

【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。

The screenshot displays a complex business application interface with multiple panes:

- 物件別負荷状況一覧照会 (Project Load Status Overview):** A table showing project details (枝, 品名, ME, 担当者, 区分) and a calendar grid (日付) from 09/06/ to 09/06/. It includes status indicators like '構想・仕様' (Concept/Spec) and '先行手配' (Advance arrangement).
- frmMain (Main Form):** A financial summary for '年度2011 コード:1102 大阪本社営業第1課'. It shows a '採算表' (Profitability Statement) with columns for months (4月 to 12月) and rows for various cost and revenue items (e.g., 総売上高, 仕切型売上, 経費合計).
- 採算表通期合計 (Profitability Statement Summary):** A table comparing '予定' (Forecast) and '実績(確定済)' (Actual/Confirmed) values for various items, along with '達成率' (Achievement Rate).
- Comparison Chart:** A line graph at the bottom right comparing '予定' (Forecast, blue line) and '実績' (Actual, red line) over time, with a y-axis ranging from 24,000,000 to 32,000,000.

## <もう一つの大きな評価ポイント>

- 使用して満足していくパフォーマンスが得られること。

## ■ 満足のない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。

順位	自治体コード	財政力指数	自治体名
1	234273	2.32	
2	014036	1.85	
3	203211	1.58	
4	143821	1.57	
5	122271	1.56	
6	083411	1.56	
7	024112	1.55	
8	155047	1.49	
9	132039	1.48	
10	082325	1.45	
11	273627	1.4	
12	194255	1.39	
13	243442	1.38	
14	413879	1.38	
15	103667	1.37	
16	112241	1.35	
17	122114	1.35	
18	232220	1.35	
19	223425	1.33	

順位	自治体コード	財政力指数	自治体名
1	234273	2.32	愛知県 飛島村
2	014036	1.85	北海道 泊村
3	203211	1.58	長野県 軽井沢町
4	143821	1.57	神奈川県 箱根町
5	122271	1.56	千葉県 浦安市
6	083411	1.56	茨城県 東海村
7	024112	1.55	青森県 六ヶ所村
8	155047	1.49	新潟県 刈羽村
9	132039	1.48	東京都 武蔵野市

実行中、マウスも含め画面応答がない為、  
処理実行中かどうか分からない。  
処理終了時に、いきなり画面が応答する。

# ■ 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。

【レベル2】 処理中、ステータスが表示されるプログラム (応答なし)

順位	自治体コード	財政力指数	自治体名
1	234273	2.32	
2	014036	1.85	
3	203211	1.58	
4	143821	1.57	
5	122271	1.56	
6	083411	1.56	
7	024112	1.55	
8	155047	1.49	
9	132039	1.48	
10	082325	1.45	
11	273627	1.4	
12	194255	1.39	
13	243442	1.38	
14	413879	1.38	
15	103667	1.37	
16	112241	1.35	
17	122114	1.35	
18	232220	1.35	
19	223425	1.33	

自治体名取得  処理中...

【レベル2】 処理中、ステータスが表示されるプログラム

順位	自治体コード	財政力指数	自治体名
1	234273	2.32	愛知県 飛島村
2	014036	1.85	北海道 泊村
3	203211	1.58	長野県 軽井沢町
4	143821	1.57	神奈川県 箱根町
5	122271	1.56	千葉県 浦安市
6	083411	1.56	茨城県 東海村
7	024112	1.55	青森県 六ヶ所村
8	155047	1.49	新潟県 刈羽村
9	132039	1.48	東京都 武蔵野市

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。

【レベル3】 進捗状況が表示されるプログラム

順位	自治体コード	財政力指数	自治体名
1	234273	2.32	
2	014036	1.85	
3	203211	1.58	
4	143821	1.57	
5	122271	1.56	
6	083411	1.56	
7	024112	1.55	
8	155047	1.49	
9	132039	1.48	
10	082325	1.45	
11	273627	1.4	
12	194255	1.39	
13	243442	1.38	
14	413879	1.38	
15	103667	1.37	
16	112241	1.35	
17	122114	1.35	
18	232220	1.35	
19	223425	1.33	

自治体名取得

処理中...

【レベル3】 進捗状況が表示されるプログラム

順位	自治体コード	財政力指数	自治体名
1	234273	2.32	愛知県 飛島村
2	014036	1.85	北海道 泊村
3	203211	1.58	長野県 軽井沢町
4	143821	1.57	神奈川県 箱根町
5	122271	1.56	千葉県 浦安市
6	083411	1.56	茨城県 東海村
7	024112	1.55	青森県 六ヶ所村
8	155047	1.49	新潟県 刈羽村
9	132039	1.48	東京都 武蔵野市

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

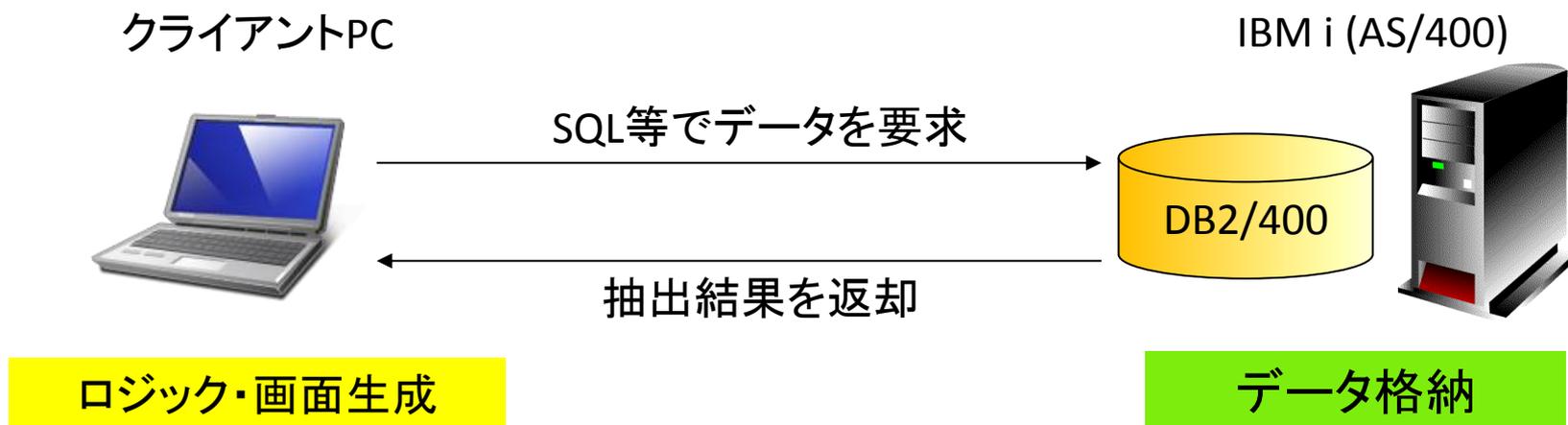
- 物理的な処理時間が短くなるような改善
  - IBMi(AS/400)上のデータを効率良く抽出するための手法
  - 事前キャッシュを利用した処理速度の向上
  - Excel出力の高速化
- 処理の待ち時間を感じさせない体感的な改善
  - マウスカーソル制御
  - マルチスレッド化による応答なし状態の回避

## 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。



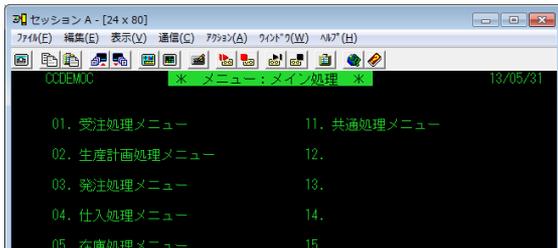
# RPGアプリとDelphi/400アプリとの比較



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

大量データであっても  
ホスト内で効率的に処理

ロジック  
RPG

DB2/400



画面ファイル  
Form

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

ロジック  
Delphi

抽出データが転送

DB2/400

Delphi/400 Exe

- 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

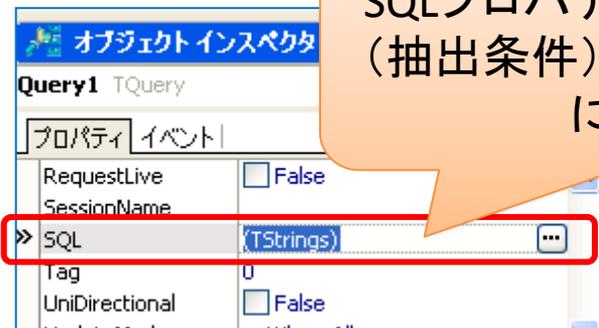
### • テーブル (TTable)



### • クエリー (TQuery)



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



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
-	URTNCN	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

オブジェクト インспекタ

Table1 TTable

プロパティ イベント

DefaultIndex  True  
Exclusive  False  
FieldDefs (TFieldDefs)  
Filter **URTFKN = '13'**  
Filtered  True  
FilterOptions []

Filterプロパティ

SQL

Query1:

文字列リストの設定

```
1行  
SELECT * FROM FURDTP  
WHERE URTFKN = '13' |
```

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。

テーブルとクエリー

テーブル

処理時間 = 2.554秒

開く

レコード移動

URDTNO	URTNC	URSYDT	URKNGK
23251	1400	20120526	
23273	1300	20120723	
23489	1900	20090520	
23516	1400	20121228	
23535	1200	20061001	
23603	1600	20090905	
23701	1900	20101216	
23731	1000	20060826	50800 13 10

クエリー

処理時間 = 0.691秒

開く

レコード移動

URDTNO	URTNC	URSYDT	URKNGK	URTFKN	UR
23149	1100	20110321		00800 13	10
23251	1400	20120526			
23273	1300	20120723			
23489	1900	20090520			
23516	1400	20121228			
23535	1200	20061001			
23603	1600	20090905			
23701	1900	20101216			
23731	1000	20060826	50800 13 10		

**【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.TDFKMN, 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(得意先コード)  
Key2 : URSYDT(処理日付)  
Key3 : URDTNO(売上NO)  
  
Select : URDFLTF (削除F)≠“1”

```
SELECT
  F.*, T.TATANM, K.TDFKMN, 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

【処理時間=約2.8秒】  
キーが適切でない為、  
処理に時間がかかる。

SQLパフォーマンス比較

得意先コード: 1005 (前方一致)

処理日付: 20070101 ~ 20070630 YYYYMMDD形式

検索方法:  物理ファイルを使用  論理ファイルを使用

処理日	担当者名	売上金額	売上NO	都道府県名	得意先CD	得意先名
2007/01/04	吉田 次郎	54,400	00115713	広島県	100500	株式会社100500
2007/01/08	吉田 次郎	62,900	00098557	徳島県	100500	株式会社100500
2007/01/13	中井 正弘	12,000	00450187	佐賀県	100500	株式会社100500
2007/01/30	山田 太郎	94,800	00190751	愛知県	100500	株式会社100500
2007/02/09	吉田 次郎	32,200	00441513	佐賀県	100500	株式会社100500
2007/02/20	前田 信二	23,300	00088408	静岡県	100500	株式会社100500

処理時間 = 2.824秒

【処理時間=約0.3秒】  
SQL文に論理ファイルを明示的に指定  
することで、処理時間が短縮

SQLパフォーマンス比較

得意先コード: 1005 (前方一致)

処理日付: 20070101 ~ 20070630 YYYYMMDD形式

検索方法:  物理ファイルを使用  論理ファイルを使用

処理日	担当者名	売上金額	売上NO	都道府県名	得意先CD	得意先名
2007/01/04	吉田 次郎	54,400	00115713	広島県	100500	株式会社100500
2007/01/08	吉田 次郎	62,900	00098557	徳島県	100500	株式会社100500
2007/01/13	中井 正弘	12,000	00450187	佐賀県	100500	株式会社100500
2007/01/30	山田 太郎	94,800	00190751	愛知県	100500	株式会社100500
2007/02/09	吉田 次郎	32,200	00441513	佐賀県	100500	株式会社100500
2007/02/20	前田 信二	23,300	00088408	静岡県	100500	株式会社100500

処理時間 = 0.270秒

## ■ 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)



frmMain

CSVファイル名 抽出パフォーマンス\_03¥JichitaiShisu.e

順位	自治体コード	財政力指数	自治体名
1	234273	2.32	
2	014036	1.85	
3	203211	1.58	
4	141221	1.57	
5	10171	1.50	

自治体コード(CHKTCD)  
自治体名称(CHKTNM)  
を保持

レコード数: 1,794

自治体コードが  
記されたCSVファイル

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

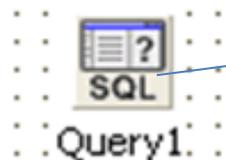
- 自治体コード1件ずつをクエリー(SQL)で問合せ。

```
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;  
end;
```

明細行毎にGetData関数を呼出して、名称を取得。

GetData関数

メイン処理



```
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;  
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

CSVファイル名

順位	自治体コード	財政力指数	自治体名
1	234273	2.32	愛知県 飛島村
2	014036	1.85	北海道 泊村
3	203211	1.58	長野県 軽井沢町
4	143821	1.57	神奈川県 箱根町
5	122271	1.56	千葉県 浦安市
6	083411	1.56	茨城県 東海村
7	024112	1.55	青森県 六ヶ所村
8	155047	1.49	新潟県 刈羽村
9	132039	1.48	東京都 武蔵野市
10	082325	1.45	茨城県 神栖市
11	273627	1.4	大阪府 田尻町
12	194255	1.39	山梨県 山中湖村
13	243442	1.38	三重県 川越町
14	413879	1.38	佐賀県 玄海町
15	103667	1.37	群馬県 上野村
16	112241	1.35	埼玉県 戸田市

自治体名 取得 (TQueryを使用してSQLで取得)

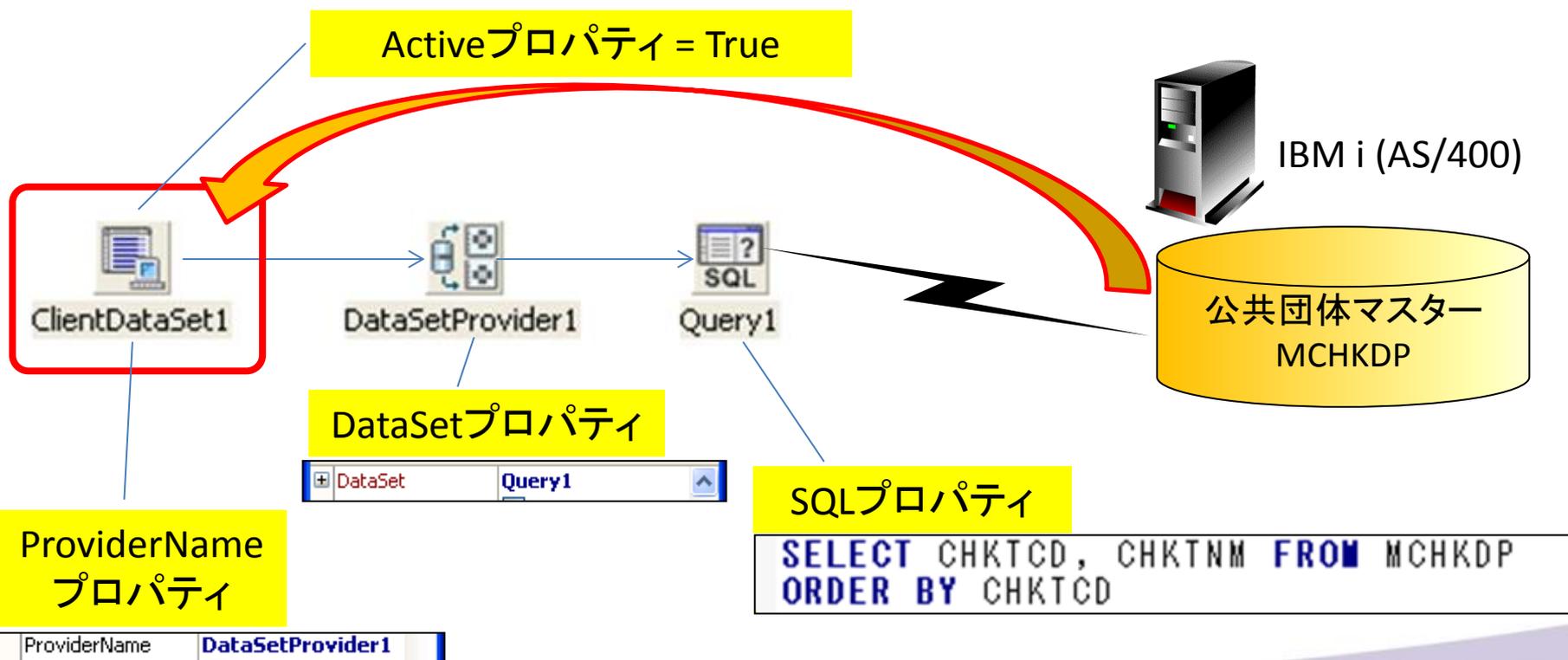
処理時間 = 10.615秒

**【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、処理に時間か  
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('CHKTCOD', 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

CSVファイル名

順位	自治体コード	財政力指数	自治体名
1	234273	2.32	愛知県 飛島村
2	014036	1.85	北海道 泊村
3	203211	1.58	長野県 軽井沢町
4	143821	1.57	神奈川県 箱根町
5	122271	1.56	千葉県 浦安市
6	083411	1.56	茨城県 東海村
7	024112	1.55	青森県 六ヶ所村
8	155047	1.49	新潟県 刈羽村
9	132039	1.48	東京都 武蔵野市
10	082325	1.45	茨城県 神栖市
11	273627	1.4	大阪府 田尻町
12	194255	1.39	山梨県 山中湖村
13	243442	1.38	三重県 川越町
14	413879	1.38	佐賀県 玄海町
15	103667	1.37	群馬県 上野村
16	112241	1.35	埼玉県 戸田市

自治体名 取得 (ClientDataを使用して取得)

**処理時間 = 0.321秒**

**【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。

IBM i (AS/400)

クライアントPC

Excel



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

エクセル(Excel.Application)

ブック(Excel.WorkBooks)

シート(WorkBook.WorkSheets)

セル(Sheet.Cells)

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にデータを出す
  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

CSVファイル名: 06¥03\_Excel出力サンプル\_OLE¥Jich

地方公共団体コード	都道府県名	市町村名
010006	北海道	
011002	北海道	札幌市
012025	北海道	函館市
012033	北海道	小樽市
012041	北海道	旭川市
012050	北海道	室蘭市
012068	北海道	釧路市
012076	北海道	帯広市
012084	北海道	北見市
012092	北海道	夕張市
012106	北海道	岩見沢市
012114	北海道	網走市
012122	北海道	留萌市
012131	北海道	苫小牧市
012149	北海道	稚内市
012157	北海道	美瑛市

Excel出力\_1      Excel出力\_2

処理時間 = 10.656秒

Microsoft Excel - Book1

A1	= 地方公共団体コード				
	A	B	C	D	E
1	地方公共	都道府県名	市町村名		
2	10006	北海道			
3	11002	北海道	札幌市		
4	12025	北海道	函館市		
5	12033	北海道	小樽市		
6	12041	北海道	旭川市		
7	12050	北海道	室蘭市		
8	12068	北海道	釧路市		

**【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。

## • バリエーション配列の使用

- 2次元のバリエーション配列を作成。
- 配列に順番に値をセット。
- Excelのセルを範囲選択し、バリエーション配列を1度にセット。

## ■ バリエーション配列の使用

### • 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); //バリエーション配列作成  
    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;  
    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。

# ■ バリエーション配列を使用したデータ出力

## ● 実行結果

CSVファイル名: 06¥03\_Excel出力サンプル\_OLE¥Jichitai.cs

地方公共団体コード	都道府県名	市町村名
010006	北海道	
011002	北海道	札幌市
012025	北海道	函館市
012033	北海道	小樽市
012041	北海道	旭川市
012050	北海道	室蘭市
012068	北海道	釧路市
012076	北海道	帯広市
012084	北海道	北見市
012092	北海道	夕張市
012106	北海道	岩見沢市
012114	北海道	網走市
012122	北海道	留萌市
012131	北海道	苫小牧市
012149	北海道	稚内市
012157	北海道	美瑛市

Excel出力\_1      Excel出力\_2

処理時間 = 2.333秒

Microsoft Excel - Book1

	A	B	C	D	E
1	地方公共団体コード	都道府県名	市町村名		
2	10006	北海道			
3	11002	北海道	札幌市		
4	12025	北海道	函館市		
5	12033	北海道	小樽市		
6	12041	北海道	旭川市		
7	12050	北海道	室蘭市		
8	12068	北海道	釧路市		

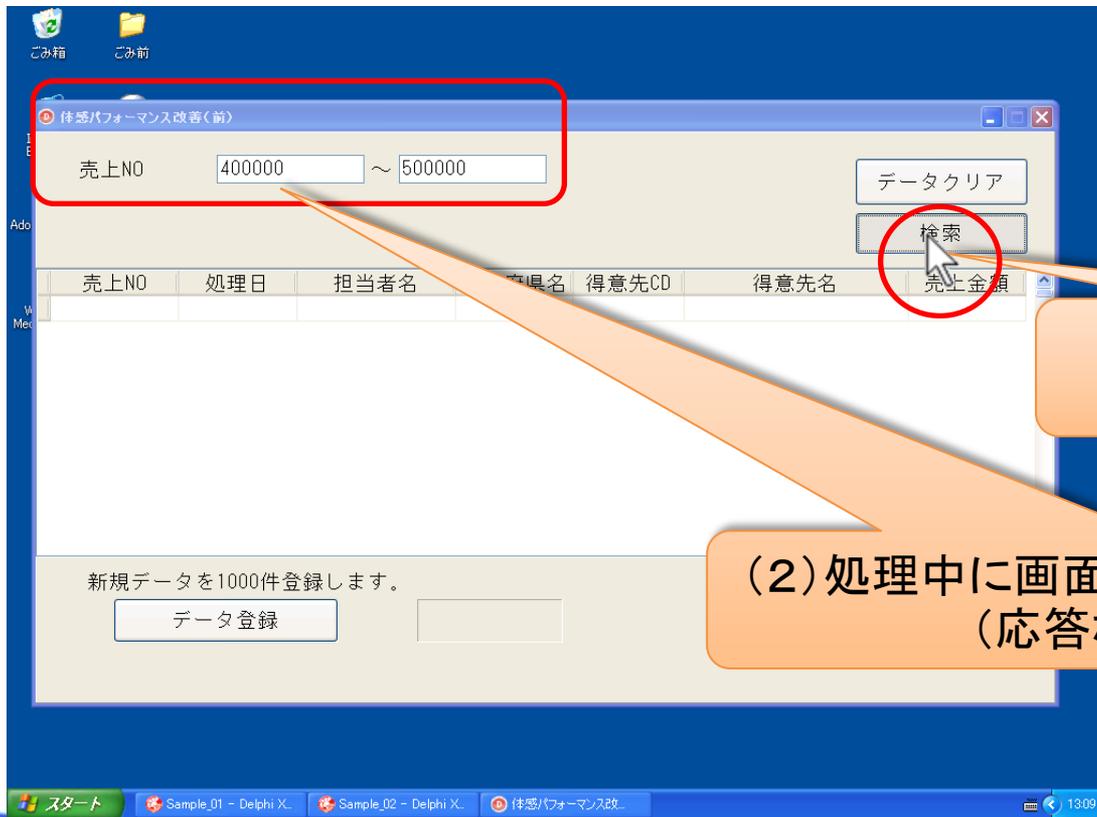
**【処理時間＝約2.3秒】**  
データの出力を1回のやりとりで実現。

- 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。

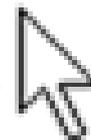
## ■ マウスカーソルによる実行通知

- 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;
```

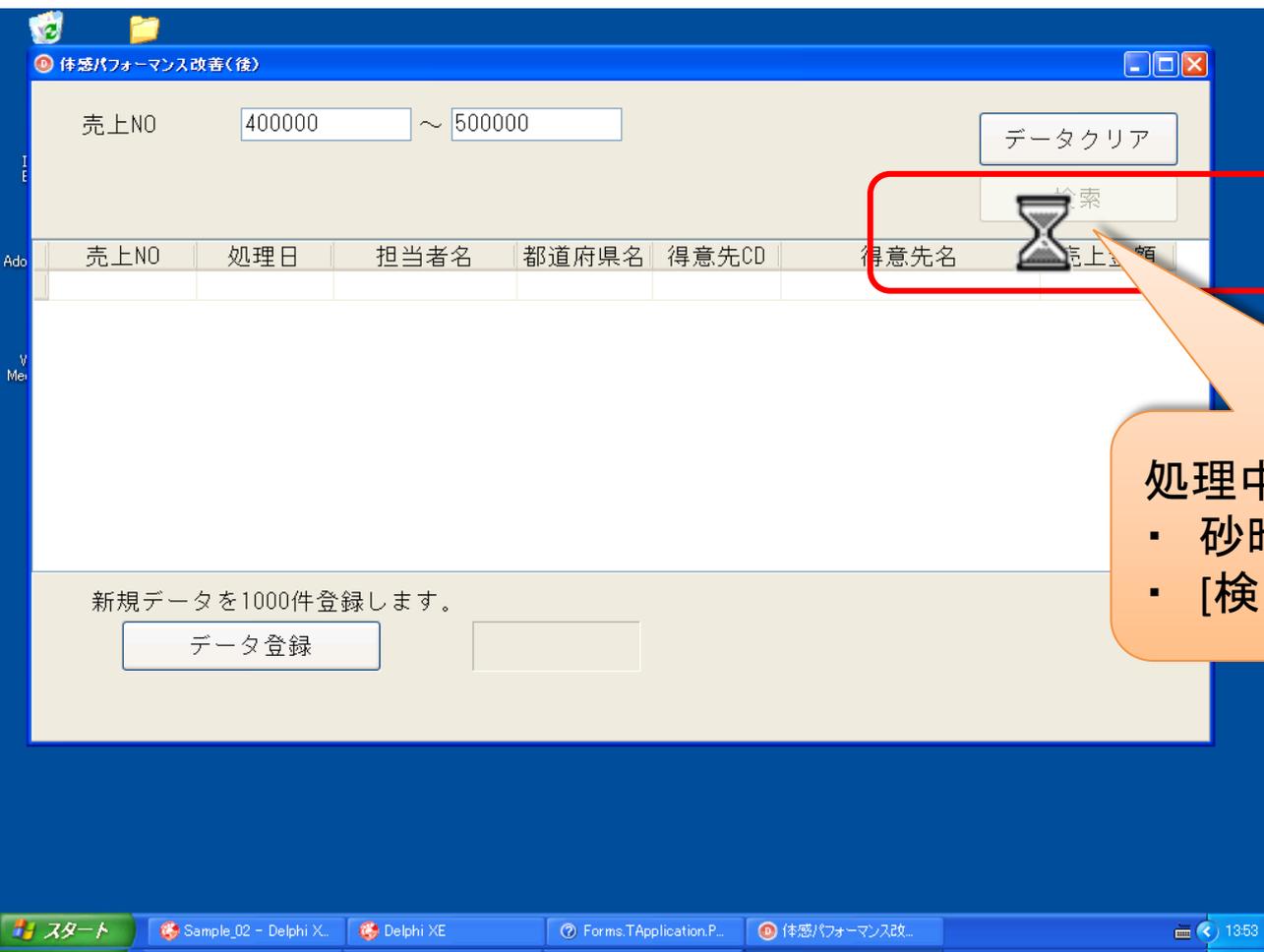


この1行を入れることにより、処理実行中に、ボタンの上で連続クリックしても、再処理されなくなる。  
この行を入れないと、処理後、再度 Button1Click が動いてしまう。



## ■ 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。

体感パフォーマンス改善(前)

売上NO  ~

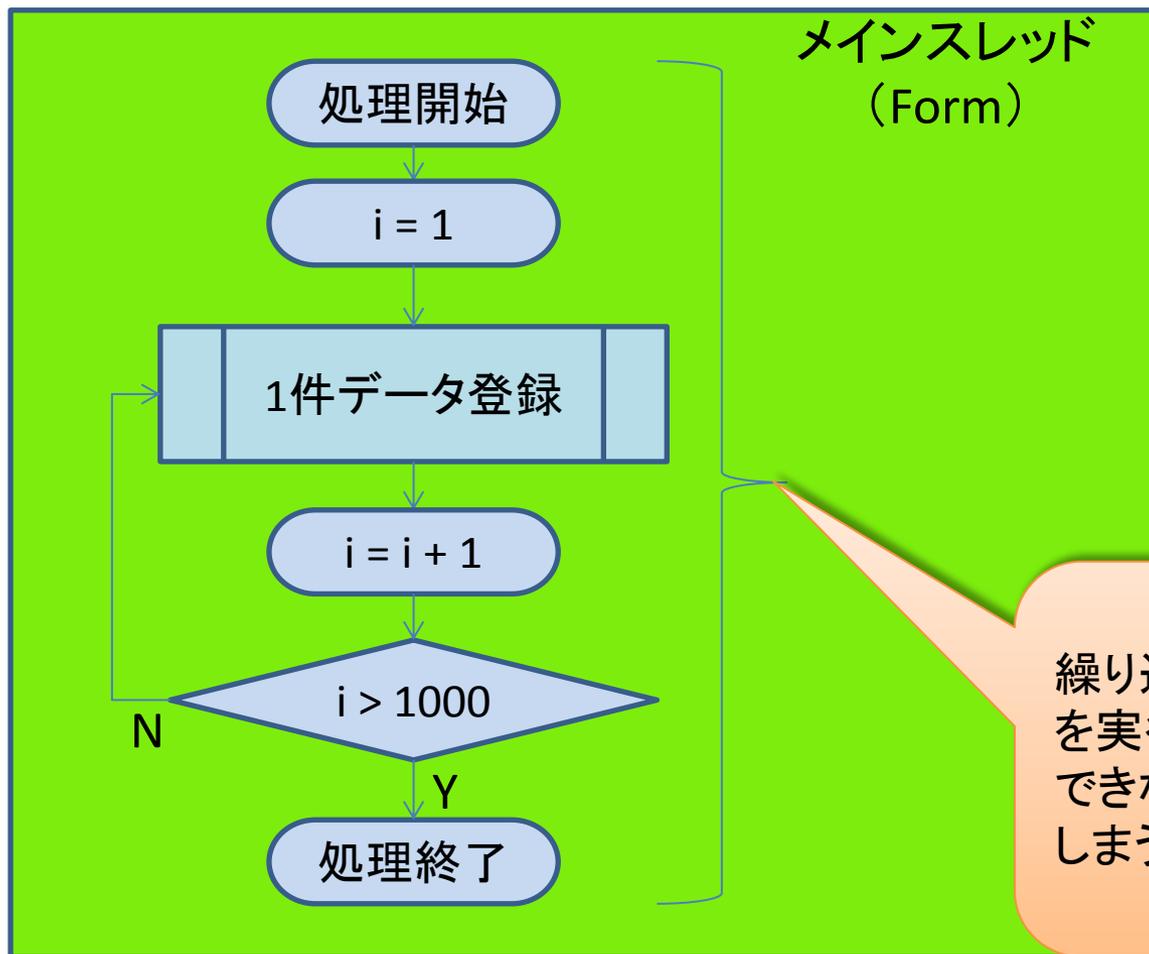
売上NO	処理日	担当者名	得意先CD	得意先名	売上金額
------	-----	------	-------	------	------

実行している間、画面の応答が無くなるため、  
検索条件の指定や、検索実行を行うことができない。

新規データを1000件登録します。

## ■ シングルスレッドアプリケーション

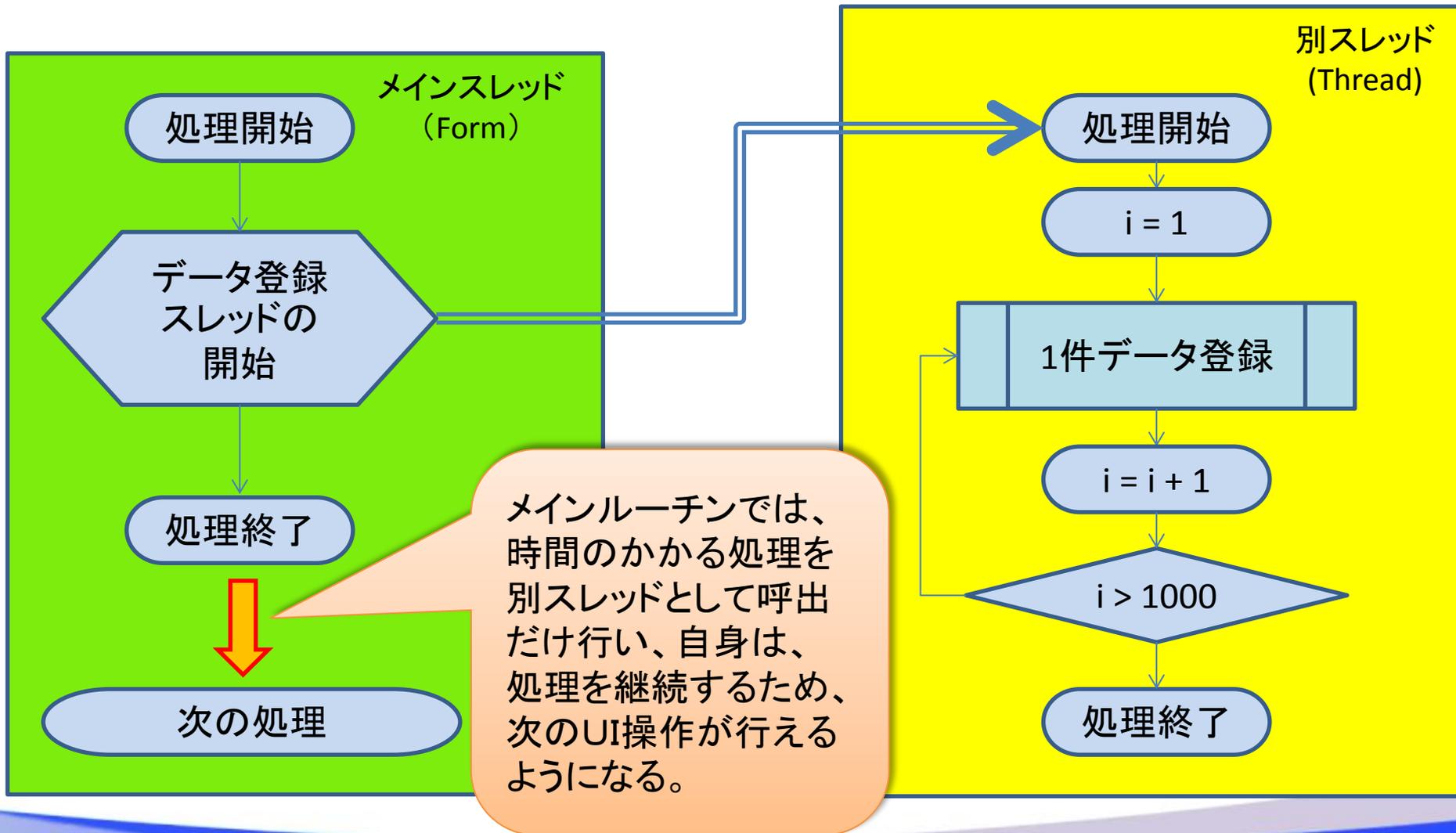
- 通常のアプリケーションは、シングルスレッド(逐次実行)である。



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

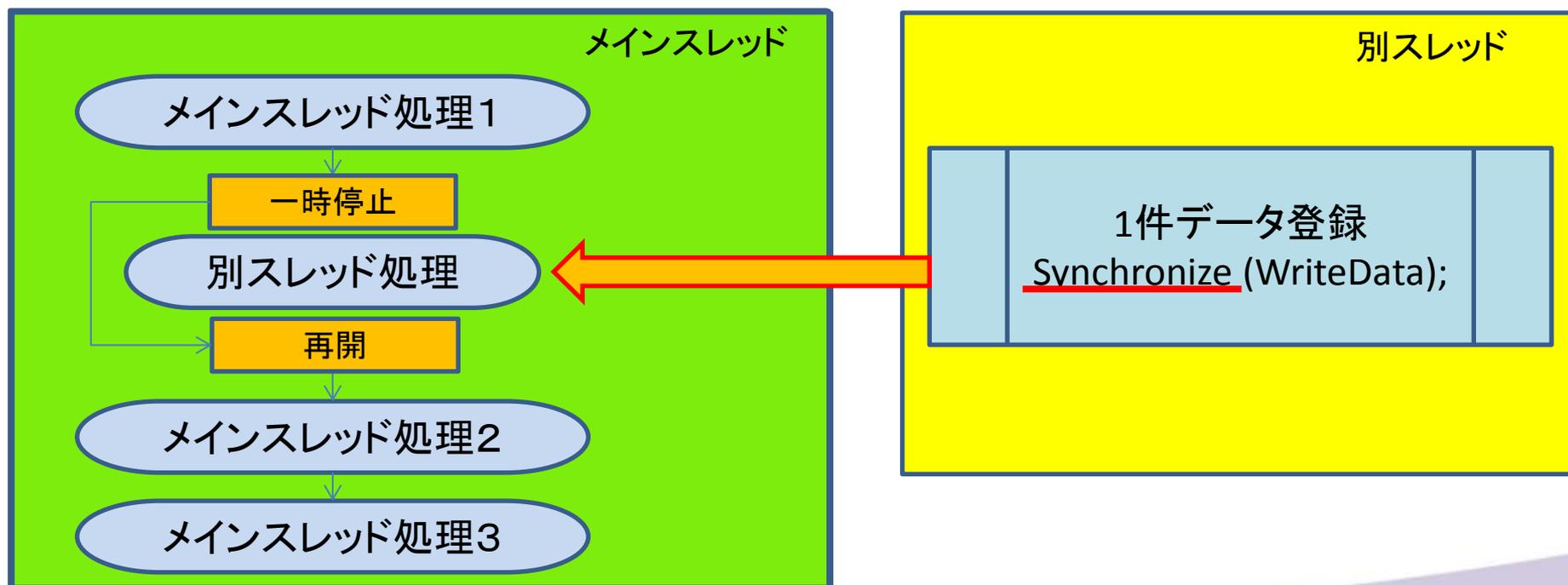
- スレッドのメイン処理 (Execute) 内では、いつでも処理が中断できるように、繰り返し処理の中で、適宜 Terminated プロパティをチェックする。

```
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。

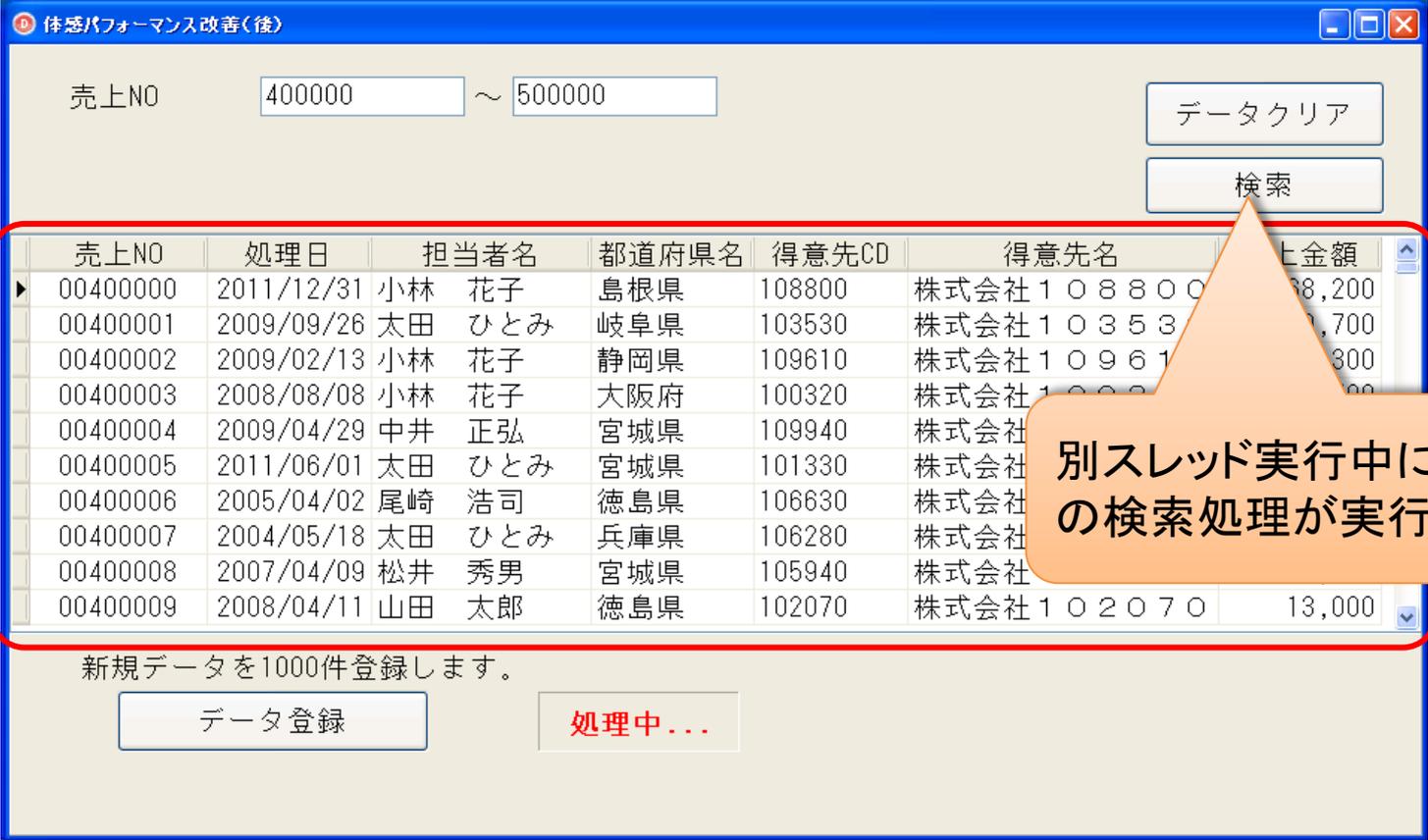
## ■ マルチスレッドアプリの注意点②

- **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。



体感パフォーマンス改善(後)

売上NO  ~

データクリア

検索

売上NO	処理日	担当者名	都道府県名	得意先CD	得意先名	売上金額
00400000	2011/12/31	小林 花子	島根県	108800	株式会社108800	8,200
00400001	2009/09/26	太田 ひとみ	岐阜県	103530	株式会社103530	1,700
00400002	2009/02/13	小林 花子	静岡県	109610	株式会社109610	300
00400003	2008/08/08	小林 花子	大阪府	100320	株式会社100320	00
00400004	2009/04/29	中井 正弘	宮城県	109940	株式会社	
00400005	2011/06/01	太田 ひとみ	宮城県	101330	株式会社	
00400006	2005/04/02	尾崎 浩司	徳島県	106630	株式会社	
00400007	2004/05/18	太田 ひとみ	兵庫県	106280	株式会社	
00400008	2007/04/09	松井 秀男	宮城県	105940	株式会社	
00400009	2008/04/11	山田 太郎	徳島県	102070	株式会社102070	13,000

新規データを1000件登録します。

データ登録

処理中...

別スレッド実行中に、メインスレッドの検索処理が実行。

# 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
  - バリエーション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。