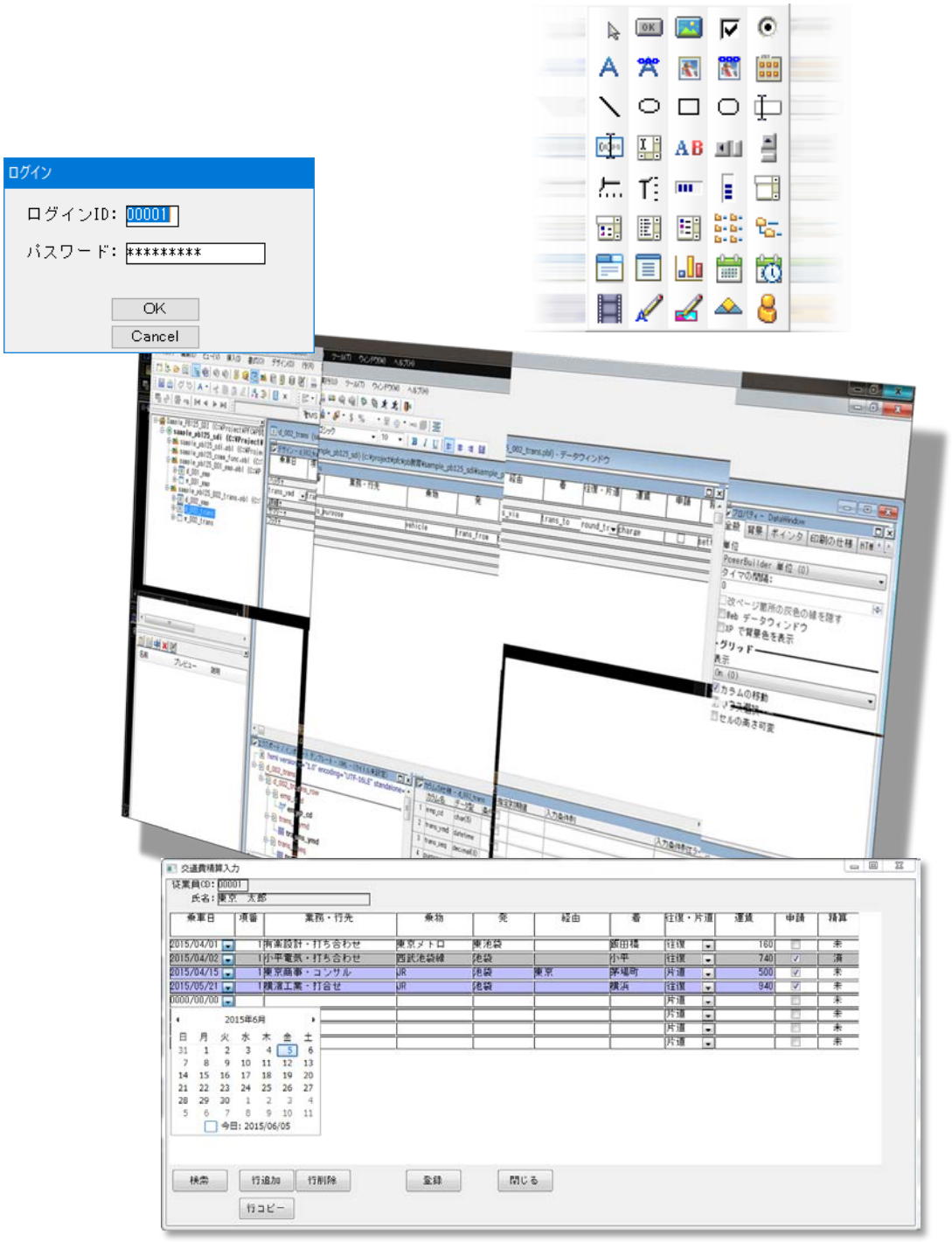
Appeon PowerBuilder 2017 入門コース



パワーフューチャー株式会社

目次

はじめに

本コースは、これから Appeon PowerBuilder 2017 の開発や運用保守を行う技術者向けに、入門レベル以上の達成度を目標に下記３ステップにより構成している。

ステップ１では、Appeon PowerBuilder 2017 を用いてプログラミングを行うために、最低限の知識として必要となるソース構成について説明する。
ソース構成を知らずしてプログラムを組み立てることはできないため、Appeon PowerBuilder 2017 が初めての場合は必読とする。

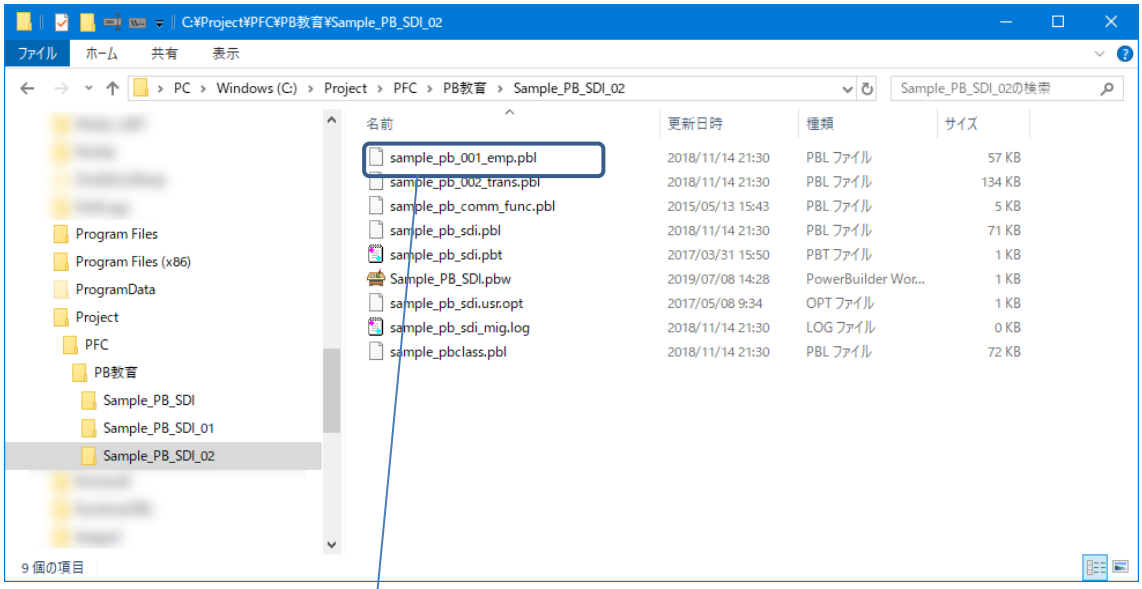
ステップ２では、データウインドウのリッチなユーザインタフェースを理解し、システム設計・構築を行う上で、優位性と効率的な視点に立てる解説を行う。

ステップ３では、Appeon PowerBuilder 2017 アプリケーションの作成方法を説明する。
IDE の操作に力点を置かず、実際にプログラムを作りながら操作方法を交えた解説をする。
本コースでは、Appeon PowerBuilder 2017 のクラスライブラリ機能を使用せずに、DB 操作アプリケーションの構築方法に注力する。

1．　PowerBuilder ソースの構成

（1）　ＰＢＬファイル

Appeon PowerBuilder 2017 のソースは、PBL ファイル（バイナリー）で PC 上に保存される。

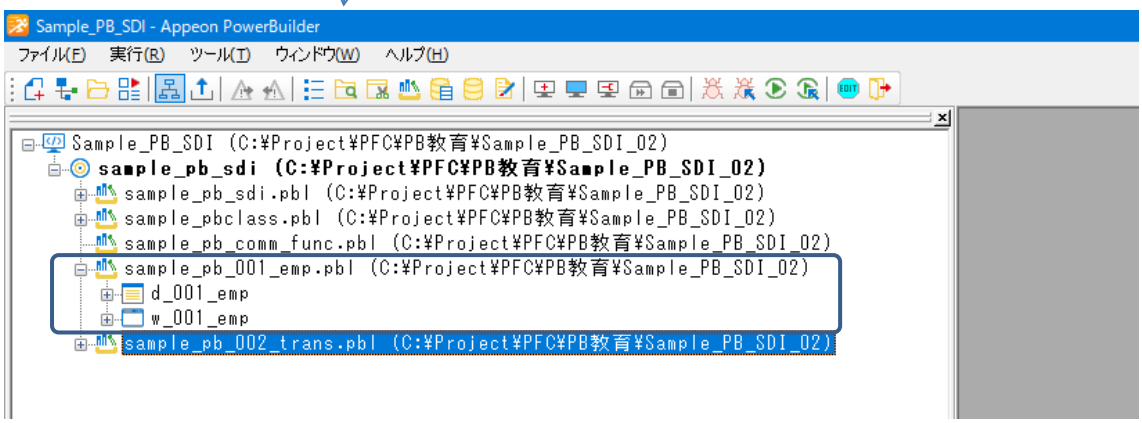


（2）　オブジェクトファイル

PBL ファイルの中には、ウインドウやデータウインドウ、関数など、アプリケーション構築に必要となるオブジェクトが登録され、Appeon PowerBuilder 2017 IDE によりソースの編集が可能になる。

各オブジェクトは、共通機能や処理画面ごとに PBL ファイルにまとめることで管理しやすくなる。

また、ソース管理ツール（VSS）を用いた開発においても、PBL 単位で役割分担がしやすくなるメリットがある。



（3） オブジェクトファイルの概要

本書では、イメージし易いように SDI（シングルドキュメントインタフェース）アプリケーションを前提に記載する。

MDI（マルチプルドキュメントインタフェース）は Office 製品にようにフレーム内で複数のウインドウが操作でき、メニュー項目を選択して操作する構成のため、説明範囲が広がり難解になるため割愛する。

① アプリケーション

アプリケーションを定義する非表示オブジェクトである。

アプリケーションの開始と終了の処理、後述するグローバル変数やグローバル外部関数の宣言を行う。

② トランザクション

Appeon PowerBuilder 2017 アプリケーション内で DB 接続、DB 操作内容を保持する非表示オブジェクトである。

デフォルトでは SQLCA トランザクションが提供されている。

複数のトランザクションが必要な際には、別名で Create して使用することができる。

その際には、ソースのハードコーディングが必要で、事前にユーザオブジェクト（後述）として作成することもできる。

CONNECT によりトランザクションが開始され、COMMIT で DB へ書込み情報を確定させ、ROLLBACK で書込み情報を破棄、DISCONNECT でトランザクションを終了する。

※重要

CONNECT 文の実行における負荷は、DB ベンダー製品によって異なるが、DB 接続が完了するまでに要する時間は大きい。

最善策としては、環境にもよるがアプリケーションの最初に 1 回だけ発行する。（但し、将来的な Web 化を前提に、画面ごとでも OK）

③ 関数

特定のプログラムを部品化し、共通関数などとして作成することができる。

④ 構造体

関数の引数や、画面間の引数用に利用できる。

WinAPI 関数（DLL）を利用する際の構造体定義もここで行う。

①⑥⑦のオブジェクト内で構造体を定義できるが、将来廃止予定のため、オブジェクト内に作成することは推奨しない。

⑤ データウインドウ

Appeon PowerBuilder 2017 の特許オブジェクトであり、DB アクセスを中心に、さまざまな表現、アクション操作が実現できる。

※「2．データウインドウ」で詳細を説明

⑥　ユーザオブジェクト

ビジュアル：　　画面上の共通コントロールをデザインできる。
（コントロールの集合体としてデザインが可能）

ノンビジュアル：トランザクションや、共通関数群として作成することができる。

⑦　ウインドウ

ユーザインタフェースを提供する主要なオブジェクトである。

データの表示、ユーザのマウス操作、キーボード操作など、様々なイベント駆動に対応した動作を実現できる。

ウインドウの種類

- メインウインドウ
- チャイルドウインドウ
- ポップアップウインドウ
- レスポンスウインドウ

ウインドウに配置できるコントロール

- コマンドボタン
- ピクチャーボタン
- チェックボックス
- ラジオボタン
- スタティックテキスト
- ピクチャー
- 直線、楕円、長方形、丸長方形
- シングルラインエディット
- エディットマスク
- マルチラインエディット
- リッチテキストエディット
- ドロップダウンリストボックス
- ドロップダウンピクチャーリストボックス
- リストボックス
- ピクチャーリストボックス
- リストビュー
- ツリーリストビュー
- タブコントロール
- データウインドウ
- グラフ
- など・・・

ウインドウにユーザインタフェース機能を搭載させるのに、上記の様々なコントロールが配置できる。

そのコントロールの一つであるデータウインドウには、ウインドウ上に配置できるコントロールと同等で、しかも DB と密接に連携したコントロール（正式にはｄｗｏ）が提供できる。

DB 連携ができていることと、ウインドウ上のコントロール数を削減できることからも、ユーザインタフェースの作成には、データウインドウを中心に設計することを推奨する。

コントロールの種類と数が減ることで、スクリプトを記載する場所が限定でき、かつ、不具合発生率の低減、メンテナンス性の向上が期待できる。

上記ｄｗｏのコントロールがあれば、ユーザインタフェースはほぼ網羅でき、チェックボックス、ラジオボタン、ドロップダウンなどは、データウインドウでリッチなユーザインタフェースとして提供することができる。

これまでの開発実績から、必要に応じてｄｗｏがあれば十分なユーザインタフェースを提供することが可能である。

⑧ メニュー

MDI アプリケーション用のため、本書では取り上げない。

（4） オブジェクトへのプログラミング

① イベント（関数、構造体、データウインドウにはない）

ウインドウでコマンドボタンの押下や、ウインドウ上の入力フィールドでの入力など、ユーザ操作により駆動するイベントにスクリプトを記載し、アプリケーションを構築する。

基本的なイベントはデフォルトで用意されており、必要に応じて別途ユーザ定義イベントを追加することができる。

例）

・Clicked	ボタン押下（クリック）
・Open	ウインドウが開くとき
・Close	ウインドウが閉じるとき
・CloseQuery	ウインドウを閉じる直前 （閉じない設定が可能）
・Doubleclicked	ダブルクリック操作
・ItemChanged	データウインドウで入力変更

② オブジェクト内関数（関数、構造体、データウインドウにはない）

ウインドウなど、オブジェクト内で関数を定義できる。

また、関数名が同一で、引数とスクリプトが異なる、関数のオーバーロードが可能であり、柔軟な開発ができるメリットがある。

例えば変数の IsNull チェックを、データ型を意識せずに判定することができる。

データ型	判定１	判定２
String	Is Null	=””
Decimal	Is Null	なし
DateTime	Is Null	=DateTime(Date(”1900/01/01”), Time(”00:00:00”))

③ 関数（関数のみ）

関数として独立して作成でき、共通関数として利用することが多い。

④ データウインドウのソース（データウインドウのみ）

SQL SELECT 文を記載し、SELECT 句に指定されたカラムがデザイン画面に展開される。

デザイン画面に配置された部品（カラムなど）のプロパティに式を記載することができ、カラムの値によって表示上の制御、色変え、入力操作可否など、リッチなデザインが簡単に実現できる。

（５） 変数の宣言

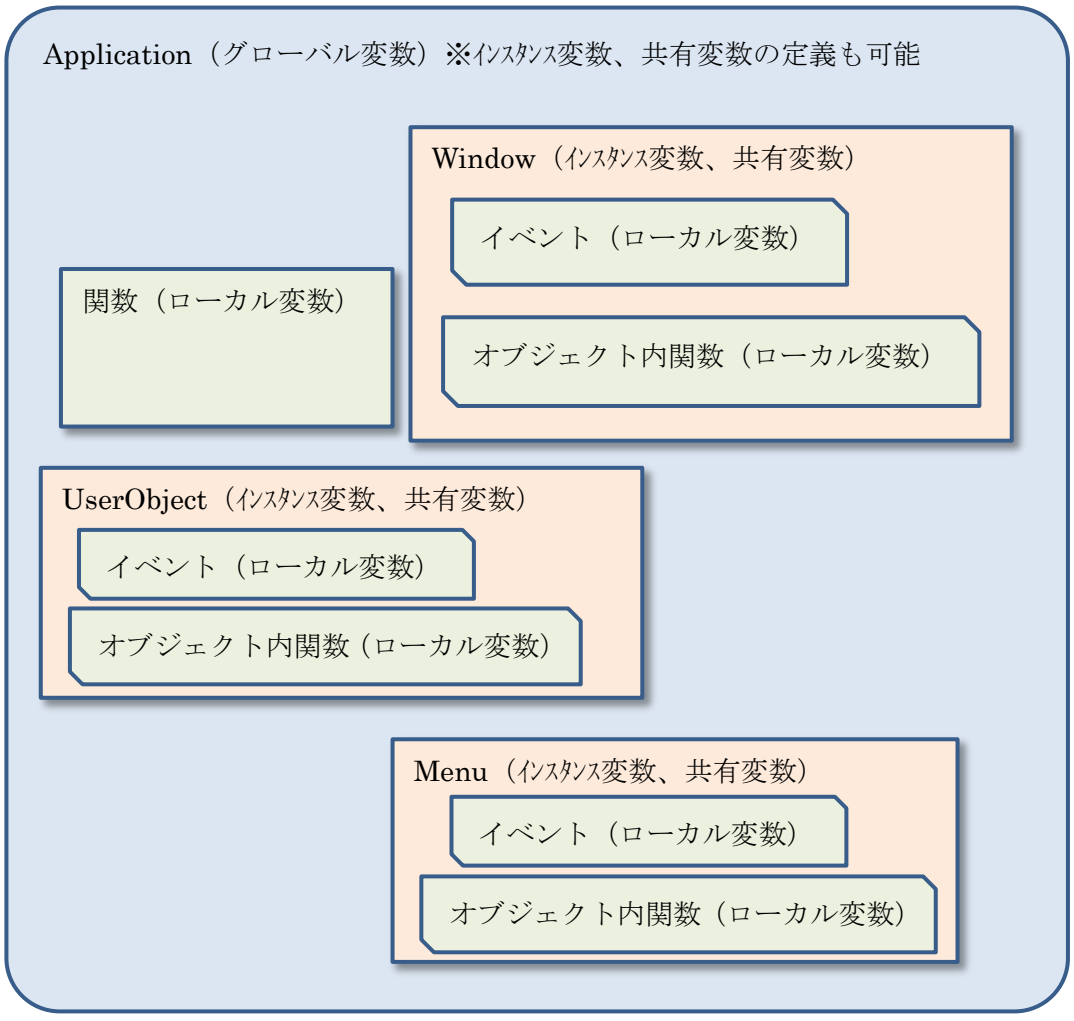
変数にはスコープの範囲がある。

① グローバル変数
アプリケーション内のどのスクリプトからでも利用できる。

② 共有変数
オブジェクト内で利用できる変数で、インスタンスを閉じて再度開いたときに値が保持されている。

③ インスタンス変数
オブジェクト内のスクリプト、その子孫のスクリプトで使用できる。

④ ローカル変数
ローカルの関数、イベント内でのみ利用できる。



（６） 外部関数の宣言

WindowsAPI など、外部 DLL の関数を利用することができる。

スコープの範囲を持っており、グローバルまたはローカル（オブジェクト内）で利用可能になる。

（７） 命名規約

アプリケーション開発時では、命名規約を定め、それに従い実施することを推奨する。
先頭文字の代表例を以下に掲載する。

① オブジェクト名

A) 関数オブジェクト	f_
B) 構造体オブジェクト	str_
C) データウインドウオブジェクト	d_
D) ユーザオブジェクト	uo_
E) ウインドウオブジェクト	w_
他割愛	

② 変数名

A)

グローバル変数（String）	gs_
グローバル変数（Integer）	gi_
グローバル変数（Long）	gl_
グローバル変数（Decimal）	gdc_
グローバル変数（DateTime）	gdt_
グローバル変数（Date）	gd_
グローバル変数（Time）	gt_
グローバル変数（Boolean）	gb_
グローバル変数（Transaction）	gt_
グローバル変数（構造体）	gstr_
グローバル変数（Datawindow）	gdw_
グローバル変数（DataStore）	gs_
グローバル変数（UserObject）	guo_
グローバル変数（Window）	gw_
他割愛	

B)

インスタンス変数（String）	is_
インスタンス変数（Integer）	ii_
インスタンス変数（Long）	il_
インスタンス変数（Decimal）	idc_
インスタンス変数（DateTime）	idt_
インスタンス変数（Date）	id_
インスタンス変数（Time）	it_
他割愛	

C)

共有変数（String）	ss_
共有変数（Integer）	si_
共有変数（Long）	sl_

共有変数（Decimal）	sdc_
共有変数（DateTime）	sdt_
共有変数（Date）	sd_
共有変数（Time）	st_
他割愛	

D) ローカル変数（String） ls_
ローカル変数（Integer） li_
ローカル変数（Long） ll_
ローカル変数（Decimal） ldc_
ローカル変数（DateTime） ldt_
ローカル変数（Date） ld_
ローカル変数（Time） lt_
他割愛

D) 引数（String） as_
引数（Integer） ai_
引数（Long） al_
引数（Decimal） adc_
引数（DateTime） adt_
引数（Date） ad_
引数（Time） at_
他割愛

③ オブジェクト内のイベント名や関数

A) アプリケーション（ユーザ定義イベント） ue_
アプリケーション（関数） af_

B) ユーザオブジェクト（ユーザ定義イベント） ue_
ユーザオブジェクト（関数） uf_

C) ウインドウ（ユーザ定義イベント） ue_
ウインドウ（関数） wf_

2．　データウインドウ

（1）　データウインドウとは

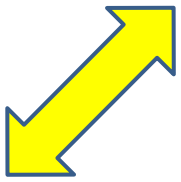
SQL SELECT の結果集合カラムを用いて、フリーフォームや一覧形式のユーザインタフェースを提供することができる。
更新特性を持たせることにより、データの追加・変更・削除が可能になり、ソースに SQL 文を記載しなくてもエントリー画面が簡単に作成することができる。

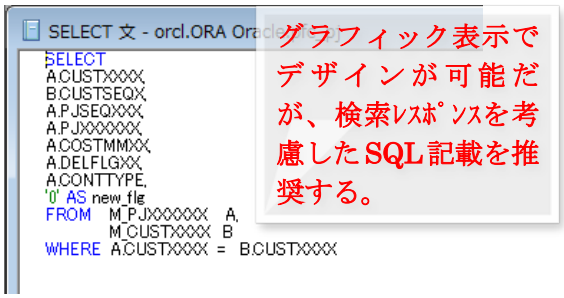
また、SQL SELECT の結果集合を帳票イメージにデザインでき、印刷プロパティを持っていることからも、柔軟かつ効率よくアプリケーションを作成することができる。

SQL 文を記載する画面「データソース」と、デザインを行う画面「データウインドウペインタ」を自由に行き来しながら開発することができる。

★

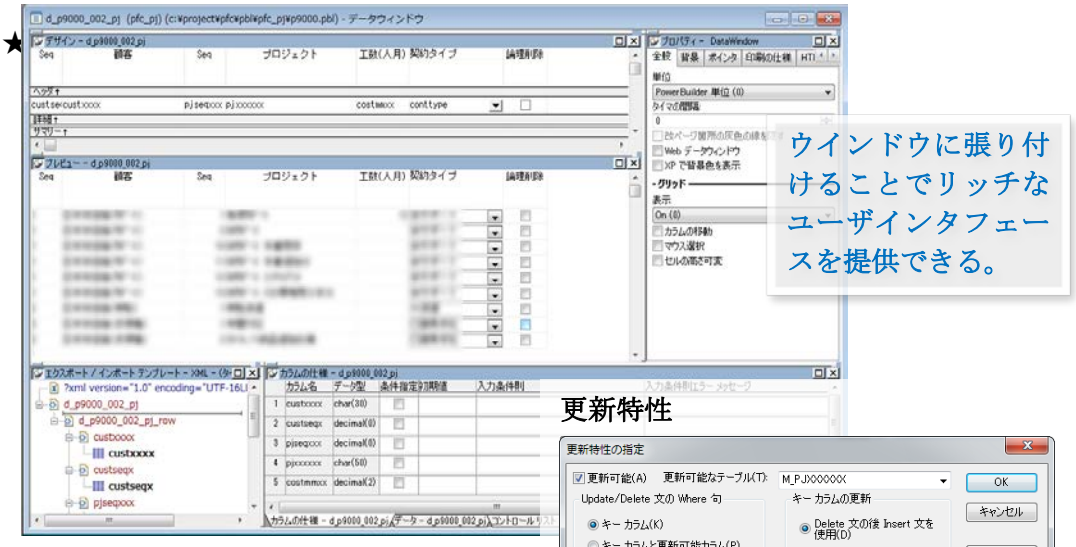
データソース（SQL 文記載画面）



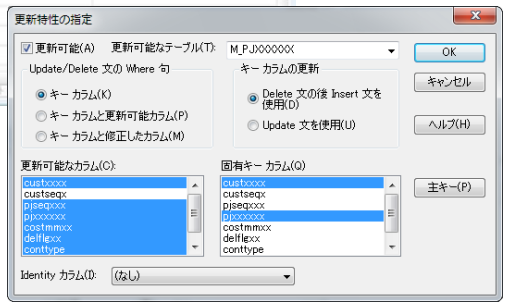


データウインドウペインタ（デザイン画面）

★



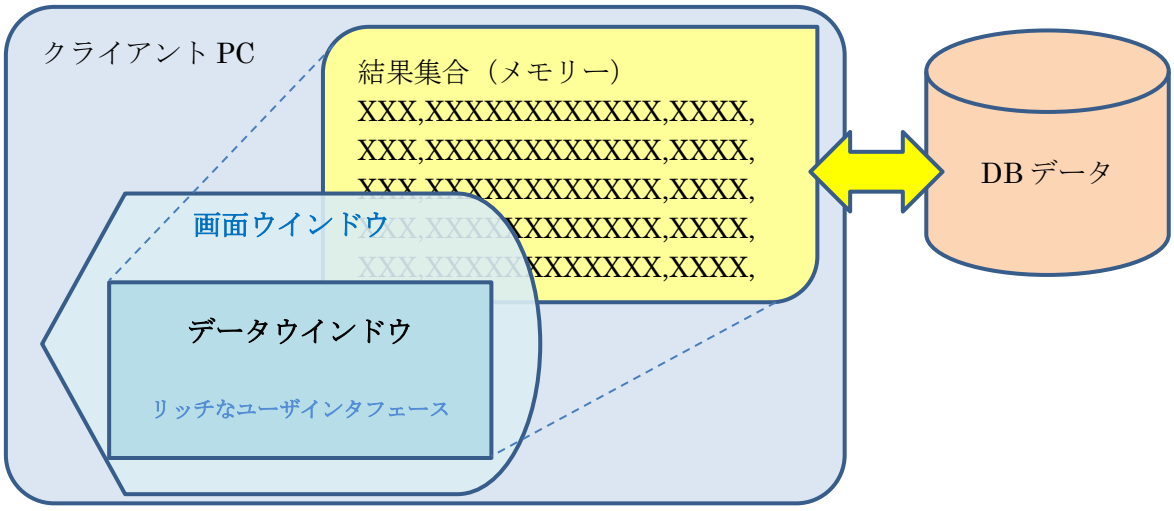
更新特性



★

（２）　データウインドウの特徴

DB から検索した結果集合はクライアント PC のメモリー上で管理され、高いレスポンシビリティでリッチなユーザインタフェースを提供することができる。



データウインドウのメモリー管理にはユーザが操作した情報を保持する領域があり、更新可能なデータウインドウでは、その情報を用いて DB サーバに SQL 文の自動送信を行っている。

① データソース
Current　：データウインドウコントロール内の現在のデータ
Original　：DB から最初に取得されたデータ

検索直後は、Current ＝ Original
入力操作で、Current ≠ Original

② バッファ
Primary　：行削除またはフィルタで除外されていないデータ
Delete　：行削除されたデータ
Filter　：フィルタで除外されたデータ

データウインドウには Update 関数があり、上記①②の状態を判断して DB サーバに SQL 文を自動送信する。

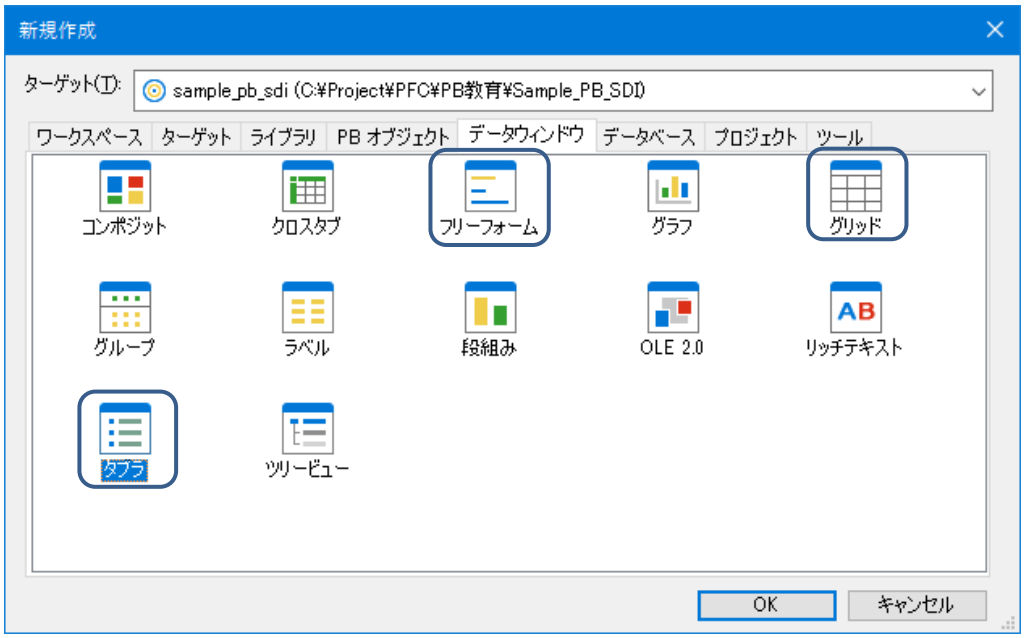
データソース Current ≠ Original のとき
値に変更があったと判断し、Update 文が送信される。
但し、更新特性の設定によっては、Delete 文&Insert 文が送信される。

新しい行のとき、Insert 文が送信される。

バッファ Delete に行があるとき
Delete 文が送信される。

（３）　データウインドウの種類

データウインドウのデザイン時に下図の選択肢があり、使用頻度が高いものを枠で囲んだ。



① フリーフォーム
結果集合１行分をページ内に自由に配置するイメージ

② グリッド
結果集合の一覧イメージで、列幅の調整が可能（エクセルイメージ）
明細行のデザインは１行分のイメージとなる。

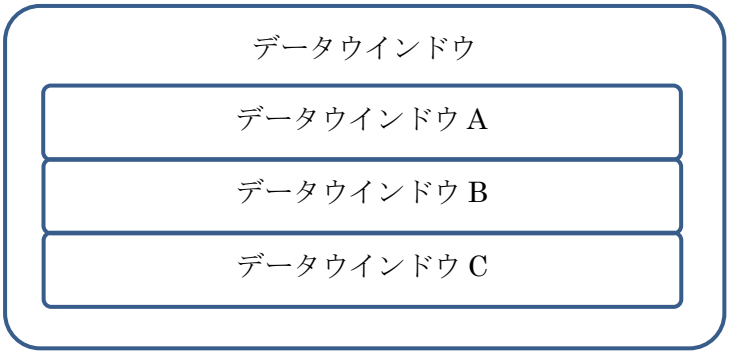
行	見出し１	見出し２	見出し３
1	データ１	データ２	データ３
2	データ１	データ２	データ３
3	データ１	データ２	データ３

③ タブラー
結果集合の一覧イメージで、列幅と位置が固定
ヘッダ見出しを上下に重ねたり、明細行のデザインを複数行のイメージにできる。

行	見出し１
	見出し２
	見出し３
1	データ１
	データ２
	データ３
2	データ１
	データ２
	データ３

④　コンポジット

データウインドウの中にデータウインドウを組み込む。
異なる帳票を左右、上下にまとめて印刷したいときに利用できる。



⑤　クロスタブ

クロスタブ集計用のイメージとなり、ウイザード形式で初期デザインが行われる。

⑥　グラフ

棒グラフ、折れ線、３Ｄグラフの表現が可能になる。

⑦　グループ

結果集合をデザイン画面でグループ集計して表現できる。
ウイザード形式で初期デザインが行われる。
実際には、タブラーで手動デザインすることが多い。

⑧　ラベル

ウイザード形式でラベル用紙を指定してデザインが行われる。

⑨　段組み

結果集合を横方向（段）に並べて配置する。

１行目データ	２行目データ	３行目データ
４行目データ	５行目データ	６行目データ
７行目データ		

⑩　ツリービュー

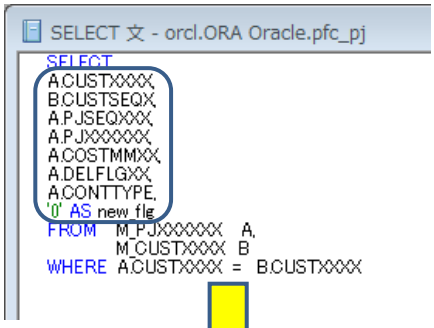
ツリービューイメージの表現ができ、ウイザード形式で初期デザインが行われる。

検索されてメモリーにあるデータをツリービューで表示できることから、.NET のツリービュー表示のレスポンスとは比べ物にならないぐらいに高速である。

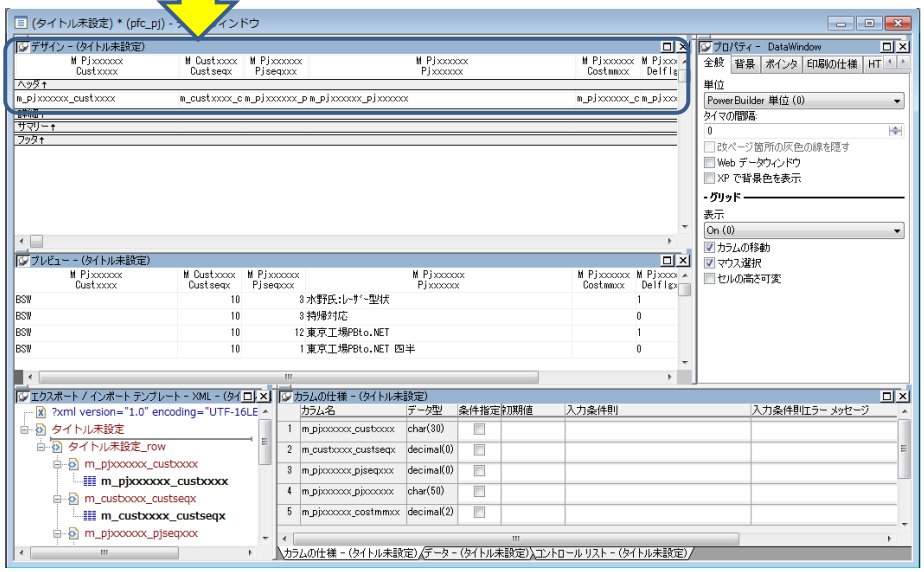
（４）　データウインドウのオブジェクト（ｄｗｏ）

データウインドウをデザインすると、SQL の結果集合カラムに相当するカラムとテキストオブジェクトが自動的に初期デザインされる。
タブラーやグリッドの場合、ヘッダには見出し（テキストオブジェクト）、詳細にカラムが配置される。

★



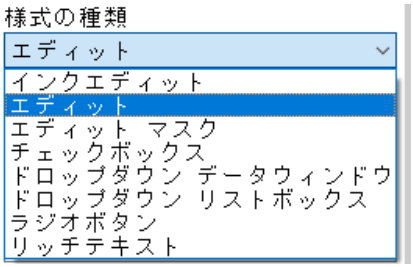
★



カラム、テキストオブジェクトの他に、計算フィールド、図形（直線、楕円、長方形、丸長方形）、グラフ、OLE、レポート（データウインドウの入れ子）などがあり、総称してｄｗｏと呼ぶ。

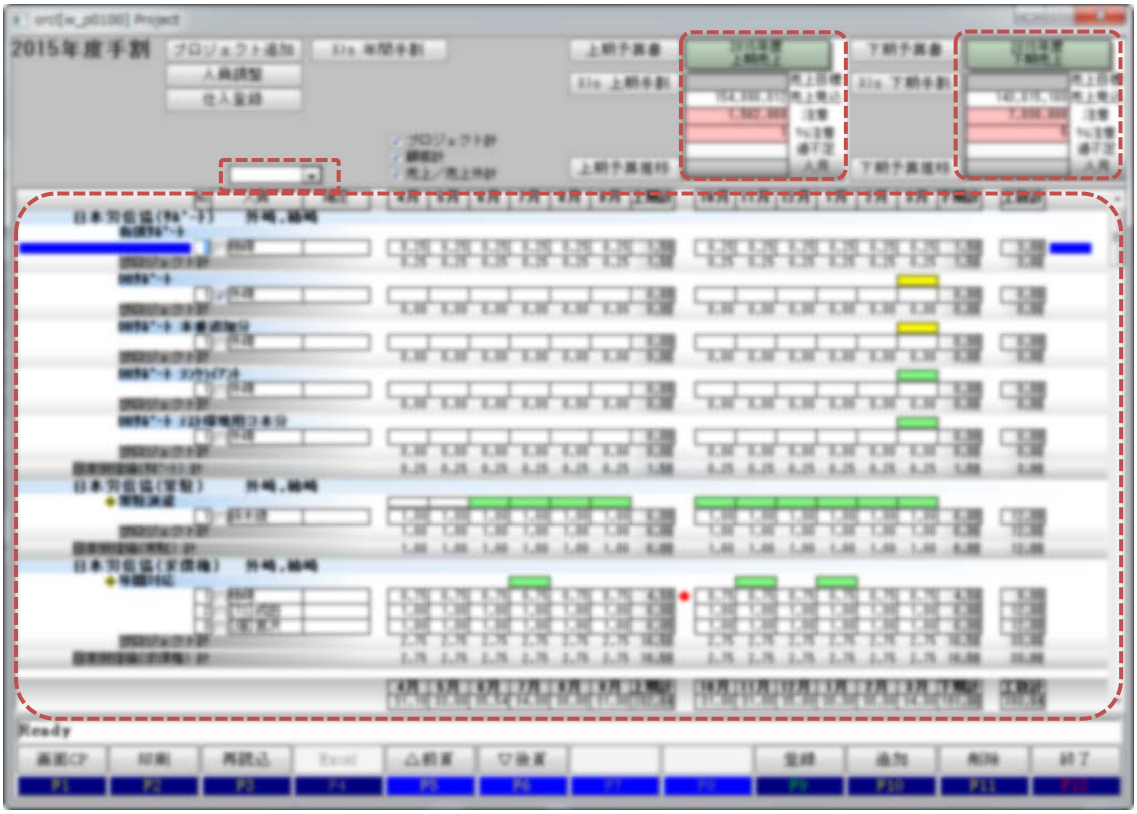
計算フィールドではカラムや計算式を用いてその結果を表示することができる。
タイトルにページ数や日付、フッダに金額の合計や、データの件数を表示できることから、レポート処理で有効利用できる。

カラムは、プロパティにある「様式の種類」を選択することで、様々な表現が可能になる。
ウインドウ用のコントロールを使用せずに、リッチなデザインが可能である。



例）自社プロジェクト管理ツール　for　PowerBuilder

で囲んでいる箇所はデータウインドウ
データウインドウだからこそ、DB データをベースとした自由な表現ができる。



（５）　データウインドウのカラムの表現方法

データウインドウカラムのプロパティを設定することで、リッチなユーザインタフェースを提供できる。

主要なものを紹介する。

① エディット（デフォルト）
作成時のデフォルトであり、最も多く使われる様式である。
書式を指定し、1,000,000 や、2015/12/31 23:59:59 などの表示が可能になり、ユーザにとって見やすい表現方法が豊かである。

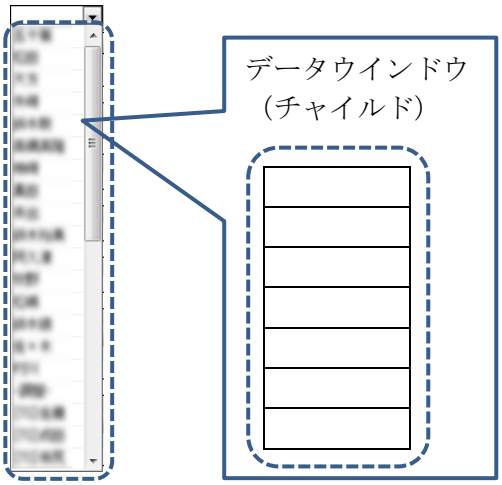
② エディットマスク
入力編集がし易いフォーマットを有している。
※カラムの Null 判定ができないため注意が必要

③ チェックボックス
フラグの ON/OFF などのチェック欄に用いる。

④ ラジオボタン
カラム１つに、複数の選択肢をラジオボタンで設けることができる。
（選択肢は手入力で設定する）

⑤ ドロップダウンリスト
カラム１つに、複数の選択肢をドロップダウンリストで設けることができる。（選択肢は手入力で設定する）

⑥ ドロップダウンデータウインドウ
カラム１つに、複数の選択肢をドロップダウンリストで設けることができる。（選択肢はチャイルドデータウインドウ）
チャイルドデータウインドウには、データウインドオブジェクトを指定する。
DB データを用いてドロップダウンのリストを生成できる。



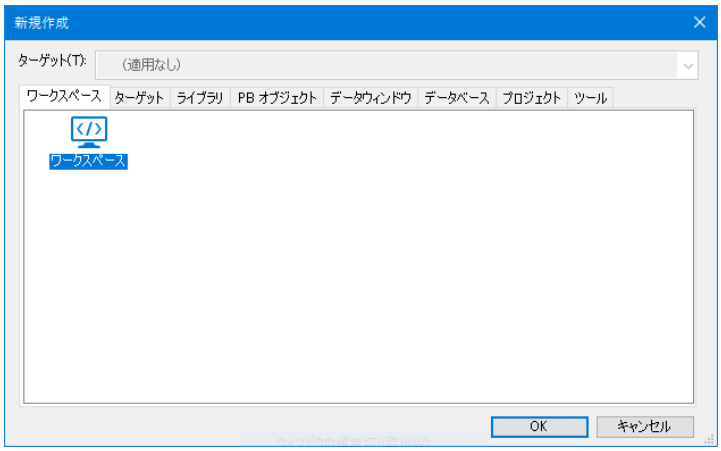
３．　Appeon PowerBuilder 2017 開発手順

（１）　ワークスペースの作成

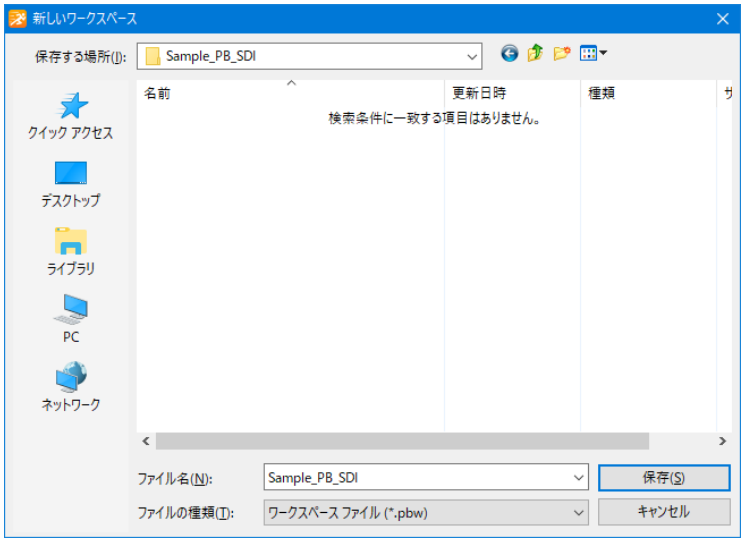
最初に、ソースを管理するための領域を作成する。

「新規作成」を押下する。

新規作成ダイアログから、ワークスペースを選択して「OK」を押下する。



ワークスペースを保存するフォルダを指定し、ファイル名（ワークスペース名）を入力して「保存」を押下する。

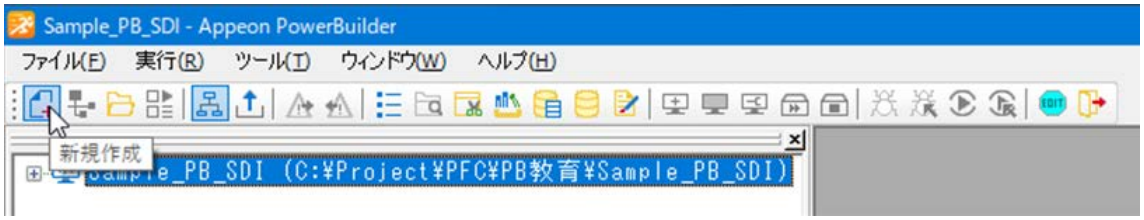


（２） ターゲットの作成

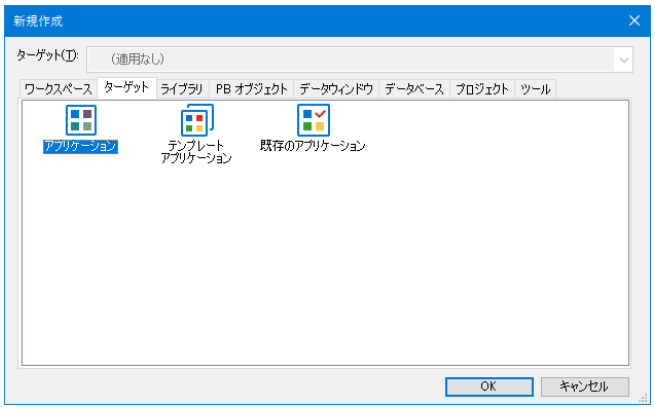
ターゲットはアプリケーションを指し、ワークスペース内に複数登録することができる。
但し、複数ターゲットはソースの管理が煩雑になる懸念があるため、１つのターゲットを推奨する。

ターゲットには、アプリケーションが１つ作成され、PBL ライブラリリストを定義する。（初期は自身の PBL のみとなり、開発を進めながら追加する）

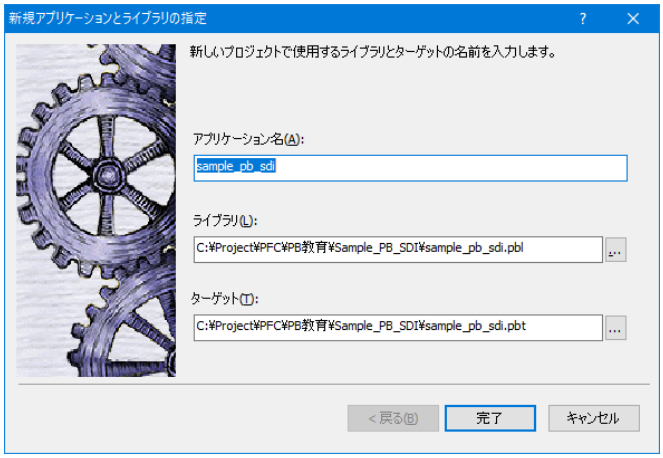
「新規作成」を押下する。



新規作成ダイアログから、ターゲット、アプリケーションを選択して「OK」を押下する。

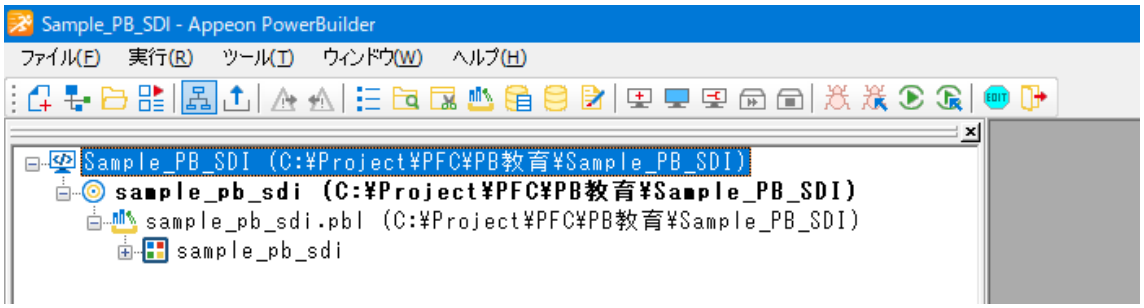


アプリケーション名を入力し、「Tab」押下により、ライブラリ、ターゲットが自動的に展開表示されるので、そのまま「完了」を押下する。



ターゲットの追加により、以下の構成が作成される。

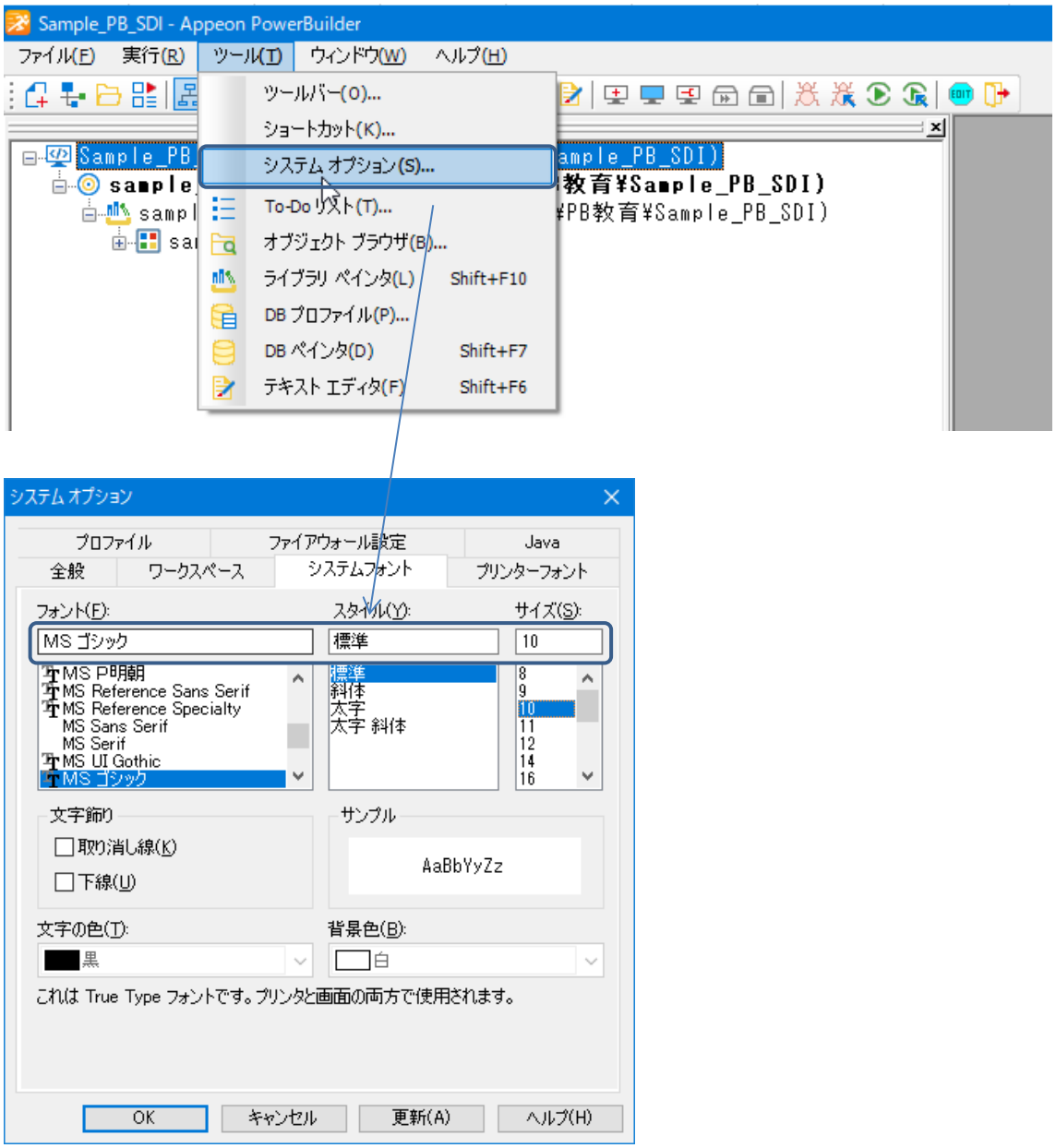
ワークスペース	Sample_PB_SDI
ターゲット	sample_pb_sdi
PBL ファイル	sample_pb_sdi.pbl
アプリケーションオブジェクト	sample_pb_sdi



（３） システムオプションのシステムフォント

インストール後のデフォルトの状態では、スクリプトの文字が小さく、文字幅が異なるため開発し辛いため、ゴシックにして文字を大きくすることを推奨する。

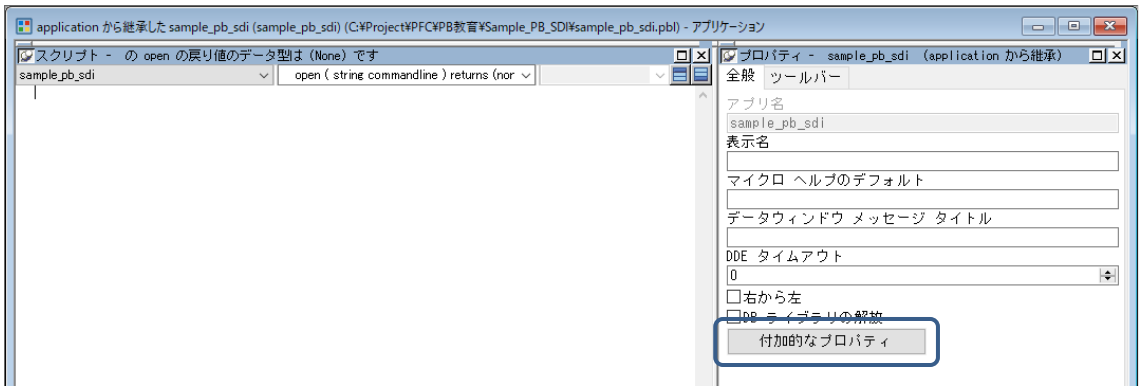
※設定後、Appeon PowerBuilder 2017 を再起動する。



（４）　アプリケーションの付加的なプロパティの設定

アプリケーションが持つ「付加的なプロパティ」を設定する。
デザイン時の文字の大きさをここで統一する。

※設定後、変更箇所を保存して Appeon PowerBuilder 2017 を再起動する。



application から継承した sample_pb_sdi (sample_pb_sdi) (C:¥Project¥PFC¥PB教育¥Sample_PB_SDI¥sample_pb_sdi.pbl) - アプリケーション

スクリプト - の open の戻り値のデータ型は (None) です

sample_pb_sdi　open (string commandline) returns (nor

プロパティ - sample_pb_sdi (application から継承)

全般　ツールバー

アプリ名: sample_pb_sdi

表示名:

マイクロ ヘルプのデフォルト:

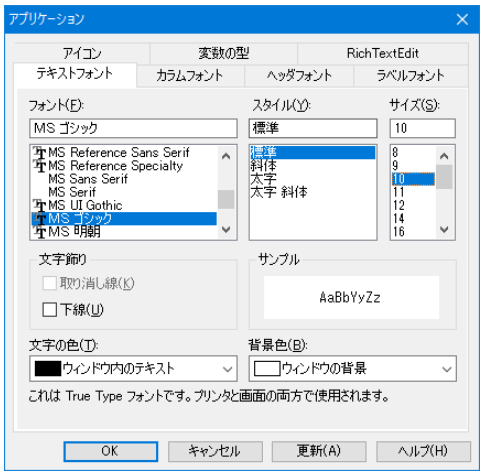
データウィンドウ メッセージ タイトル:

DDE タイムアウト: 0

☐右から左

☐DB ライブラリの解放

付加的なプロパティ



アプリケーション

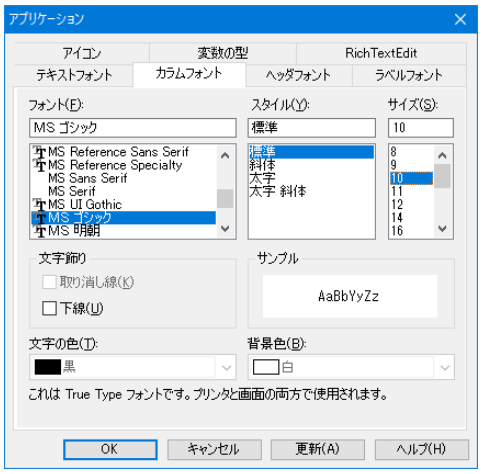
アイコン　変数の型　RichTextEdit
テキストフォント　カラムフォント　ヘッダフォント　ラベルフォント

フォント(F): MS ゴシック
スタイル(Y): 標準
サイズ(S): 10

文字飾り: ☐取り消し線(K)　☐下線(U)
サンプル: AaBbYyZz
文字の色(T): ウィンドウ内のテキスト
背景色(B): ウィンドウの背景

これは True Type フォントです。プリンタと画面の両方で使用されます。

OK　キャンセル　更新(A)　ヘルプ(H)



アプリケーション

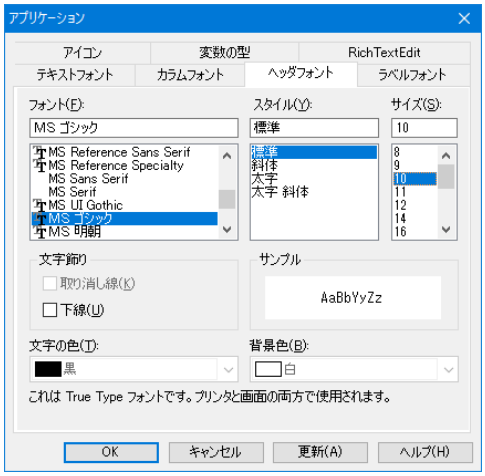
アイコン　変数の型　RichTextEdit
テキストフォント　**カラムフォント**　ヘッダフォント　ラベルフォント

フォント(F): MS ゴシック
スタイル(Y): 標準
サイズ(S): 10

文字飾り: ☐取り消し線(K)　☐下線(U)
サンプル: AaBbYyZz
文字の色(T): 黒
背景色(B): 白

これは True Type フォントです。プリンタと画面の両方で使用されます。

OK　キャンセル　更新(A)　ヘルプ(H)



アプリケーション

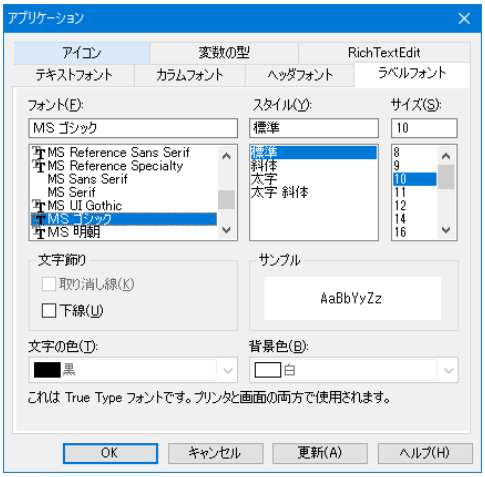
アイコン　変数の型　RichTextEdit
テキストフォント　カラムフォント　**ヘッダフォント**　ラベルフォント

フォント(F): MS ゴシック
スタイル(Y): 標準
サイズ(S): 10

文字飾り: ☐取り消し線(K)　☐下線(U)
サンプル: AaBbYyZz
文字の色(T): 黒
背景色(B): 白

これは True Type フォントです。プリンタと画面の両方で使用されます。

OK　キャンセル　更新(A)　ヘルプ(H)



アプリケーション

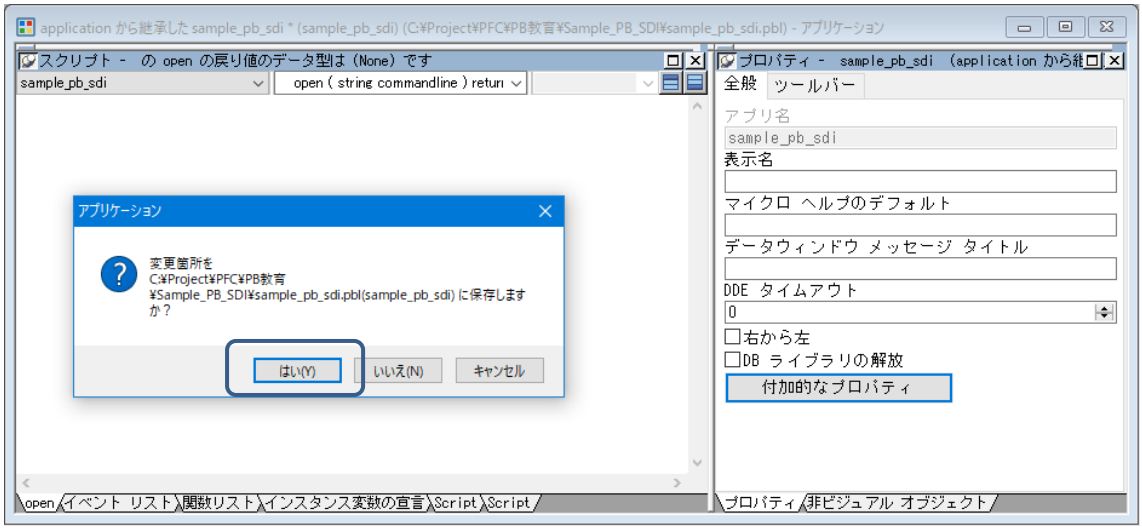
アイコン　変数の型　RichTextEdit
テキストフォント　カラムフォント　ヘッダフォント　**ラベルフォント**

フォント(F): MS ゴシック
スタイル(Y): 標準
サイズ(S): 10

文字飾り: ☐取り消し線(K)　☐下線(U)
サンプル: AaBbYyZz
文字の色(T): 黒
背景色(B): 白

これは True Type フォントです。プリンタと画面の両方で使用されます。

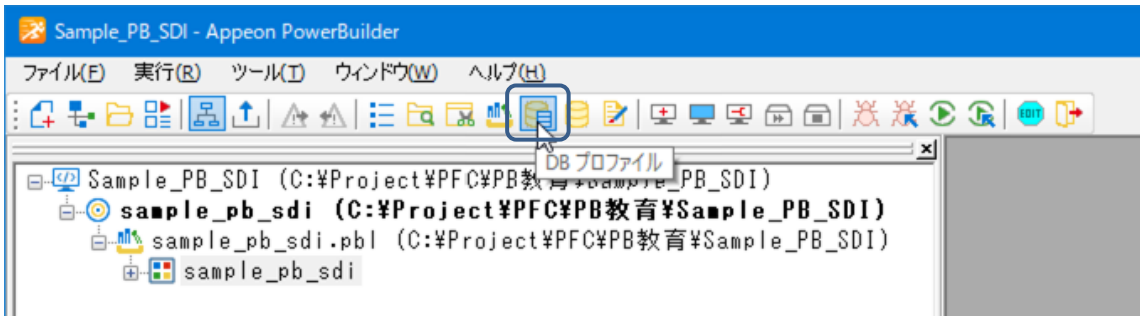
OK　キャンセル　更新(A)　ヘルプ(H)

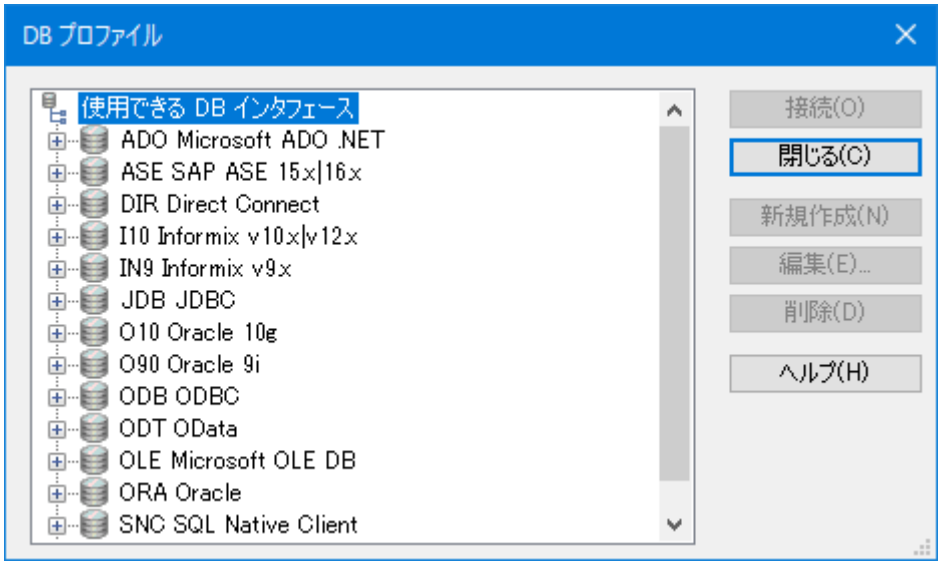

application から継承した sample_pb_sdi * (sample_pb_sdi) (C:¥Project¥PFC¥PB教育¥Sample_PB_SDI¥sample_pb_sdi.pbl) - アプリケーション
スクリプト - の open の戻り値のデータ型は (None) です
sample_pb_sdi
open (string commandline) return
アプリケーション
変更箇所を
C:¥Project¥PFC¥PB教育
¥Sample_PB_SDI¥sample_pb_sdi.pbl(sample_pb_sdi) に保存します
か？
はい(Y)
いいえ(N)
キャンセル
open
イベント リスト
関数リスト
インスタンス変数の宣言
Script
Script
プロパティ - sample_pb_sdi (application から継
全般
ツールバー
アプリ名
sample_pb_sdi
表示名
マイクロ ヘルプのデフォルト
データウィンドウ メッセージ タイトル
DDE タイムアウト
0
右から左
DB ライブラリの解放
付加的なプロパティ
プロパティ
非ビジュアル オブジェクト

（５） DB 接続

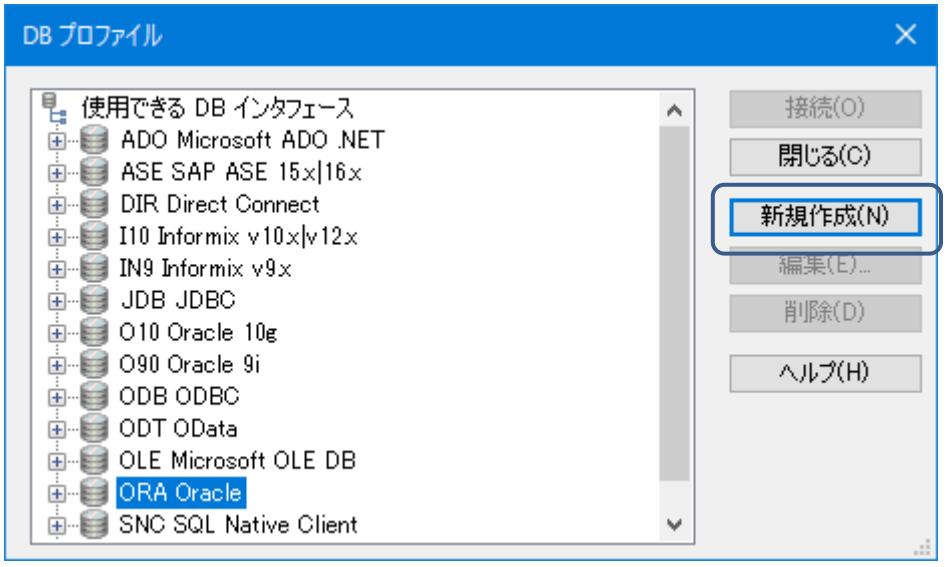
コーディングの前に、IDE を DB に接続する。

「DB プロファイル」を押下する。



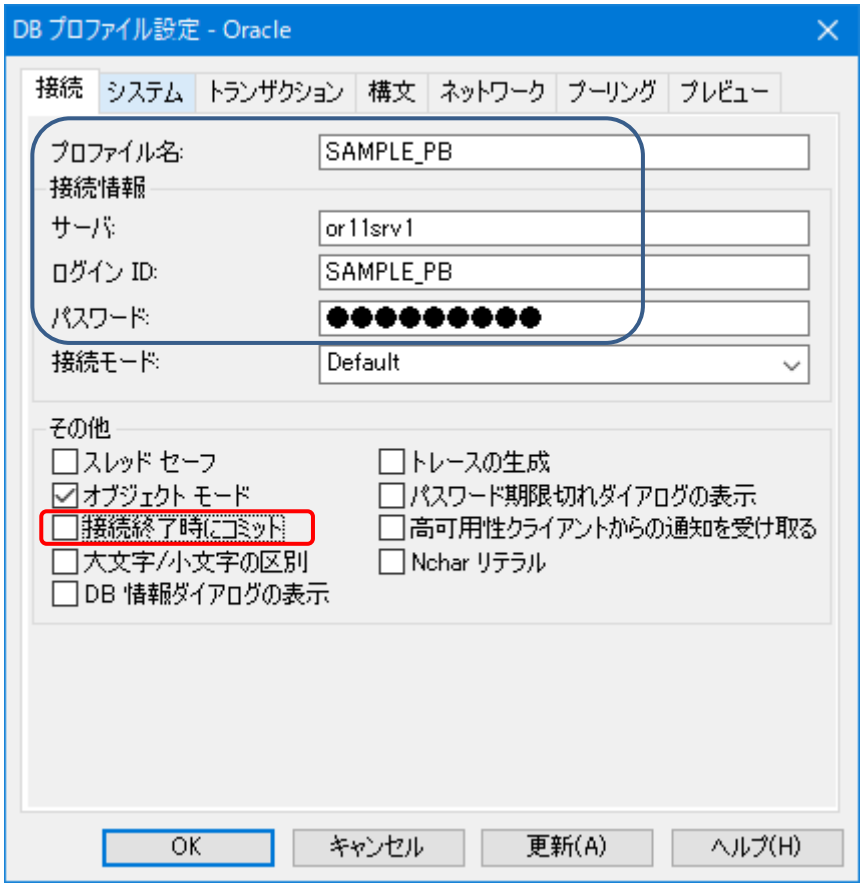


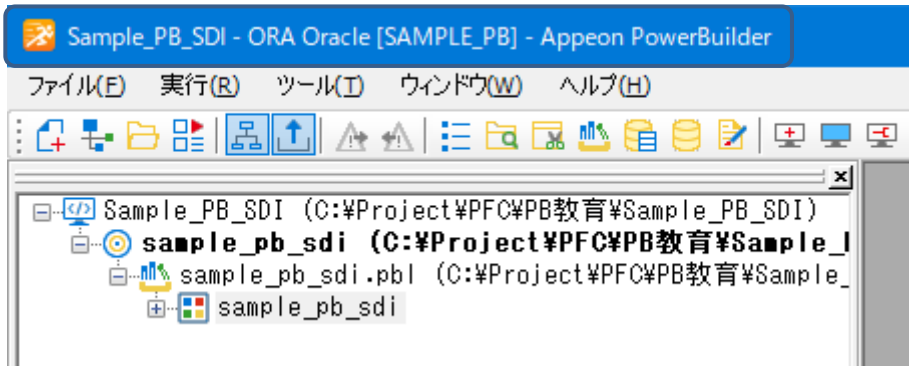
Oracle11g への接続として設定する。



「接続」の設定を行う。

プロファイル名：　DB プロファイルの一覧に表示される。
接続情報：　DB ベンダーごとに設定する。
その他：　接続終了時にコミット
　ON のままでアプリケーションが異常終了した場合、その時点まで実行された SQL 文が COMMIT されるため、外すことを推奨する。

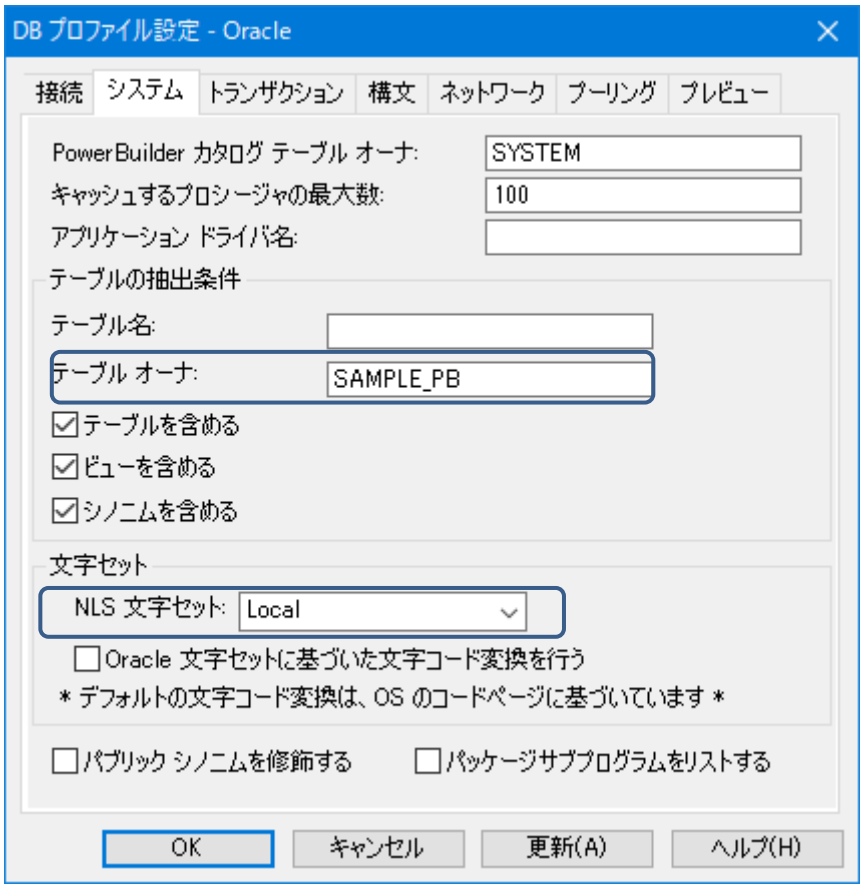




「システム」の設定を行う。

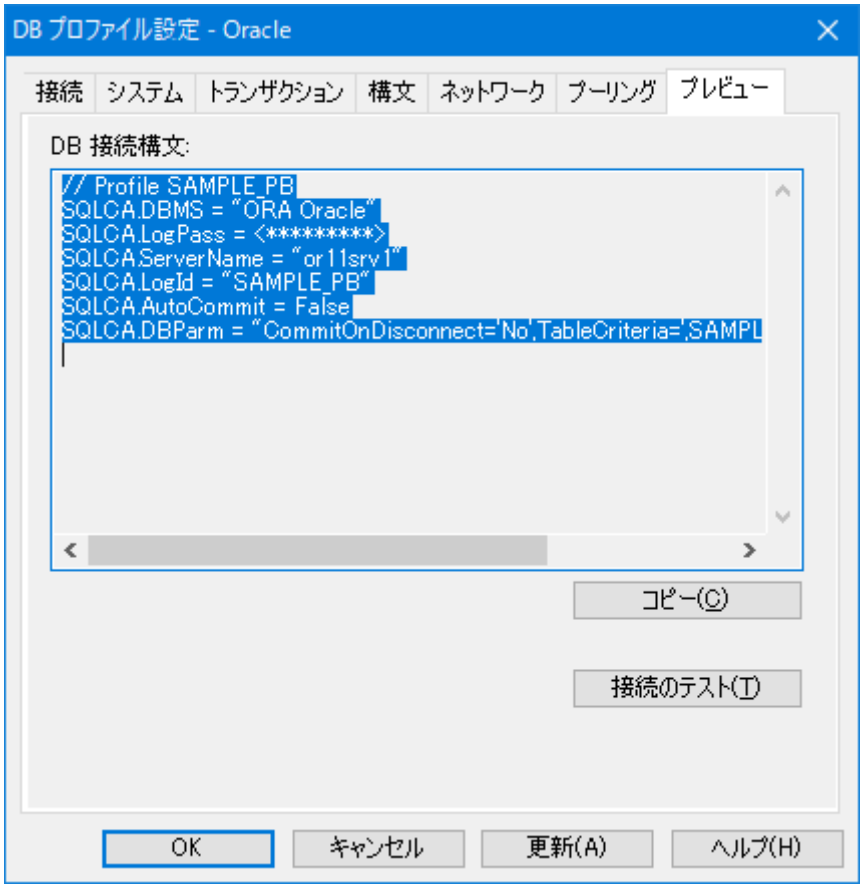
テーブル抽出条件：　テーブルオーナーを指定する。
テーブルオーナーを指定しておくことで、DB ペインターでテーブル操作する際に、他のオーナーのテーブルが表示されないため、目的のテーブルが探しやすくなる。

文字セット：　NLS 文字を指定する。
DB の文字コードが SJIS のとき、Local を指定する。
但し、Build 5006J は不具合があるため、開発時は Unicode を指定し、実行時は Local を指定すること。



「プレビュー」について

DB 接続構文をプログラミングに利用することができる。

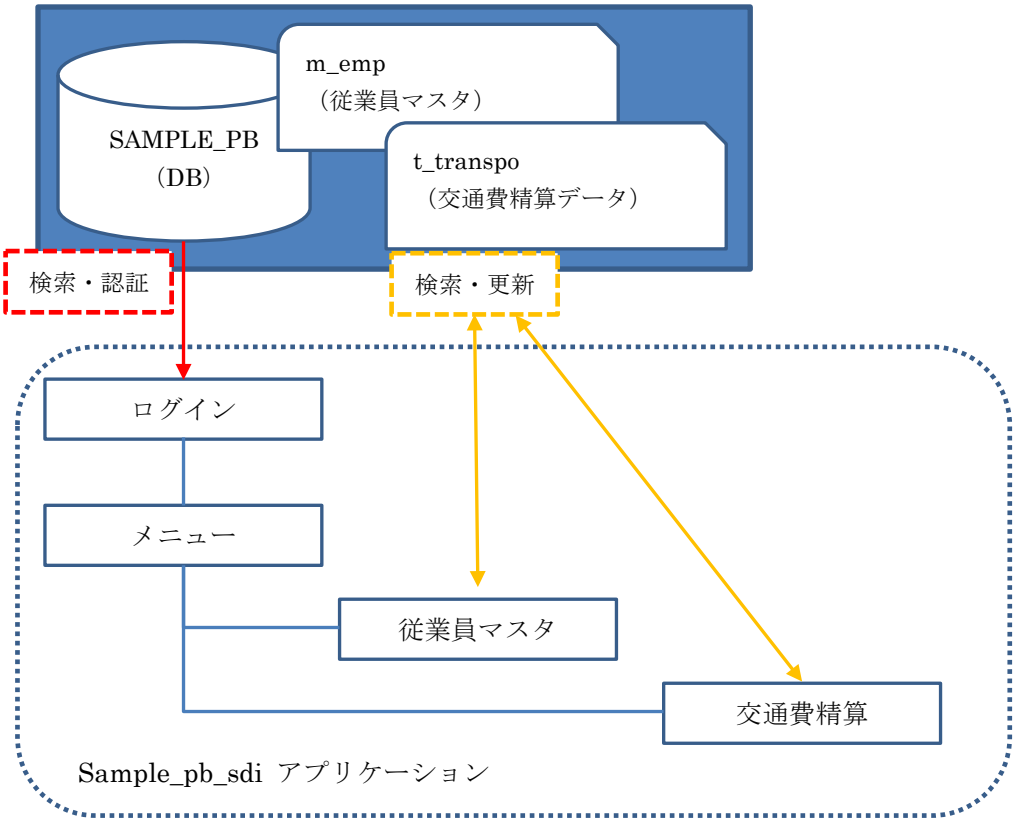


（６）　アプリケーションのコーディング

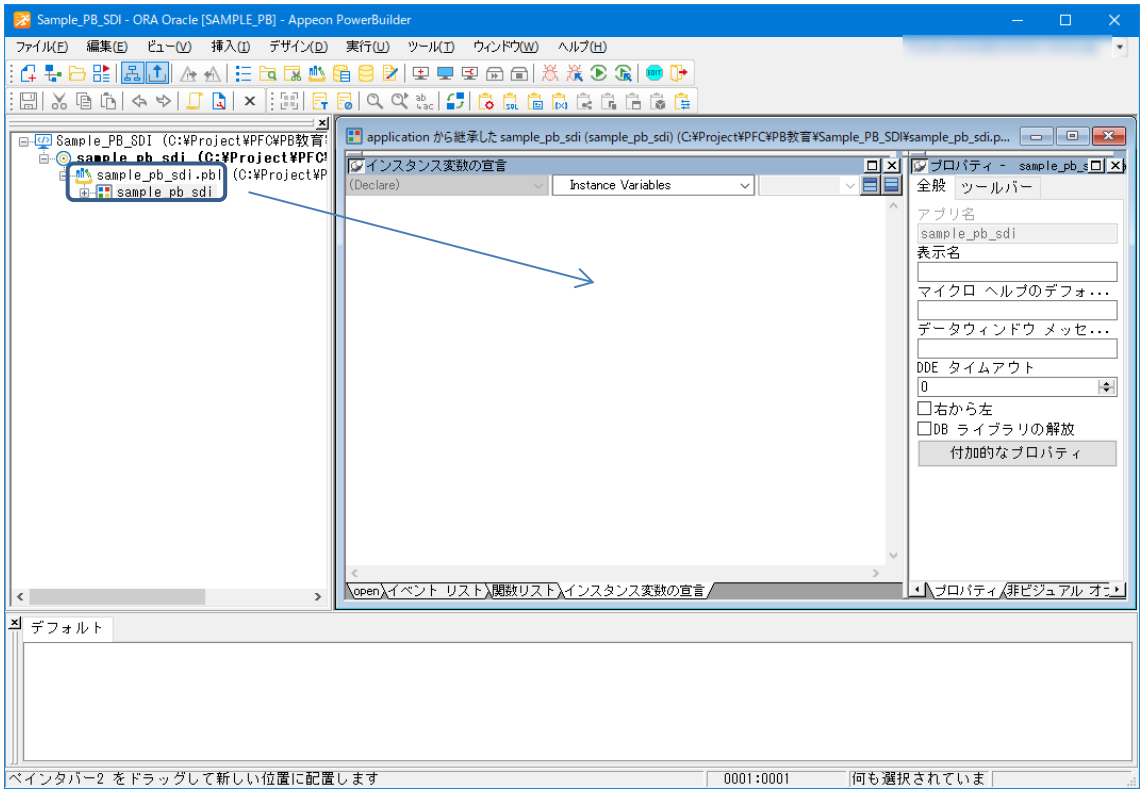
上記手順により作成されたアプリケーションオブジェクトに、基本ソースをコーディングする。

■アプリケーションの概要
DB（Sample_PB）に用意されたテーブル『従業員マスタ』『交通費精算』からデータの検索や更新（登録・変更・削除）を行う画面を作成する。

■画面フローイメージ

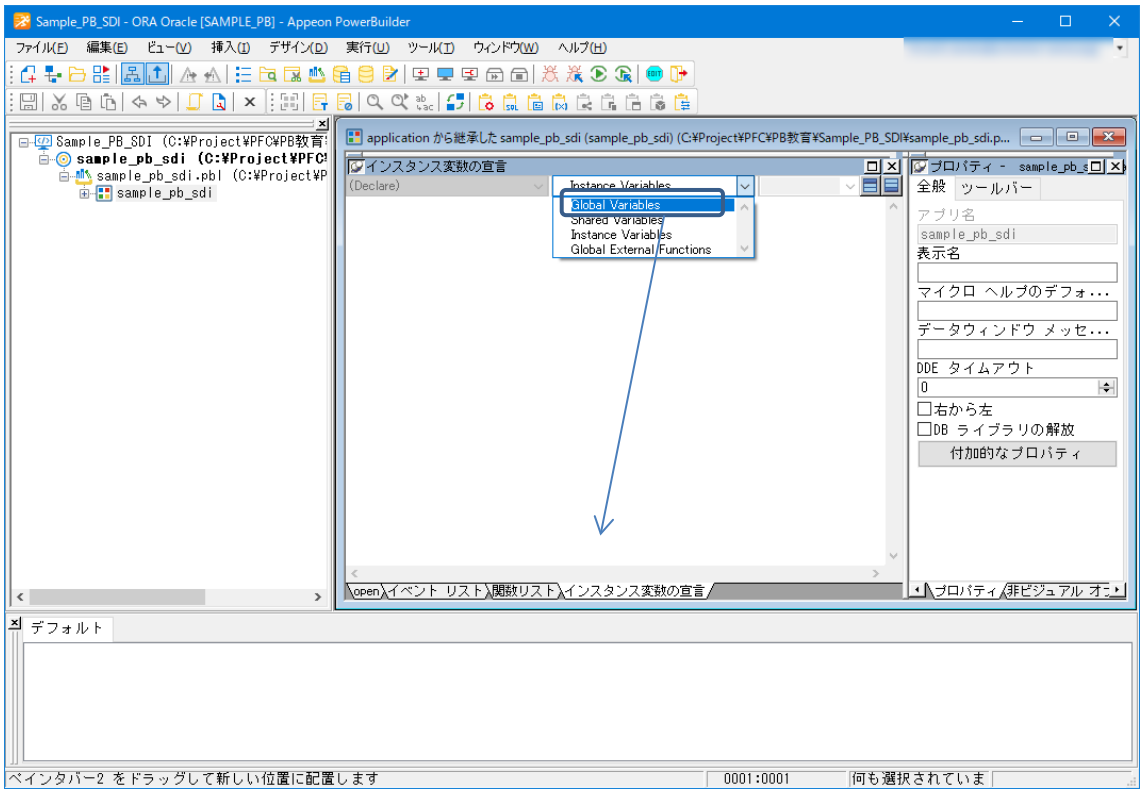


アプリケーションオブジェクトは、PB アプリケーションの入り口であり、出口である。



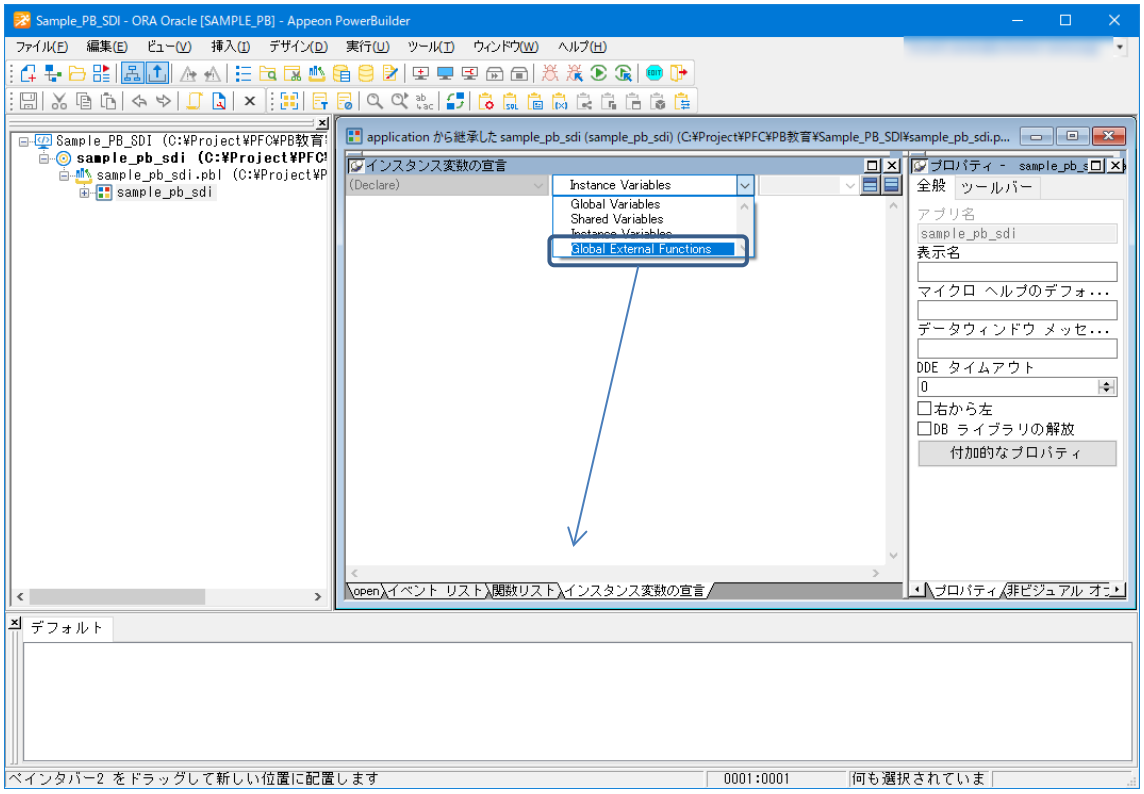
① グローバル変数宣言

下図操作により、グローバル変数宣言が可能になる。



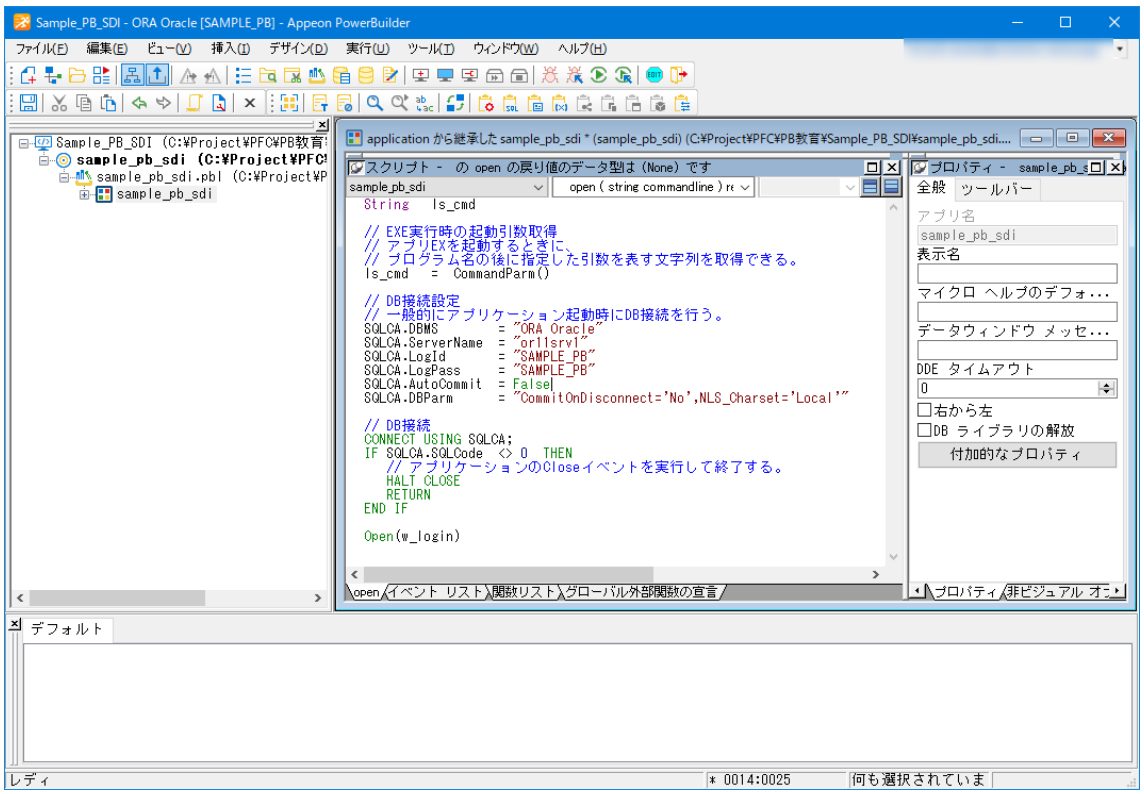
② グローバル外部関数宣言

下図操作により、グローバル外部関数変数宣言が可能になる。



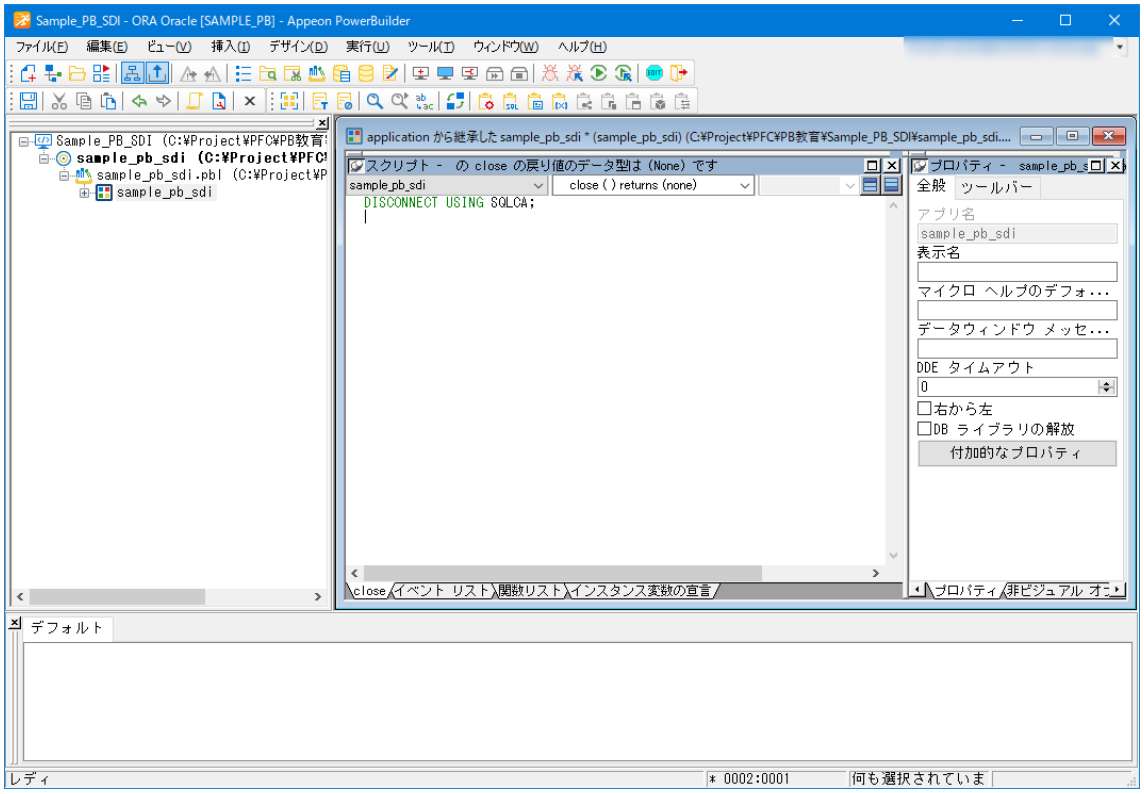
③ Open イベント

アプリケーション起動時の処理を記載する。



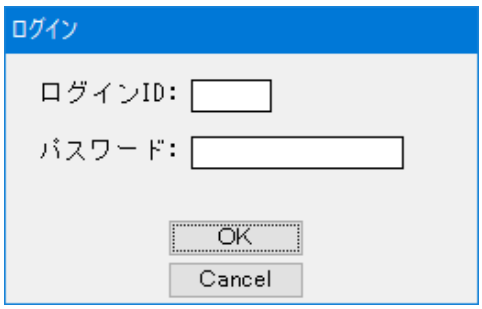
Close イベント

アプリケーション終了時の処理を記載する。



（７） ログイン画面の作成

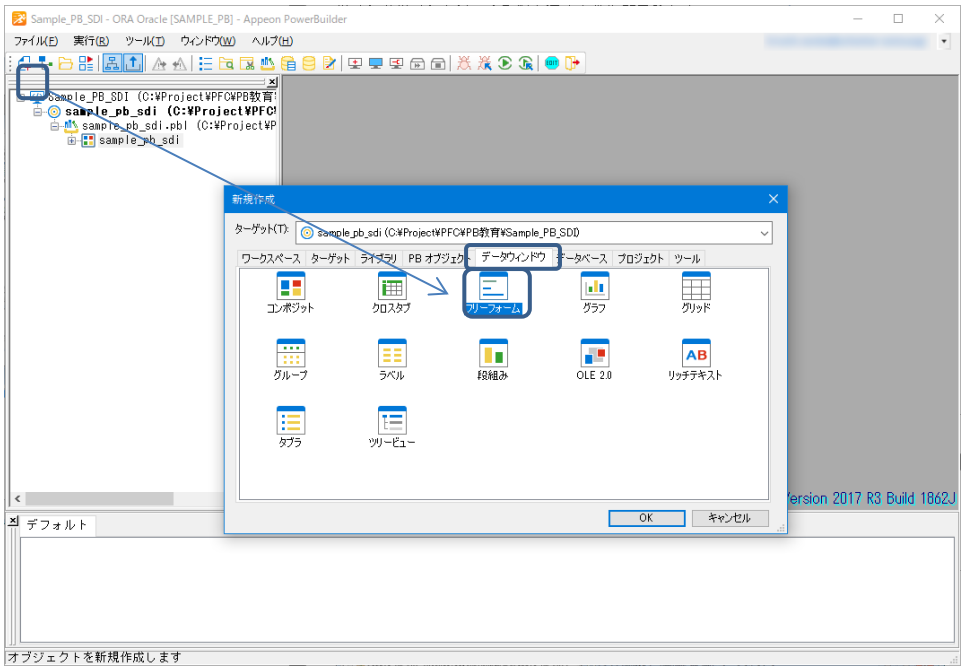
作成するウインドウのイメージ

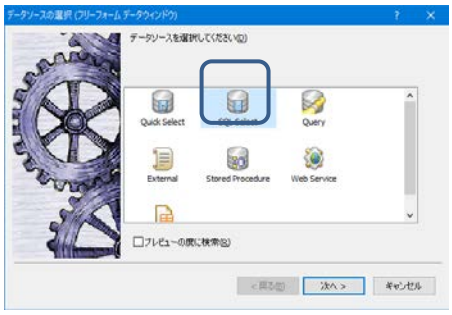


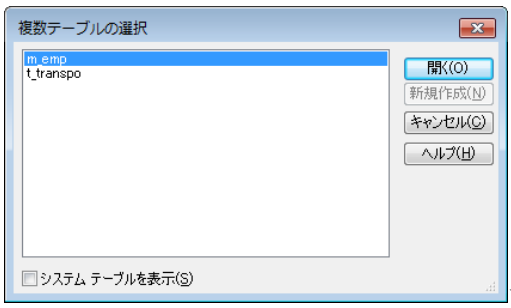
① データウインドウの作成

画面上に配置できるコントロールは様々あるが、今回はデータウインドウで入力フィールドを作成する。

検索・更新がなく入力専用であっても、データウインドウを用いることでフィールドデザインが容易に行えるメリットがある。



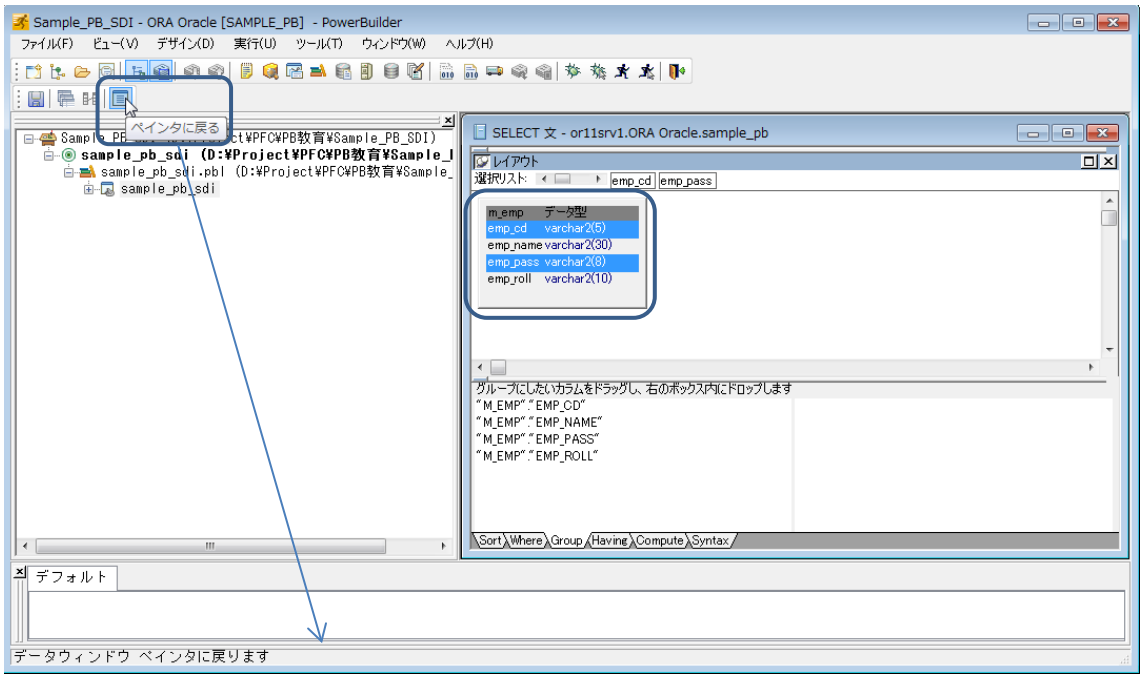


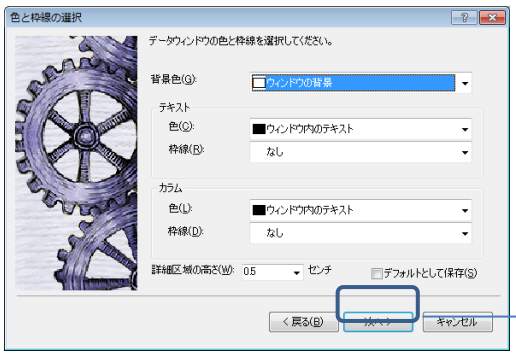


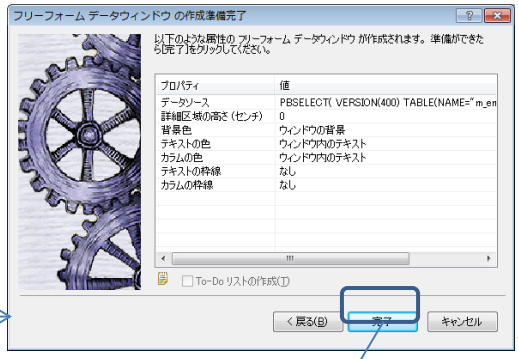
★

テーブルからカラムを指定して、ペインタに戻る。

★

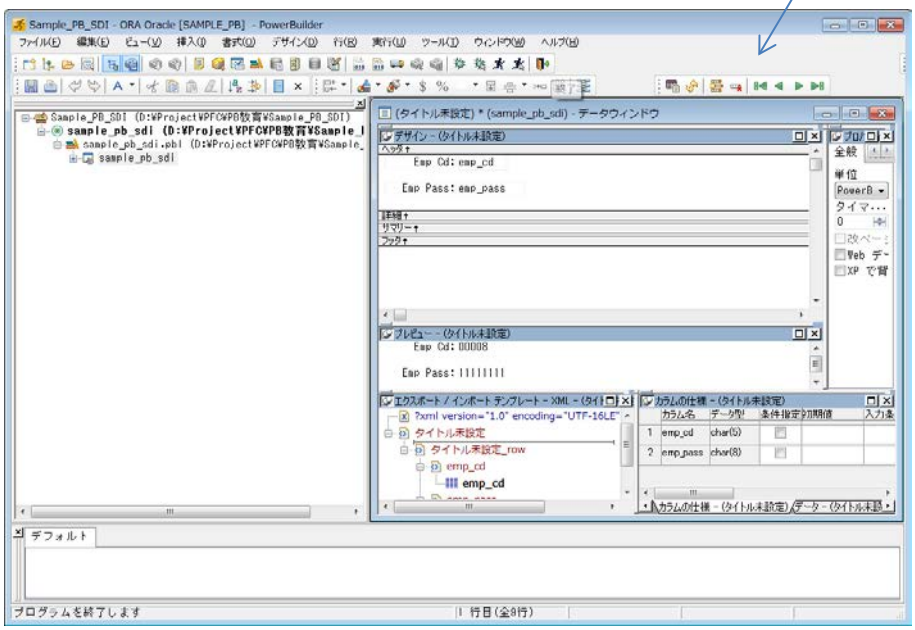




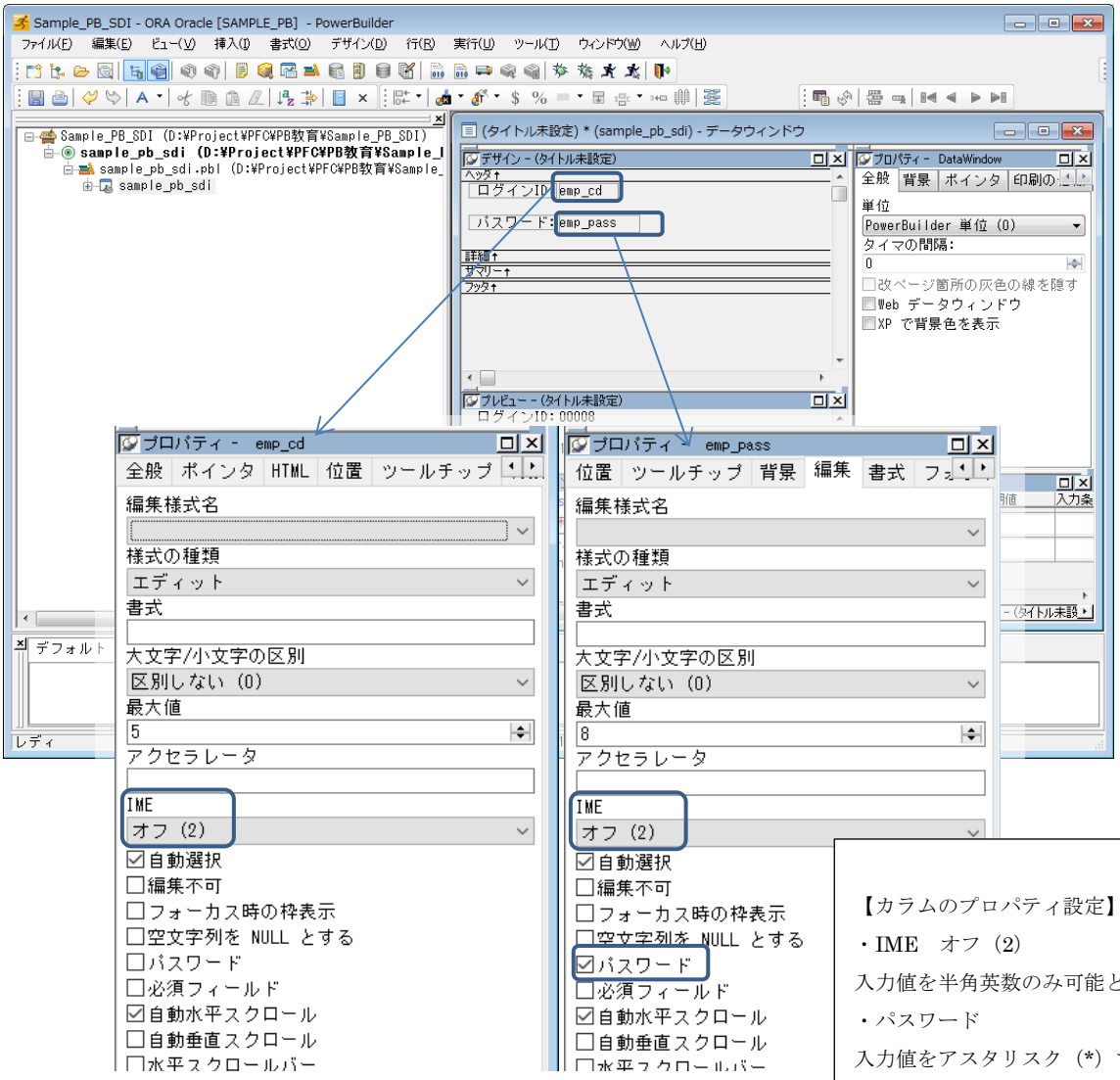


★

ウインドウコントロールを配置するよりも簡単である。
よって、開発スピードとメンテナンス性が向上する。★

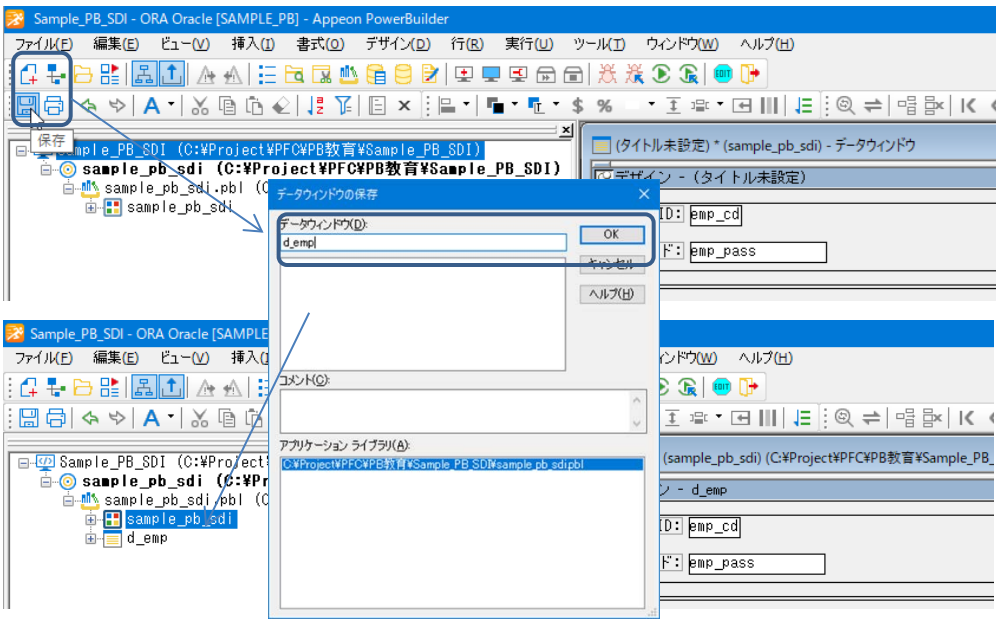


体裁を整える。★



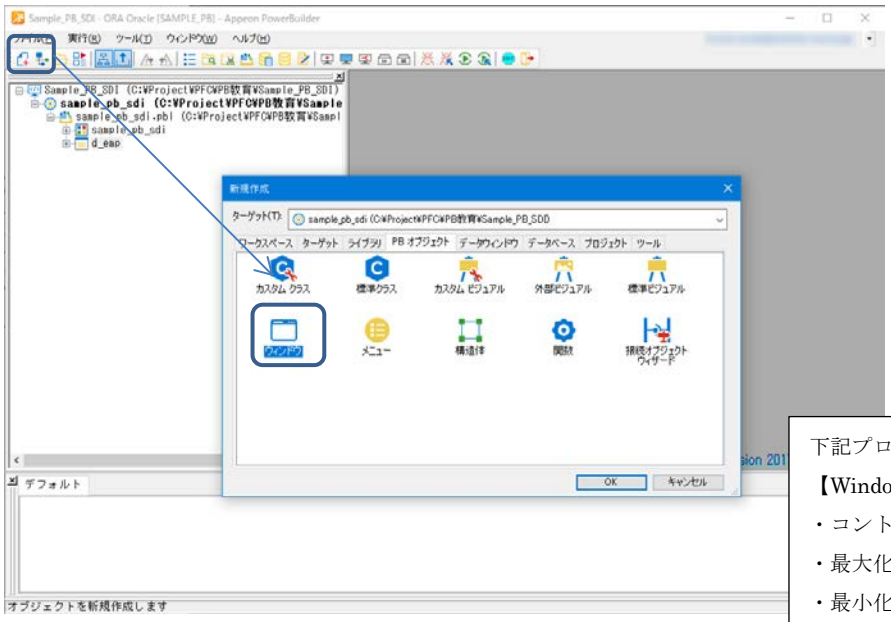
【カラムのプロパティ設定】
・IME　オフ（2）
入力値を半角英数のみ可能とする
・パスワード
入力値をアスタリスク（*）で表示する

保存する。



② ウインドウの作成

ウインドウを新規で開き、コントロールを配置する。



下記プロパティを無効（チェック OFF）にする

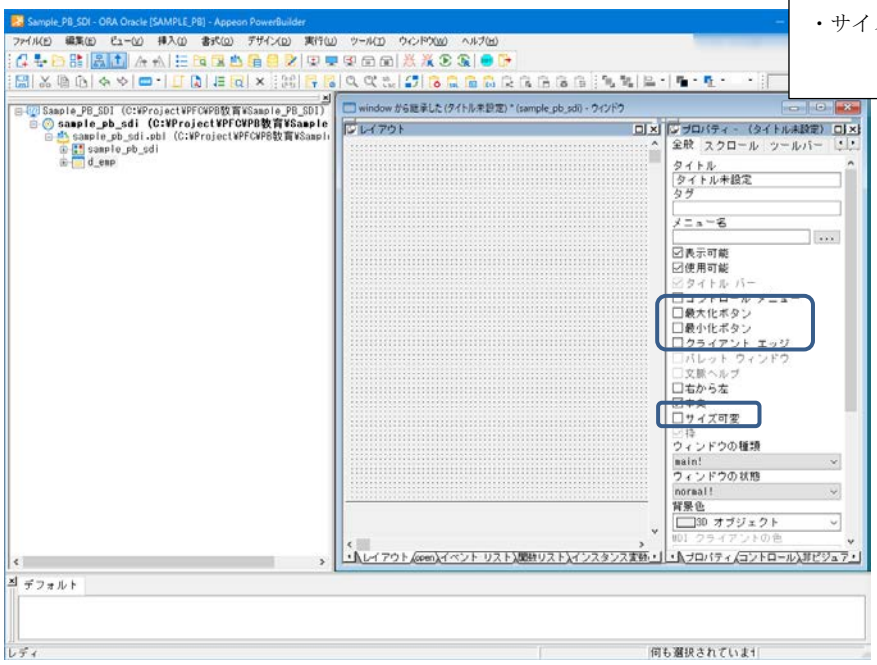
【Window コントロールのプロパティ設定】

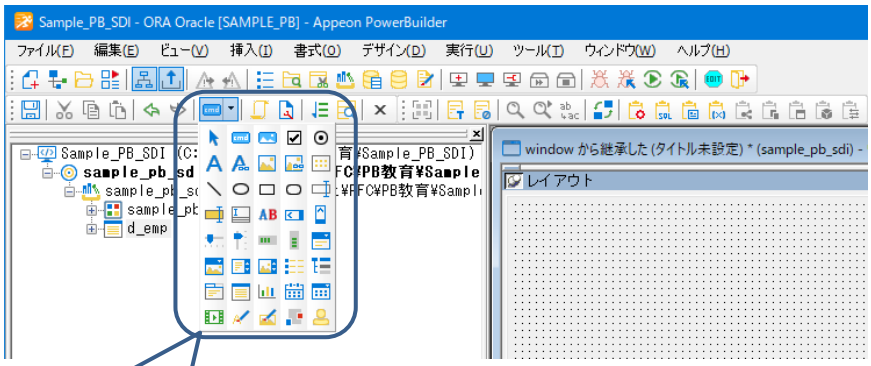
・コントロールメニュー

・最大化ボタン

・最小化ボタン

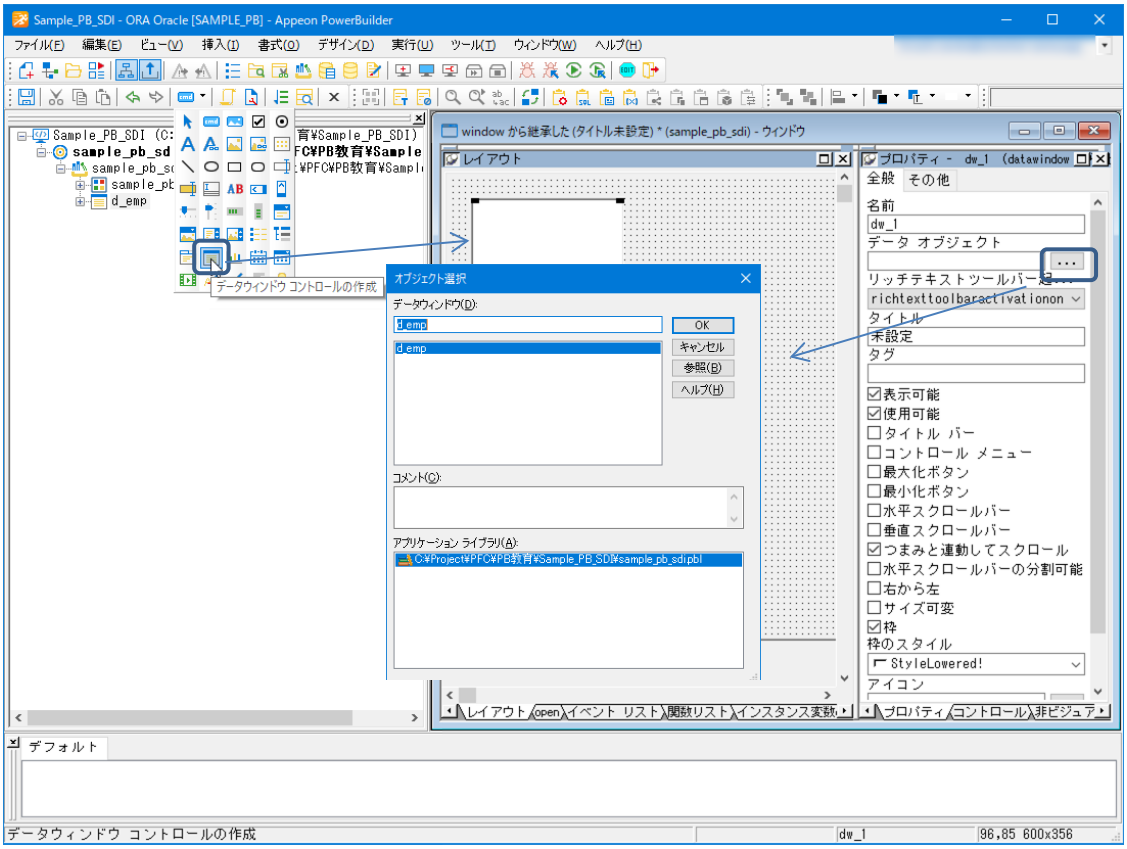
・サイズ可変



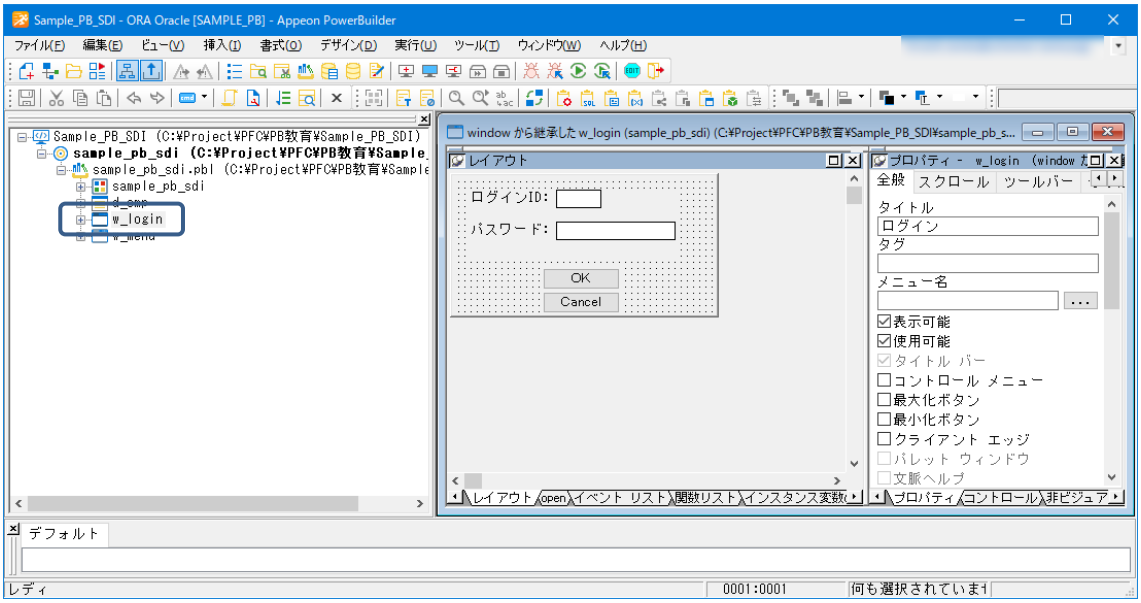


配置できるコントロール

データウインドウを配置する。



コマンドボタンも貼り付け、体裁を整え、保存する。(w_login)

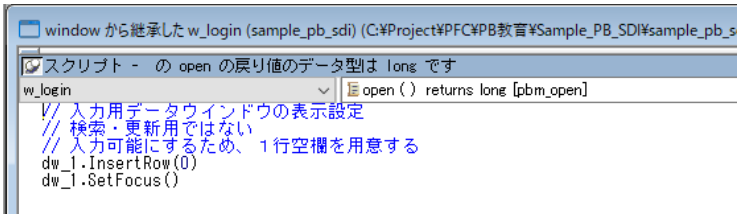


③ スクリプトの記載箇所

w_login：ウインドウのイベント
dw_1：データウインドウのイベント
cb_1：OK ボタンのイベント
cb_2：Cancel ボタンのイベント

④ w_login

A) open イベント

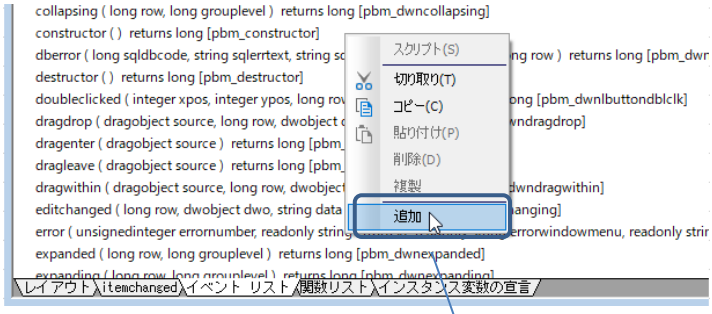


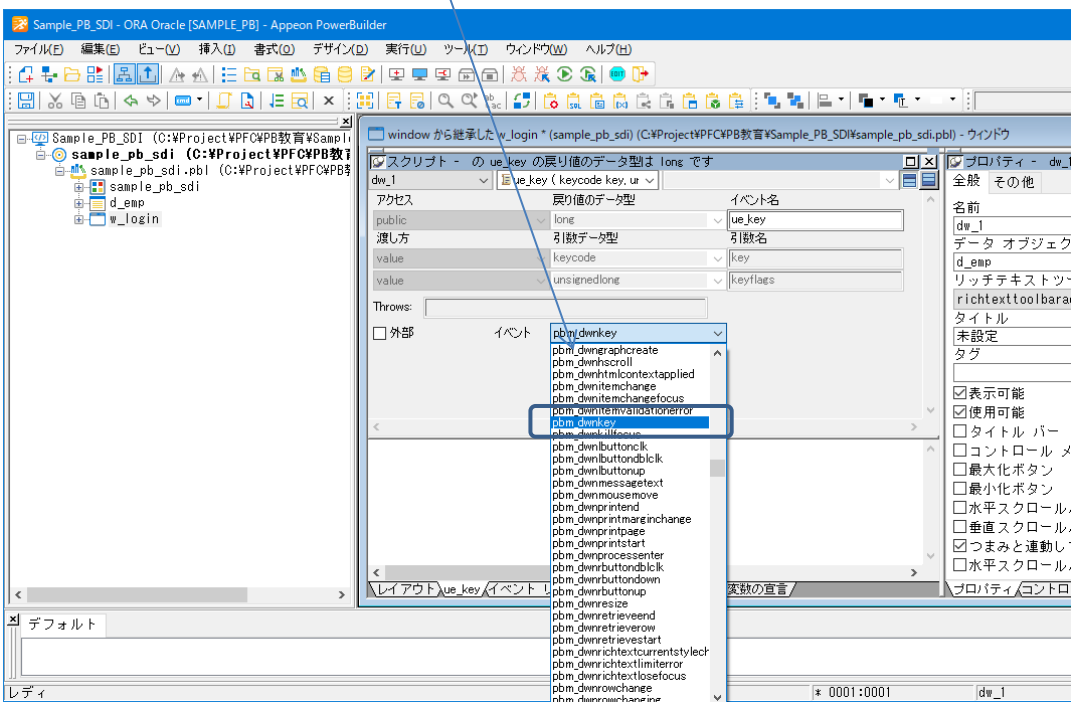
⑤ dw_1

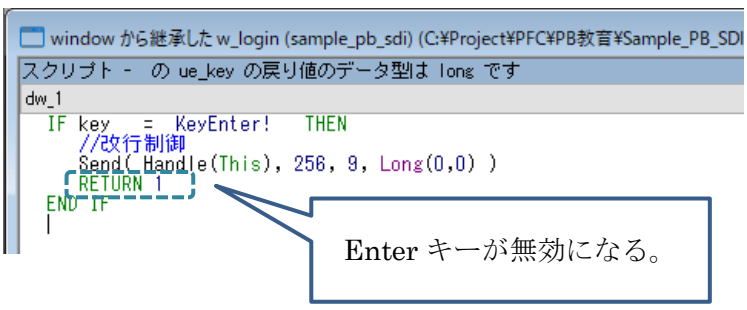
A) ユーザイベントの作成　ue_key

Enter キー押下でカラム移動する仕組みを入れる。

ウインドウズ標準では、Enter キー押下で次行移動となり操作性が悪いため、Enter キー押下で Tab キーを発行する。







⑥ cb_1：OK

A) clicked イベント

ログイン、パスワードのチェックを行う。

データウィンドウの基本動作

≪重要キーワード≫

・データ管理に使用するバッファ

Original / Primary / Filter /Delete バッファ

・データ操作スクリプト

カラムデータの取得方法

①関数によるアクセス

dw_1.GetItemX[データ型](行番号, ”カラム名”)

②ドット表記によるアクセス

dw_1.Object.カラム名[行番号]

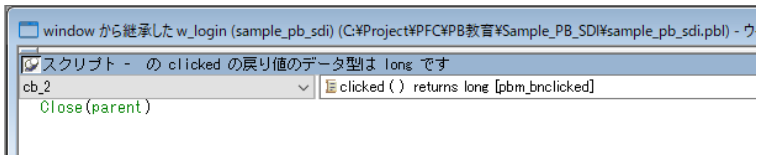
スクリプト内で SQL 文を記載時
取得値を定義済の変数「ls_emp_pass_db」に格納できる。

⑦ cb_2：Cancel

A) clicked イベント

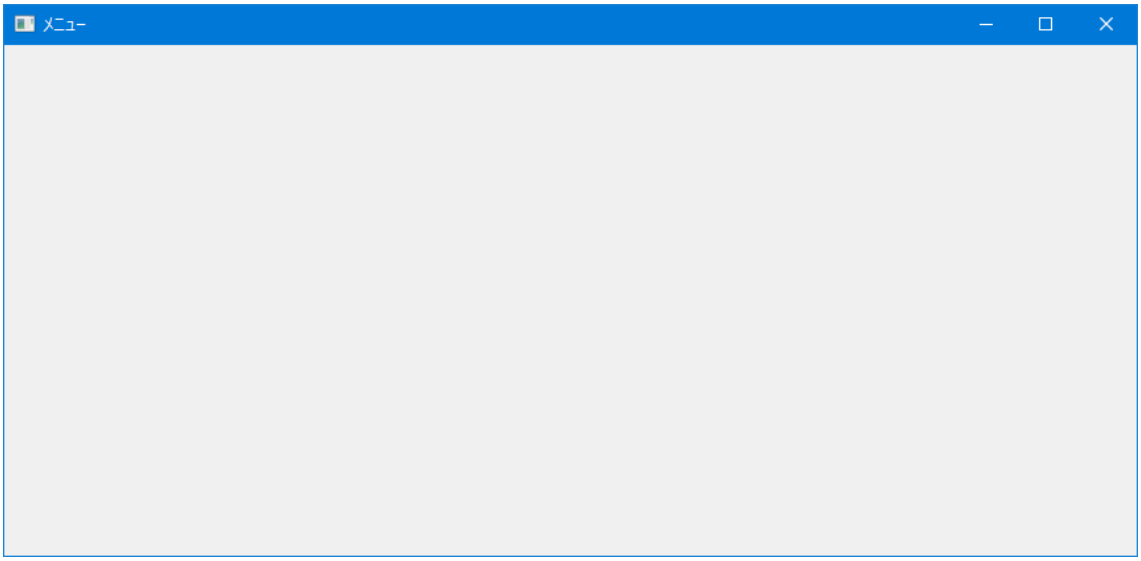
画面を閉じる。

Close(ウインドウ名)になるが、Parent 代名詞を使用している。
Parent：コントロールから見て、張り付いている親（Parent）を意味する。（自分自身は This になる）



（８） メニュー画面（暫定）の作成

作成するメニューのイメージ（暫定）

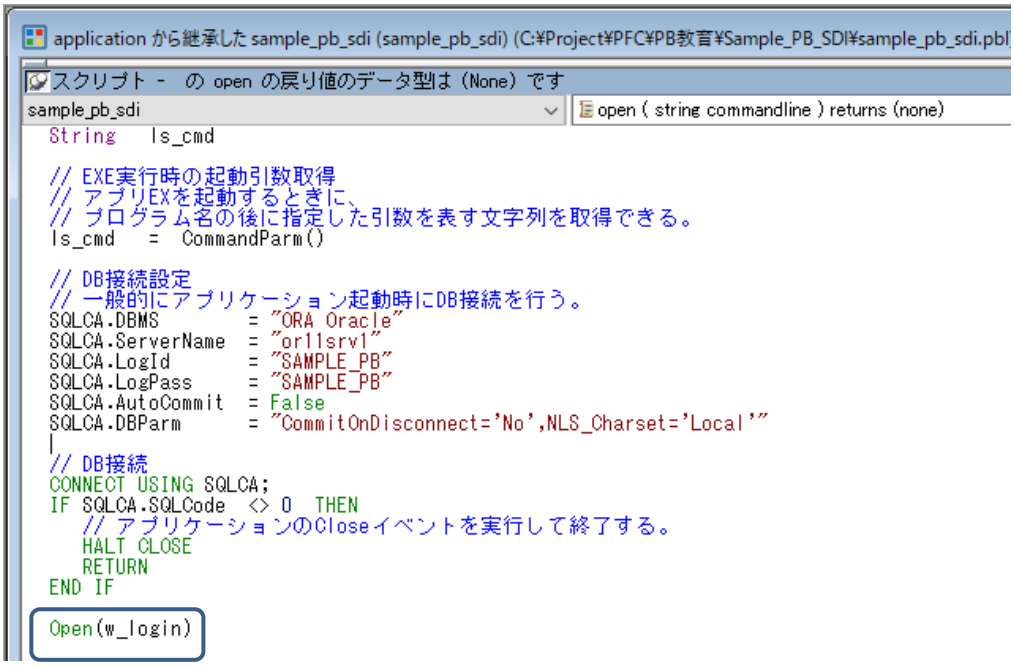


w_menu で保存する。

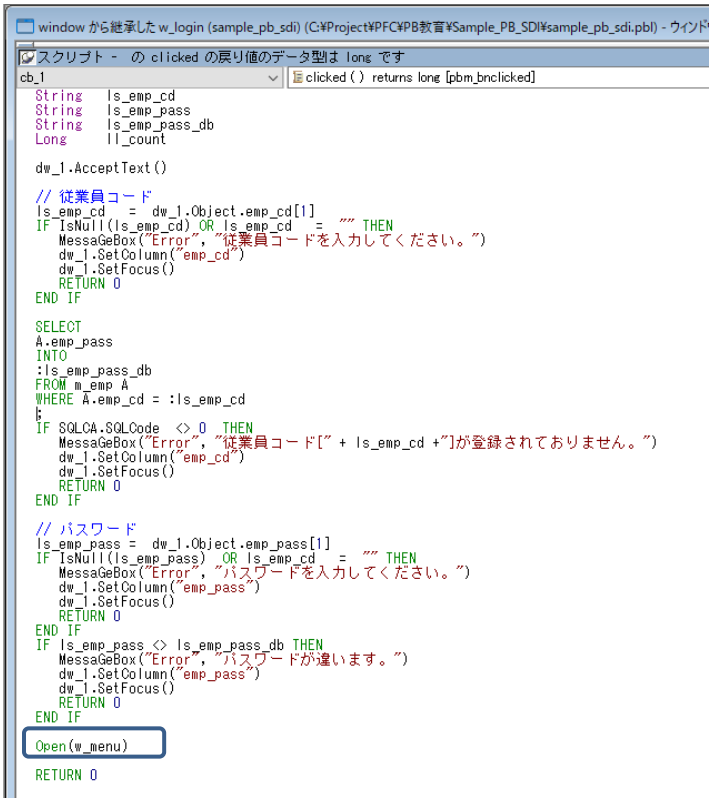
（９） アプリ、ログイン、メニューの連結

これまで作成したオブジェクトの動作を連結する。

① アプリケーション：sample_pb_sdi



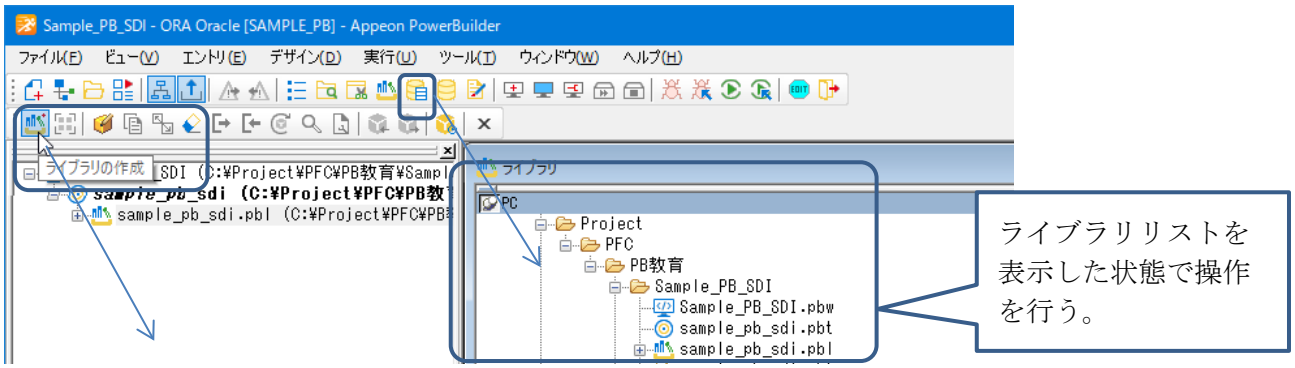
② ログインウインドウ：w_login

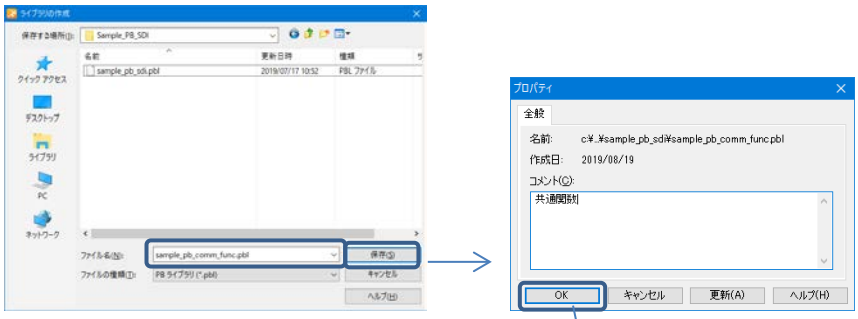


（１０）PBL ファイルの追加

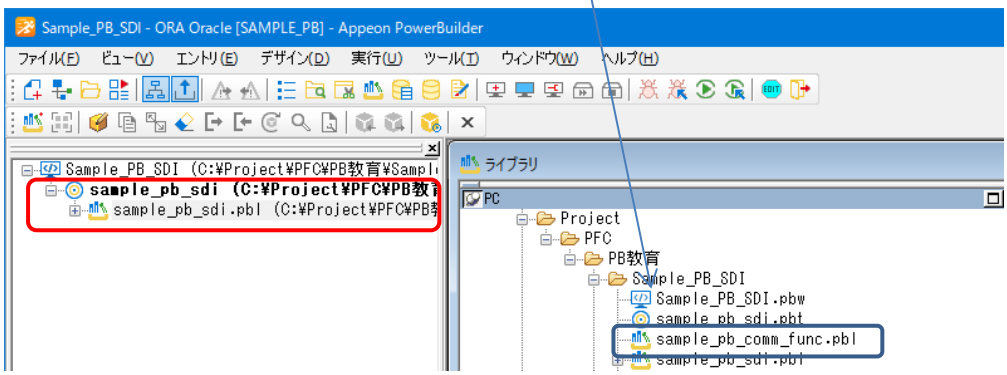
暫定メニューまで作成が済んだ段階で、次のウインドウ作成のために、PBL ファイルを追加する。
機能単位、画面や処理の単位に PBL ファイルを分けた開発を推奨する。

PBL ファイルを追加した後、既存ターゲットのライブラリリストに PBL パスを追加する必要がある。

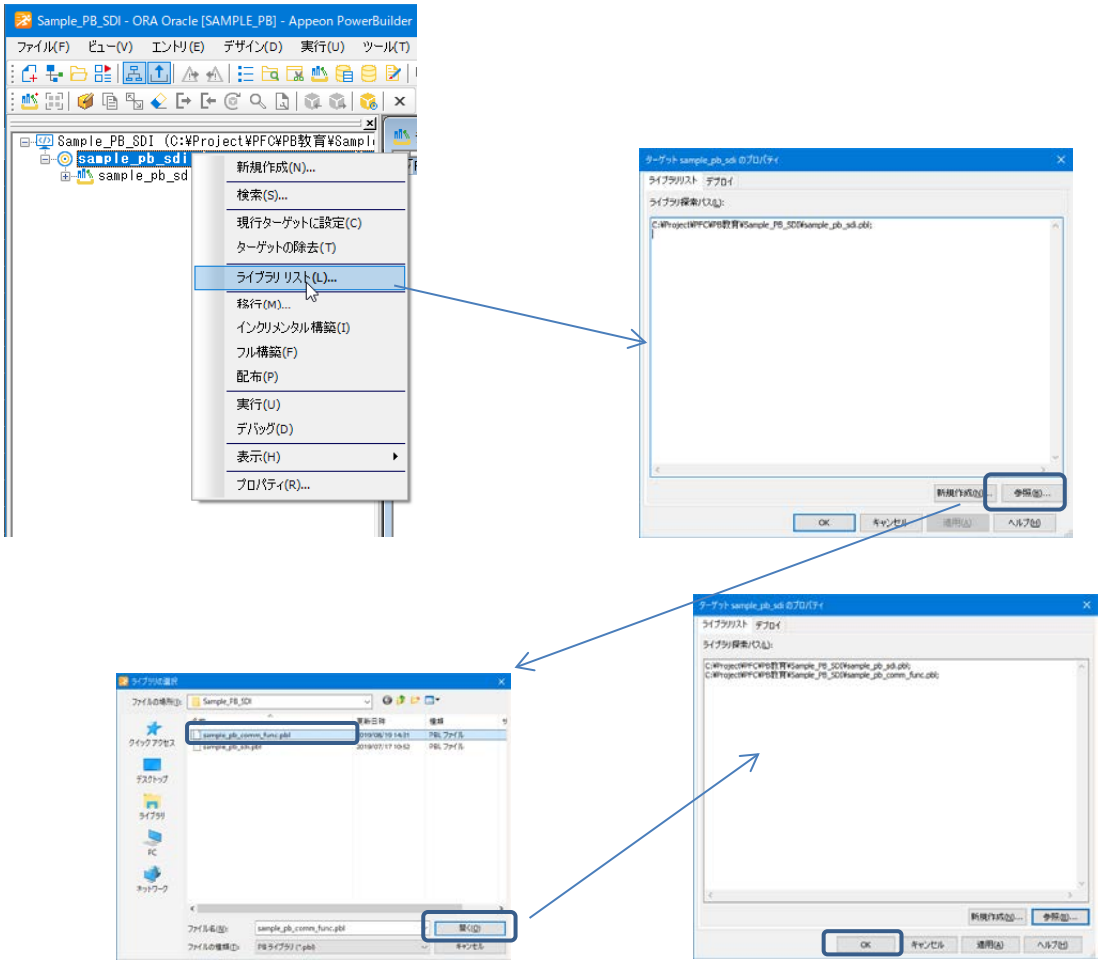




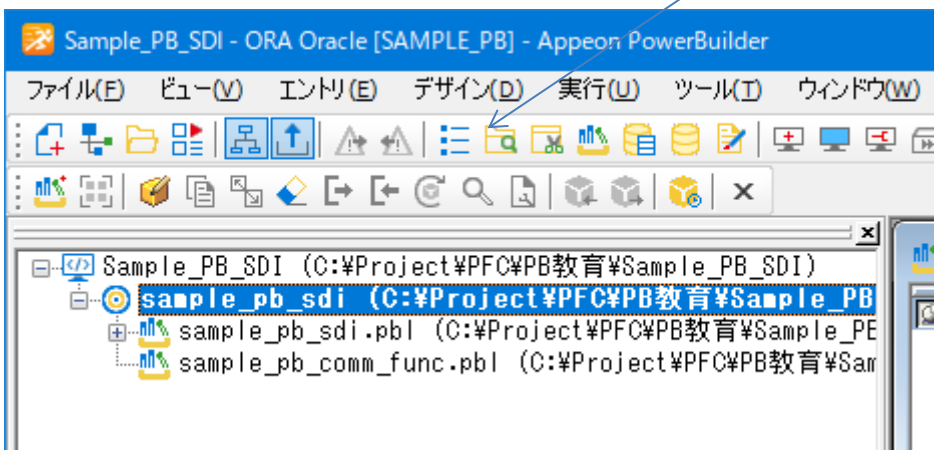
PBL は作成されるが、ライブラリリストにない状態



ライブラリリストに登録する。



ライブラリリストに登録された。

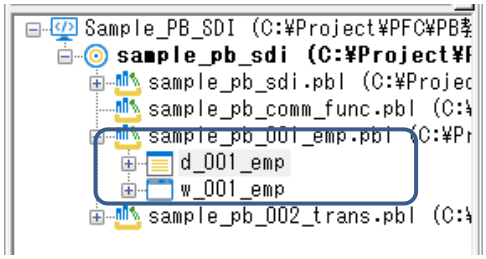


sample_pb_comm_func.pbl と同じ手順で下記 PBL を追加する。

・sample_pb_001_emp.pbl
・sample_pb_002_trans.pbl

（１１）従業員メンテナンスの作成

オブジェクト保存の例

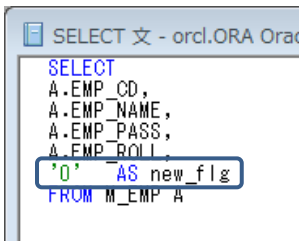


作成するウインドウのイメージ

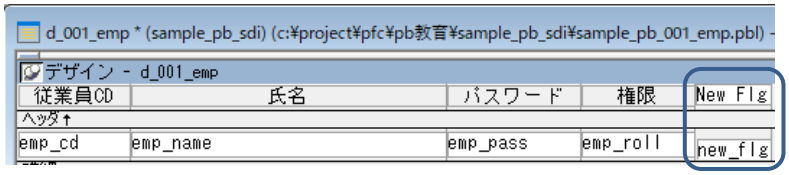
① データウインドウの作成（d_001_emp）

グリッドで作成する。

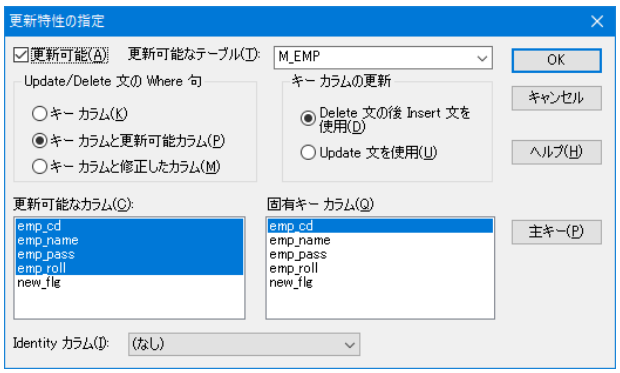
ここでは、制御用のフラグ new_flg を select 文に組み込む。★



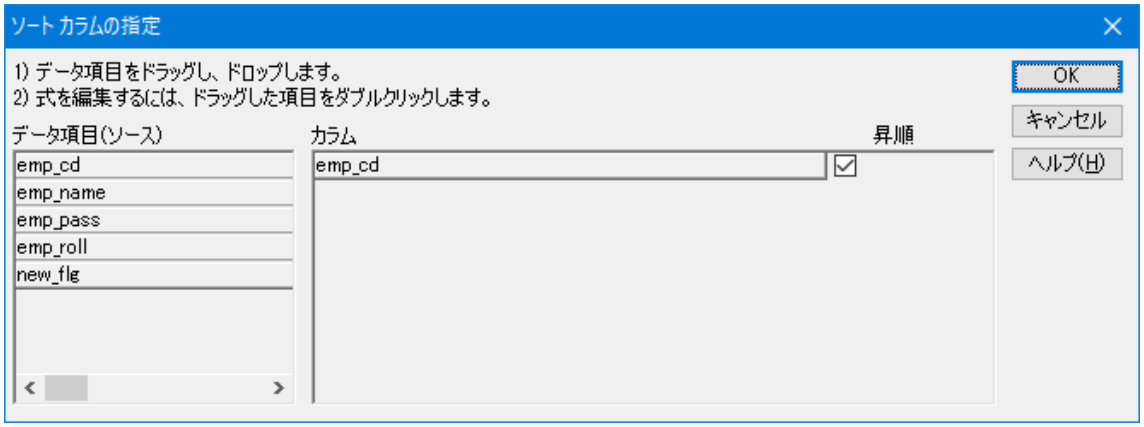
見出しの体裁を整え、制御用のフラグはデザイン画面から削除する。



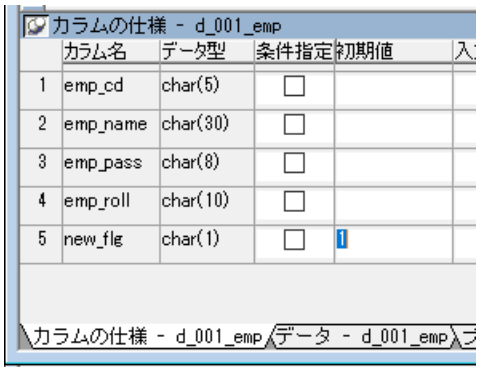
制御用のフラグ（new_flg）は更新特性から外す。



ソートカラムの指定



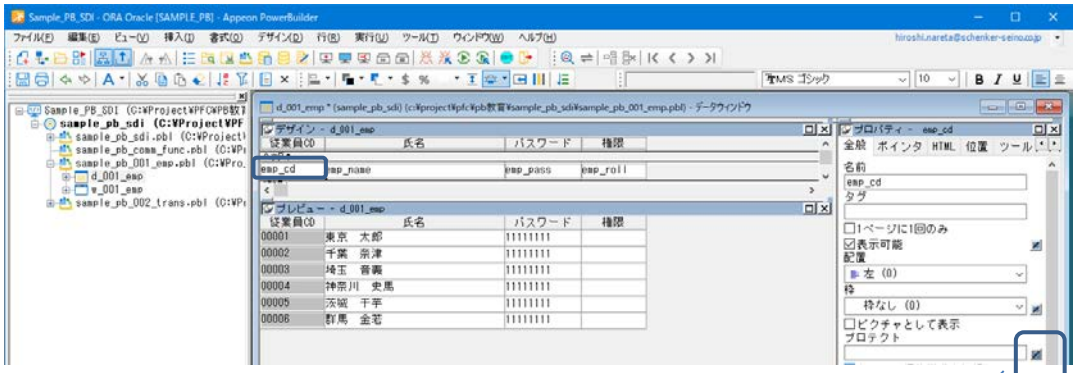
制御用のフラグ（new_flg）の初期値に 1 を設定する。



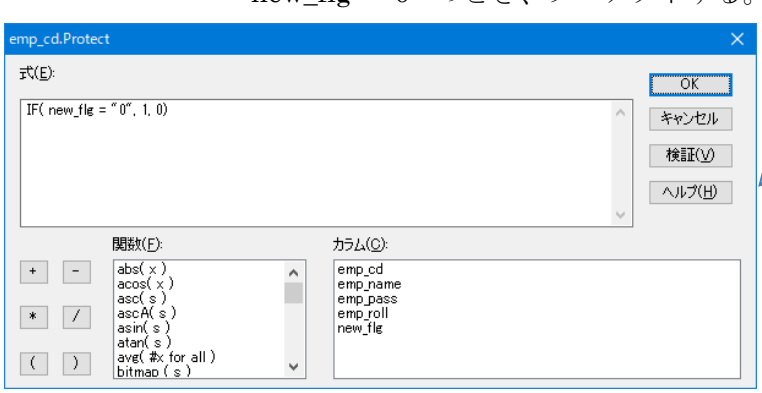
制御用のフラグ（new_flg）の意味

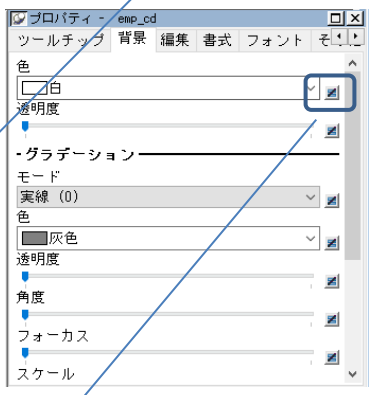
一つの開発手法であり、マスタメンテナンス画面用にデータウインドウを作成するときに役立つ。

SQL SELECT で検索した行は、new_flg = 0 である。
新規に行追加したときは、new_flg = 1 になる。
この情報を用いて、主キーに相当するカラムをプロテクトし、登録済のデータの主キー変更を許可しない方式をとれる。
また、背景色を変えることで既存行、新規行の区別が確認できる。

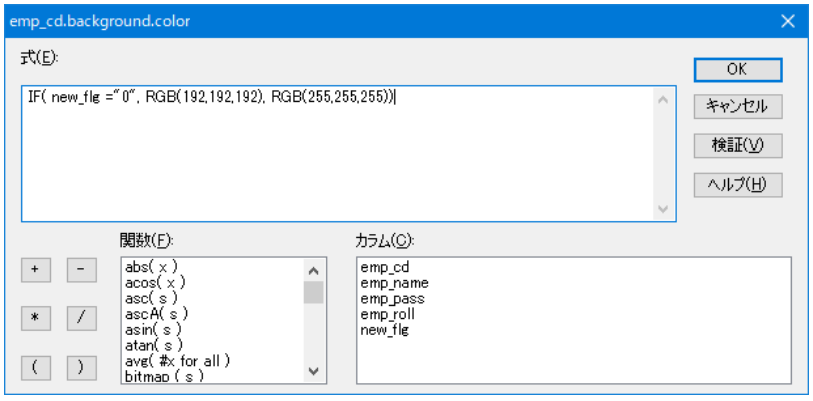


new_flg = “0” のとき、プロテクトする。

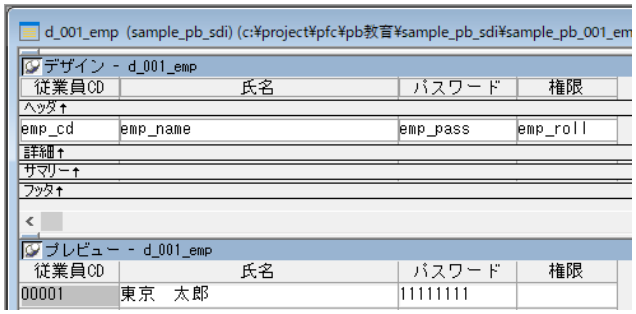




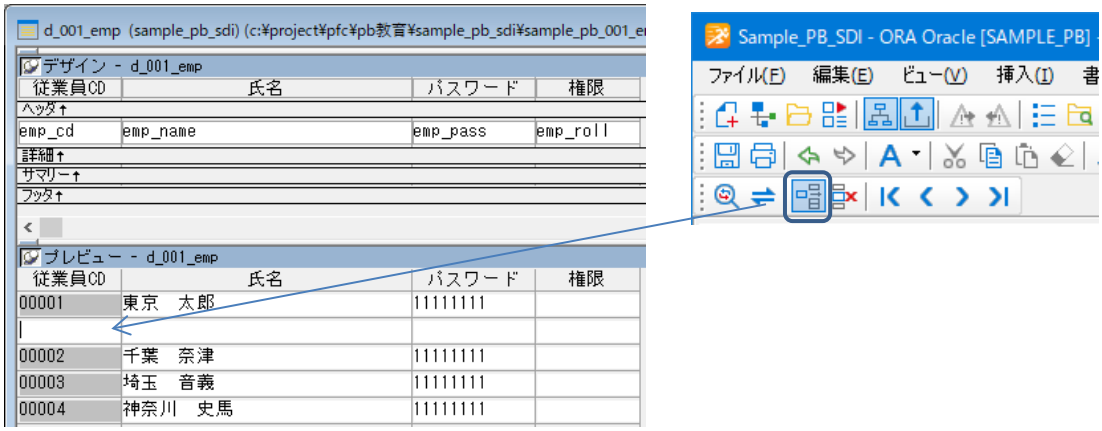
背景色を変更する。



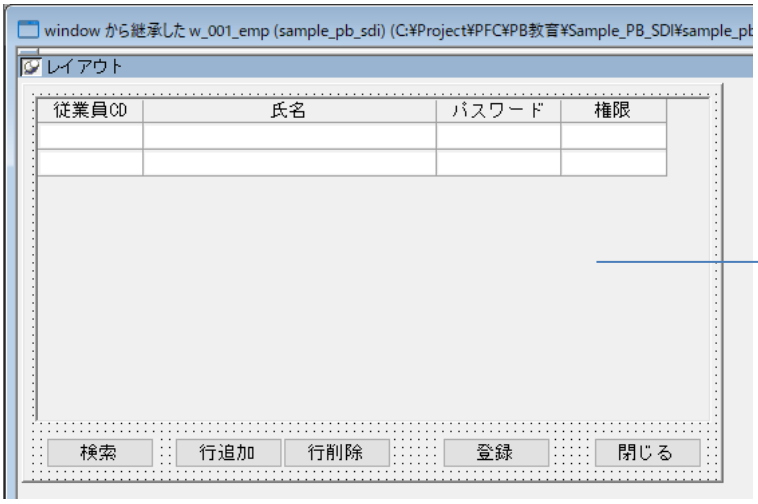
検索データの従業員 CD は、入力ができなく灰色になっている。

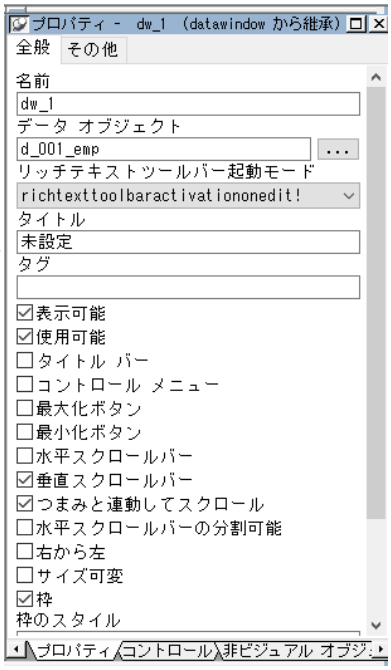


行追加された場合は、従業員 CD が入力可能になっている。



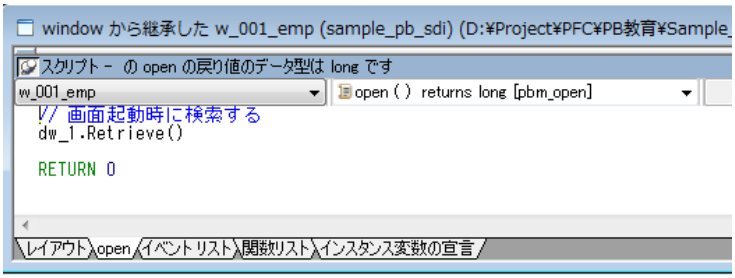
② ウインドウの作成（w_001_emp）
新規作成し、コントロールを配置する。



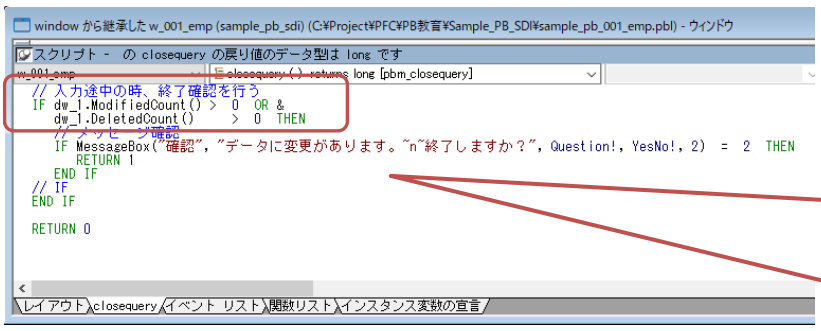


③ スクリプトの記載箇所
w_001_emp：ウインドウのイベント
dw_1：データウインドウのイベント
cb_1：検索ボタンのイベント
cb_2：行追加ボタンのイベント
cb_3：行削除ボタンのイベント
cb_4：登録ボタンのイベント
cb_5：閉じるボタンのイベント

④ w_001_emp
A) open イベント
データを検索する。



B) closequery イベント
画面を閉じる際に、画面データが編集中のときは、確認メッセージを表示する。



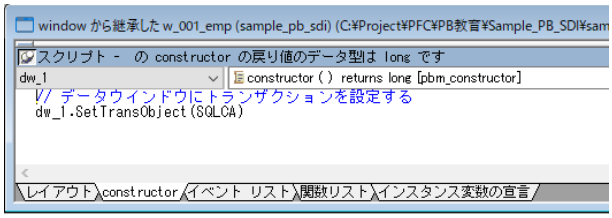
データ行の状態を取得
・dw_1.ModifiedCount()
修正済かつ未更新の変更データ行数を取得
Primary と Filter バッファ内データが対象
・dw_1.DeletedCount()
削除予定かつ未更新のデータ行数を取得
Delete バッファ内データが対象

⑤ dw_1：データウインドウ

A) constructor イベント

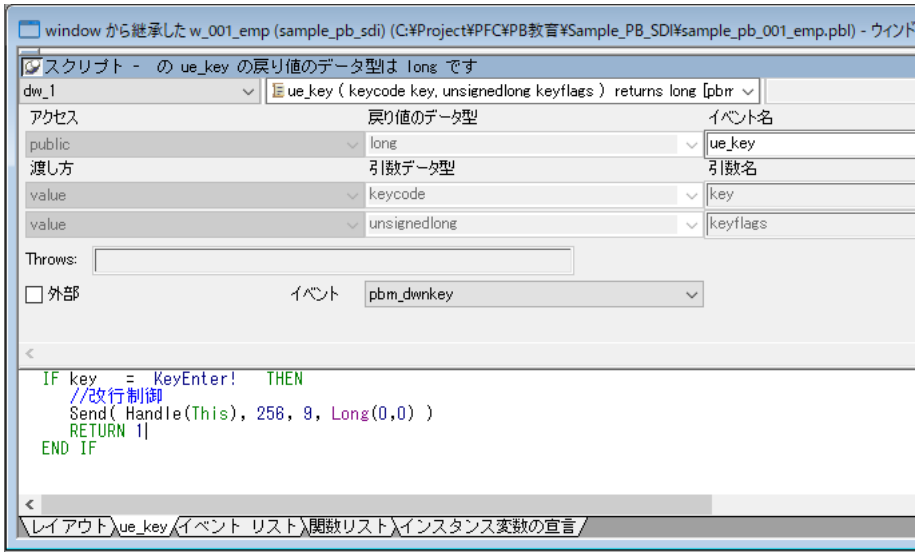
データウインドウをトランザクションで利用可能にする。
データウインドウで DB 操作をする際には必須である。

※当イベントは、ウインドウの open イベントの前に動作する。



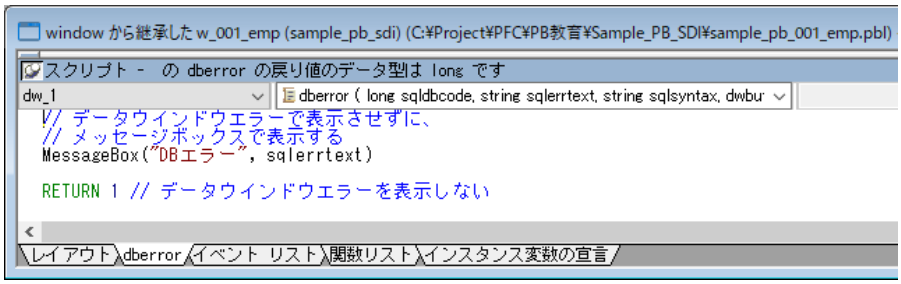
B) ユーザイベントの作成　ue_key

Enter キー押下でカラム移動する仕組みを入れる。



C) dberror イベント

データウインドウエラーの発生時、SQL ベンダーのエラー情報を表示する。

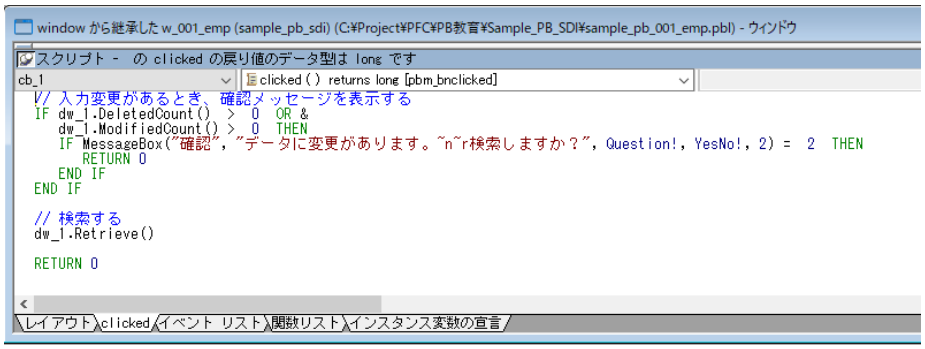


⑥ cb_1：検索

A) clicked イベント

データを検索する。

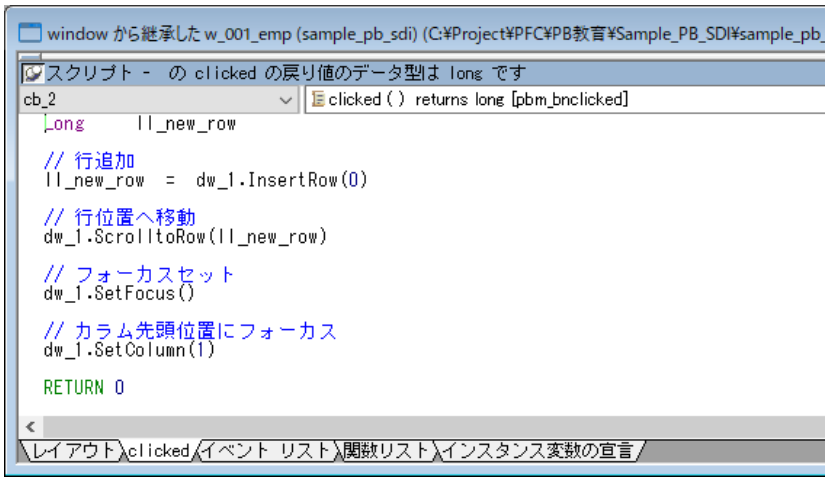
データウインドウに変更があるとき、確認メッセージを表示する。



⑦ cb_2：行追加

A) clicked イベント

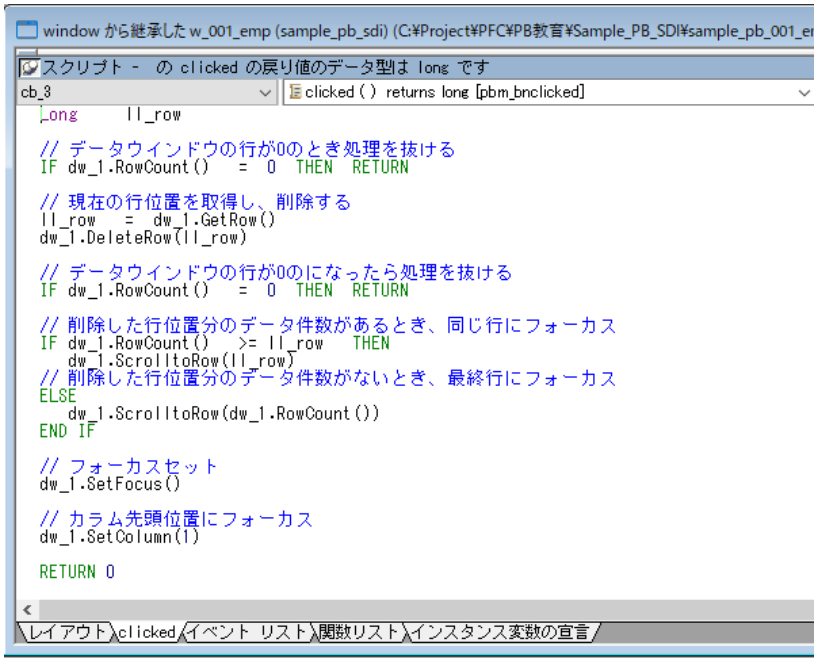
データウインドウに新規行を追加し、その行にフォーカスする。



⑧　cb_3：行削除

A)　clicked イベント

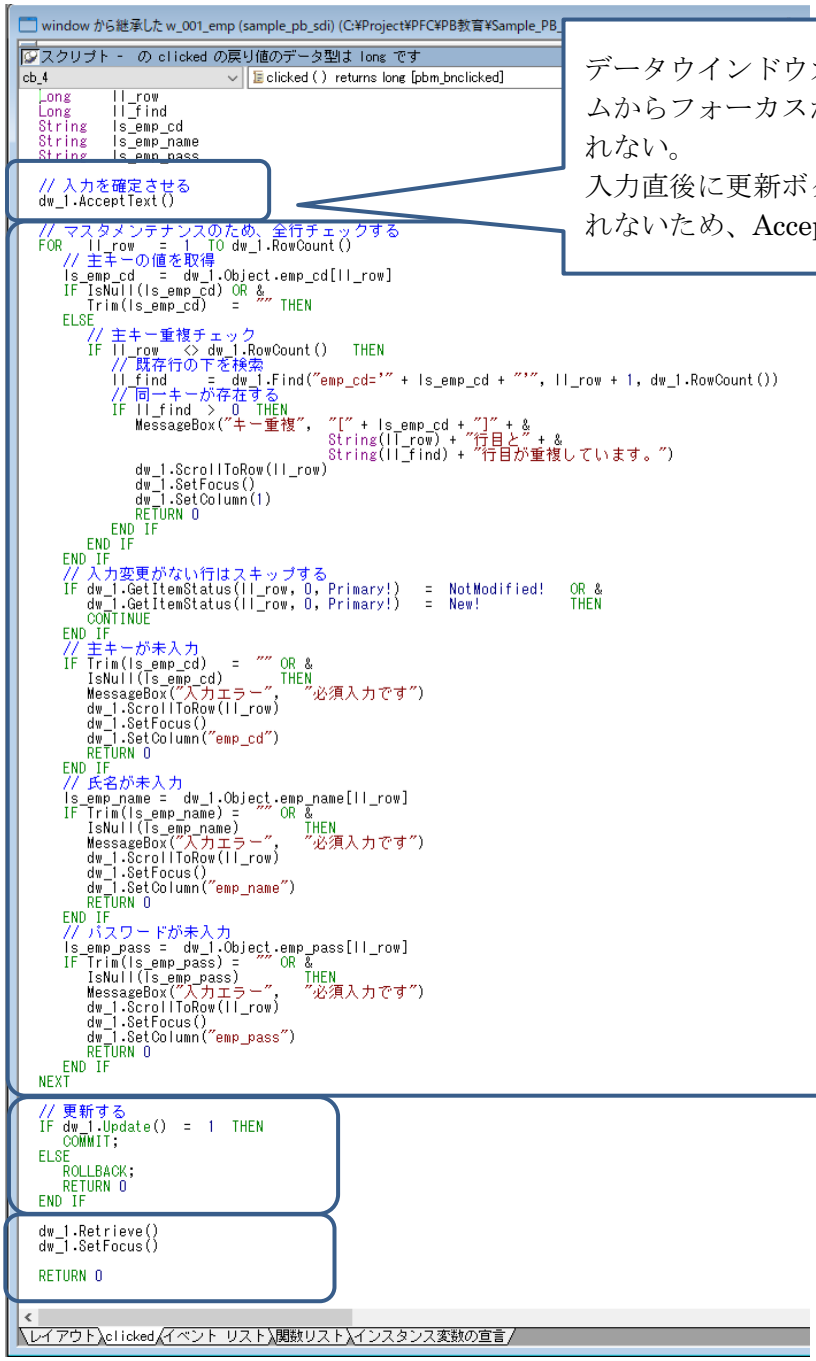
データウインドウの既存行を削除し、フォーカスする。



⑨ cb_4：更新

A) clicked イベント

データの入力チェックを行い、データを更新する。

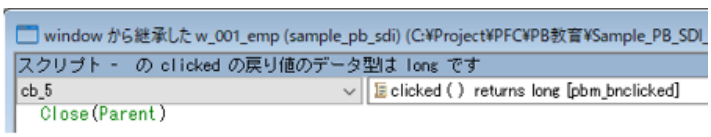


データウインドウカラムに入力した値は、カラムからフォーカスが移らなければ入力が確定されない。
入力直後に更新ボタンを押しても入力が確定されないため、AcceptText 関数を発行する。

⑩ cb_5：閉じる

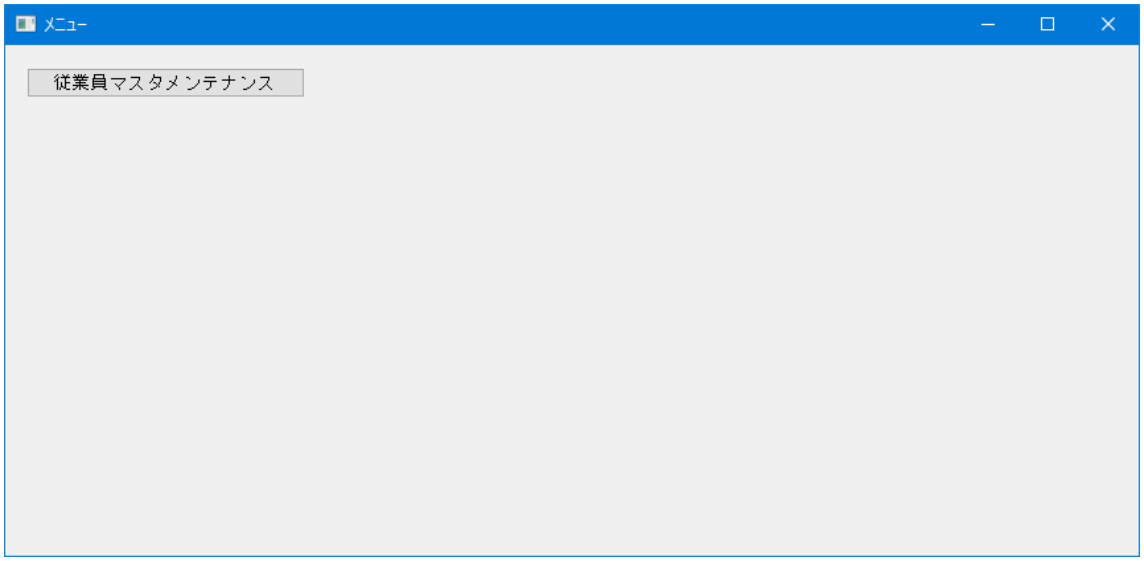
A) clicked イベント

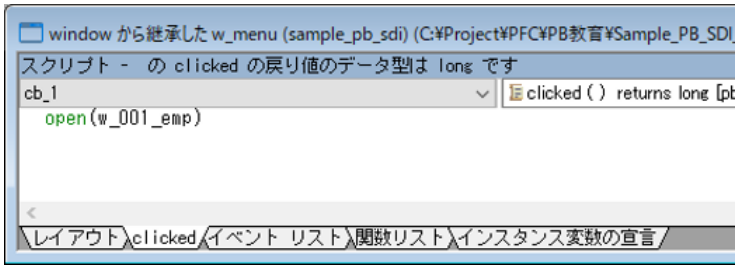
ウインドウを閉じる。



（１２）従業員マスタメンテナンスをメニューから起動

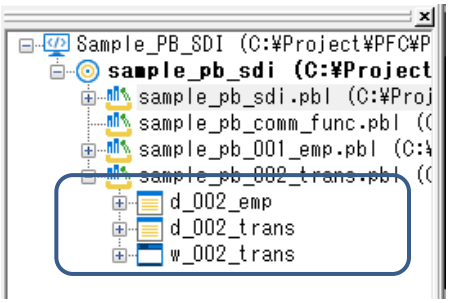
作成するウインドウのイメージ





（１３）交通費精算画面の作成

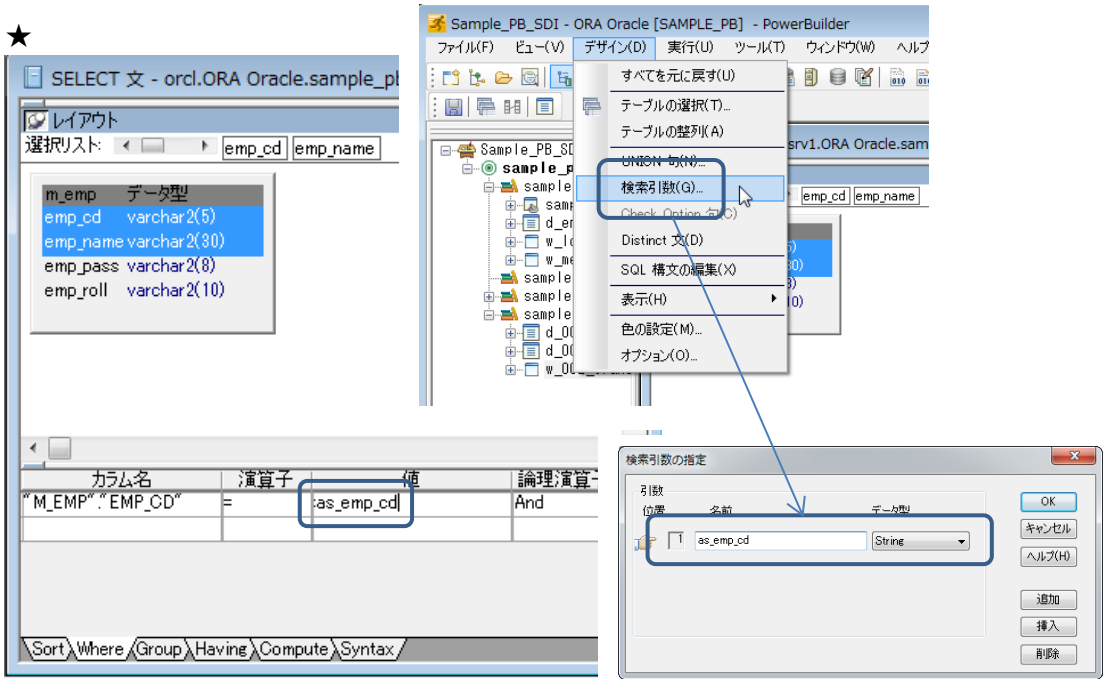
オブジェクト保存の例



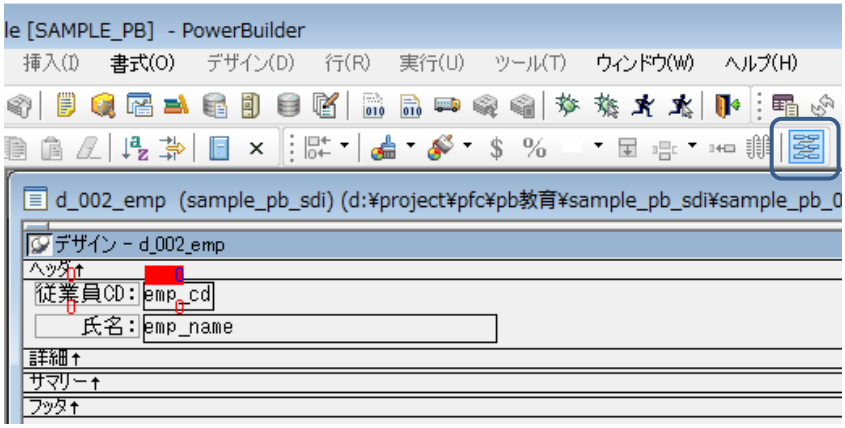
作成するウインドウのイメージ

① データウインドウ（画面ヘッダ部用）の作成（d_002_emp）
検索引数を持たせる。

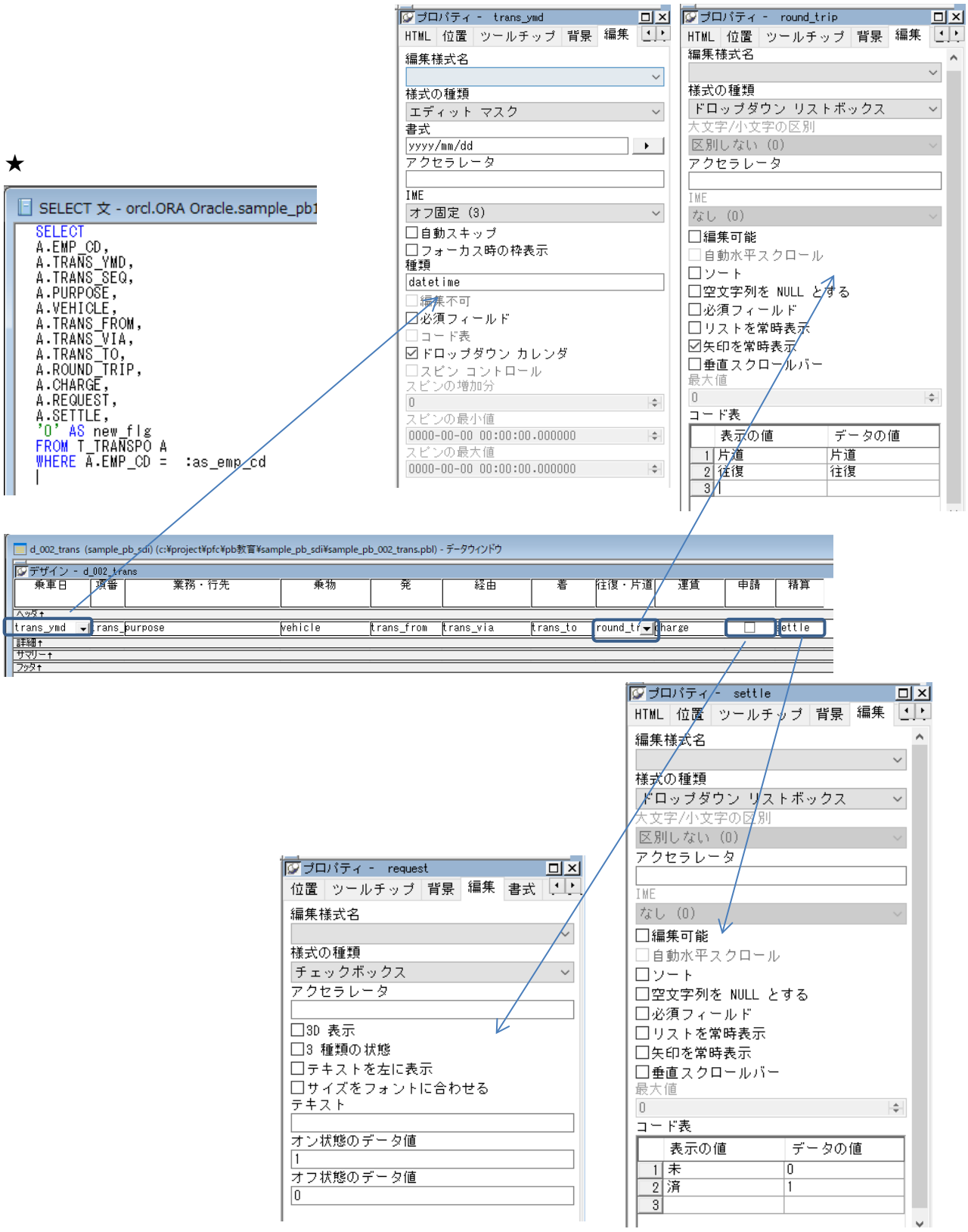
★



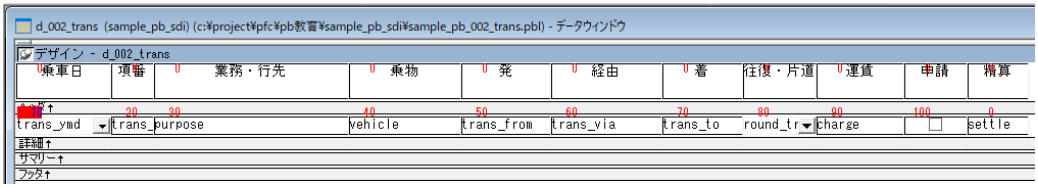
デザイン画面で入力不可能にする。
（タブシーケンスを全て 0 にする）★



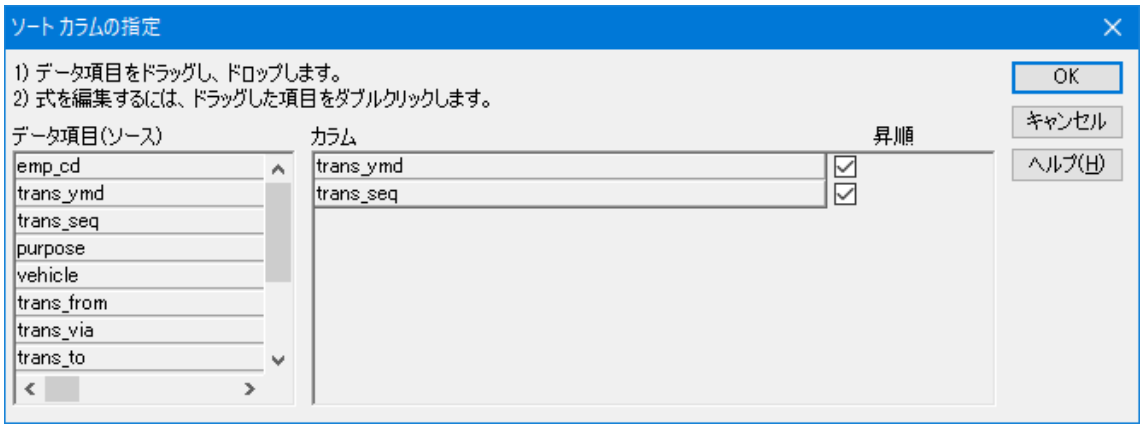
② データウインドウ（画面明細部用）の作成（d_002_trans）
（画面ヘッダ部用）と同様に検索引数を持たせる。



タブオーダー



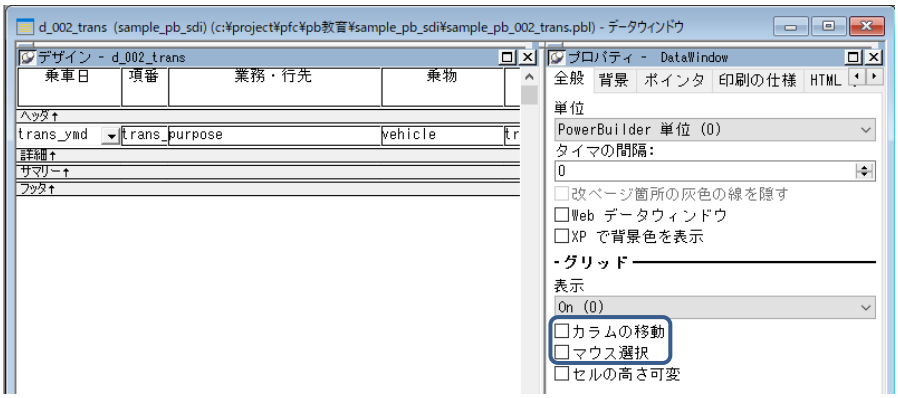
ソートカラムの指定



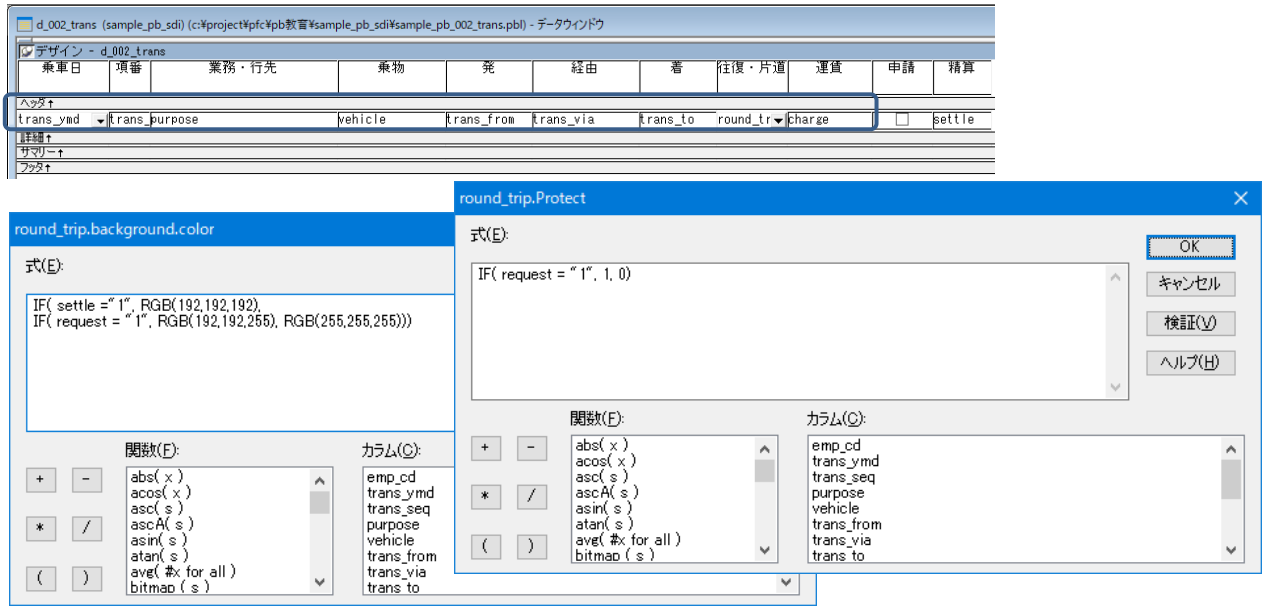
カラムの移動、マウスの選択のチェックを OFF

入力不可能な行であっても、表面上カラム選択ができているように見えてしまい、その行を削除するつもりで画面上の「削除ボタン」を押下したとき、実際にフォーカスが当たっている（入力可能な行）が削除されることから、意図しない動作として錯覚してしまうケースが考えられる。

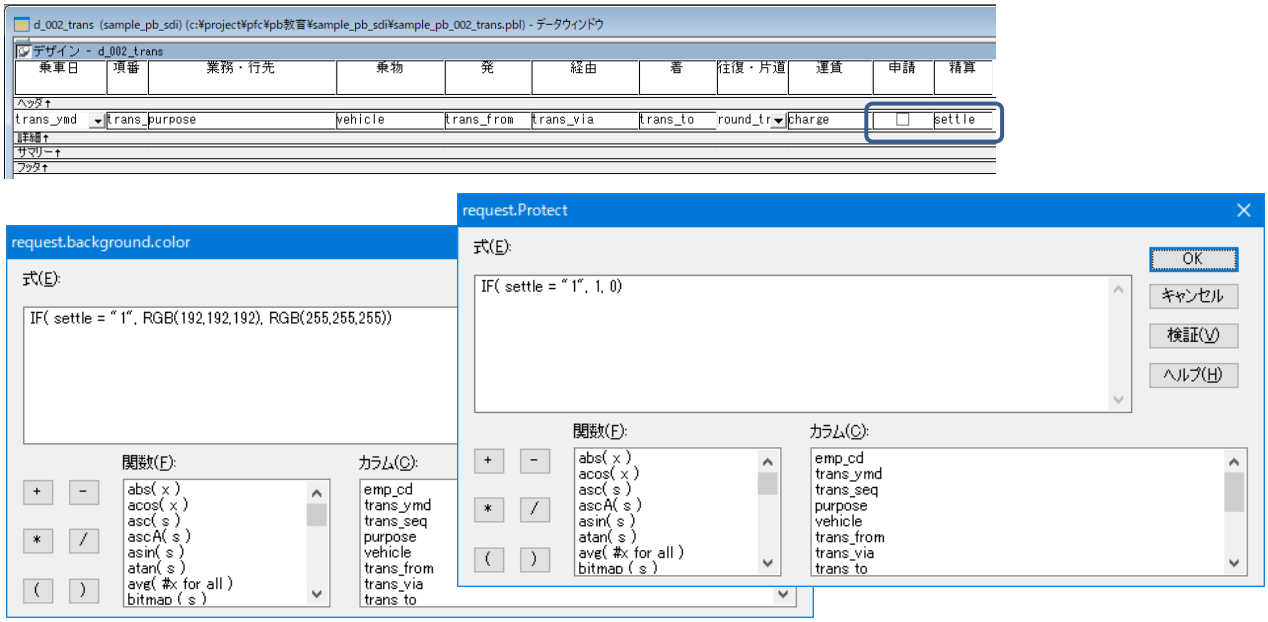
フォーカスが当たる、当たらない行が混在するケースの場合は、カラムの移動、マウスの選択のチェックを OFF にすることを推奨する。



プロテクト、背景色（乗車日～運賃）

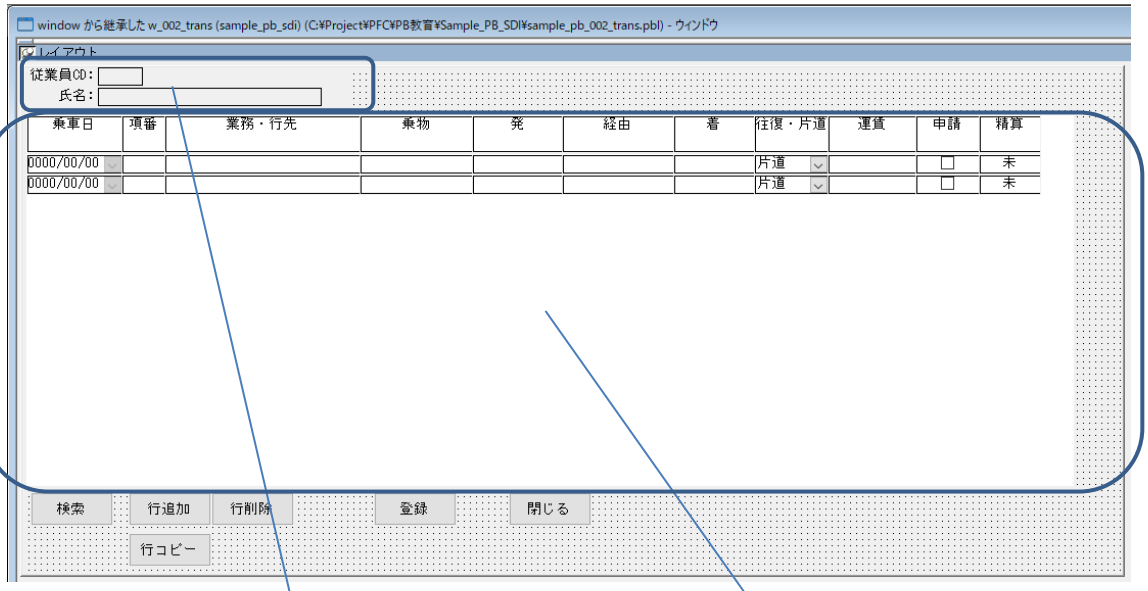


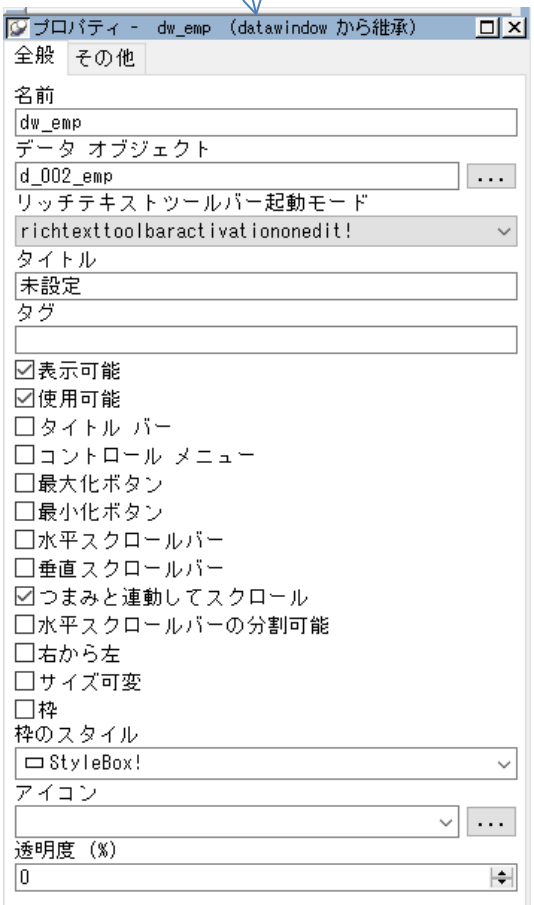
プロテクト、背景色（申請）

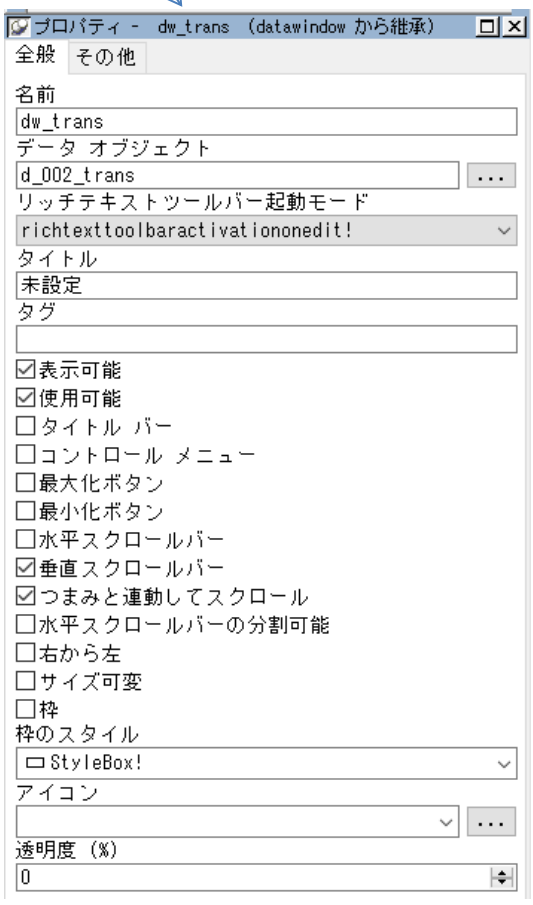


③　ウインドウの作成（w_002_trans）

新規作成し、コントロールを配置する。







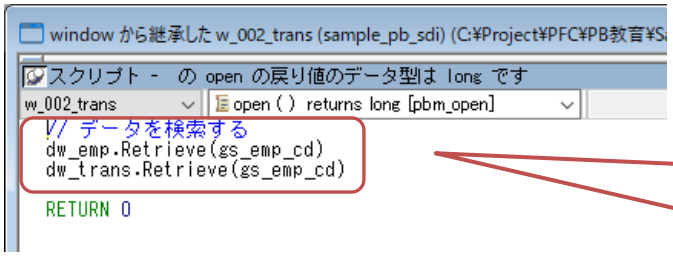
④　スクリプトの記載箇所

w_002_trans：ウインドウのイベント
dw_emp：画面ヘッダ部データウインドウのイベント
dw_trans：画面明細部データウインドウのイベント
cb_1：検索ボタンのイベント
cb_2：行追加ボタンのイベント
cb_3：行削除ボタンのイベント
cb_4：行コピーボタンのイベント
cb_5：登録ボタンのイベント
cb_6：閉じるボタンのイベント

⑤　w_002_trans

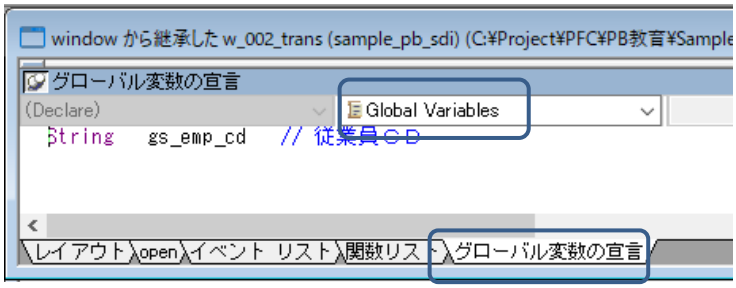
A)　open イベント

ログインユーザでデータ検索する。

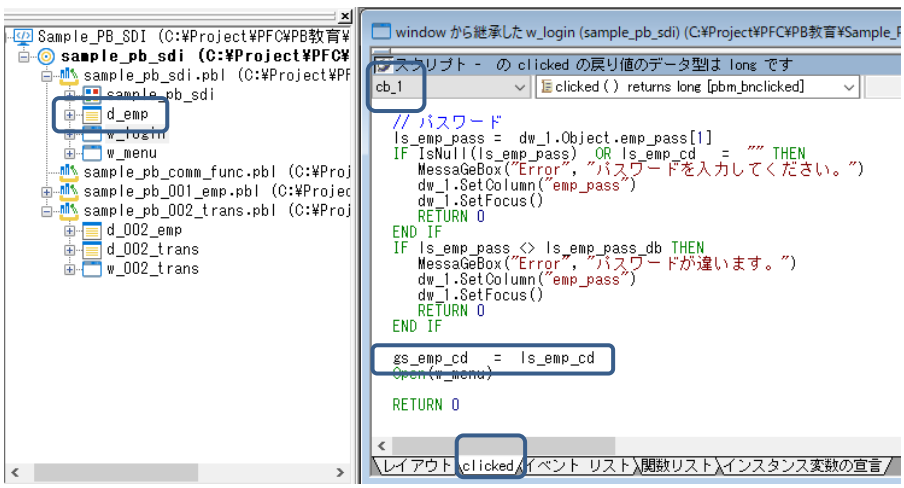


データウィンドウ側で用意した
検索引数「：as_emp_cd」に
変数「gs_emp_cd」の値を条件にして
各データウィンドウを検索する。

上記の記載前に・・・
gs_emp_cd のグローバル変数を宣言

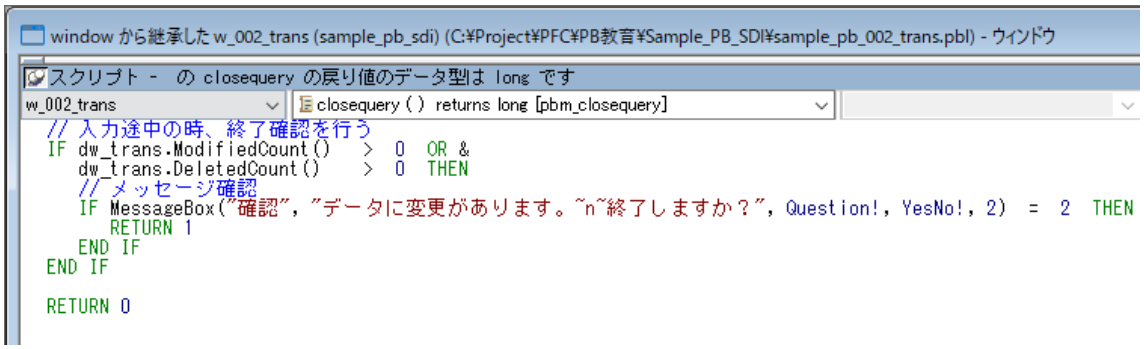


ログインが成功したタイミングでグローバル変数に値をセット



B) closequery イベント

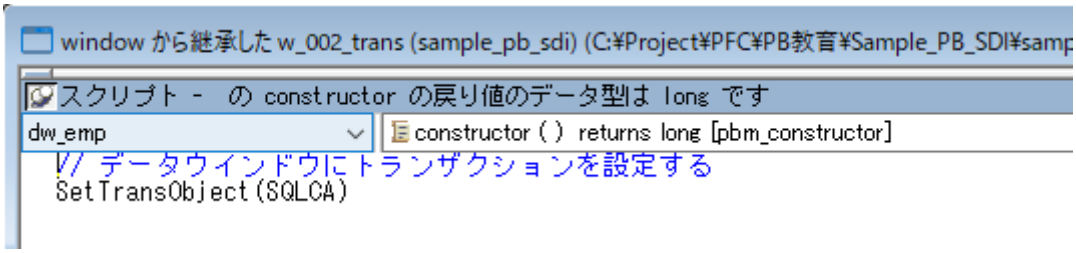
画面を閉じる際に、画面データが編集中のときは、確認メッセージを表示する。



⑥ dw_emp：画面ヘッダ部データウインドウ

A) constructor イベント

データウインドウをトランザクションで利用可能にする。

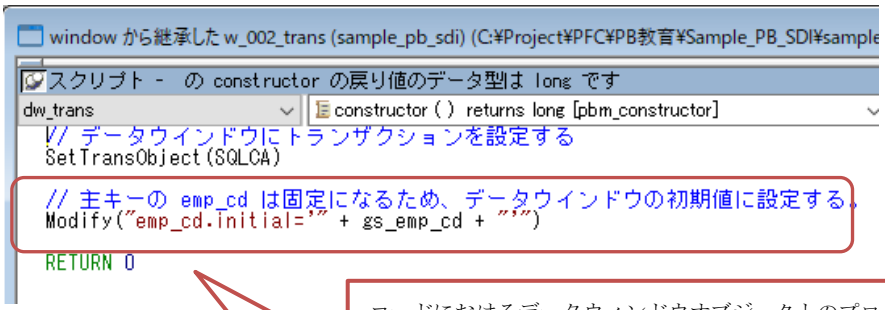


⑦ dw_trans：画面明細部データウインドウ

A) constructor イベント

データウインドウをトランザクションで利用可能にする。

emp_cd はログイン ID のため、データウインドウの初期値として設定する。



コードにおけるデータウィンドウオブジェクトのプロパティ値のアクセス方法

例）①新たに挿入された行の従業員カラムの初期値を設定する場合

②カラムの初期値を取得する場合

・メソッドによるアクセス

Describe メソッドと Modify メソッドは文字列を使用して、プロパティ名を指定できる

①Modify("カラム名.initial = ' " ～)

②Describe("カラム名.initial ")

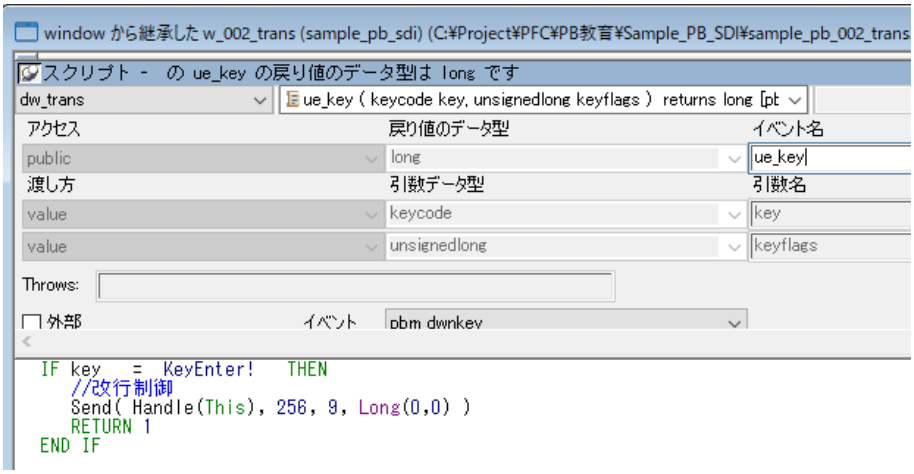
・Object プロパティとドット（.）表記

①dw_trans.Object.emp_cd.initial = gs_emp_cd

②ls_initial = dw_trans.Object.emp_cd.initial

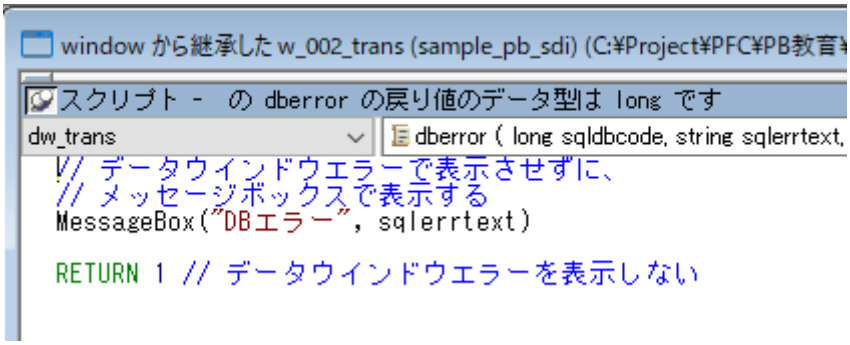
B) ユーザイベントの作成 ue_key

Enter キー押下でカラム移動する仕組みを入れる。



C) dberror イベント

データウインドウエラーの発生時、SQL ベンダーのエラー情報を表示する。

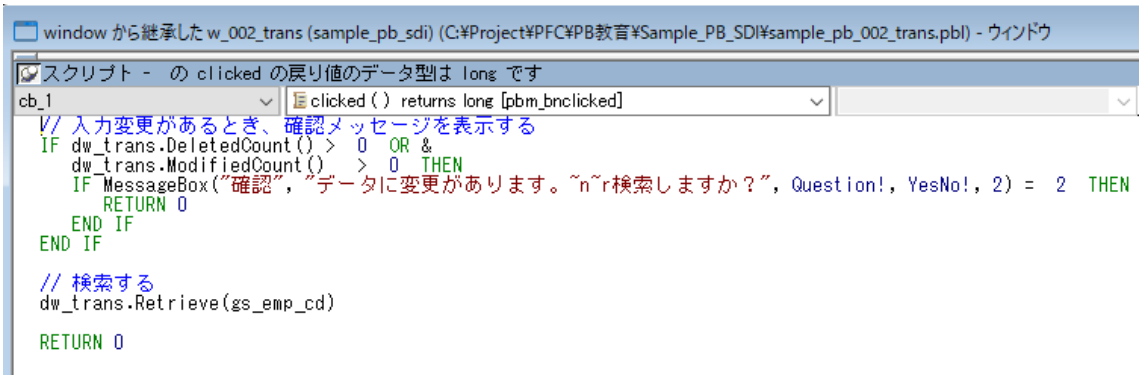


⑧ cb_1：検索

A) clicked イベント

データを検索する。

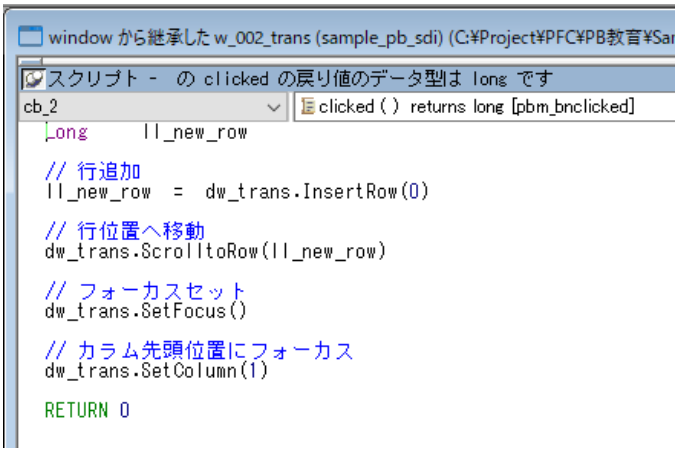
データウインドウに変更があるとき、確認メッセージを表示する。



⑨ cb_2：行追加

A) clicked イベント

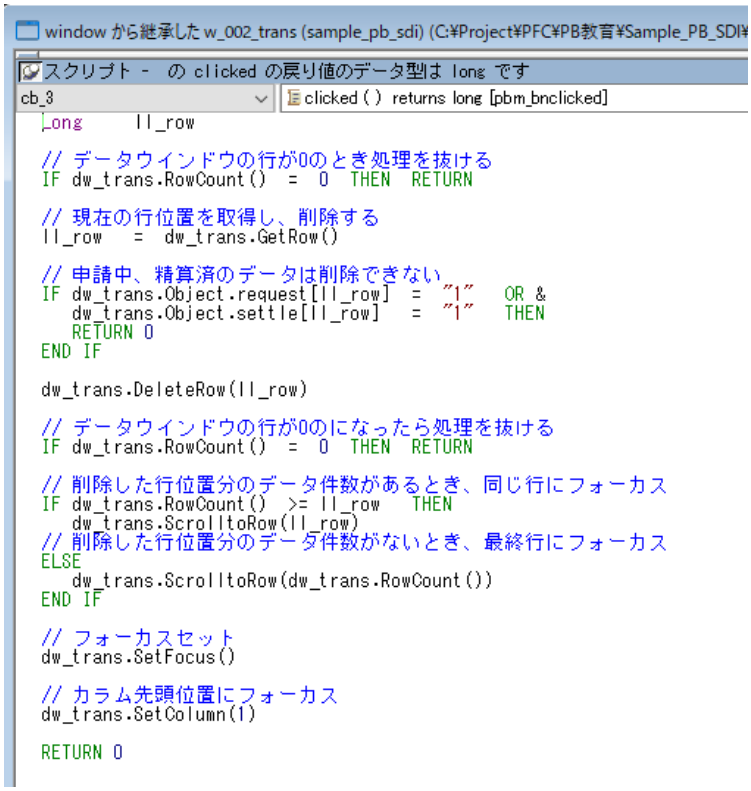
データウインドウに新規行を追加し、その行にフォーカスする。



⑩ cb_3：行削除

A) clicked イベント

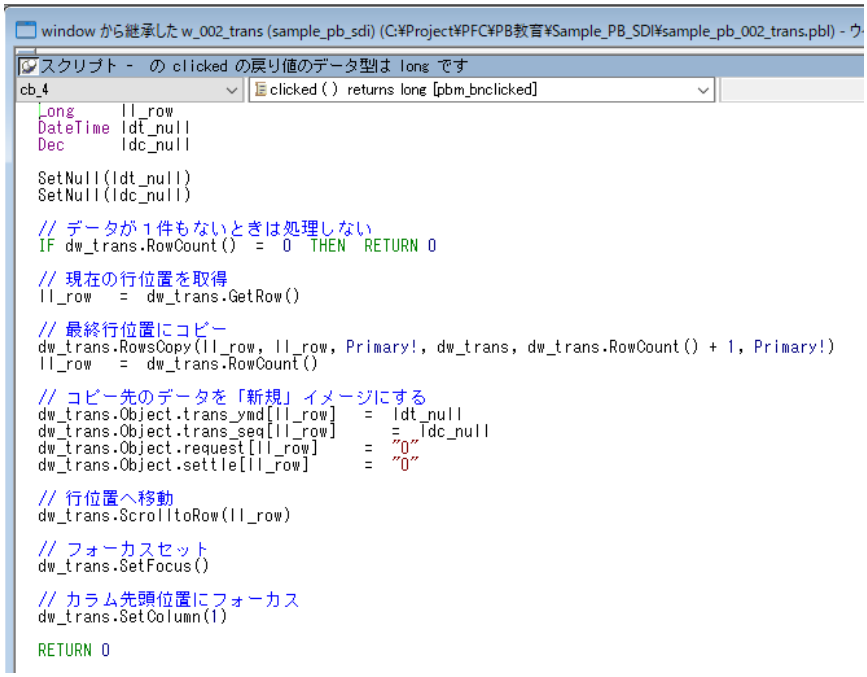
データウインドウの既存行を削除し、フォーカスする。
申請中、精算済のデータは削除できない。



⑪ cb_4：行コピー

A) clicked イベント

選択された行をコピーし、新規データとして編集中の状態にする。



⑫ cb_5：更新

A) clicked イベント

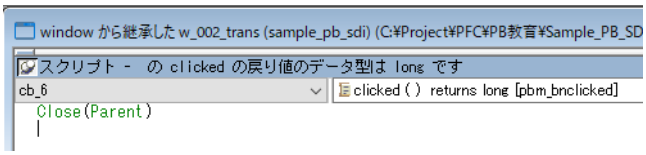
データの入力チェックを行い、データを更新する。


```
window から継承した w_002_trans (sample_pb_sdi) (C:¥Project¥PFC¥PB教育¥Sample_PB_SDI¥sample_pb_002_trans.pbl) - ウィンドウ
スクリプト -  の clicked の戻り値のデータ型は long です
cb_5                                clicked ( ) returns long [pbm_bnclicked]
  Long     ll_row
  Long     ll_find
  DateTime ldt_trans_ymd
  Long     ll_trans_seq
  String   ls_string
  Dec      ldc_decimal

  // 入力を確定させる
  dw_trans.AcceptText()

  // 全行チェックする
  FOR   ll_row   = 1  TO dw_trans.RowCount()
     // 主キー重複チェック
     ldt_trans_ymd  =  dw_trans.Object.trans_ymd[ll_row]
     ll_trans_seq   =  dw_trans.Object.trans_seq[ll_row]
     // 項番のNull扱う
     IF ll_trans_seq   =  0  THEN
        SetNull(ll_trans_seq)
     END IF
     IF IsNull(ldt_trans_ymd)   AND   IsNull(ll_trans_seq)    THEN
     ELSE
        IF ll_row   <> dw_trans.RowCount()  THEN
           // 既存行の下を検索
           ll_find     =  dw_trans.Find( "trans_ymd=DateTime(Date('"  + String(ldt_trans_ymd, "yyyy/mm/dd") + "'),Time('00:00:00'))"  &
                         +  "  AND  trans_seq="  + String(ll_trans_seq), ll_row + 1, dw_trans.RowCount())
           // 同一キーが存在する
           IF ll_find  >  0  THEN
              MessageBox("キー重複",  String(ldt_trans_ymd, "yyyy/mm/dd") + "[" + String(ll_trans_seq) + "]" + &
                                      String(ll_row) + "行目と" + &
                                      String(ll_find) + "行目が重複しています。")
              dw_trans.ScrollToRow(ll_row)
              dw_trans.SetFocus()
              dw_trans.SetColumn(1)
              RETURN 0
           END IF
        END IF
     END IF

     // 入力変更がない行はスキップする
     IF dw_trans.GetItemStatus(ll_row, 0, Primary!)  =  NotModified!   OR &
        dw_trans.GetItemStatus(ll_row, 0, Primary!)  =  New!           THEN
        CONTINUE
     END IF

     // 乗車日のチェック
     ldt_trans_ymd  =  dw_trans.Object.trans_ymd[ll_row]
     IF IsNull(ldt_trans_ymd)   THEN
        MessageBox("入力エラー",  "必須入力です")
        dw_trans.ScrollToRow(ll_row)
        dw_trans.SetFocus()
        dw_trans.SetColumn("trans_ymd")
        RETURN 0
     END IF

     // 項番のチェック
     ll_trans_seq   =  dw_trans.Object.trans_seq[ll_row]
     IF IsNull(ll_trans_seq) OR &
        ll_trans_seq   <= 0  THEN
        MessageBox("入力エラー",  "必須入力です（1,2,3・・の順で入力）")
        dw_trans.ScrollToRow(ll_row)
        dw_trans.SetFocus()
        dw_trans.SetColumn("trans_seq")
        RETURN 0
     END IF

     // 業務・行先が未入力
     ls_string   =  dw_trans.Object.purpose[ll_row]
     IF Trim(ls_string)   =  "" OR &
        IsNull(ls_string)       THEN
        MessageBox("入力エラー",  "必須入力です")
        dw_trans.ScrollToRow(ll_row)
        dw_trans.SetFocus()
        dw_trans.SetColumn("purpose")
        RETURN 0
     END IF
     // 発が未入力
     ls_string   =  dw_trans.Object.trans_from[ll_row]
     IF Trim(ls_string)   =  "" OR &
        IsNull(ls_string)       THEN
        MessageBox("入力エラー",  "必須入力です")
        dw_trans.ScrollToRow(ll_row)
        dw_trans.SetFocus()
        dw_trans.SetColumn("trans_from")
        RETURN 0
     END IF
     // 着が未入力
     ls_string   =  dw_trans.Object.trans_to[ll_row]
     IF Trim(ls_string)   =  "" OR &
        IsNull(ls_string)       THEN
        MessageBox("入力エラー",  "必須入力です")
        dw_trans.ScrollToRow(ll_row)
        dw_trans.SetFocus()
        dw_trans.SetColumn("trans_to")
        RETURN 0
     END IF
     // 運賃が未入力
     ldc_decimal =  dw_trans.Object.charge[ll_row]
     IF ldc_decimal =  0    OR &
        IsNull(ldc_decimal)  THEN
        MessageBox("入力エラー",  "必須入力です")
        dw_trans.ScrollToRow(ll_row)
        dw_trans.SetFocus()
        dw_trans.SetColumn("charge")
        RETURN 0
     END IF
  NEXT

  // 更新する
  IF dw_trans.Update() =  1  THEN
     COMMIT;
  ELSE
     ROLLBACK;
     RETURN 0
  END IF

  dw_trans.Retrieve(gs_emp_cd)
  dw_trans.SetFocus()

  RETURN 0
```

⑬　cb_6：閉じる

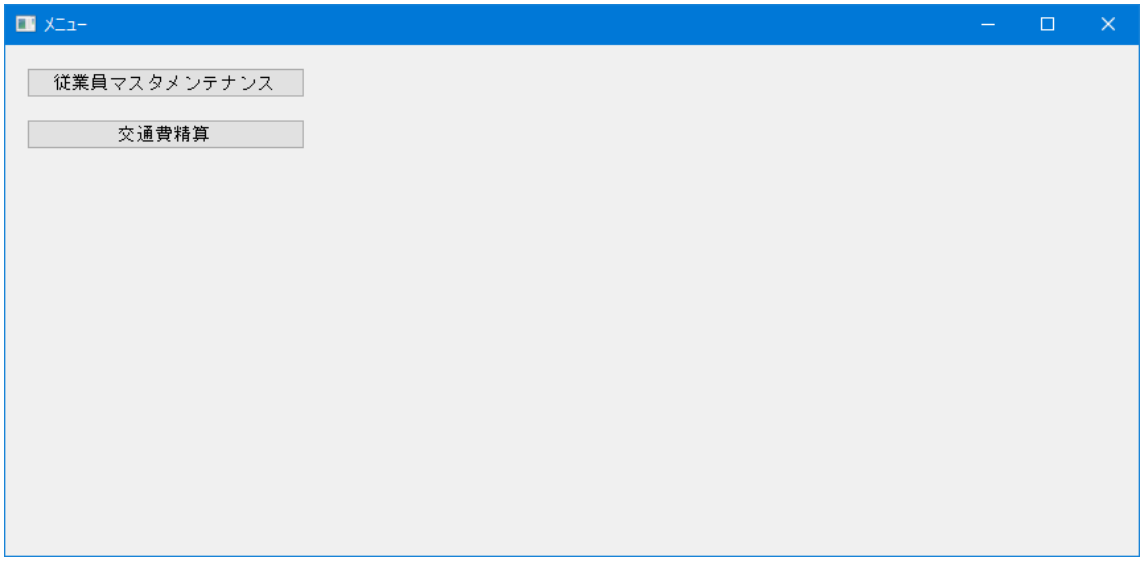
A)　clicked イベント

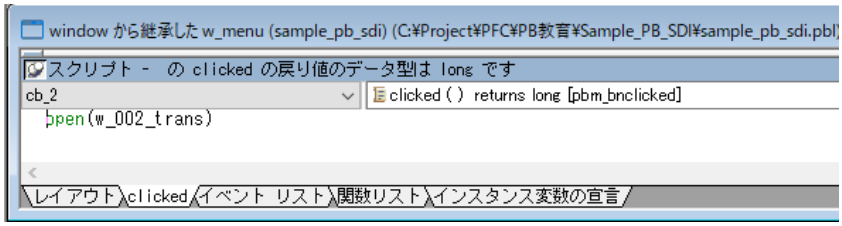
ウインドウを閉じる。



（１４）交通費精算画面をメニューから起動

作成するウインドウのイメージ





４．　まとめ

本書にて、Appeon PowerBuilder 2017 アプリケーションの作成について解説してきた。

Appeon PowerBuilder 2017 を用いた初歩的な手法であるが、C/S システムの構築方法が理解できたと想定し、次回のコースでは開発効率を向上するための手法について説明する。

Appeon PowerBuilder 2017 クラスを用いるメリットと、データウインドウの有効利用について解説する。

以上