回のポイント

- 物理的な処理時間が短くなるような改善
  - IBMi(AS/400)上のデータを効率良く抽出するための手法
  - 事前キャッシュを利用した処理速度の向上
  - Excel出力の高速化
- 処理の待ち時間を感じさせない体感的な改善
  - マウスカーソル制御
  - マルチスレッド化による応答なし状態の回避

2. データ抽出パフォーマンス向上テクニック

業務アプリケーションの基本

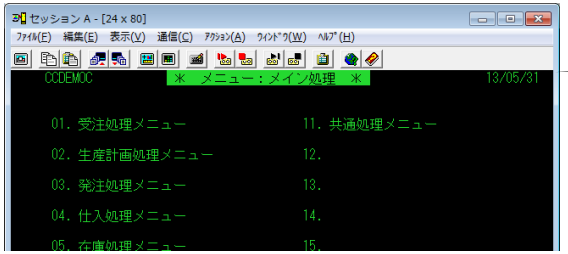
- Delphi/400を使用したIBMi(AS/400)を活用する業務アプリケーション
  - クライアントPC（Delphi/400 Exe）は、リモートサーバであるIBMi(AS/400)にデータを要求し、抽出された結果を受け取って処理を行う。



RPGアプリとDelphi/400アプリとの比較

クライアントPC

IBM i (AS/400)



エミュレータ

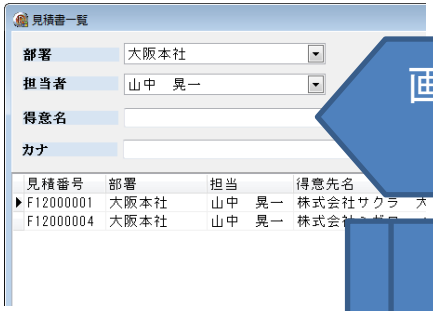
画面情報が転送

画面ファイル
DSPF

ロジック
RPG

DB2/400

大量データであっても
ホスト内で効率的に処理



画面ファイル
Form

ロジック
Delphi

Delphi/400　Exe

抽出データが転送

DB2/400

大量データを要求すると、
大量の通信が発生し、
パフォーマンスが低下

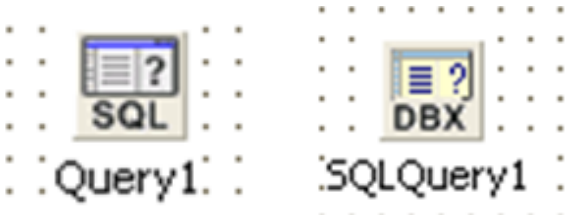
- 必要なデータを効率よく抽出することが重要！

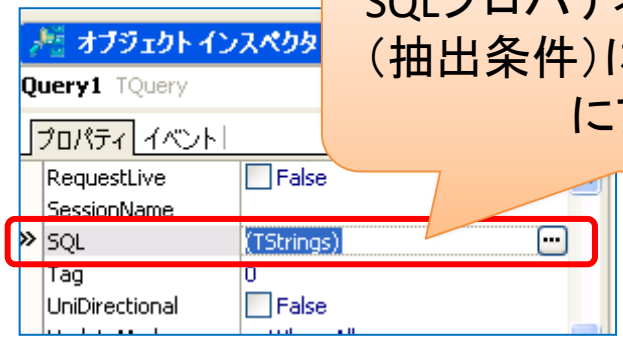
テーブル と クエリー

- テーブル（TTable)

TableNameプロパティに指定した物理ファイル/論理ファイルに全件アクセス

- クエリー（TQuery)





SQLプロパティに指定したSQL文（抽出条件）に合致するレコードにアクセス

- IBMi(AS/400)への問合せは、クエリーの使用が基本。
- RPG処理結果として出力したQTEMP上のファイルへのアクセスは、テーブルを使用すると便利。

テーブルとクエリーでのパフォーマンス比較

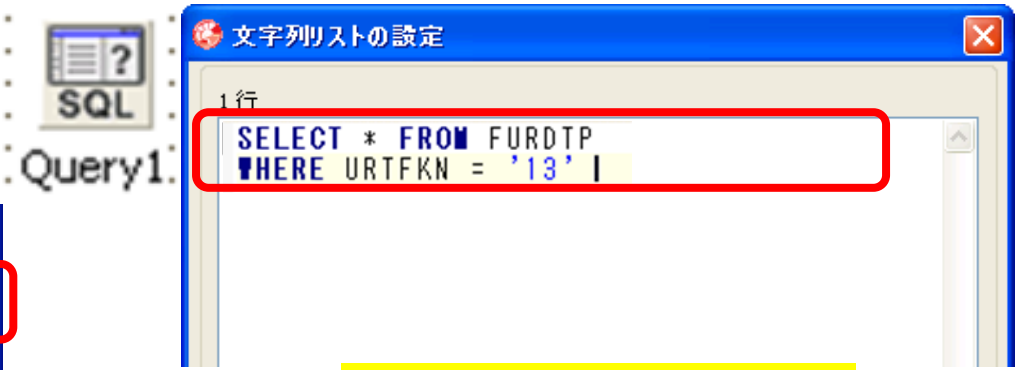
- 500,000件のデータから都道府県＝”13”のデータを抽出

売上データファイル
FURDTP
都道府県フィールド
URTFKN

物理ファイル D4TECLIB/FURDTP　様式名 FURDTR　レコード
様式記述 売上データファイル
5= 詳細

選択	項目名	桁数	属性	キー順	開始	終了	テキスト記述
_	URDTNO	9 0	S		1	9	売上ＮＯ
_	URTNCD	4	A		10	13	担当者ＣＤ
_	URSYDT	8 0	S		14	21	処理日付
_	URKNGK	11 0	P		22	27	売上金額
_	URTFKN	2	A		28	29	都道府県
_	URTKCD	6	A		30	35	得意先ＣＤ
_	URDLTF	1	A		36	36	1= 削除

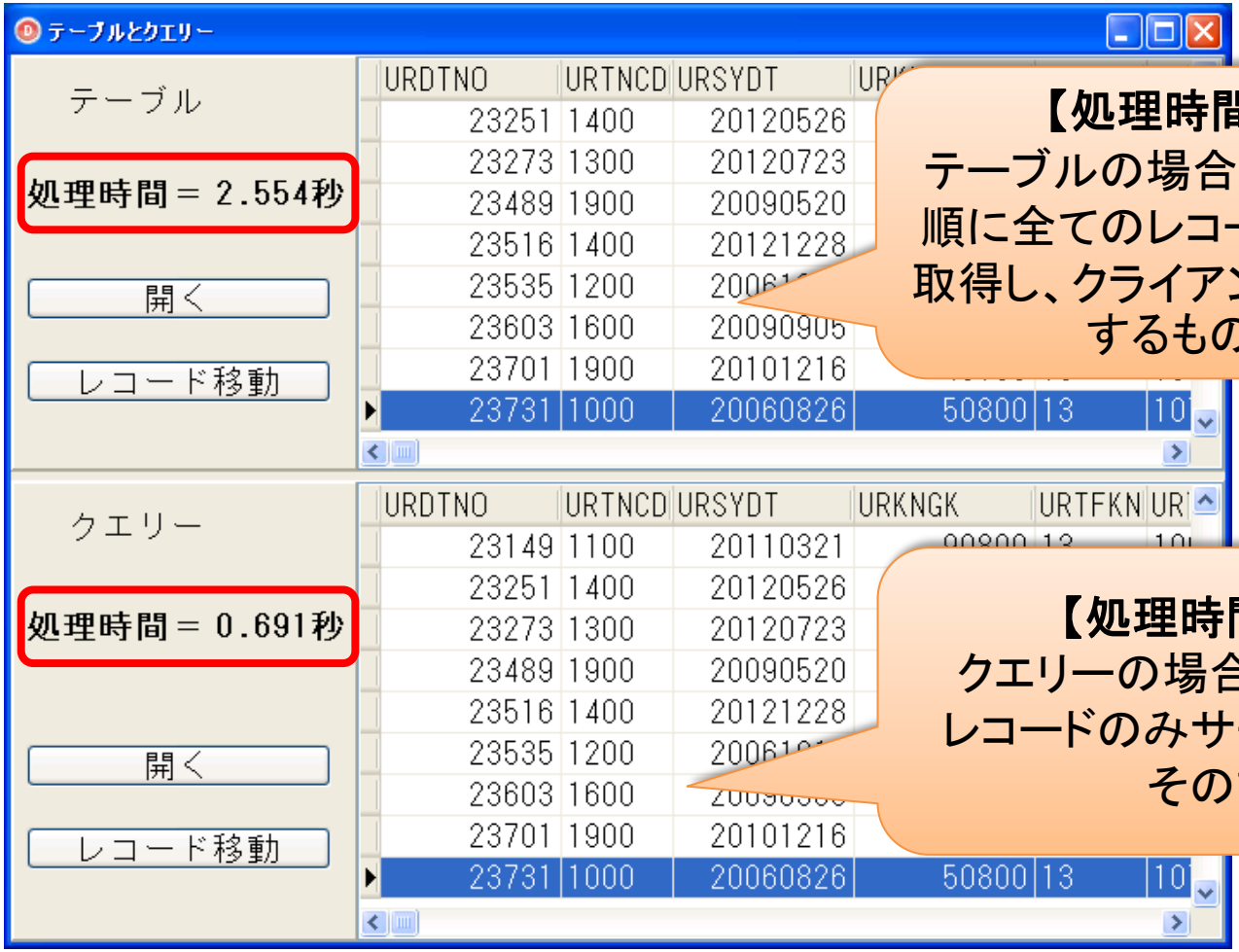
Filterプロパティ



SQLプロパティ

テーブルとクエリーでのパフォーマンス比較

- 条件に合致するレコードを順に500件読み込む。



【処理時間＝約2.5秒】
テーブルの場合、先頭レコードから順に全てのレコードをサーバーから取得し、クライアント上で条件に合致するもののみ表示

【処理時間＝約0.7秒】
クエリーの場合、条件に合致するレコードのみサーバーから取得し、そのまま表示

SQLパフォーマンス向上

- 論理ファイル（LF）の活用

売上データファイル
物理ファイル：FURDTP

（キー指定なし）

【抽出SQL】

```
SELECT
  F.*, T.TATANM, K.TDFKNM, M.TKTKNM
FROM FURDTP F

LEFT JOIN MTANTP T ON F.URTNCD = T.TATACD
LEFT JOIN MTOKUP M ON F.URTKCD = M.TKTKCD
LEFT JOIN MTDFKP K ON F.URTFKN = K.TDFKCD
WHERE
    F.URTKCD LIKE :URTKCD
AND F.URSYDT >=   :URSYDT_FM
AND F.URSYDT <=   :URSYDT_TO
AND F.URDLTF <> '1'

ORDER BY F.URTKCD, F.URSYDT, F.URDTNO
```

売上データファイル
論理ファイル：FURDTL01

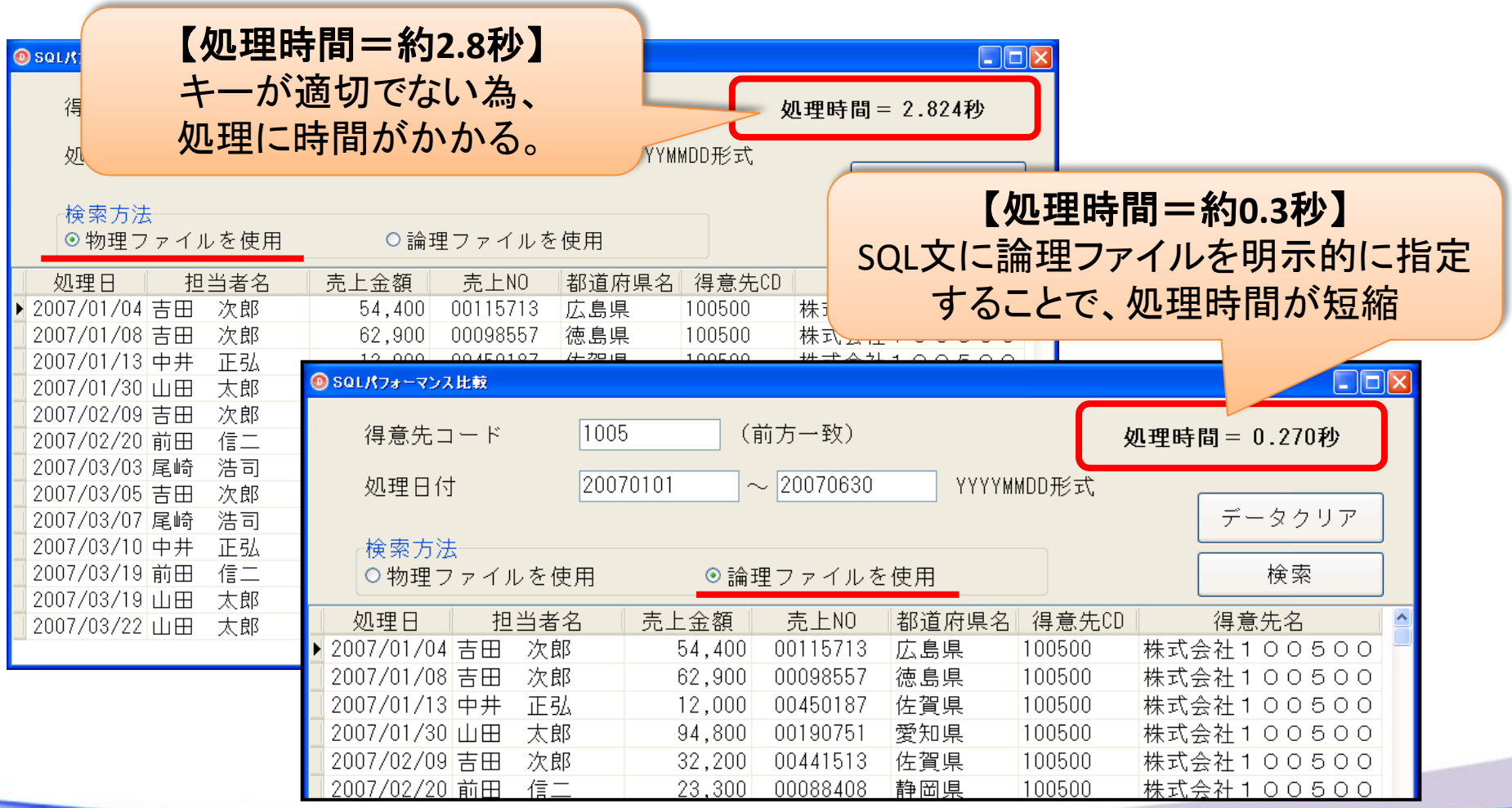
Key1 ： URTKCD（得意先ｺｰﾄﾞ）
Key2 ： URSYDT（処理日付）
Key3 ： URDTNO（売上NO）

Select： URDFLTF（削除F）≠ “1”

```
SELECT
  F.*, T.TATANM, K.TDFKNM, M.TKTKNM
FROM FURDTL01 F

LEFT JOIN MTANTP T ON F.URTNCD = T.TATACD
LEFT JOIN MTOKUP M ON F.URTKCD = M.TKTKCD
LEFT JOIN MTDFKP K ON F.URTFKN = K.TDFKCD
WHERE
    F.URTKCD LIKE :URTKCD
AND F.URSYDT >=   :URSYDT_FM
AND F.URSYDT <=   :URSYDT_TO

ORDER BY F.URTKCD, F.URSYDT, F.URDTNO
```

SQLパフォーマンス向上

- 画面で指定した条件に合致するレコードを抽出



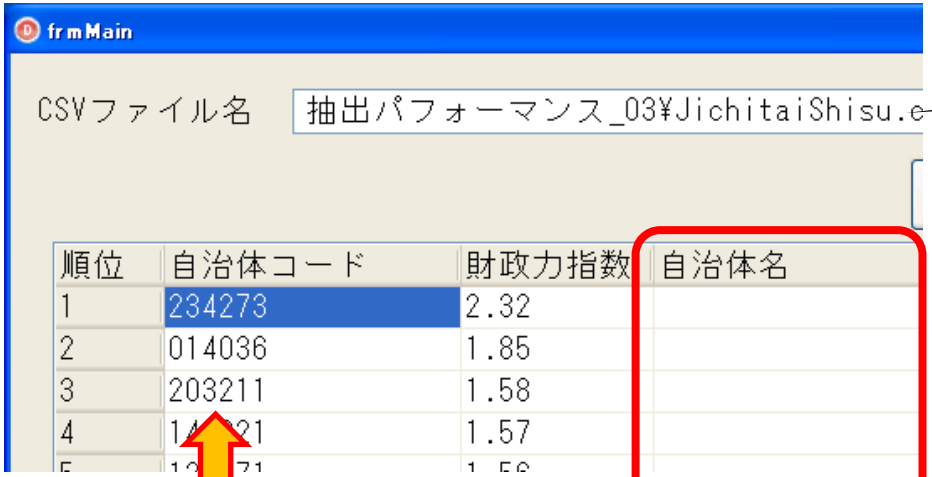
事前キャッシュによる高速化

- IBMi(AS/400)上にあるマスターを参照する。

クライアントPC

IBM i (AS/400)

公共団体マスター
MCHKDP



自治体コードが
記されたCSVファイル

自治体コード(CHKTCD)
自治体名称(CHKTNM)
を保持

レコード数:1,794

IBMi(AS/400)上のマスターを参照し、
自治体名を取得する。
(取得件数:250件)

■ クエリーによる参照

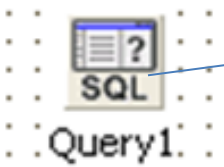
- 自治体コード1件ずつをクエリー(SQL)で問合せ。

```
procedure TfrmMain.btnGetDataClick(Sender: TObject);
begin
  //各行の自治体コードより自治体名を取得して画面にセットする
  for iRow := 1 to sgGrid.RowCount - 1 do
  begin
    sgGrid.Cells[3, iRow] := GetData(sgGrid.Cells[1, iRow]);
  end;
end;
```

明細行毎にGetData関数を呼出して、名称を取得。

メイン処理

GetData関数




```
SELECT CHKTNM FROM MCHKDP
WHERE CHKTCD = :CHKTCD
```

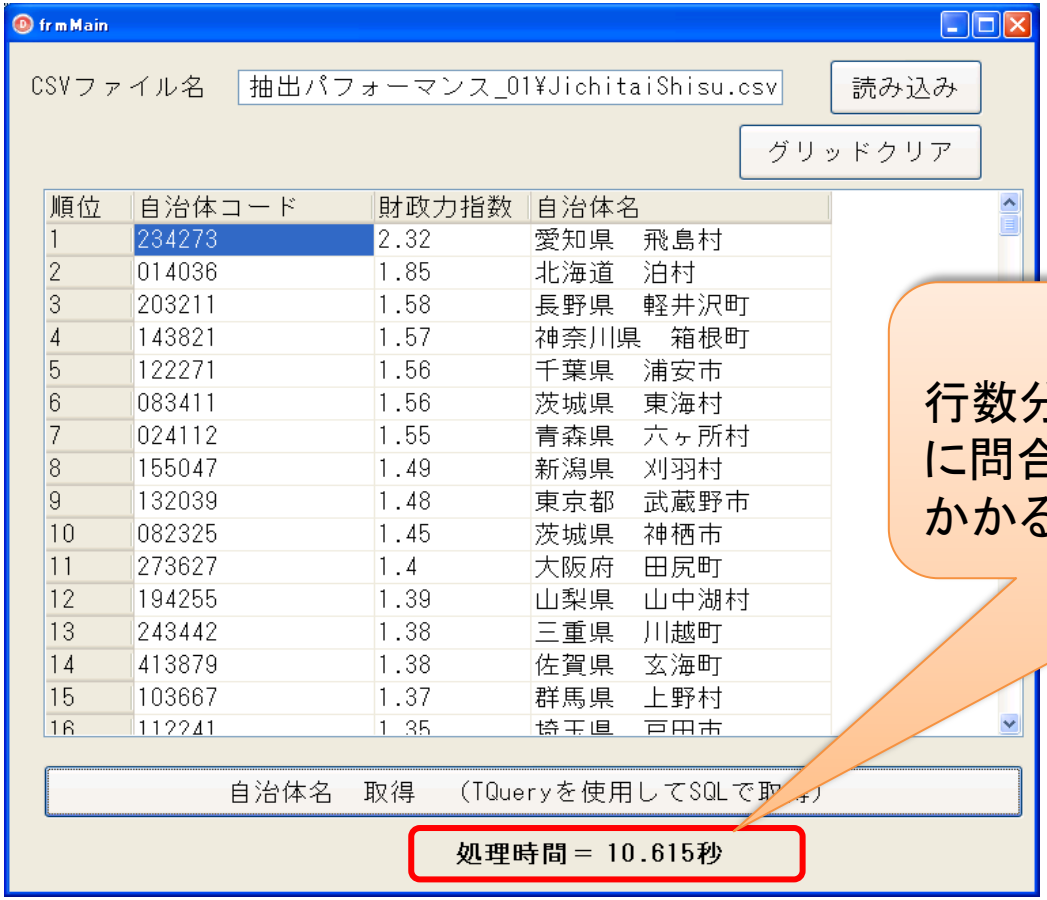
SQLプロパティ

```
function TfrmMain.GetData(ACode: String): String;
begin
  with Query1 do
  begin
    //パラメータの指定
    ParamByName('CHKTCD').AsAnsiString := ACode;
    //データセットを開き値を取得
    Active := True;
    try
      Result := FieldByName('CHKTNM').AsString;
    finally
      Active := False;
    end;
  end;
end;
```

クエリーを実行し、IBMi(AS/400)に問合せを実行

■ クエリーによる参照

- 実行結果

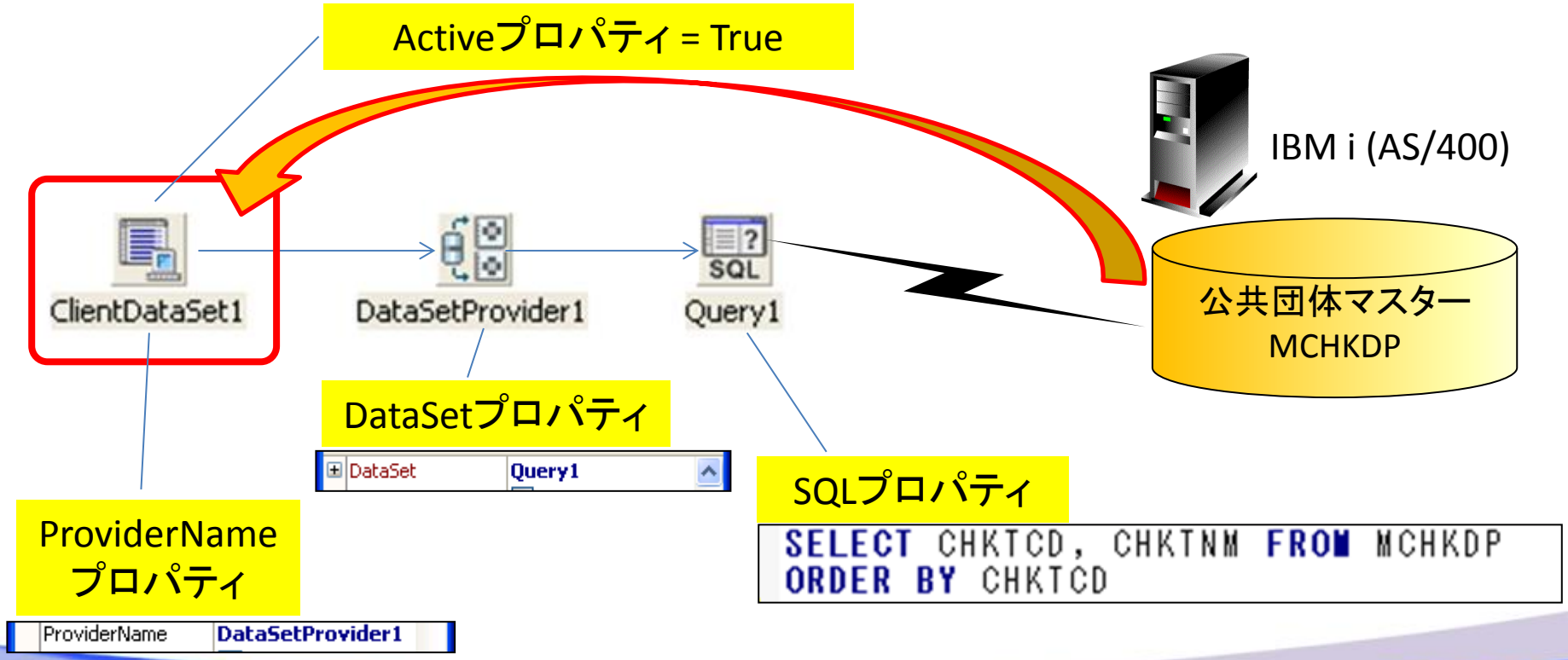


【処理時間＝約10秒】
行数分だけ、SQL文をIBMi(AS/400)に問合せする為、処理に時間がかかる。

- パフォーマンスをあげられないだろうか？

クライアントデータセットによる事前キャッシュ

- クライアントデータセットを使用。
  - メモリにキャッシュされるデータセット。
  - Activeプロパティを Trueにした際、DataSetProvider経由で接続された クエリーが実行され、結果のデータセットが全件メモリ上に格納される。



クライアントデータセットによる事前キャッシュ

- 起動時に、メモリにキャッシュ。

```
procedure TfrmMain.FormCreate(Sender: TObject);
begin
          ・・・・
  //クライアントデータセットを開き、メモリ上に情報を格納する
  ClientDataSet1.Active := True;
end;
```

起動時にIBMi(AS/400)に問合せを行い、全件取得する。

フォーム生成処理

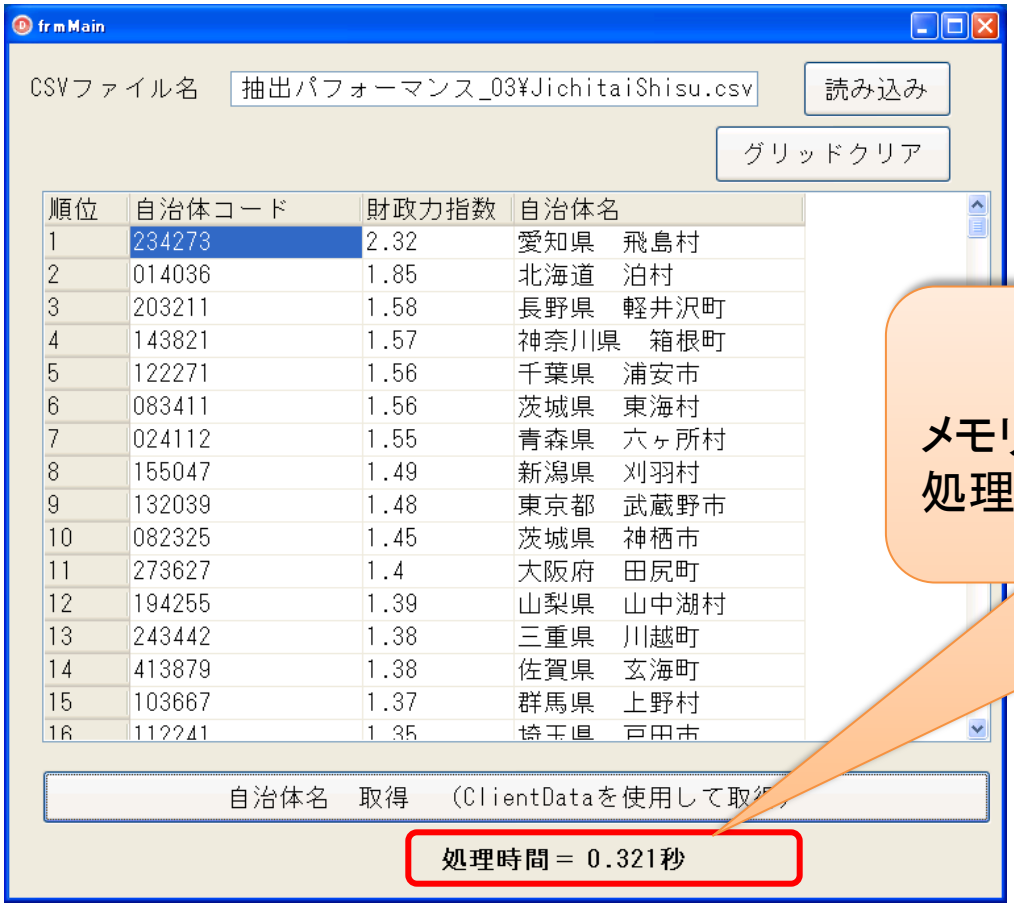
GetData関数

```
function TfrmMain.GetData(ACode: String): String;
begin
  //メモリ上のデータセットよりデータを取得
  with ClientDataSet1 do
  begin
    //データを検索する
    if Locate('CHKTCD', ACode, []) then
    begin
      //値を取得する
      Result := FieldByName('CHKTNM').AsString;
    end;
  end;
end;
```

Locateメソッドにより、メモリ上のデータセットを検索し、名称を取得。

■ クライアントデータセットによる事前キャッシュ

- 実行結果



【処理時間＝約0.3秒】
メモリ中のデータセットに問合せする為、
処理時間の短縮が可能！

- 事前キャッシュによりパフォーマンスが向上

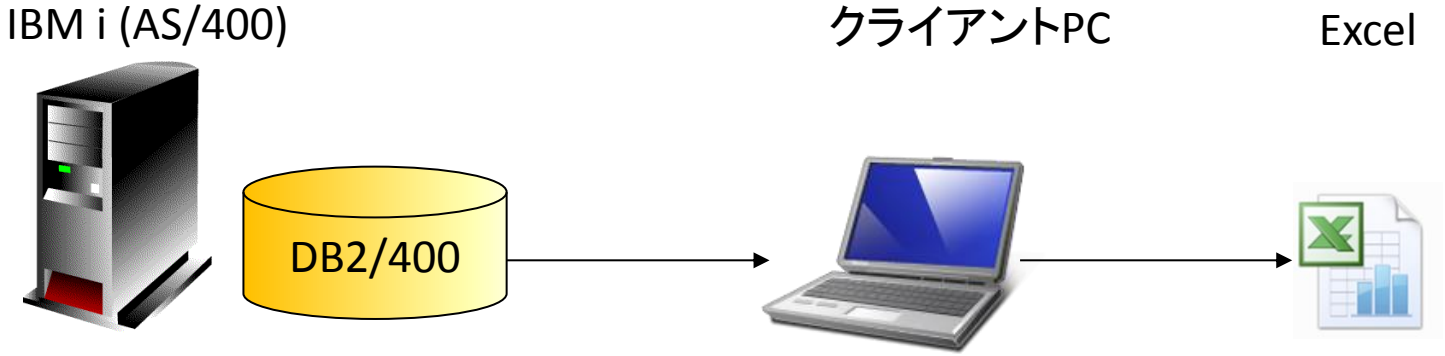
クライアントデータセットによる事前キャッシュ

- 事前キャッシュが有効に活用できるパターン
  - 同じマスターを何度も参照する必要がある。
    - 明細行単位にマスター参照が必要な場合に有効。
  - マスターのレコード件数が比較的少ない。
    - 件数が多いと、キャッシュに時間がかかり効果が低い。
    - 全レコードが不要な場合は、SQLで絞り込みを行いキャッシュする。
  - 頻繁にマスターの値が更新されない。
    - 組織マスターや住所マスターのように頻繁に更新がかからないものに有効。
    - あくまで、キャッシュしたコピーデータを使用するため、データの鮮度が重要な場合は不向き。

3．Excel出力パフォーマンス向上テクニック

Excelとの連携

- Delphi/400におけるExcel利用形態
  - ユーザーが加工しやすいよう業務データをホストからダウンロードして、Excelに出力する。
  - 帳票として、雛形のExcelにデータを差し込んで出力する。



データの出力先として
（加工が容易）

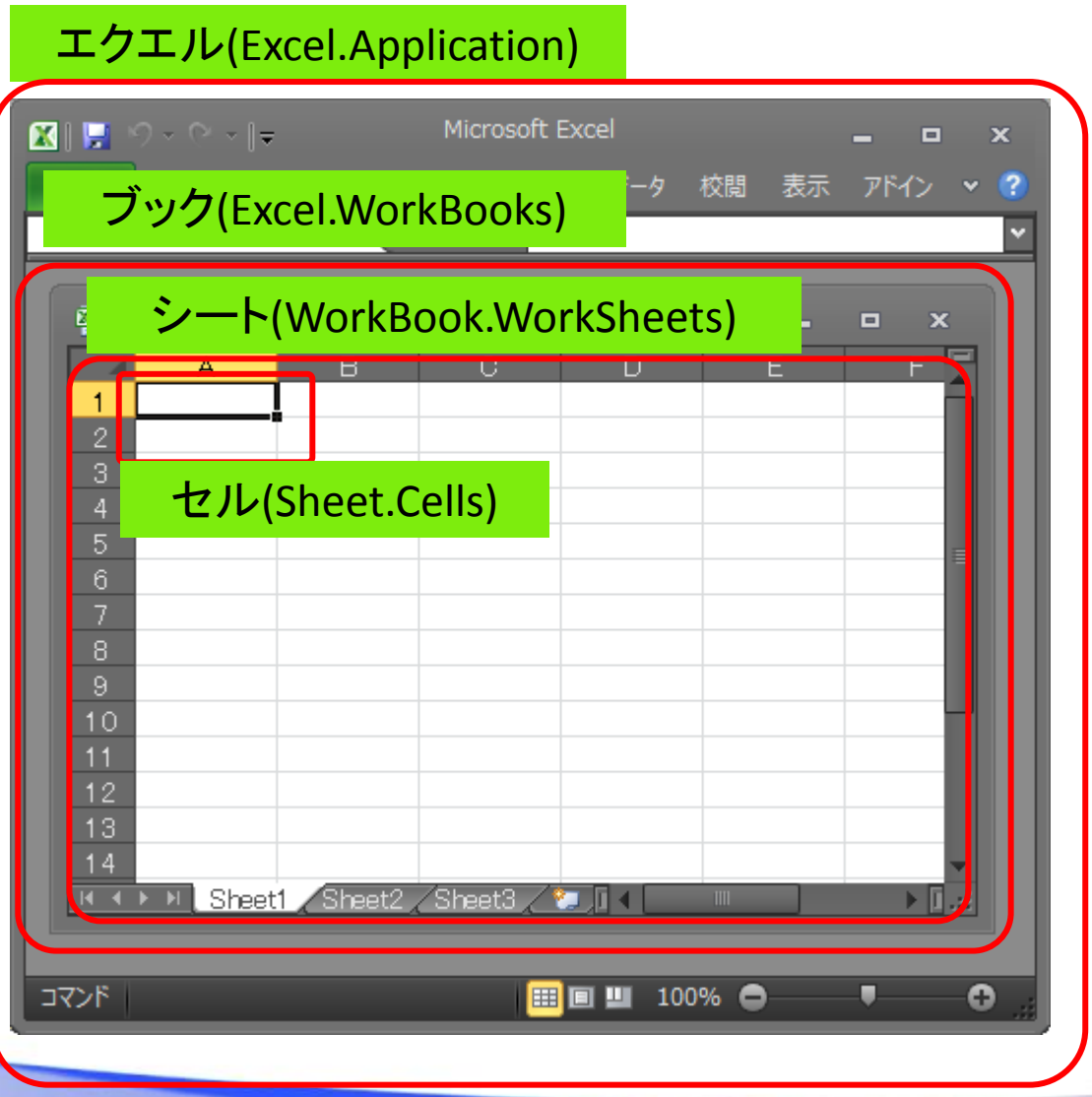
帳票ツールとして
見積書、注文書　等
（レイアウト作成が容易）

Excelへ出力する方法

- OLE(Object Linking and Embedding)の使用
  - 複数のアプリケーション間で、データの転送や共有を行う仕組み。
  - Excelは、OLEサーバとなり、他のアプリ（Delphi/400等）から、操作することが可能。（操作する為のメソッドが用意されている）
  - 実行する為には、Excel環境が必要。

- VB-Report/ExcelCreator等3rdParty製品の使用
  - Excelファイルの作成や帳票出力が可能。
  - OLEより高速なことが多い。
  - 実行する為に 、Excel環境が不要。

OLE操作方法




```
Excel := CreateOleObject('Excel.Application');
```

OLEでExcelを起動

```
Book := Excel.WorkBooks.Add;
```

Excelに新規ブックを追加

```
Sheet := Book.ActiveSheet;
```

現在有効なシートを選択

```
Sheet.Cells[1, 1] := 'ミガロ. ';
```

A1セルに文字列を代入

- Uses節に、comObjを追加
- 変数は、OleVariant型として定義

OLEを使用したデータ出力

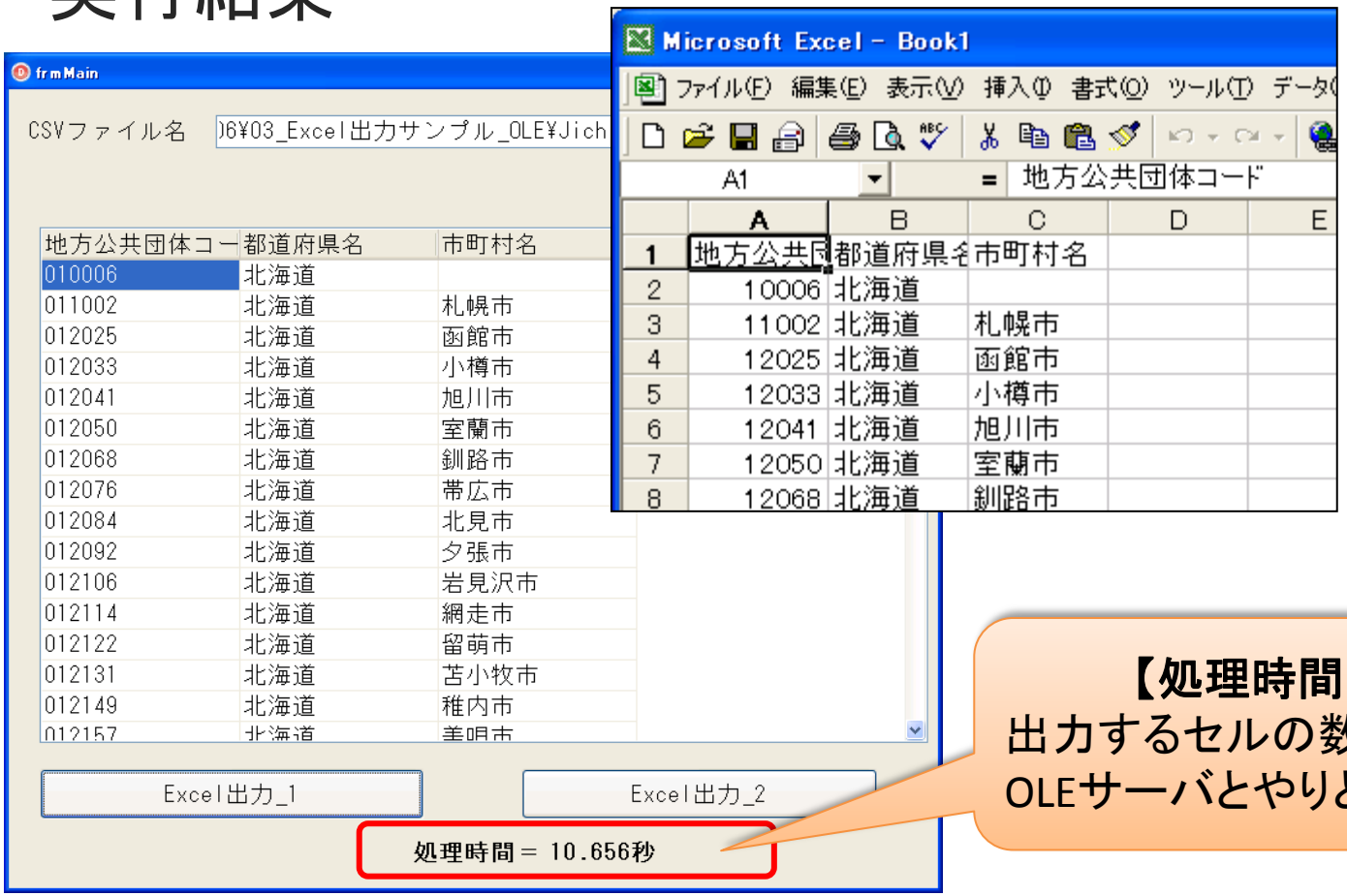
- StringGridの内容をExcelへ出力

```
procedure TfrmMain.ExcelOut1;
var
  iRow, iCol: Integer;
  Excel: OleVariant;
begin
  Excel := CreateOleObject('Excel.Application');
  Excel.WorkBooks.Add;
  Excel.ActiveSheet.Cells.Select;

  //StringGridの情報を元にExcelにデータを出力する
  for iRow := 0 to sgGrid.RowCount - 1 do
    for iCol := 0 to sgGrid.ColCount - 1 do
      //セルにデータ出力
      Excel.ActiveSheet.Cells[iRow + 1, iCol + 1].Value
        := sgGrid.Cells[iCol, iRow];

  //A1セルを選択状態にする
  Excel.ActiveSheet.Cells[1, 1].Select;
  // エクセルを表示
  Excel.Visible := True;
end;
```

StringGridの内容を各行、各列ループで値を取得しながら、OLEで順番にセルに代入。

OLEを使用したデータ出力

- 実行結果



【処理時間＝約10秒】
出力するセルの数だけ、
OLEサーバとやりとりを実施。

- パフォーマンスをあげられないだろうか？

Excel出力のパフォーマンスをアップする方法

- OLEサーバ(Excel)との通信回数をできるだけ少なくする。
  - セル一つ一つの代入で通信は行わず、複数セルの情報をまとめて通信するとパフォーマンスがアップ。

- クリップボードの活用　第1回Delphi/400テクニカルセミナーでご紹介
  - 複数セルの情報をタブ区切りの文字列として作成。
  - 作成した文字列をクリップボードにコピー。
  - 貼りつけ位置を選択し、貼り付けを行う。

  → クリップボードの内容がロジックによって置き換えられてしまう。

- バリアント配列の使用
  - 2次元のバリアント配列を作成。
  - 配列に順番に値をセット。
  - Excelのセルを範囲選択し、バリアント配列を1度にセット。

バリアント配列の使用

- StringGridの内容をExcelへ出力

```
procedure TfrmMain.ExcelOut2;
var
      ～（途中省略）～
  OleArray  : OleVariant;
begin
  OleArray := VarArrayCreate([0, sgGrid.RowCount - 1,
    0, sgGrid.ColCount - 1], varVariant);  //ﾊﾞﾘｱﾝﾄ配列作成
  try
      ～（途中省略）～
    for iRow := 0 to sgGrid.RowCount - 1 do
      for iCol := 0 to sgGrid.ColCount - 1 do
        //配列にデータ出力
        OleArray[iRow, iCol] := sgGrid.Cells[iCol, iRow];

    //エクセルに配列データを送る
    Excel.ActiveSheet.Range[Excel.ActiveSheet.Cells[1, 1],
      Excel.ActiveSheet.Cells[sgGrid.RowCount,sgGrid.ColCount]
      ].Value := OleArray;
      ～（途中省略）～
  finally
    VarClear(OleArray);       //配列を空にする
  end;
end;
```

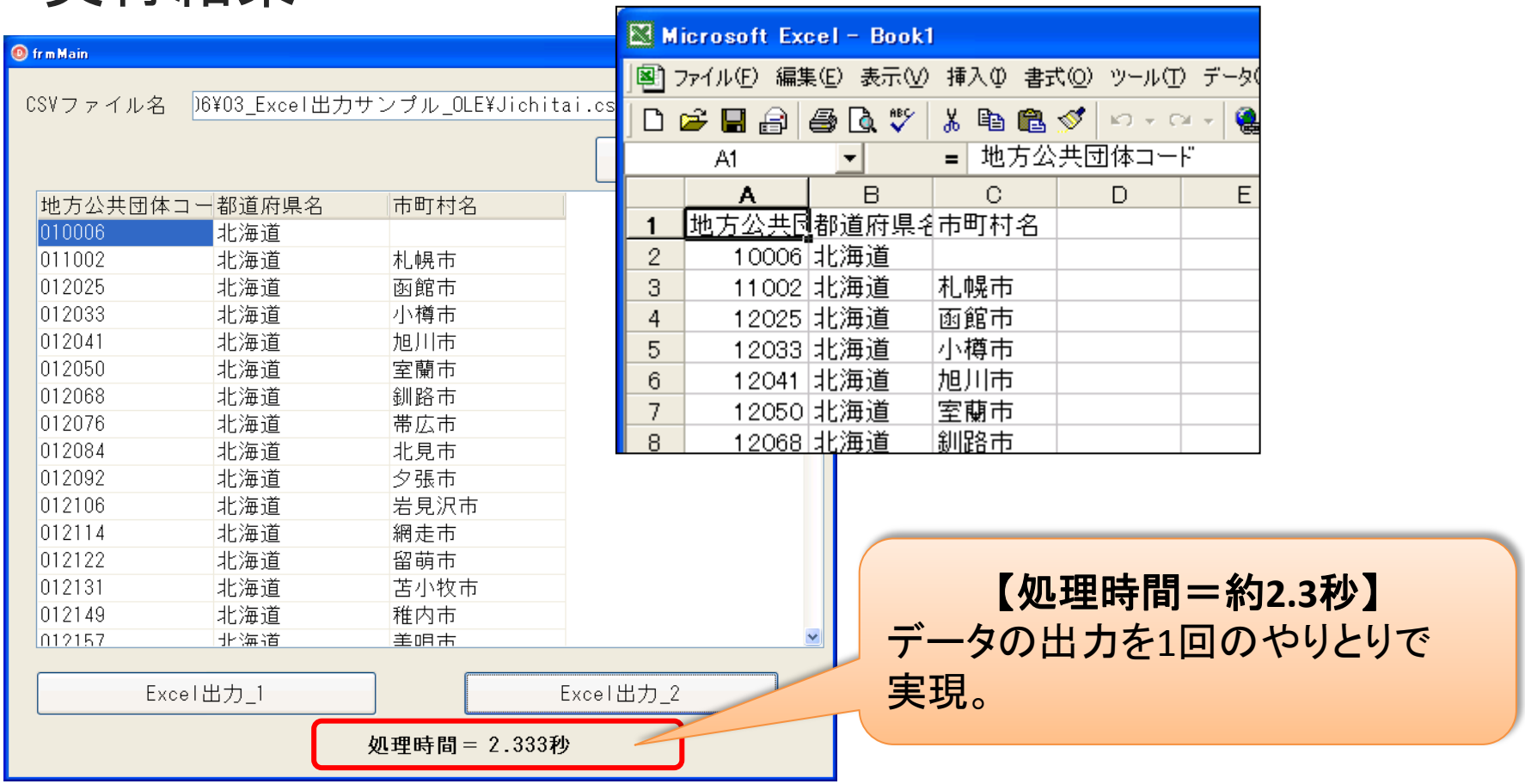
StringGridの行列分の2次元配列を作成し、変数OleArrayにセット。

配列に順番にStringGridのセル内容をセット。

StringGridの行列数分にあわせたExcelのセル範囲(Range)を指定し、配列の内容をセット。

■ バリアント配列を使用したデータ出力

- 実行結果



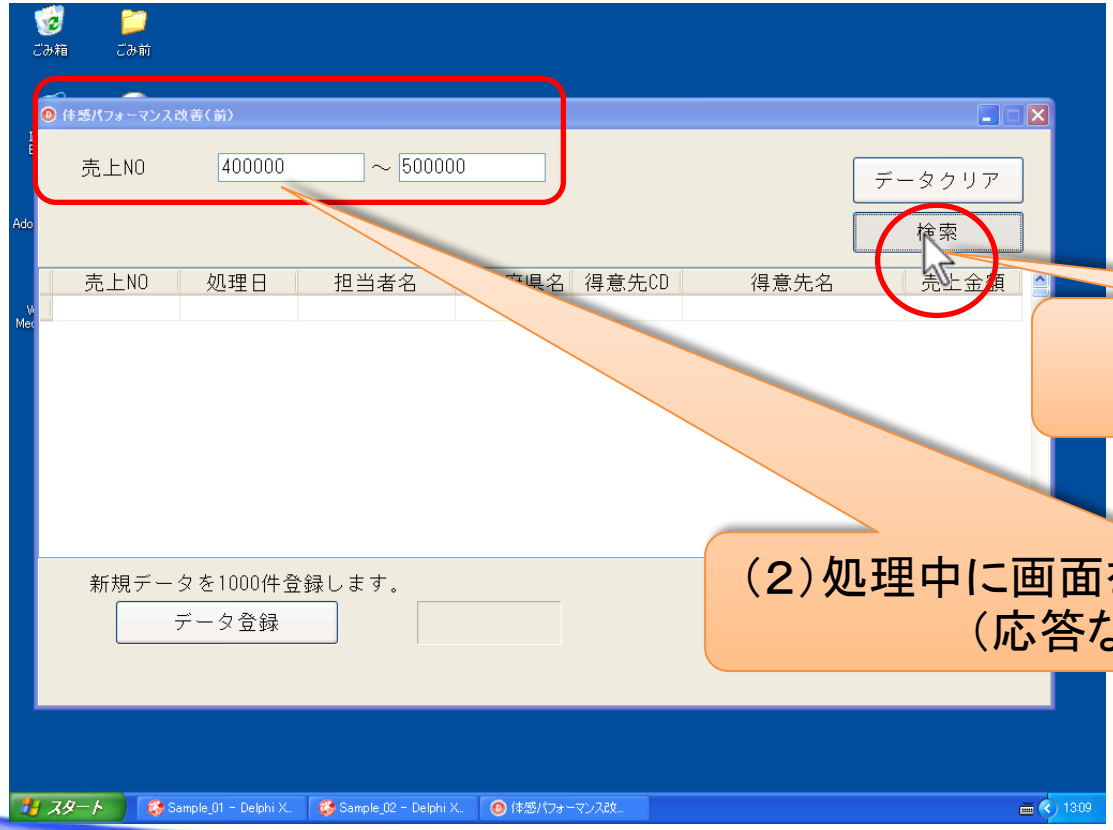
【処理時間＝約2.3秒】
データの出力を1回のやりとりで実現。

- OLEサーバとの通信回数を減らす事でパフォーマンス向上

4. 体感パフォーマンス向上テクニック

パフォーマンスにイライラする原因

- UI(ユーザーインターフェース)が不親切
  - 「実行」ボタンを押下しても、画面に変化がない。
  - 処理中に画面が固まってしまい、何も応答しない。



(1)ボタンを押下したか、
していないかが分からない。

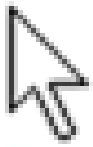
(2)処理中に画面を触ろうとすると、反応がなく、
(応答なし)と表示される。

マウスカーソルによる実行通知

- 処理中は、マウスカーソルを砂時計に変更する
  - Screen.Cursor にマウスカーソルのアイコン番号(定数)を指定。

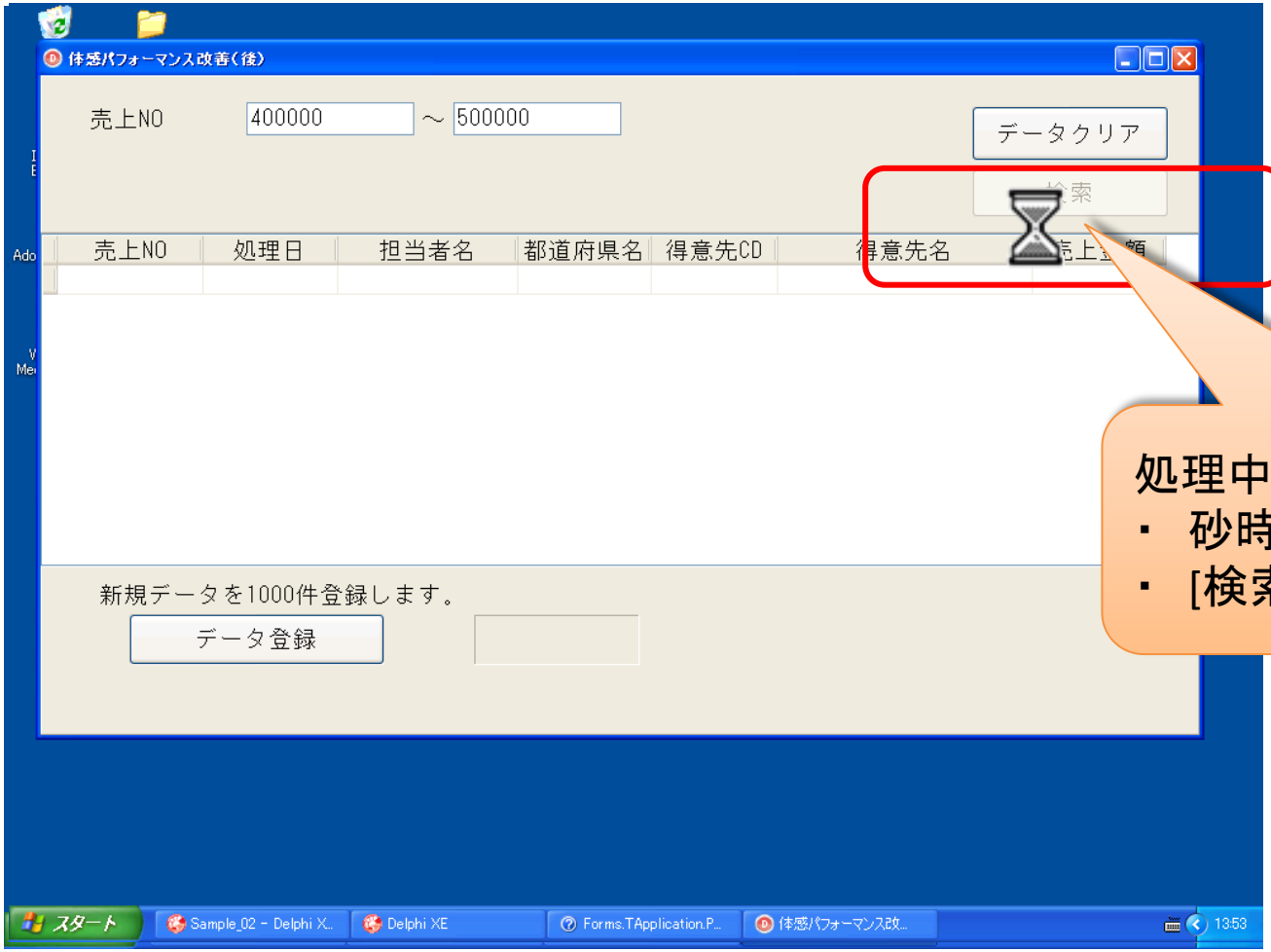
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //連続実行されないよう、ボタンを使用不可とする
  Button1.Enabled := False;
  //マウスカーソルを砂時計にかえる
  Screen.Cursor := crHourGlass;
  try

    ((( 時間のかかる処理 )))

  finally
    //実行中にたまったメッセージキューを処理する
    Application.ProcessMessages;
    //砂時計を元に戻す
    Screen.Cursor := crDefault;
    //ボタンを使用可能に戻す
    Button1.Enabled := True;
  end;
end;
```

この1行を入れることにより、
処理実行中に、ボタンの上で
連続クリックしても、再処理
されなくなる。
この行を入れないと、処理後、再度
Button1Clickが動いてしまう。



マウス制御後の動作

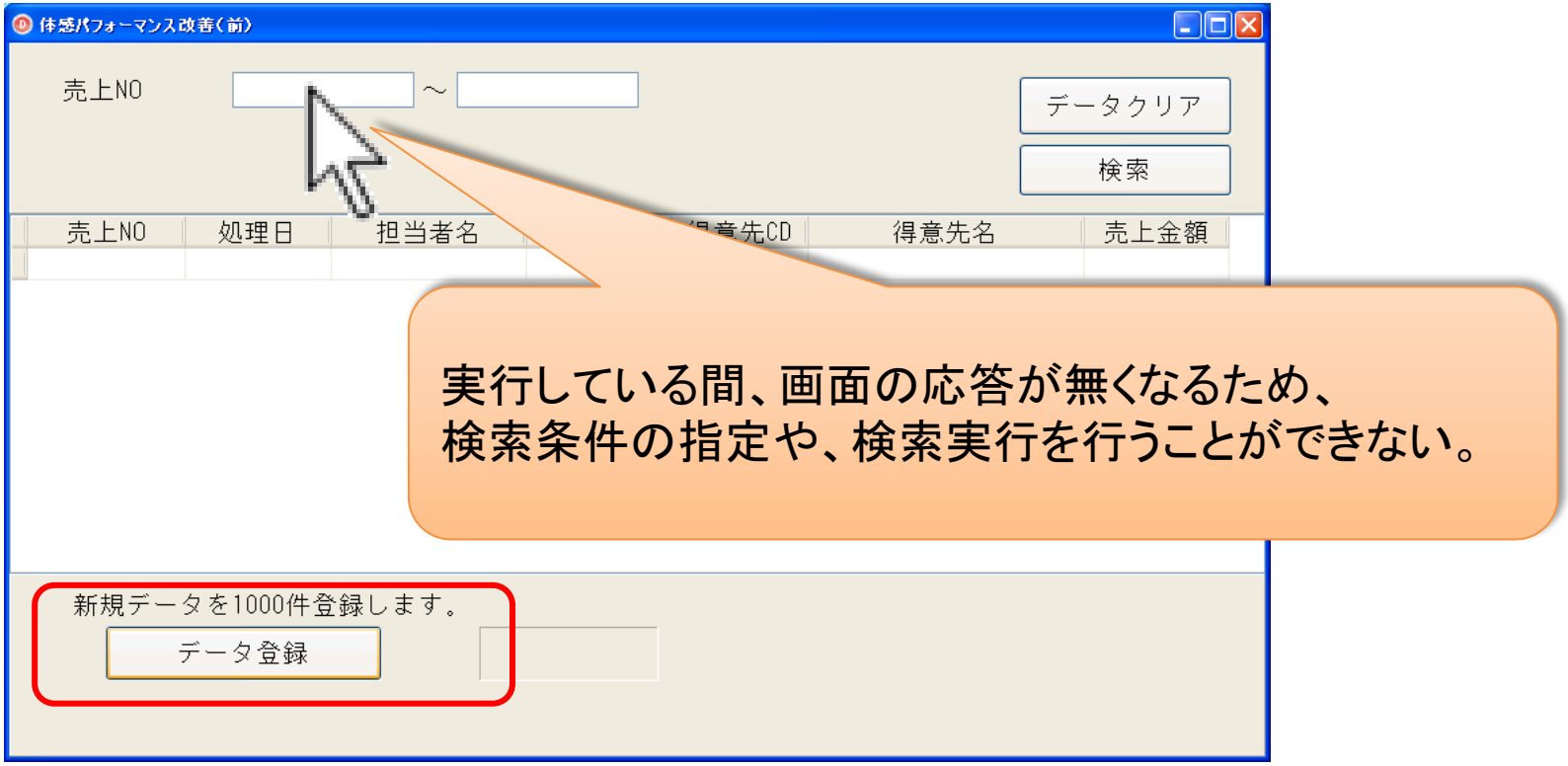
- マウスカーソルの変更により実行中をユーザーに通知。



処理中であることを以下で通知。
- 砂時計アイコン
- [検索]ボタン押下不可

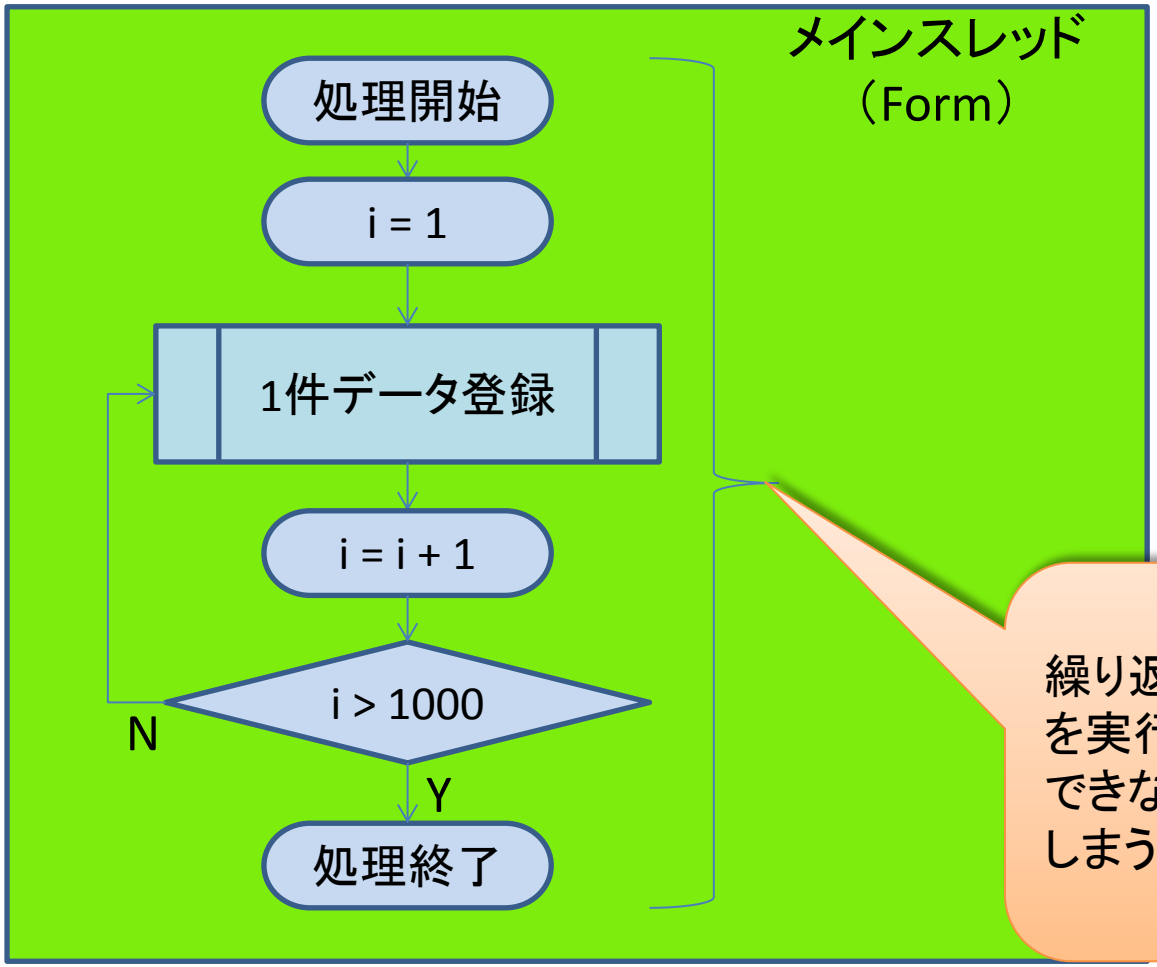
重い処理実行中の動作

- 時間のかかる処理を動かしている間、他の処理ができない。



シングルスレッドアプリケーション

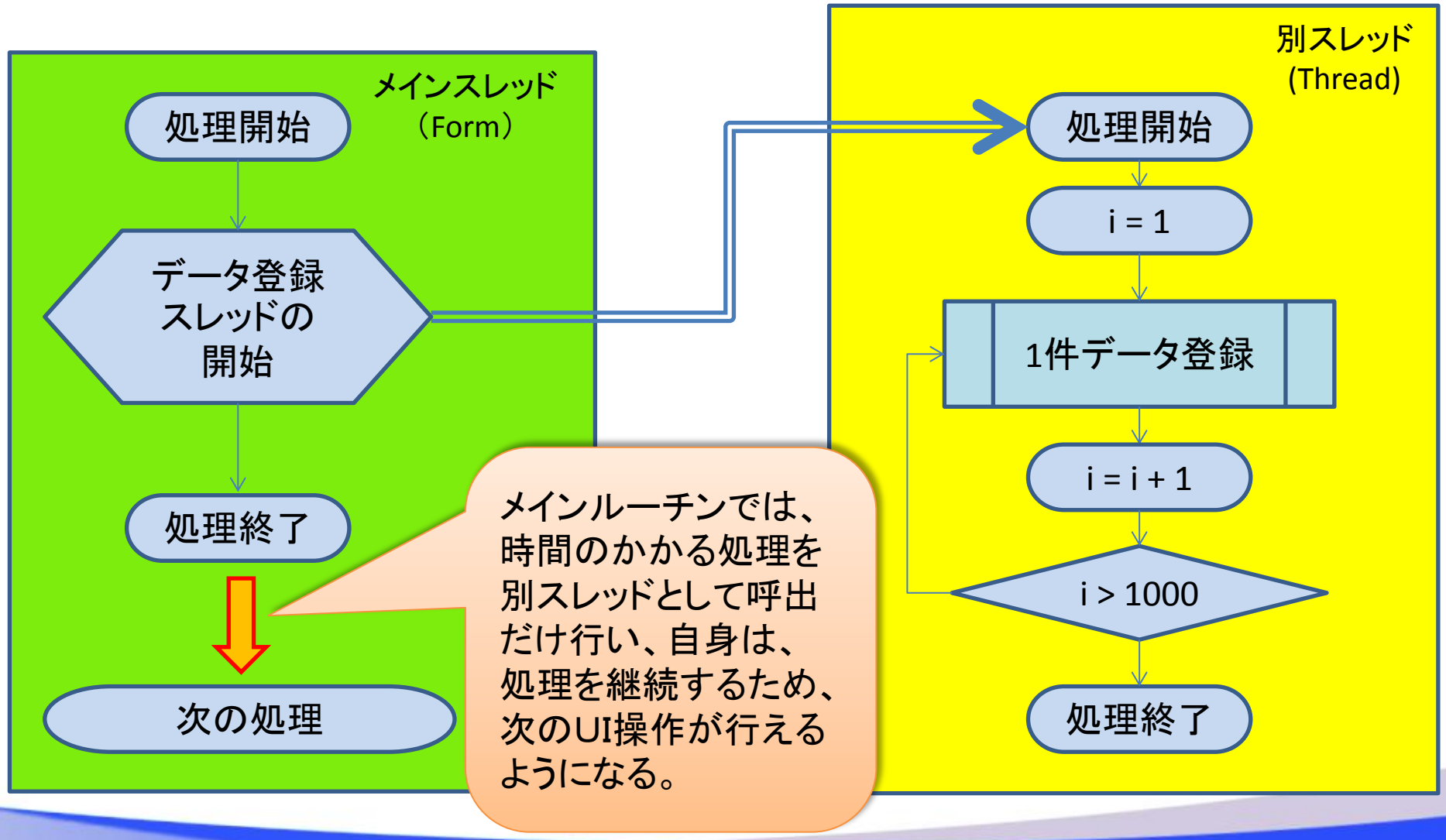
- 通常のアプリケーションは、シングルスレッド（逐次実行）である。



繰り返し処理等、時間がかかる処理を実行すると、他の処理が実行できないため、画面が固まってしまう。

マルチスレッドアプリケーション

- 時間がかかる処理を並列実行できるアプリケーション。



スレッドクラス（TThread）

- Delphi/400でマルチスレッドを行うには、TThreadを使用。

宣言部

```
type
  //データ登録用スレッド
  TDataEntryThread = class(TThread)
  private

    ((スレッド内で使用する変数や手続きを宣言))

  protected
    procedure Execute; override;
  public
    constructor Create(パラメータリスト); virtual;
  end;
```

TThreadを継承するクラスとして宣言する。

スレッドのメイン処理を記述。

スレッド生成（開始）時の処理を記述。
受取ったパラメータを元に初期処理を行う。

スレッドクラス（TThread）

実装部

```
constructor TDataEntryThread.Create(パラメータリスト);
begin
  //初期化
    ((( 変数の初期化等　を実施 )))

  //継承元のCreateを呼出（パラメータFalseで即時スレッド実行）
  inherited Create(False);

  //FreeOnTerminateプロパティをTrueにする。
  //(Trueにすると、スレッド終了時にスレッドオブジェクトが破棄される)
  FreeOnTerminate := True;
end;

------------------------------------------------------

procedure TDataEntryThread.Execute;
begin
  inherited;
  //繰り返しデータ登録を行う。
    ((( データ登録のメイン処理を記述　)))
end;
```

Create後、即時Executeメソッドを実行する。

スレッド終了時に自動的に破棄される。

メインスレッドから、別スレッドの実行

メインスレッド(Form)

```
procedure TfrmMain.btnDataEntryClick(Sender: TObject);
begin
    //登録処理のスレッドを生成する
    TDataEntryThread.Create(受け渡しパラメータ);

end;
```

スレッドクラスのオブジェクトを動的に生成すれば良い。
ボタンクリックイベントは、そのまま終了する。

- スレッドオブジェクトを生成すれば、別スレッドが実行可能

マルチスレッドアプリの注意点①

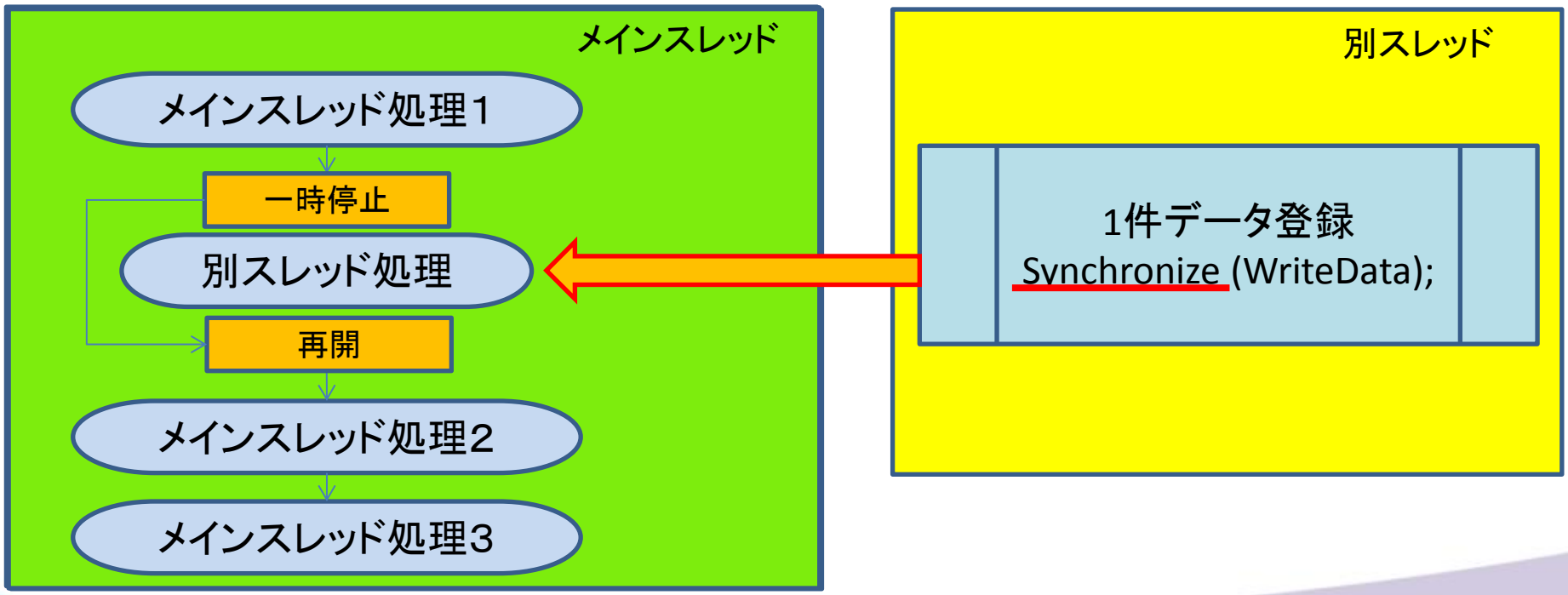
- スレッドのメイン処理（Execute）内では、いつでも処理が中断できるよう、繰り返し処理の中で、適宜Terminatedプロパティをチェックする。

```
procedure TDataEntryThread.Execute;
begin
  inherited;

  while (not Terminated) and (継続条件) do
  begin
     (( この中に、繰り返し処理を記述 ))
  end;
end;
```

Terminatedプロパティ = True となった場合は、いつでも処理が中断できるようにする。

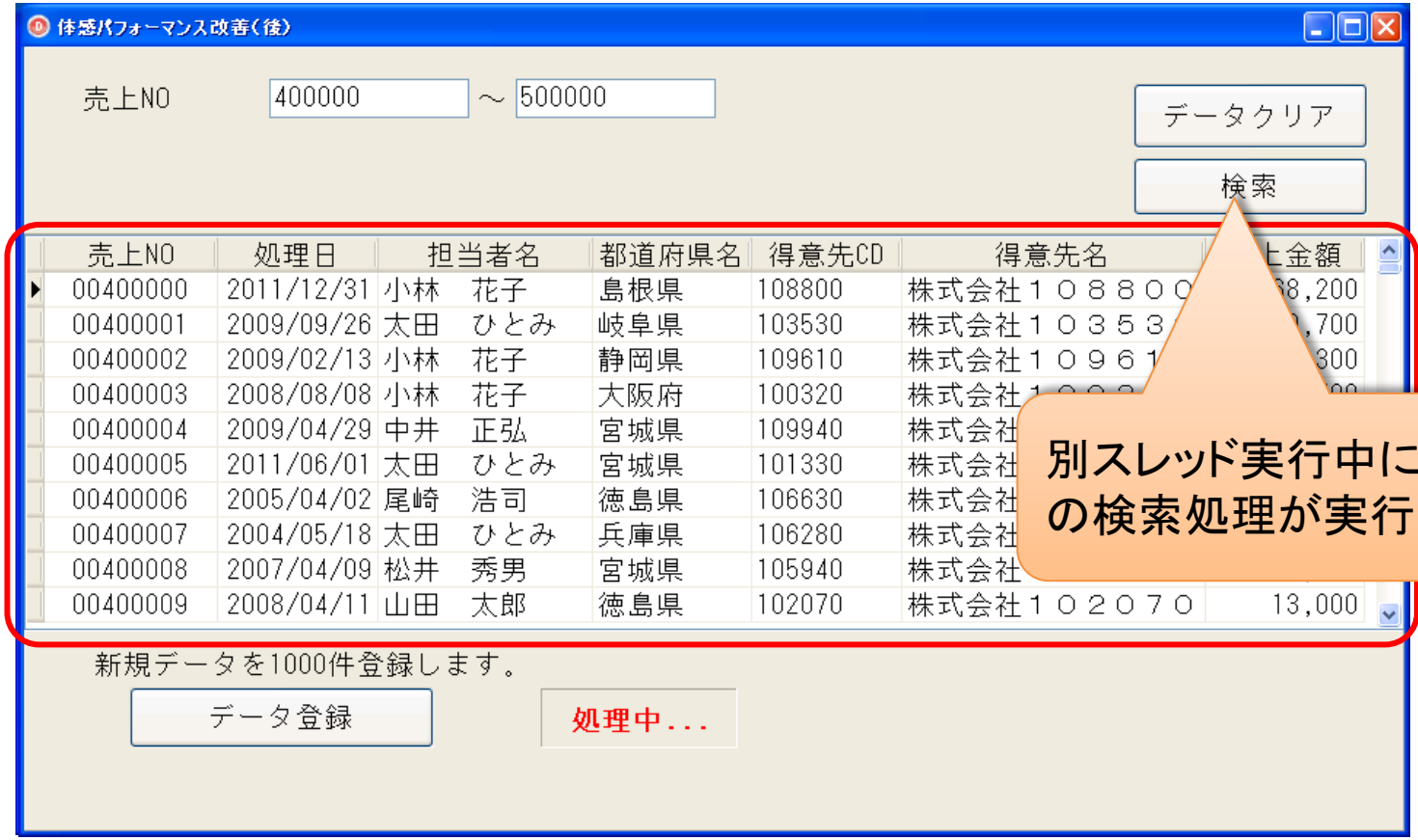
マルチスレッドアプリの注意点②

- **VCL(コンポーネント)が使用できるのは、メインスレッドのみ**である。
  別スレッド側からコンポーネントを使用する場合、メインスレッド側のサブルーチンを呼び出す必要がある。
- 別スレッド側からメインスレッドのコンポーネントにアクセスする場合、**Synchronize手続き**を使用し、メインスレッド側を一時停止した上で、メインスレッド側のサブルーチンを呼び出す。



マルチスレッド対応プログラム

- 時間のかかる処理を動かしている間も、別の処理が実行可能。



5. まとめ

まとめ

- データ抽出パフォーマンス向上テクニック
  - クエリーの使用
  - 論理ファイルの活用
  - クライアントデータセットを使用した事前キャッシュ
- Excel出力パフォーマンス向上テクニック
  - バリアント配列を使用した処理の高速化
- 体感パフォーマンス向上テクニック
  - マウスカーソル制御
  - マルチスレッドアプリケーションの実装

ご清聴ありがとうございました。