Public

SQL Anywhere データベース移行

-Power Builderアプリ　概要編

SAPジャパン株式会社

Sep. 28, 2016

Agenda

- 本セッションの対象
- 移行手順概要
- テーブル・データの移行
- SQLの移行とポイント
- 参考資料：SAP Exodusのご紹介
- Additional：プロシージャ・ユーザー定義関数の移行概要

本セッションの対象

本セッションの対象

本セッションではPower Builderアプリケーションで使用しているデータベースをOracleからSQL Anywhereに移行する上での概要を解説します。

- SQL AnywhereはWATCOM SQLとTransact SQLの2種類のSQLを使用可能ですが、このセッションではWATCOM SQLへの移行を解説しています。

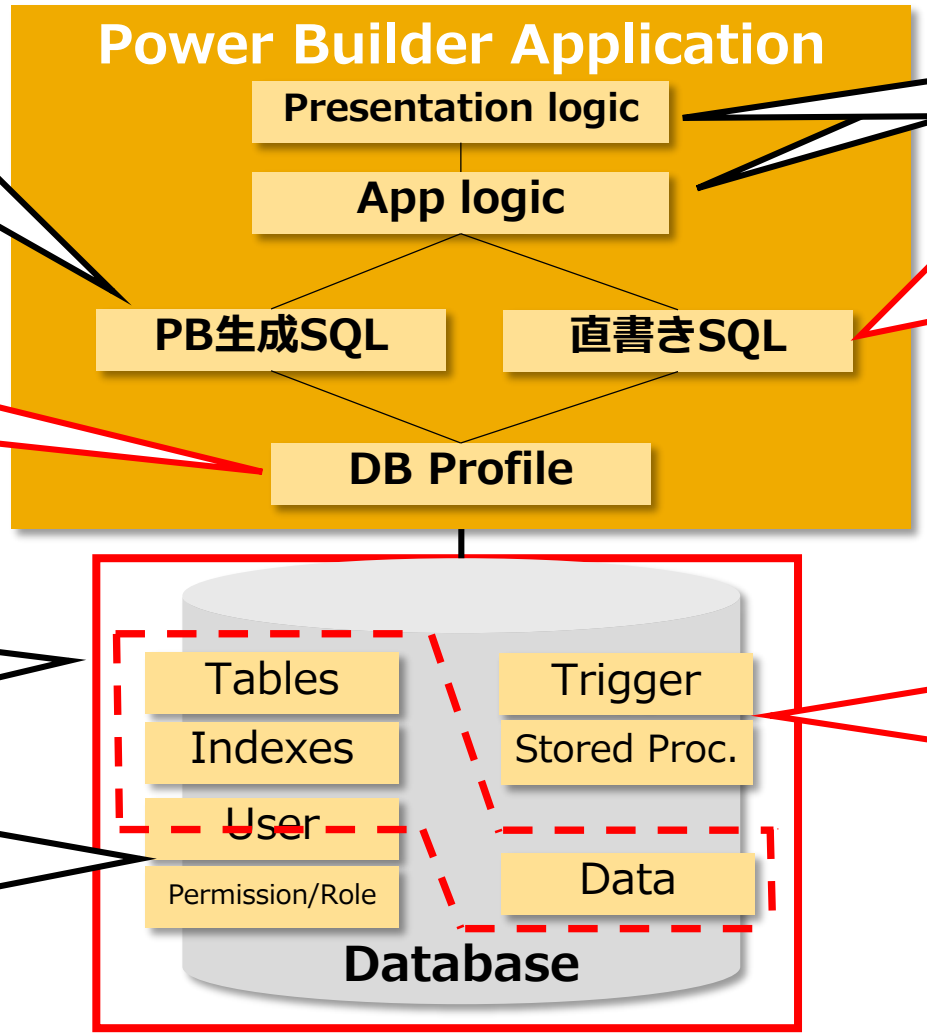
定期的に行われているSQL Anywhereセミナーでは

- SQL Anywhereデータベース移行セッション
- SQL Anywhereデータベース移行セッション Advanced

という別セッションもあります。Power Builderアプリでも適用できる内容がありますので是非ご参加ください。

移行手順概要

Power Builderアプリケーションでの DB移行で移行するもの（From Oracle）



コネクションを変更することで自動で変換がかかる

例：QuickSelectのDatawindow

基本的に変更の必要なし

SQL Anywhere用に変更

先程のNCS様セッションで解説

コード中にSQLを直接書いてしまっている部分は全てチェック要

SQL CentralのDB移行ツールで移行

（点線はその際に移行できる）

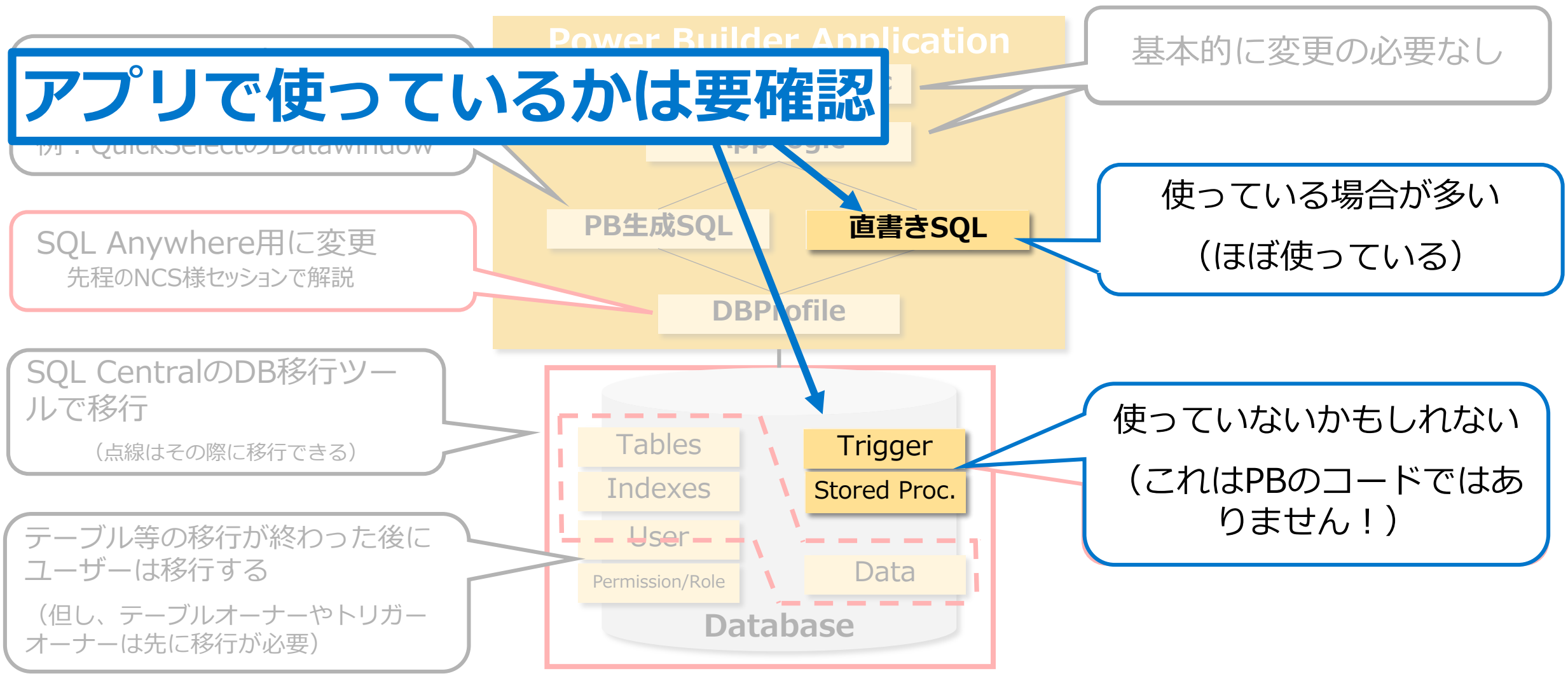
移行要

量が多ければ大変

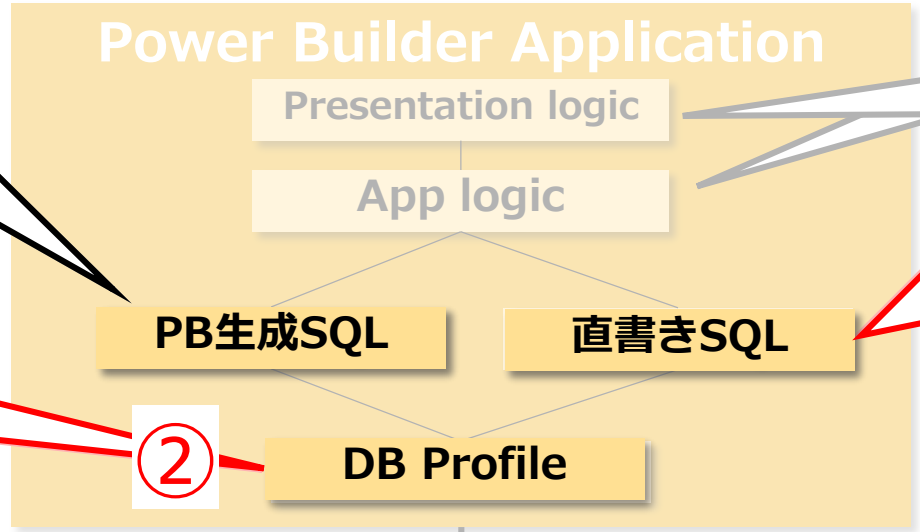
テーブル等の移行が終わった後にユーザーは移行する

（但し、テーブルオーナーやトリガーオーナーは先に移行が必要）

移行するアプリで確認してください！



大まかな移行順番



コネクションを変更することで自動で変換がかかる ③

例：QuickSelectのDatawindow

SQL Anywhere用に変更 ②

先程のNCS様セッションで解説

SQL CentralのDB移行ツールで移行 ①

（点線はその際に移行できる）

テーブル等の移行が終わった後にユーザーは移行する

（但し、テーブルオーナーやトリガーオーナーは先に移行が必要）

基本的に変更の必要なし

コード中にSQLを直接書いてしまっている部分は全てチェック要

③ 同時進行可能、あるいは連係が必要

移行要

量が多ければ大変

他の項目は④となる（順序に決まりはない）

テーブル・データの移行

移行元のバックアップ＆小環境の準備を

アプリケーションを含めた移行を行う場合は、元のDBやアプリで動作していた内容が「正」となります。移行後のアプリケーションが移行前と同じ動作をするか確認するために移行元のデータベースが開発環境にあると便利です。

- 同一スキーマである必要がありますが、サイズやマシンスペックは同じである必要はありません。

これを作成するための方法は

- **移行元データベースをバックアップして、開発環境でリストア**
- **移行元データベースをエクスポートして、開発環境でインポート**

が挙げられます。移行元データベースは本番のバックアップや、その移行前の開発で使用していた開発用DBが使用できるでしょう。移行元DBのマニュアルを参照して、同じDBをコピーする方法を確認して開発環境でも使用できるようにして下さい。

- **データの変更がかかることも考慮し、そのバックアップやエクスポートしたデータは保管しておくと良いです。**

移行先DB作成

まずは移行先となるSQL Anywhereデータベースを作成する必要があります。

この際に意識しなければならない（作成時に指定しなければならない）のは
- 文字コード

です。

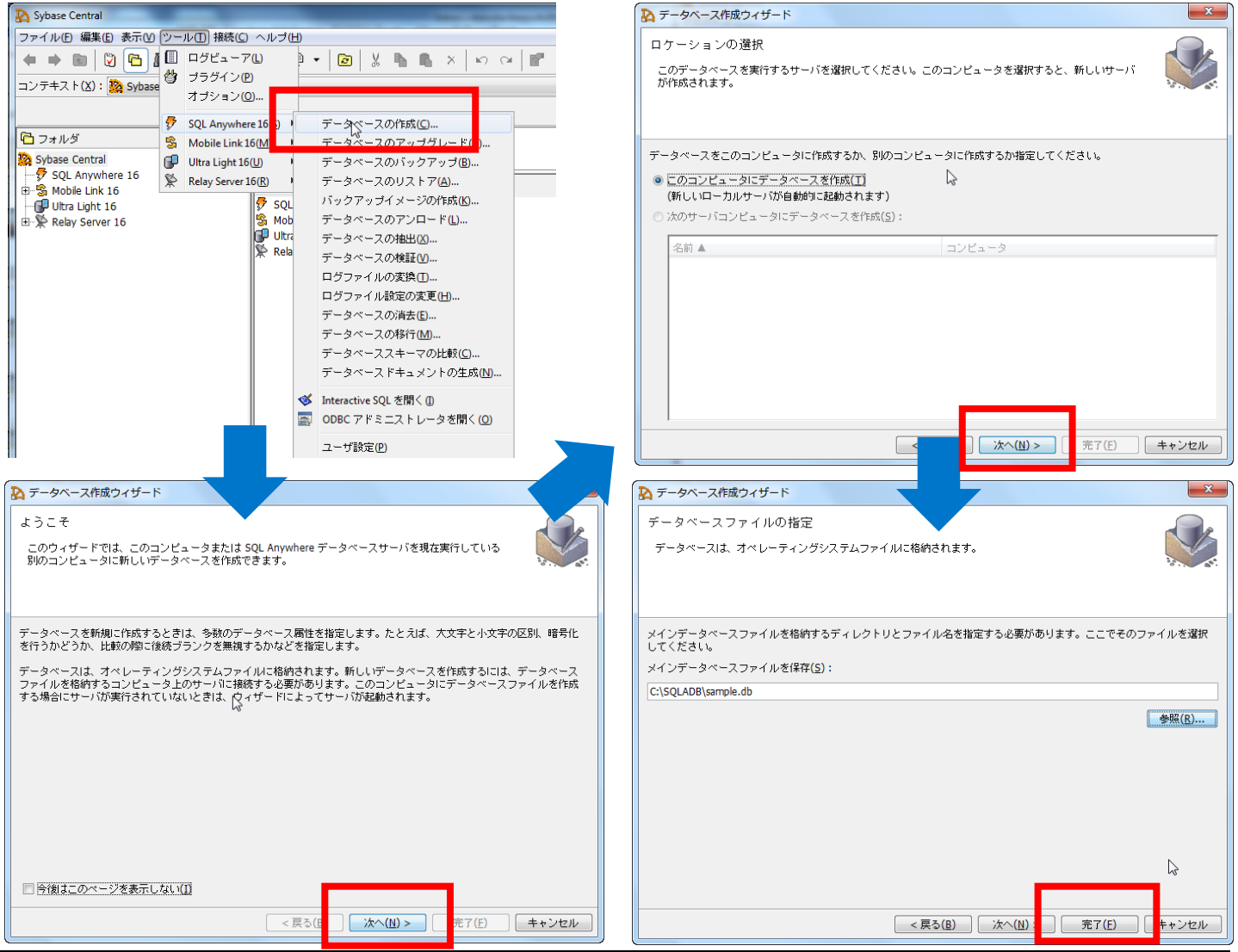
上記ならびにDB作成時に指定するパラメータ類は、あとから変更できる、あるいは本番で使用する際に指定しなおせば良いというパラメータでも有ります。
- DBUNLOAD/DBLOADコマンド等SQL Anywhereはそのデータベースを DDLに直して新規のデータベースに再ロードするコマンドが予め用意されています。

まずは移行元のデータベースの文字コードを確認して、同義の文字コードを使用してDBを作成して下さい。

このDBは開発用のDBであり、本番で使用するデータベースではありません。

データベースの作成

- **データベースを作成するには以下の手順を実行します**
  1. Sybase Centralのメニューから”ツール” > “SQL Anywhere 16” > “データベースの作成”を選択する
  2. ようこそ画面で[次へ]ボタンをクリックする
  3. ロケーションの選択画面でデータベースを作成する場所を指定し、[次へ]ボタンをクリックする
  4. メインデータベースファイルの格納先を指定し、[次へ]ボタンをクリックする

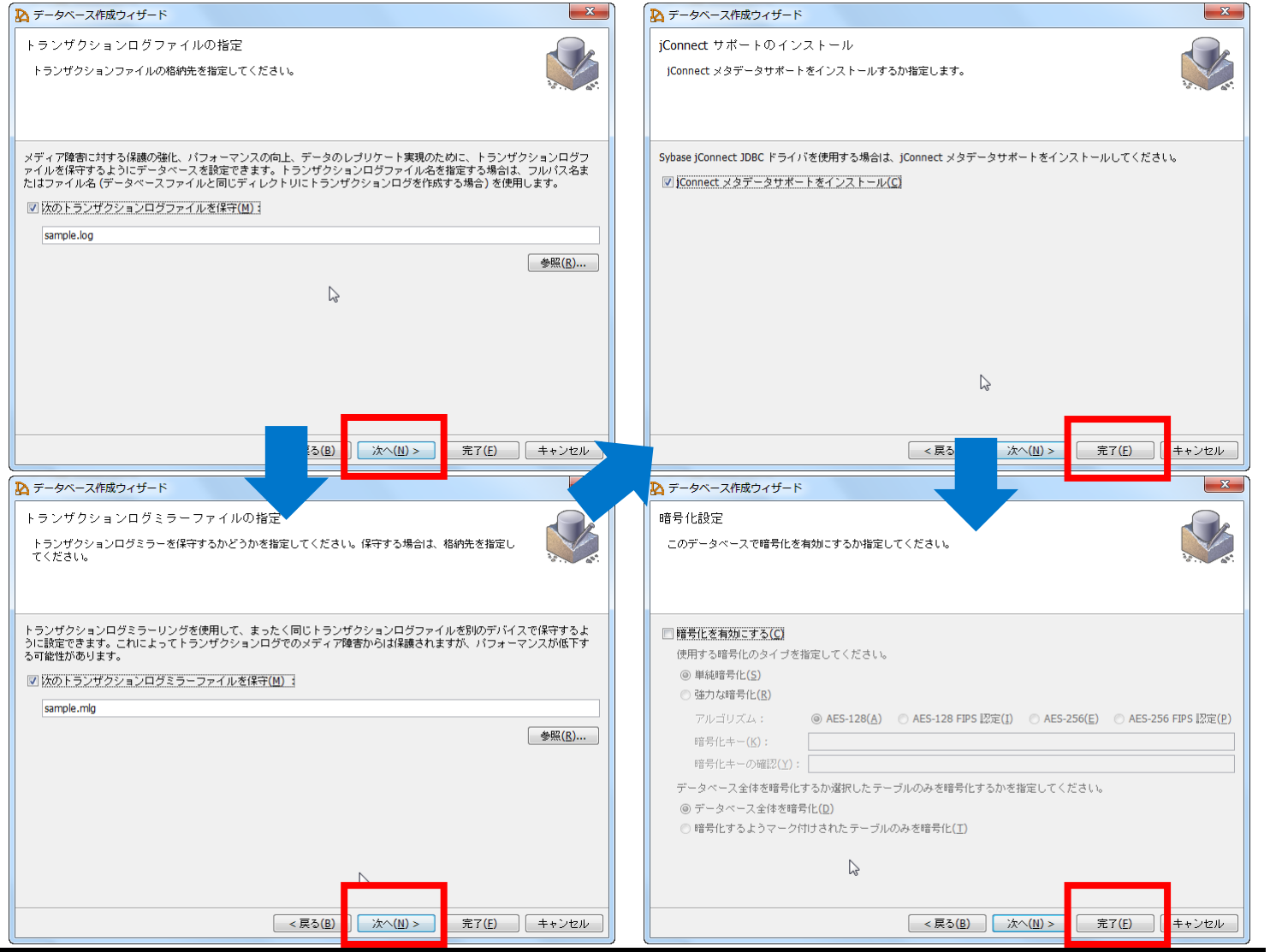


- **～続き～**

5. トランザクションログファイルの格納先を指定し、[次へ]ボタンをクリックする
6. トランザクションログミラーファイルを保守する場合は、チェックボックスにチェックし、トランザクションログミラーファイルの格納先を指定し、[次へ]ボタンをクリックする
7. Sybase jConnect JDBCドライバを使用する場合は、jConnectメタデータサポートをインストールにチェックせず、[次へ]ボタンをクリックする

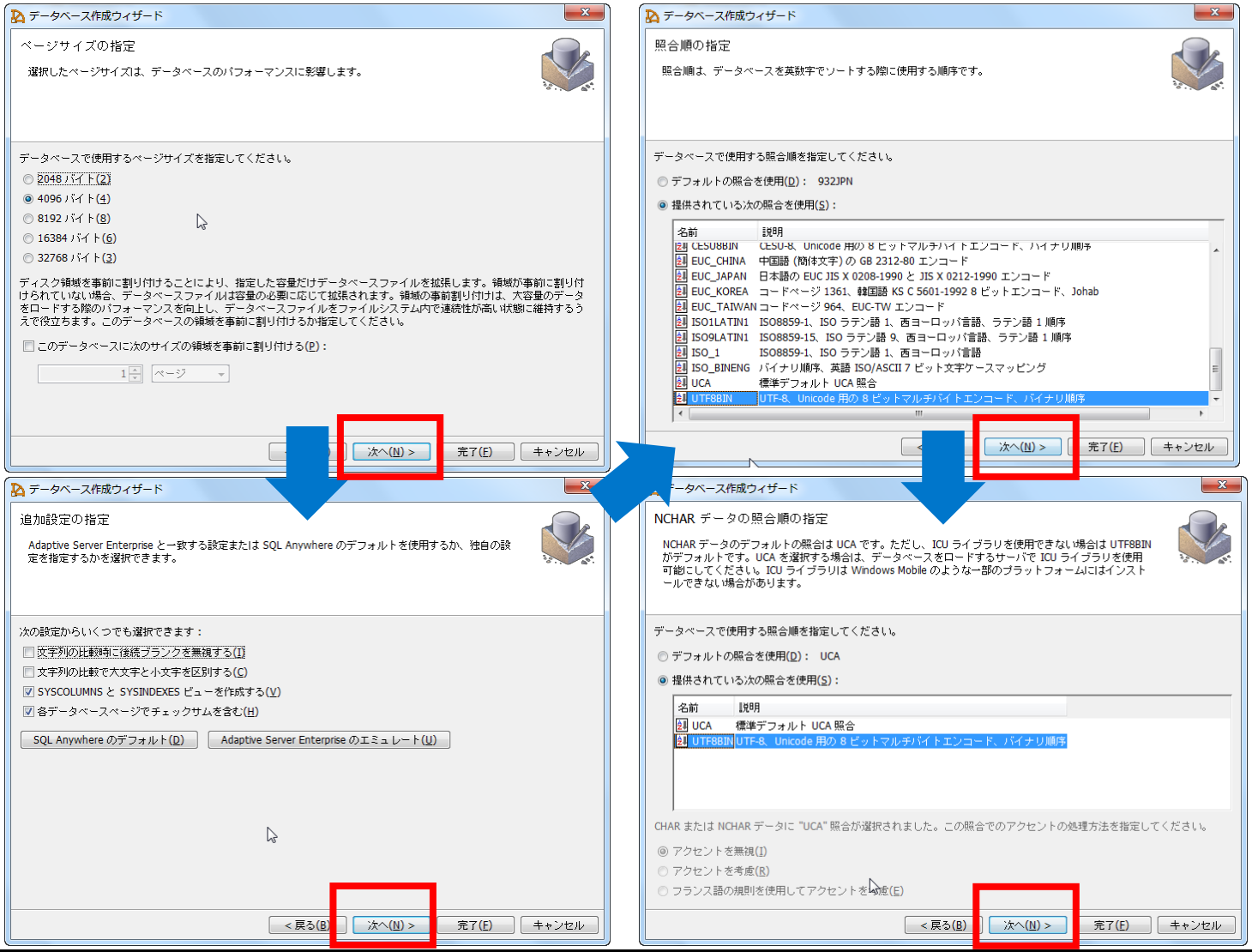
　＊PowerBuiderアプリの場合は「別の外部プログラムでJDBCを使用する場合を除き」チェックを外して良い。

8. データベースを暗号化する場合は暗号化を有効化し、暗号化設定をおこない、[次へ]ボタンをクリックする



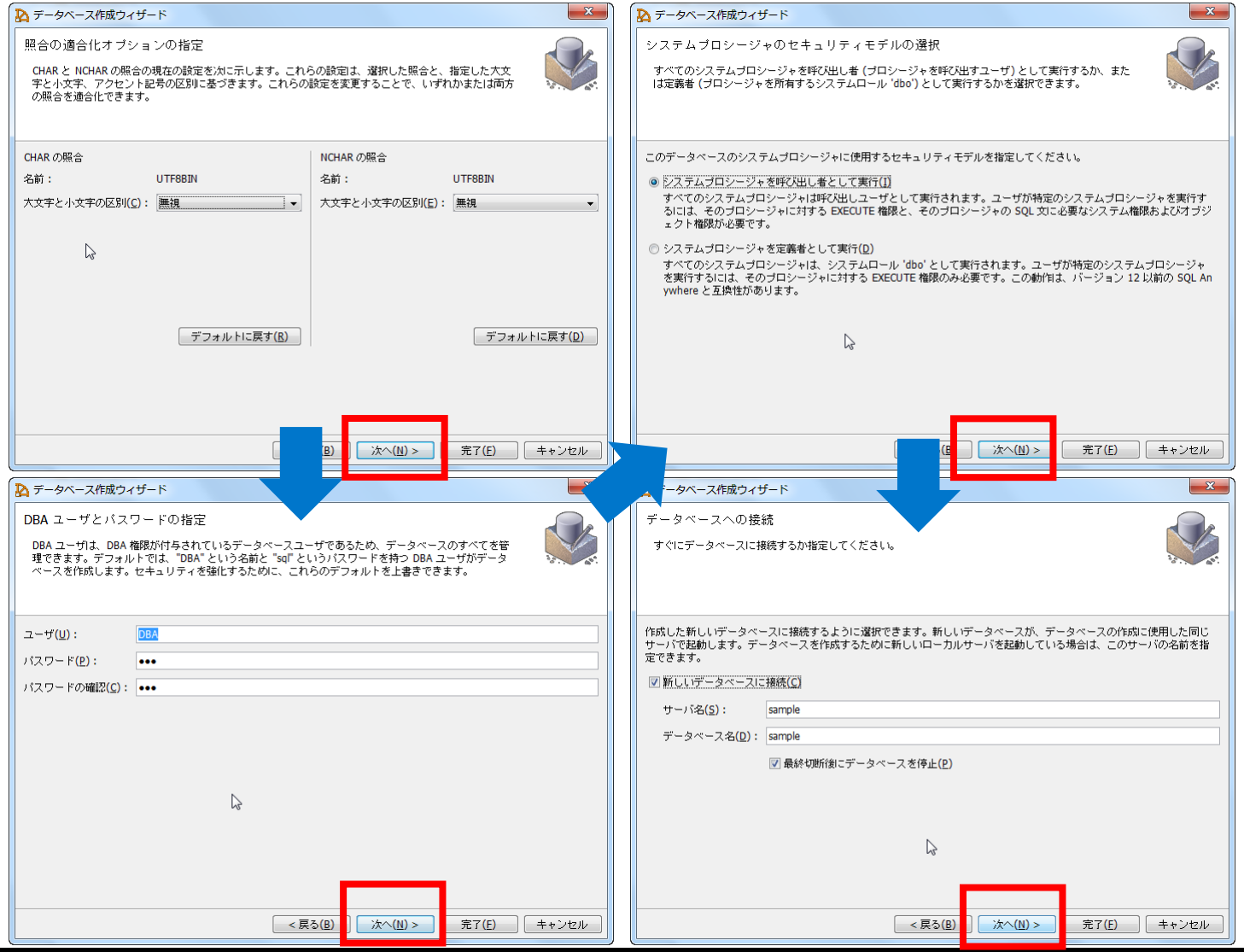
- **～続き～**

9. ページサイズを指定し、[次へ]ボタンをクリックする
10. 追加設定を指定する場合は指定し、[次へ]ボタンをクリックする
11. データベースの照合順を指定し、[次へ]ボタンをクリックする
12. NCHARデータの照合順を指定し、[次へ]ボタンをクリックする



- **～続き～**

13. 照合の適合化オプションを指定し、[次へ]ボタンをクリックする
14. DBAユーザとパスワードを指定し、[次へ]ボタンをクリックする
    - デフォルトユーザ名: DBA
    - デフォルトパスワード: sql
15. システムプロシージャに使用するセキュリティモデルを選択し、[次へ]ボタンをクリックする
16. すぐにデータベースに接続するかどうかを指定し、[次へ]ボタンをクリックする



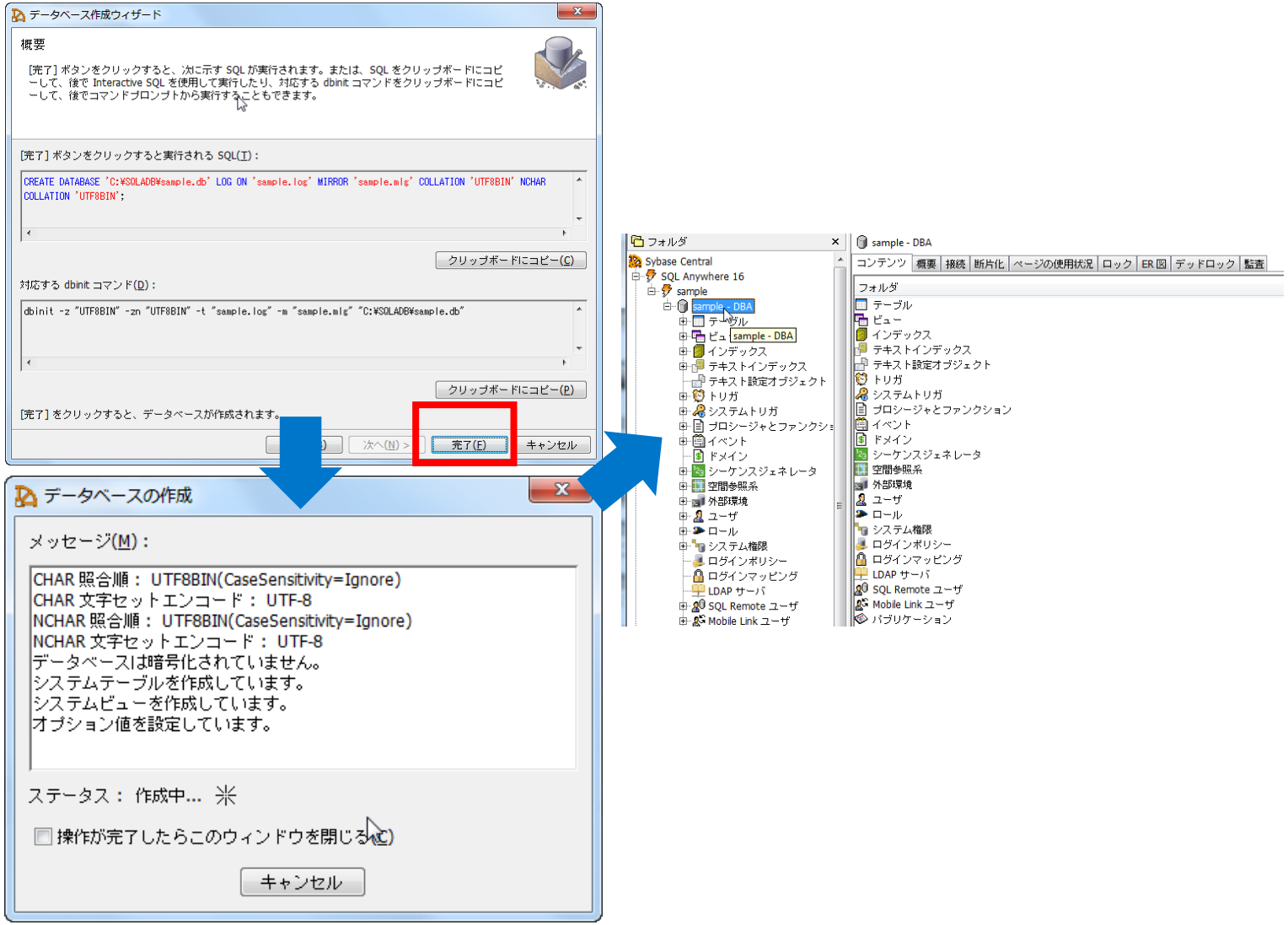
- **～続き～**

17. 概要を確認し、[完了]ボタンをクリックして、データベースを作成する

- **CREATE DATABASEコマンドやdbinitユーティリティを使用してデータベースを作成することも可能です**

注）この時点で作るデータベースは、移行作業用のデータベースという意味合いのものであるため、ページサイズなどのパラメータは特に考えずディフォルトのままでも良いです。

- 本番環境で使用するデータベースはこのデータベースを基にして、新規のものを作成することが出来ます。



ユーザーの移行

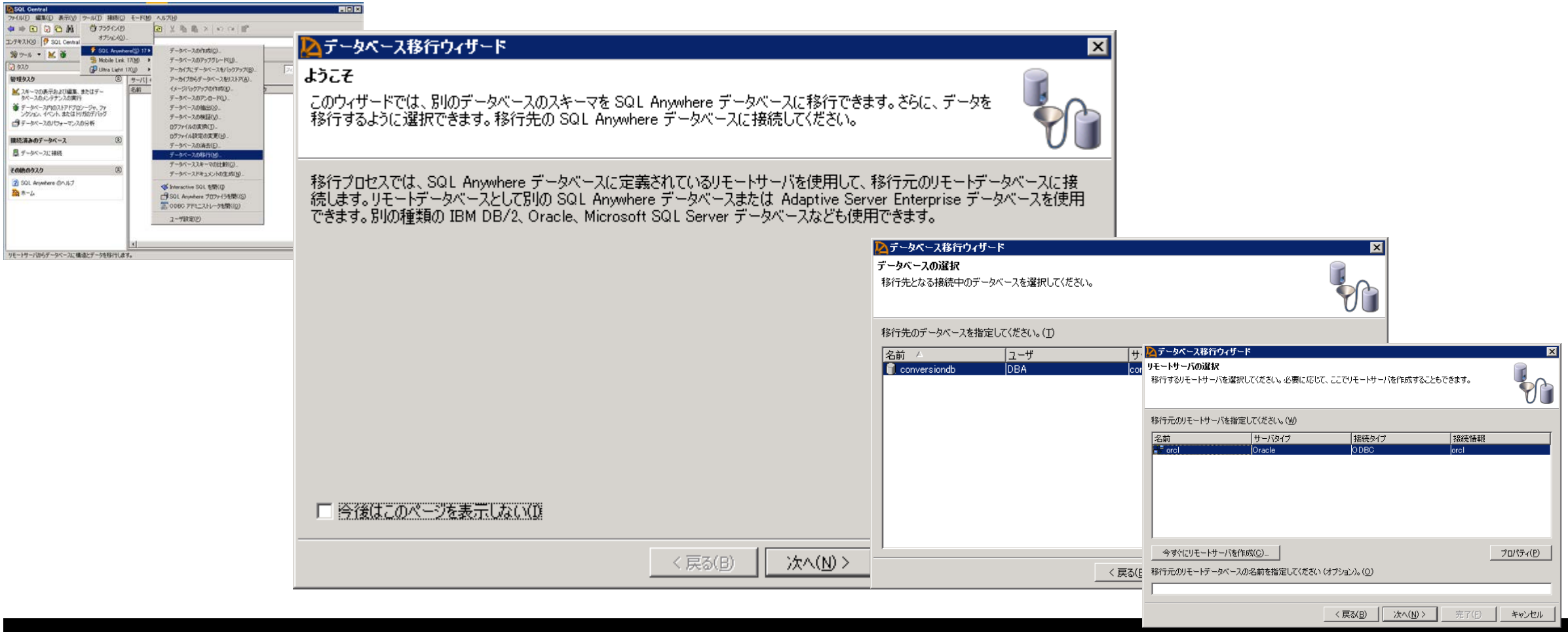
作成したSQL Anywhereデータベースにはユーザー（&グループ）を移行する必要があります。この作業はツールを使用して移行出来ません。但し、この時点で“全て”行うかは検討が必要です。

- テーブルなどのオブジェクトはユーザーに属するので、まずユーザーを移行するところから始めないとテーブルをDBAユーザーの所有にせざるをえなくなります。
  - これが問題なければユーザー移行を最初に行う必要はありません。
  - この時点ではOracle上でテーブルなどのオブジェクトを所有するユーザーだけ移行しても良いでしょう。
    - 通常は管理者ユーザー、オブジェクトの所有ユーザー、開発しながらの小テストに使用する数名の一般ユーザーがいれば移行フェーズでは事が足りる事が多いです。
    - この後のテーブルの移行時に移行先のSQL Anywhere上にユーザーを作成することが可能です。
- Oracleからの場合、（Oracleの）USER_USERSテーブルからユーザーデータをエクスポートし、それをSQL AnywhereのCREATE USER文に直す必要があります。
- **Oracleで使用していたパスワードはエクスポート出来ません。**何らかの初期パスワードを設定してユーザーを作成する必要があります。

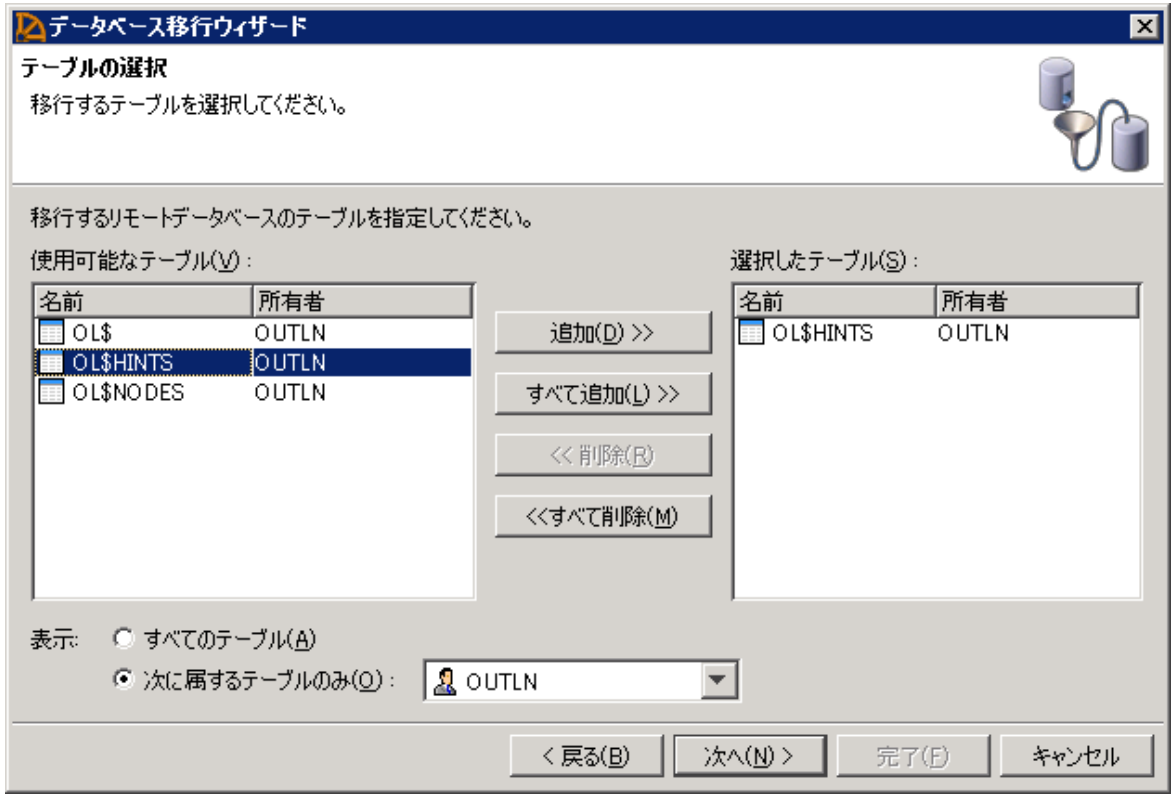
- 権限に関してもOracleからの場合はdba_sys_privsテーブルからエクスポート可能です。SQL AnywhereのGRANT文に直す必要があります。
  - この時点では管理者ロールを与えてしまうのも良いでしょう。

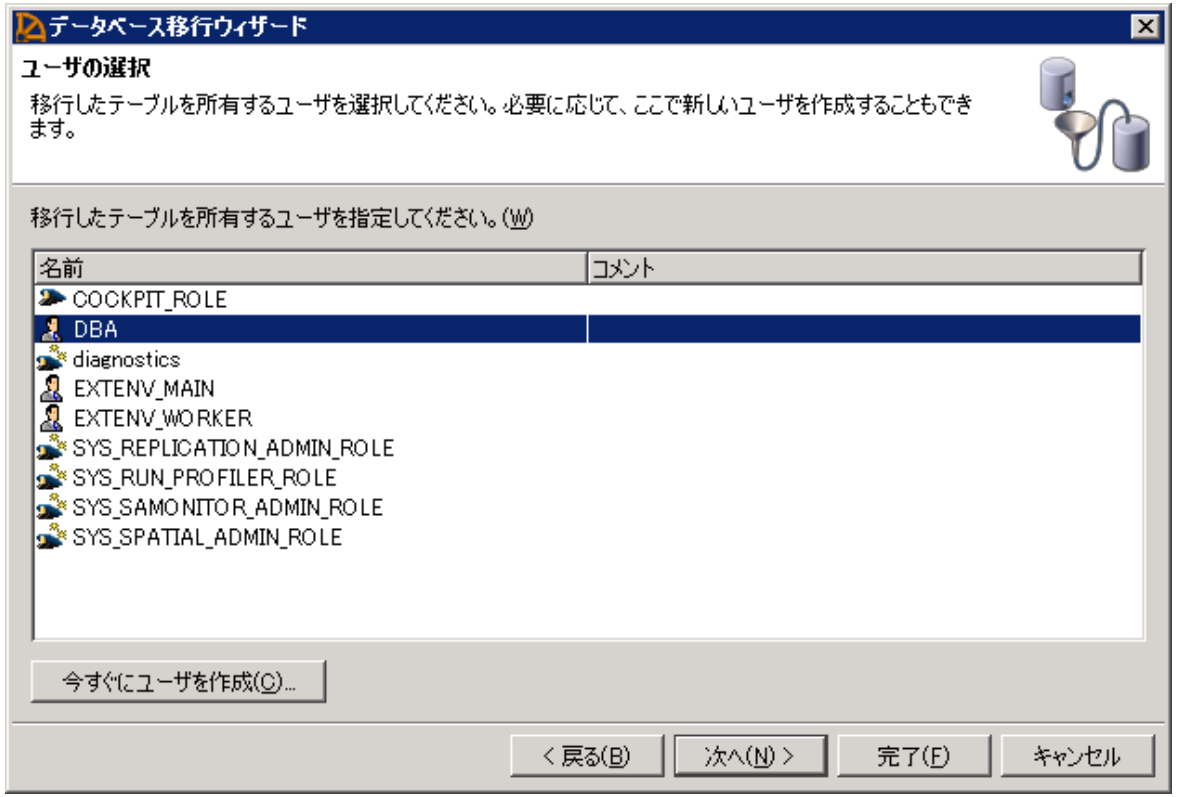
データベース移行機能

他社のDBのテーブル・インデックス・データをSQL Anywhereデータベースへ移行する機能がSQL Centralには含まれます。

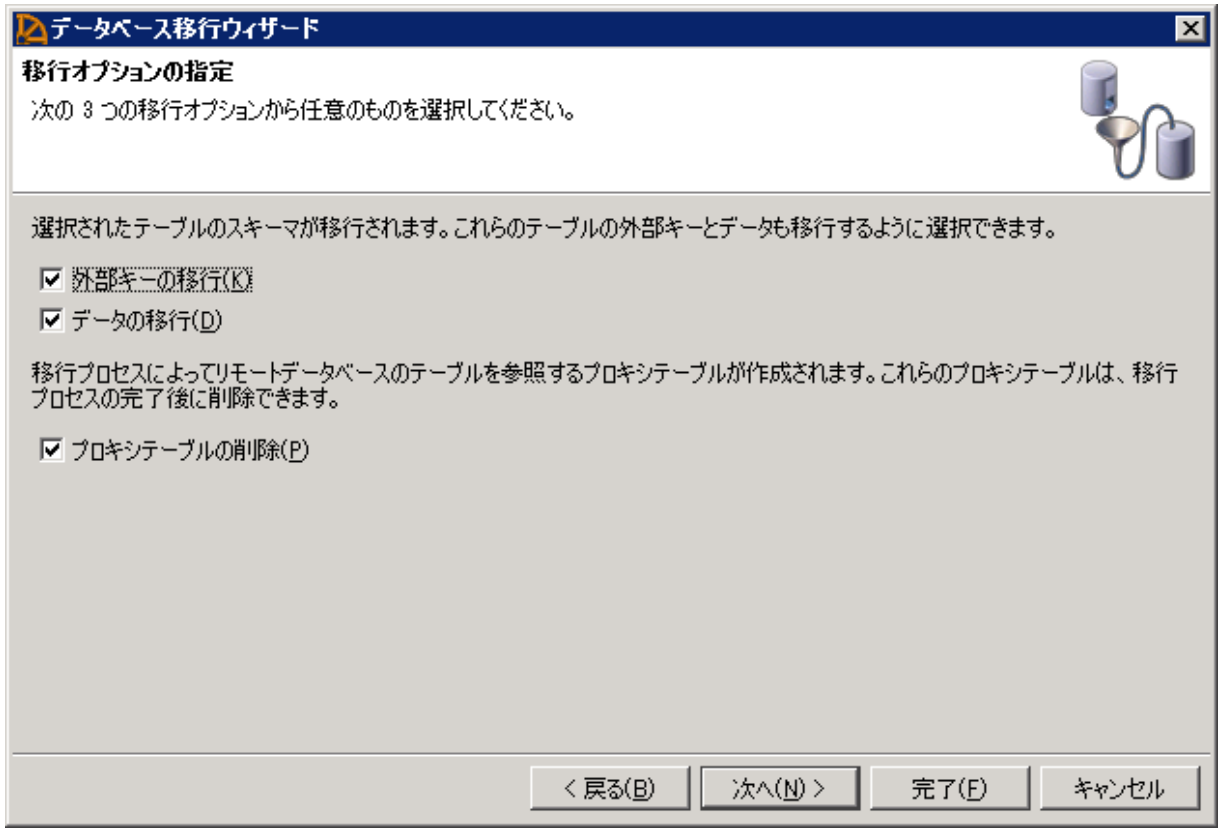


テーブル内の各カラムはSQL Anywhereへ互換性の有る型で移行されます。インデックス・外部キー、並びにデータも移行が可能です。





ここでそのテーブルを保有するユーザーを作成可能です。



CSVファイルをロードするでも良いです

前頁のDB移行ウィザードでのデータ移行は移行元ODBC経由でデータをINSERT文に直しての移行です。

大量のデータでは効率が悪いため移行元でデータをCSV等でアンロードし、SQL Anywhere側ではDB移行ウィザードでデータの移行をせずに移行を行い、後からCSVファイルを各テーブルにロードするという手法も活用して下さい。

データ移行後にバックアップしておきましょう

開発用に移行元の小環境をキープして、それと同じデータを移行完了したのであれば、その時点の”データ”をバックアップしておいたほうが良いでしょう。これはアプリ上の処理・SQLの実行結果の確認に使用します。元のアプリと同じように変更されるか、あるいはデータが出力されるかを確認することで正しく移行できているかの確認ができます。

- **移行元と同じデータでないと、データが正しく変更されているかがわかりません**
  - 開発中のテストで結構データは変わります。
- **データベースファイルをバックアップしておいても良いし、dbunloadコマンド等でCSVファイルとして保管しておいても良いです。**
  - 開発でプロシジャなども移植するのであれば、CSVファイルに直したほうが楽でしょう。（削除してデータを入れなおせば良い）

SQLの移行とポイント

SQLの移行 (From Oracle)

- SQL AnywhereのSQLは構文としてANSI書式、Watcom SQL書式、Transact SQL書式を使用することが出来ます。
- OracleではANSI書式とOracle書式（PL/SQL書式）を使用することが出来ます。

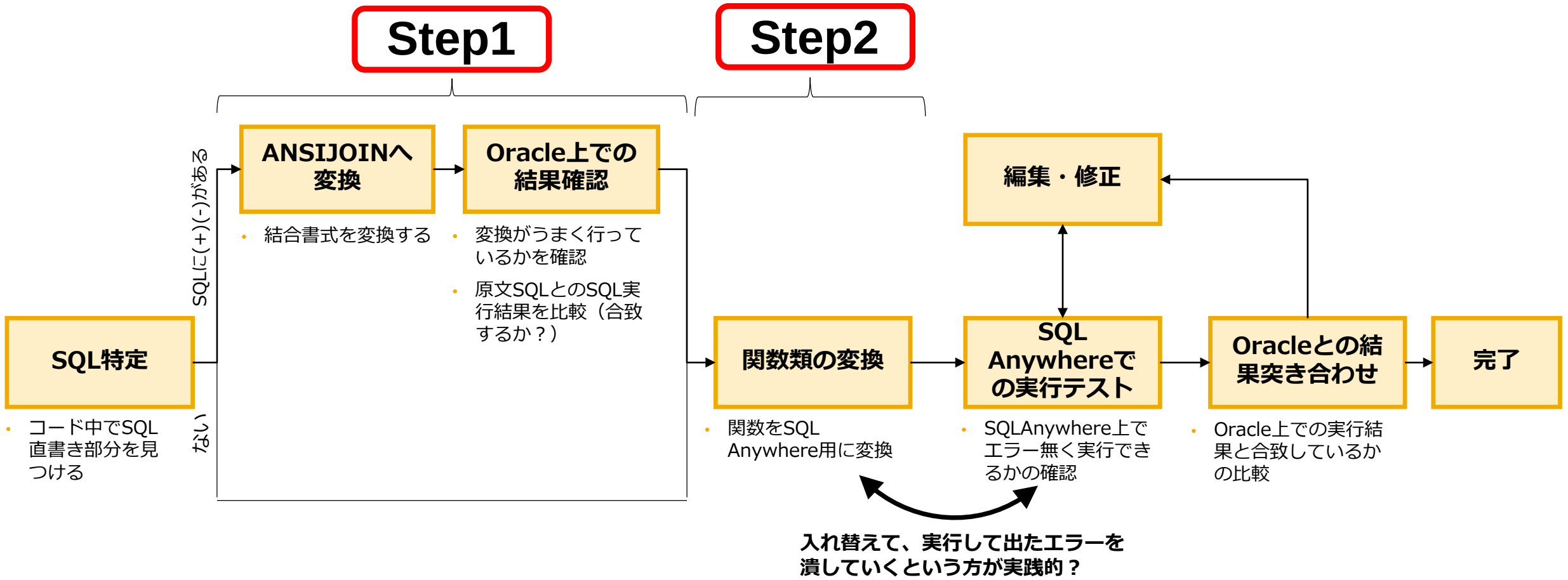
双方で使用できるANSISQLに変換することが移行の第１ステップです。

重要：”移行”においては”SQLが同じ結果を返す”ことが重要です。

- SQL結果確認テスト用にOracleのデータベースを用意しておいて下さい。
  - 9i以上を用意するのが良い
    - 9i以上はANSISQLが使えるので移行途中の確認が可能、それ以前の場合途中の確認が難しい。

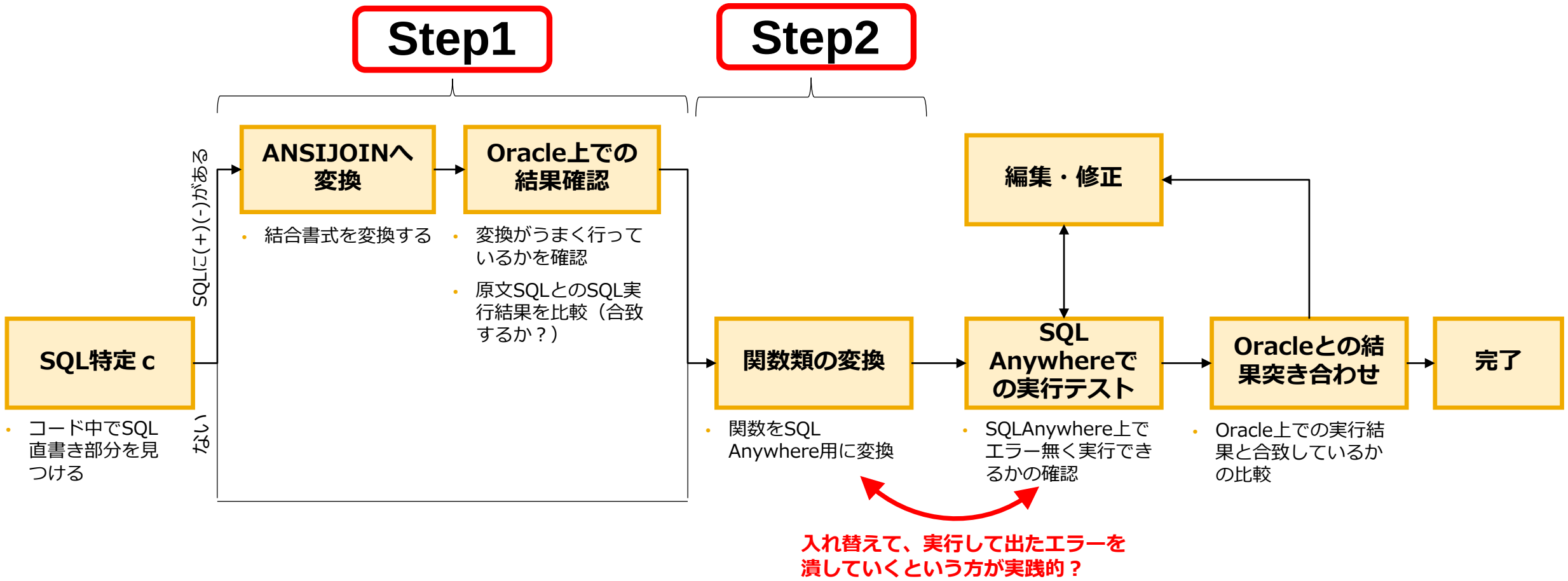
大まかな変換手順：SQL（SELECT文）の場合

※テーブルやデータ等のデータ移行が済んでいる前提

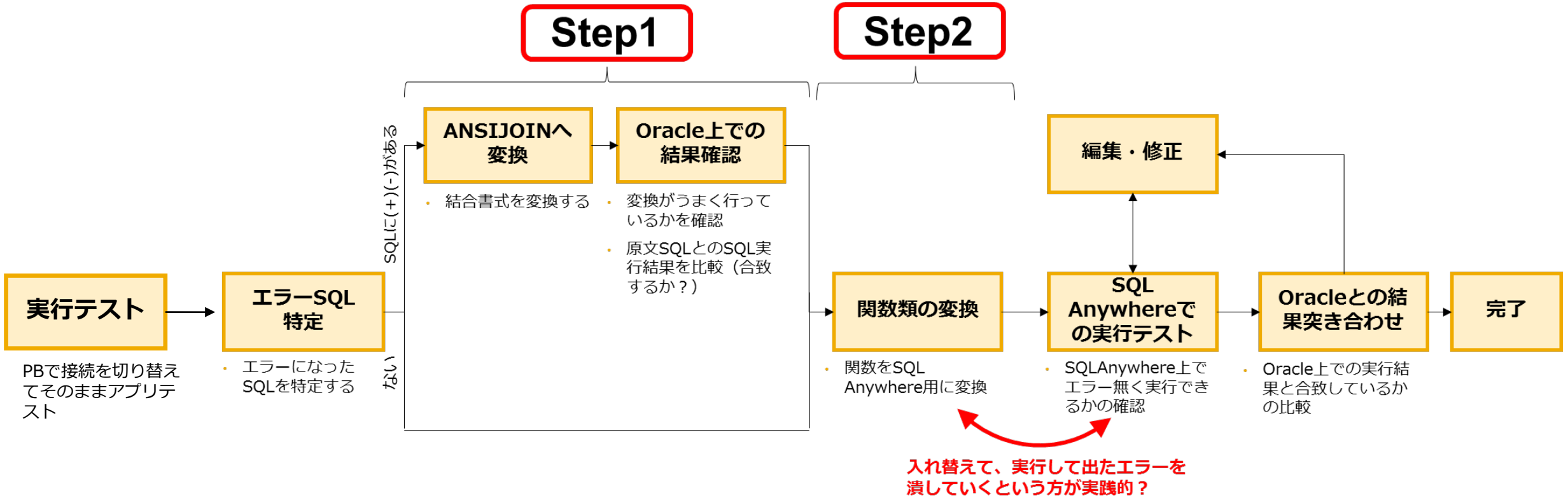


実践的には？

「こうでなければならない」というものではないので、下記のように一部の手順を入れ替える方法や



PB上で接続を切りかえて、そのままアプリケーションのテストを行い、エラーや結果がおかしい部分を潰していく…という手法でもあります。



小規模アプリ、あるいは少人数での変更作業向き。大規模アプリの場合はその前の部分がエラーになるために担当部分のコードが実行できないなどが起こりうるので非効率になるかも．．．．。

Step1:ANSIJOINへの変換
(+)(-)による結合

Oracleの独特な書式として(+)、(-)によるテーブルの結合が有ります。これはOracle独自のものであり、他のRDBMSでは動作しません。この書式をSQL Anywhereでも使用できるようANSI書式に直す必要があります。

もし、そのSQLが（+）(-)を使用していなければ、この項はスキップできます。

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B
WHERE
A.COL1 = B.COL1(+)
AND A.COL3 = 'xxxxxx'
```

まずはSQLにこのような(+)、(-)が有るかを確認して下さい。

- これはSQL文の構造から変わる変更であり、文字列置換のレベルで機械的に置換することが不可能です。
- 但し、現在は必ずしもこの書式が使用されているわけではありません。Oracleでも９以降ではANSI書式が使用可能で、最近のバージョンでは(+)(-)の結合ではなくANSI書式の使用を推奨しています。

(-)はあまり使われないのでこの資料では(+)に絞って解説します。

(-)は基本的に(-)を削除するだけで対応可能です。

Step1：ANSIJOINへの変換

（+）の外部結合をANSI形式のジョインに直します。

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B
WHERE A.COL1 = B.COL1(+)
AND A.COL3 = 'xxxxxx'
```

ANSI書式への変換

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A LEFT OUTER JOIN TABLEB B ON
A.COL1 = B.COL1
WHERE A.COL3 = 'xxxxxx'
```

変換の基本

- 右辺に(+)なら左辺のテーブルにLEFT OUTER JOIN
- 左辺に(+)なら右辺のテーブルにRIGHT OUTER JOIN
  - FROMでのテーブルの順番は合わせる
- ON句にそのテーブル同士を結合する条件を指定
- テーブル同士の結合に関わらない条件はWHERE句の後に指定

Step1の完了

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B
WHERE A.COL1 = B.COL1(+)
AND A.COL3 = ‘xxxxxx'
```

ANSI書式への変換

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A LEFT OUTER JOIN TABLEB B ON
A.COL1 = B.COL1
WHERE A.COL3 = ‘xxxxxx'
```

変換が完了したら、OracleDB上でsql*plusなどから変換後のSQLを実行し、同じ結果が出力されるか確認してください（上記の場合、DECODE関数の問題でこの時点ではSQL Anywhere上で実行できません）

同じ結果になればStep2へ。そうでなければどこかが間違っているので見直しましょう。

Step1:ANSIジョインへの変換・様々なパターン

- **左辺が(+)の場合はRIGHT OUTER JOIN**

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B
WHERE A.COL1(+) = B.COL1
AND A.COL3 = ‘xxxxxx'
```

ANSI変換

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A RIGHT OUTER JOIN TABLEB B on A.COL1
= B.COL1
WHERE A.COL3 = ‘xxxxxx'
```

※勿論等価になるよう左右を入れ替えても良い

- **複数条件の場合はon句の後を（ ）で囲んでAND/OR**

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B
WHERE A.COL1 = B.COL1(+) AND A.COL2=B.COL2(+)
AND A.COL3 = ‘xxxxxx'
```

ANSI変換

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A LEFT OUTER JOIN TABLEB B on ( A.COL1
= B.COL1 AND A.COL2=B.COL2)
WHERE A.COL3 = ‘xxxxxx'
```

- **複数条件で片方が(+)を忘れている場合は内部結合になってしまう。(Oracleでエラーにならない)**

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B
WHERE A.COL1 = B.COL1(+) AND A.COL2=B.COL2
AND A.COL3 = ‘xxxxxx'
```

ANSI変換

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA ,TABLEB B WHERE A.COL1 = B.COL1 AND
A.COL2=B.COL2
WHERE A.COL3 = ‘xxxxxx'
```

※この場合は(+)を消すだけで対処可能だが、これが意図的なものかは確認が必要。（潜在的なバグの可能性がある）

- **３表の場合**

```
SELECT A.COL1,B.COL2,C.COL4,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B,TABLEC C
WHERE A.COL1 = B.COL1(+) AND B.COL1 = C.COL1(+)
AND A.COL3 = ‘xxxxxx'
```

ANSI変換

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A LEFT OUTER JOIN TABLEB B ON A.COL1 =
B.COL1  LEFT OUTER JOIN TABLEC C ON B.COL1 =C.COL1
WHERE A.COL3 = ‘xxxxxx'
```

※OUTER JOINした後にさらにOUTER JOINする

- **３表の場合の(+)忘れで内部結合になるパターン（Oracleでエラーにならない）**

```
SELECT A.COL1,B.COL2,C.COL4,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B,TABLEC C
WHERE A.COL1 = B.COL1(+) AND B.COL1 = C.COL1
AND A.COL3 = ‘xxxxxx'
```

ANSI変換

```
SELECT A.COL1,B.COL2,C.COL4,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B,TABLEC C
WHERE A.COL1 = B.COL1 AND B.COL1 = C.COL1
AND A.COL3 = ‘xxxxxx'
```

※これが意図的なものかは確認が必要。（潜在的なバグの可能性がある）

- **一部のみ外部結合の場合**

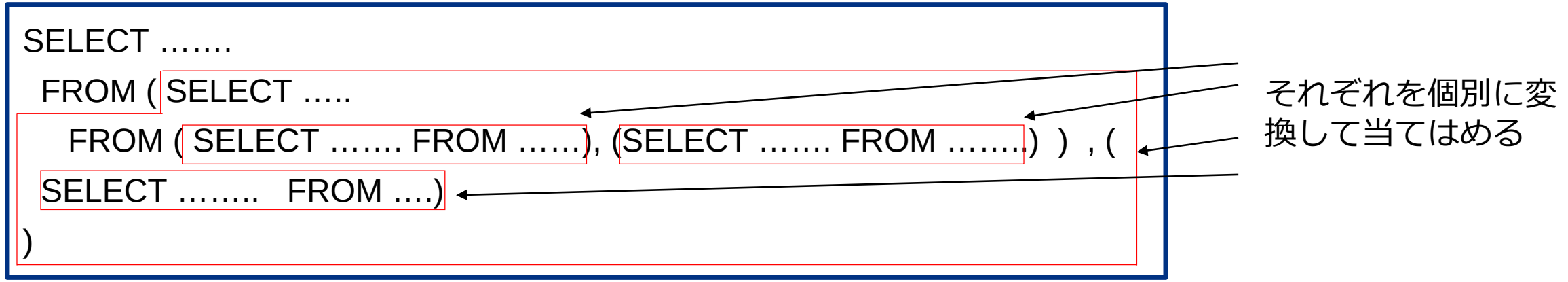
```
SELECT A.COL1,B.COL2,C.COL4, D.COL1,E.COl2
FROM TABLEA A,TABLEB B,TABLEC C,TABLED D,TABLEE E
WHERE A.COL1 = B.COL1(+) AND B.COL1 = C.COL1(+)
AND D.COL1=E.COL1 AND A.COL1=D.COl1
AND A.COL3 = ‘xxxxxx'
```

ANSI変換

```
SELECT A.COL1,B.COL2,C.COL4, D.COL1,E.COl2
FROM TABLEA A LEFT OUTER JOIN TABLEB B ON A.COL1 =
B.COL1  LEFT OUTER JOIN TABLEC C ON B.COL1
=C.COL1,TABLED,TABLEE
WHERE D.COL1=E.COL1 AND A.COL1=D.COl1
AND A.COL3 = ‘xxxxxx'
```

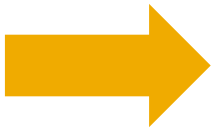
※(+)のところのみ直す

- **長いSQL（サブクエリー使用）はクエリー毎に区切って考えると楽（？）**



- サブクエリーを使った複雑な、長いSQLでは、サブクエリー毎に分解し、ANSI変換を行う事を推奨
- サブクエリーでOracle特有書式を使われていない場合はそのサブクエリーは変換する必要はない
- テーブル名のシノニム（FROM TABLEA A)を使用している場合は、そのSQL内の全てのテーブル・サブクエリー結果にシノニムを付与してください。シノニム経由のカラム参照を使用しない場合でもシノニムを付与する必要があります。

```
SELECT …….
  FROM ( SELECT …..
     FROM ( SELECT ……. FROM ……) Tab1, (SELECT ……. FROM ……..) ) , (
  SELECT ……..   FROM ….)
)
```



```
SELECT …….
  FROM ( SELECT …..
     FROM ( SELECT ……. FROM ……) Tab1, (SELECT ……. FROM ……..)
TAB2 ) , (
  SELECT ……..   FROM ….) TAB3
)  TAB4
```

Step1:追記

- **“OUTER”は省略可能です。**
  - LEFT OUTER JOIN → LEFT JOIN / RIGHT OUTER JOIN → RIGHT JOIN　でもOK

- **Transact SQL (SQL Server系）の “*=“ 及び “=*“ による外部結合はSQL Anywhereは処理できます。**
  - tsql_outer_joins オプション をONにする。
  - 同一SQL文内でON句による外部結合との混在は非サポートです。
  - 将来的にはこの機能は削除される予定ですので早めに（この機会に）ANSI形式に直したほうが良いです。

Step2：関数の変換

SQL内で関数が使用されている場合、その関数がSQL Anywhereで使用できるかが重要です。

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A LEFT OUTER JOIN TABLEB B ON
A.COL1 = B.COL1
WHERE A.COL3 = ‘xxxxxx'
```

例えば、上記で使用されているDECODE関数はSQL Anywhereでは使用できませんので、SQL Anywhereが提供する同義のもので置き換えることになります。

- これは単純文字列置換で可能な場合もありますし、不可能な場合もあります。
- また、同名称・同定義のものも存在しますので変換不要な場合もあります。

これを判別するために、置換前にそのまま実行してしまうのも手です。SQL Anywhereに存在しない、あるいは定義が異なるのであればエラーになりますのでそれで判別が出来ます。

Step2：関数の変換　例

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A LEFT OUTER JOIN TABLEB B ON
A.COL1 = B.COL1
WHERE A.COL3 = ‘xxxxxx'
```

Oracle:DECODE関数

DECODE (expr , search_and_result_list [, default])

expr　　NULL を設定可能な式（数値式、文字列式、日付式、etc）
search_and_result_list　検索式と結果式リスト [search , result]
default　　検索式に一致しない場合の結果式　　default NULL

return [第3引数（search_and_result_list の最初の result 式） のデータ型]

式 expr が search_and_result_list の第1番目の要素の search1 と同値なら result1 を戻す。search2 と同値なら result2 を戻す。これを 検索式と結果式リスト [search , result] のある分だけ繰り返す。最後まで一致する要素が見つからない場合にはオプション指定の 式 default の値を戻す。

上記例では　“B.COL3はYの場合は1、そうでなければ0を返す” という動きとなる。

SQL AnywhereではDECODEという関数は存在しません。類似の動きをするものとして“CASE式”があります。これを利用して変換します。

```
CASE expression-1 WHEN expression-2 THEN expression-3, ...
[ ELSE expression-4 ]
{ END | END CASE }
```

SQL Anywhere :CASE式

CASE 句に続く式が WHEN 句に続く式と等しい場合、THEN 文の後の式に復帰します。それ以外の場合、
ELSE 文があればそれに続く式に復帰します。
ELSE 句が存在しなく、expression-1 が expression-2...expression-n のいずれの値とも一致しない場合、
CASE 式は NULL を返します。

“B.COL3がYの場合は1、そうでなければ0を返す。” は実現可能

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A LEFT OUTER JOIN TABLEB B ON
A.COL1 = B.COL1
WHERE A.COL3 = ‘xxxxxx'
```

CASE式を用いて同義変換

```
SELECT A.COL1,B.COL2,CASE B.COL3 WHEN ‘Y’
THEN 1 ELSE 0 END
FROM TABLEA A LEFT OUTER JOIN TABLEB B ON
A.COL1 = B.COL1
WHERE A.COL3 = ‘xxxxxx'
```

※CASE式には変形でCASE WHEN <式> THEN …　や　NULLとの比較に特化した　ISNULL　も有ります。

Step2:その他の関数

その他の関数の場合の例を幾つか列挙します。

Oracle 関数名	ADD_MONTHS	
説明	日付に月数を加算	
SQL Anywhere 対応関数名	DATE_ADD	
構文	Oracle	ADD_MONTHS(日付式, 月数)
	SQL Anywhere	DATEADD (日付要素(*1), 加算する値, 日付値)
使用例1	col1 に格納されている日付から 12 ヵ月前の日付値を求める	
	Oracle	ADD_MONTHS(TO_DATE(col1,'yyyymmdd'),-12)
	SQL Anywhere	DATEADD(MONTH,-12,col1)

※Oracleは日付演算は計算単位により関数名が異なるが、SQL Anywhereは一つの関数で引数で計算単位を変更する。

Oracle 関数名	MONTHS_BETWEEN	
説明	2 つの日付の間の月数を求める	
SQL Anywhere 対応関数名	DATEDIFF	
構文	Oracle	MONTHS_BETWEEN(日付1, 日付2)
	SQL Anywhere	DATEDIFF(日付要素(*1), 開始日付, 終了日付)
使用例1	date1 カラムの日付から date2 カラムの日付までの月数を返す	
	Oracle	MONTH_BETWEEN(date1, date2)
	SQL Anywhere	DATEDIFF(month, date1, date2)

※Oracleは日付演算は計算単位により関数名が異なるが、SQL Anywhereは一つの関数で引数で計算単位を変更する。

Oracle 関数名	SYSDATE	
説明	現在の日時を求める	
SQL Anywhere 対応関数名	NOW, GETDATE	
構文	Oracle	SYSDATE
	SQL Anywhere	NOW() GETDATE()
使用例1	現在の日時を求める	
	Oracle	SYSDATE
	SQL Anywhere	NOW() GETDATE()

Oracle 関数名	NVL	
説明	NULL 値を別の値に変換	
SQL Anywhere 対応関数名	ISNULL	
構文	Oracle	NVL(式, 値)
	SQL Anywhere	ISNULL(式, 値)
使用例1	col1 カラムの値が NULL 値ならば “0” を返す	
	Oracle	NVL(col1, 0)
	SQL Anywhere	ISNULL(col1,0)

Oracle 関数名	TO_CHAR	
説明	数値／日付を文字列型に変換	
SQL Anywhere 対応関数名	CONVERT	
構文	Oracle	TO_CHAR(式 [, フォーマット [, NLS パラメータ]])
	SQL Anywhere	CONVERT(データ型 [(長さ)], 値 [, 日付形式(*2)])
使用例1	数値を文字列型に変換する	
	Oracle	TO_CHAR(12345)
	SQL Anywhere	CONVERT(CHAR(20), 12345)
使用例2	今日の日付をフォーマットを指定して文字列型に変換する	
	Oracle	TO_CHAR(SYSDATE, 'MON DD, YYYY')
	SQL Anywhere	CONVERT(CHAR(20), NOW(), 107)

※Oracleはデータ型変換は変換先のデータ型により関数名が異なるが、SQL AnywhereはCONVERT関数の引数で変換先を指定する。

http://ftp2.ianywhere.jp/tech/Oracle_to_SQLAnywhere_Migration_Schema&Data_SQL.pdf
にて他の幾つかの関数に関しても公開されています。

要注意事項：Oracleのrowid

もし、Oracleで使用しているSQLにおいてrowid値が使われている場合、SQL Anywhereでもrowid関数で置換可能ですが、これは推奨しておりません。

- **Oracleにおいてはrowidを使用した「1行の取得」は最速のアクセスパスとされていますが、SQL Anywhereにおいては最速ではありません。**
- **SQL Anywhereではrowidの不変性は保証していません（Oracleでも変わる場合がある）**
- **主キー、あるいは一意に特定できるカラムの値（の組み合わせ）で代用するようにしてください。**

Oracle:SELECT col1 FROM TableA WHERE ROWID = 'XXXXXXXX' ;
　→SQL Anywhere:SELECT col1 FROM TableA WHERE ROWID(TABLEA)='XXXXXXX'
で置換可能ですが、メリットはありません。また、ROWIDの不変性は保証しません。

SELECT col1 FROM TableA WHERE key1= X AND key2 = Y;
等で代用するようにしてください。

その他の文

- **INSERT文、UPDATE文、DELETE文などはOracleとSQL Anywhereで文型に大きな違いがないためそのまま適用できる場合が殆どです。**
  - ANSI形式であればそのままOK。「UPDATE TBL1 SET (col1. col2. col3) = (SELECT col1. col2, col3 FROM tbl2 WHERE...) WHERE tbl1.col1=xxx」 のような形式はダメです。ANSI形式になおしてください。
  - 関数の存在、あるいはサブクエリーやINSERT~SELECT文のようにSELECTが含まれる場合はそのSELECT文を先に開設したとおりに変更する等は必要です。

- **MERGE文はSQL AnywhereにもMERGE文が存在するのでそれで置換できます。**
  - 対象テーブルには主キーが必要です。
  - Oracleの独自構文で文型を変えなくてはならないパターンがあります。（WHERE文が存在する場合）
    - 次スライドにて解説

- **OracleでのDUAL表はSQL AnywhereではDUMMY表となります。**

  Oracle : SELECT SYSDATE FROM DUAL → SQL Anywhere : SELECT TODAY(*) FROM DUMMY

  *但し、SQL Anywhereでは SELECT TODAY(*) だけでも実行可能です。

MERGE文

MERGE文は基本形が同じであるためそのまま使用できます。

関数類が使用されている場合はSELECTと同じように対応が必要です。

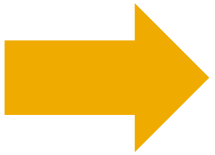
```
merge into EMP  t
using EMP_DAILY f
on ( t.EMPNO = f.EMPNO )
when matched then
                update set
                                t.ENAME    = f.ENAME,
                                t.JOB      = f.JOB,
                                t.MGR      = f.MGR,
                                t.HIREDATE = f.HIREDATE,
                                t.SAL      = f.SAL,
                                t.COMM     = f.COMM,
                                t.DEPTNO   = f.DEPTNO
when not matched then
                insert (t.EMPNO, t.ENAME, t.JOB, t.MGR, t.HIREDATE, t.SAL, t.COMM, t.DEPTNO)
                values (f.EMPNO, f.ENAME, f.JOB, f.MGR, f.HIREDATE, f.SAL, f.COMM, f.DEPTNO):
```

※省略形も使用可能です。

ON句でのマッチ・非マッチの後で条件が設定されている場合はMATCHED/NOT MATCHDED AND 句で条件を指定するように書き換えが必要です。

Oracle

```
merge into EMP t
using EMP_DAILY f
on ( t.EMPNO = f.EMPNO )
when matched then
                update set
                                t.ENAME    = f.ENAME,
                                t.JOB      = f.JOB,
                                t.MGR      = f.MGR,
                                t.HIREDATE = f.HIREDATE,
                                t.SAL      = f.SAL,
                                t.COMM     = f.COMM,
                                t.DEPTNO   = f.DEPTNO
                where f.EMPNO between (7000 AND 8000)
when not matched then
                insert (t.EMPNO, t.ENAME, t.JOB, t.MGR,
t.HIREDATE, t.SAL, t.COMM, t.DEPTNO)
                values (f.EMPNO, f.ENAME, f.JOB, f.MGR,
f.HIREDATE, f.SAL, f.COMM, f.DEPTNO)
                where f.EMPNO between (9000 AND 10000)
```



SQL Anywhere

```
merge into EMP t
using EMP_DAILY f
on ( t.EMPNO = f.EMPNO )
when matched AND f.EMPNO between (7000 AND 8000)
then
                update set
                                t.ENAME    = f.ENAME,
                                t.JOB      = f.JOB,
                                t.MGR      = f.MGR,
                                t.HIREDATE = f.HIREDATE,
                                t.SAL      = f.SAL,
                                t.COMM     = f.COMM,
                                t.DEPTNO   = f.DEPTNO
when not matched AND f.EMPNO between (9000 AND
10000)
then
                insert (t.EMPNO, t.ENAME, t.JOB, t.MGR,
t.HIREDATE, t.SAL, t.COMM, t.DEPTNO)
                values (f.EMPNO, f.ENAME, f.JOB, f.MGR,
f.HIREDATE, f.SAL, f.COMM, f.DEPTNO)
```

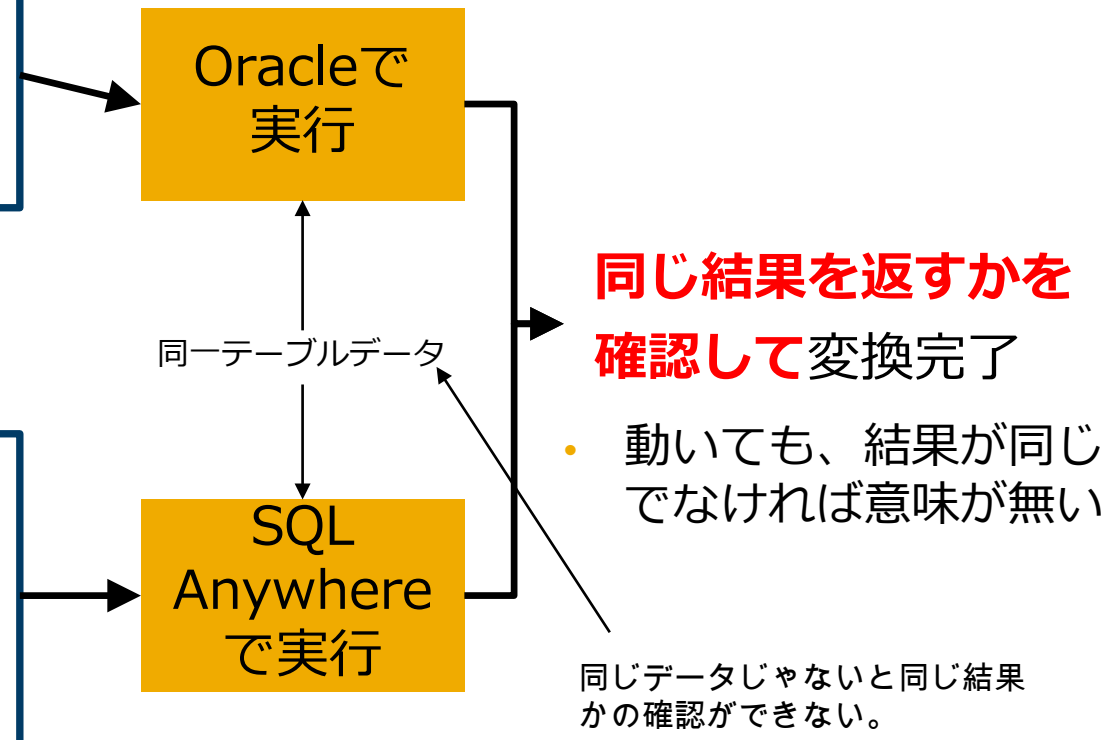
最後に：実行して確認してください！

元のOracle用SQL

```
SELECT A.COL1,B.COL2,DECODE(B.COL3,'Y',1,0)
FROM TABLEA A,TABLEB B
WHERE A.COL1 = B.COL1(+)
AND A.COL3 = ‘xxxxxx'
```

SQL Anywhere用に変換したSQL

```
SELECT A.COL1,B.COL2,CASE B.COL3 WHEN NULL
THEN 0 ELSE B.COL3 END
FROM TABLEA A LEFT OUTER JOIN TABLEB B ON
A.COL1 = B.COL1
WHERE A.COL3 = ‘xxxxxx'
```



同じ結果を返すかを確認して変換完了

- 動いても、結果が同じでなければ意味が無い

そして変換したSQLをPowerBuilderのコードに貼り付ければ完了です。

1文の変換にどのくらいかかる？

これまでのお客様の御意見では・・・・

- **勿論文の長さや複雑さ、あるいは担当者のスキルや慣れに依存するが、1文あたり10分から15分もあれば十分**
  - また、アプリケーション内では”似たようなSQL”が複数登場する場合が多く、一つを直せば他にも適用出来るパターンが存在するので、トータルで見るともっと短くなる可能性がある
    - 同一構造のSQLで対象テーブル違い、WHERE句のみ違う
    - 出力カラムが若干違う　等
- **アプリを開発するのではなく、「移行」なのでメンテ業務に近い**
  - Oracleだってバージョン間でうまく動かなくなるコマンドは存在するので、それを確認し、直すのと同じ
- **重要なのは「答え合わせ」が出来る環境を用意すること**
  - 同一データのOracleとSQL Anywhereデータベース環境を用意する。
  - 実行して同じ結果かを確認できる環境を用意することが重要

是非トライして下さい

- **今回ご紹介した方法でデータベースのスキーマ構造とデータは移行できます。**

コネクションを変更して、ご使用中のアプリが動くかどうか確認しながら行ってみて下さい。

SQL Anywhere の無期限無償のDeveloper Editionはこちらからダウンロードできます。
http://www.sqlanywhere.jp/dl/

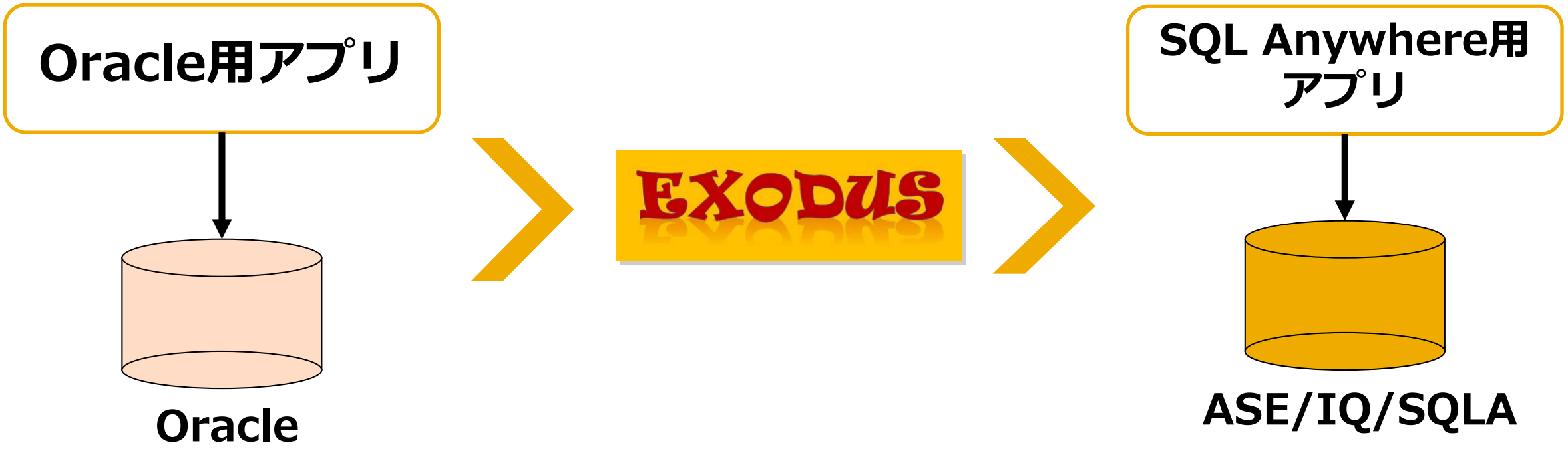
Windowsアプリケーション開発ツール
PowerBuilder® × SQL Anywhere®

SAP Exodusの御紹介

SAPパートナー様にのみ提供されるデータベースアプリケーション移行アシストツール

What is Exodus ?

SAP ExodusはOracleやDB2、SQL Server用に開発されたデータベースアプリケーションをASE、IQ、SQL Anywhere用への変換をアシストするツールです。



Why Exodus ?

WindowsPCで稼働するアプリを想定

対応・変換・改造

Oracle向けに記述したアプリ

App

Oracle DB

アプリ

ロジック部分は変更不要

DB周りの変更が必要。

接続API

汎用規格（ODBC/JDBC/.NET等）であれば変更量少

SQL

ANSI規格に従っていれば共通だが独自構文の場合もある。

DB構造

何らかの互換性のある形で移行できる場合が殆ど。ツールも豊富。

DB内ロジック（プロシジャ等）

独自言語、長文

出来なくはないが、面倒

SQL Anywhere向けに記述したアプリ

App

SQL Anywhere DB

移行するデータベースアプリケーションの種類

SAP アプリケーション (例：SAP Business Suite)

- SAP Rapid Deployment Servicesを使用することで移行できます。

ORマッパーのような中間層のフレームワークを使用した非SAPのカスタムアプリケーション (例：Hibernate)

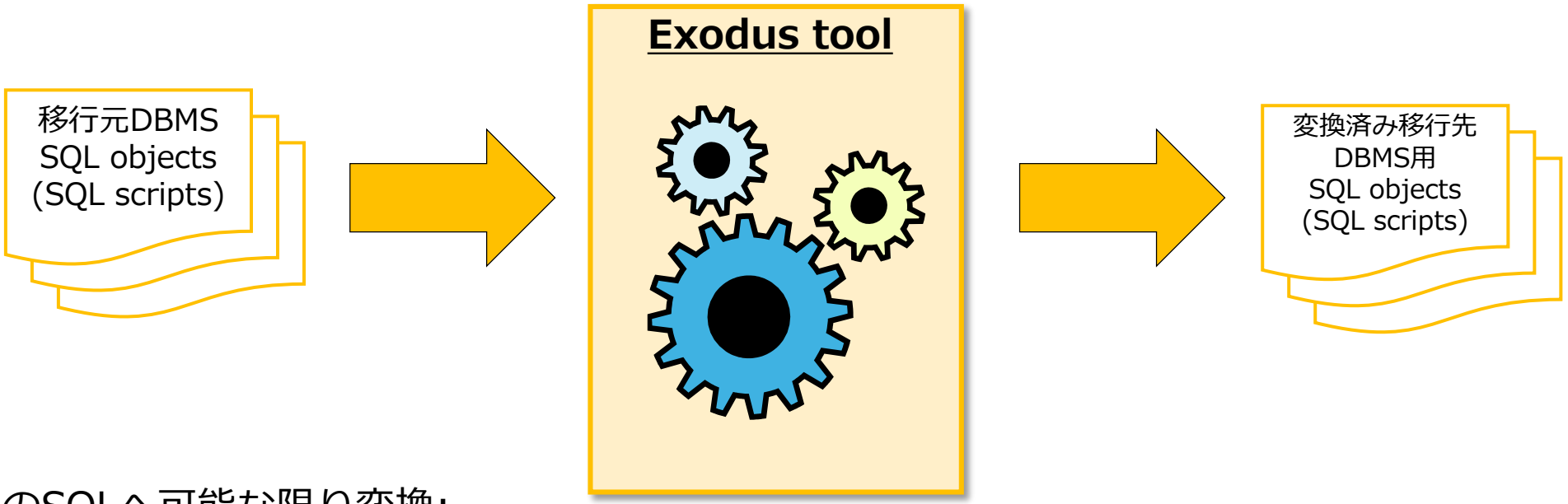
- そのフレームワークが移行先のDBをサポートしていれば簡単。
- この種のアプリはストアドプロシジャやユーザー定義関数等を使用していることは少ない。

CやJava、.NET等で開発されたカスタムアプリケーション

- アーキテクチャやSQLコード、API、DBMS毎の
  独自機能の使用など多岐に渡る。
- この種のアプリはストアドプロシジャやユーザー定義関数等を使用していることが多い
- サードベンダーライブラリを使用している場合は注意

Exodusが対象とするもの

Exodusによる変換



移行先のSQLへ可能な限り変換:

```
-- example: Oracle
SELECT SYSDATE from DUAL;
```

```
-- example: SAP SQL Anywhere
SELECT CURRENT TIMESTAMP
```

いくつかの関数では代用関数を提供:

```
-- example: Oracle
SELECT INITCAP(msg) from MyTable;
```

```
-- example: SAP SQL Anywhere
SELECT
dbo.sp_f_dbmtk_capitalize_word(msg)
from MyTable
```

※OracleのINITCAP関数に相当するものはSQL Anywhereに存在しないので、代用となる関数をExodusが提供し、それで置換する

Exodusが提供する関数

SAPがパートナー向けに提供しているDB移行プログラム：SAP Exodusでは、Oracleにあり、SQL Anywhereに相当する関数が存在しない場合の対処用に、それらの関数をユーザー定義関数として提供するようにしています。これはユーザー定義関数ですのでアプリからも使用可能です。

一例

dbo.sp_f_dbmtk_bits_to_number
dbo.sp_f_dbmtk_capitalize_word
dbo.sp_f_dbmtk_convert_UTC_2_local
dbo.sp_f_dbmtk_convert_local_2_UTC
dbo.sp_f_dbmtk_convert_minutes_offset (@offset varchar(100))
dbo.sp_f_dbmtk_convert_offset_minutes (@offset varchar(100))
dbo.sp_f_dbmtk_convert_timezone
dbo.sp_f_dbmtk_create_datetime_tz
dbo.sp_f_dbmtk_create_datetime_tz_enhanced (@date datetime , @offset varchar(100))
dbo.sp_f_dbmtk_current_proc_name
dbo.sp_f_dbmtk_datetime_add_tz (@date dbmtk_datetime_tz , @offset varchar(100))
dbo.sp_f_dbmtk_datetime_part
dbo.sp_f_dbmtk_datetime_part_interval
dbo.sp_f_dbmtk_datetime_part_timezone
dbo.sp_f_dbmtk_decimal_2_interval_ds
dbo.sp_f_dbmtk_dow
dbo.sp_f_dbmtk_escape_html_chars
dbo.sp_f_dbmtk_format_basic_numeric_to_string
dbo.sp_f_dbmtk_format_currency_numeric_to_string
dbo.sp_f_dbmtk_format_currency_string_to_numeric
dbo.sp_f_dbmtk_format_datetime_to_string
dbo.sp_f_dbmtk_format_hexadecimal_numeric_to_string
dbo.sp_f_dbmtk_format_hexadecimal_string_to_numeric
dbo.sp_f_dbmtk_format_numeric_to_string
dbo.sp_f_dbmtk_format_regular_numeric_to_string
dbo.sp_f_dbmtk_format_regular_string_to_numeric
dbo.sp_f_dbmtk_format_roman_numeric_to_string
dbo.sp_f_dbmtk_format_scientific_numeric_to_string
dbo.sp_f_dbmtk_format_scientific_string_to_numeric
dbo.sp_f_dbmtk_format_string_to_datetime
dbo.sp_f_dbmtk_format_string_to_datetime_date
dbo.sp_f_dbmtk_format_string_to_numeric
dbo.sp_f_dbmtk_hex_to_string(@inStr varchar(16384), @allInputAsHex int = 0)
dbo.sp_f_dbmtk_int_to_string
dbo.sp_f_dbmtk_invert_case_example

Exodusが提供する関数の例

Exodusが提供する関数：dbo.sp_f_dbmtk_capitalize_word

OracleのINITCAP関数に相当：引数に与えられた文字列の1文字目を大文字へ変換

- SQL Anywhereには全体を大文字（UCASE)・全体を小文字（LCASE）というのは存在するが、1文字目のみという関数は存在しない。

```
select dbo.sp_f_dbmtk_capitalize_word('hello')
```

```
dbo.sp_f_dbmtk_capitalize_word('hello')
----------------------------------------------------------------------------------------------------------------------------------
Hello
```

※標準関数の組み合わせでも出来なくはないが、地味に面倒

```
select UCASE(LEFT(‘hello',1))+RIGHT(‘hello',LENGTH('hello')-1)
```

```
UCASE("LEFT"('hello',1))+"RIGHT"('hello',LENGTH('hello')-1)
------------------------------------------------------------
Hello
```

Exodusが提供する関数：dbo.sp_f_dbmtk_last_day_of_month

OracleのLASTDAY関数に相当：引数に与えられた日付の月の最終日を返す

SQL Anywhereにはこれを行う関数は存在しない。

select dbo.sp_f_dbmtk_last_day_of_month(‘2015/08/15’)

```
dbo.sp_f_dbmtk_last_day_of_month('2015/08/15')
------------------------------------------
2015-08-31 00:00:00.0
```

※現在の仕様では時間の引き継ぎは行われないことに注意

※標準関数の組み合わせで行うとすれば、月に+1、日を1にして1日マイナス？

```
Select DATEADD(day,-1,DATEADD(month,1,DATEADD(DAY,-(DAY('2015/08/15'))+1,'2015/08/15')))
```

```
DATEADD(day,-1,DATEADD(month,1,DATEADD(day,-
(DAY('2015/08/15'))+1,'2015/08/15')))
-------------------------------------------------------------------------
2015-08-31 00:00:00.0
```

プロシジャ

注：変換後のスクリプトには変換前の行がコメントアウトした状態で含まれます。右記は変換前のコメントアウトした行は削除してあります。

オリジナル：Oracle PL/SQL

```
create procedure ml_set_listening(
   p_name    in  varchar2,
   p_device    in  varchar2,
   p_listening    in  varchar2,
   p_ignore_tracking  in  varchar2,
   p_source    in  varchar2 )
as
   d      varchar2( 255 );
begin
   begin
 select device_name into d from ml_listening where name = p_name;
   exception
     when NO_DATA_FOUND then
     d := NULL;
   end;
   if d is null then
 insert into ml_listening( name, device_name, listening, ignore_tracking, source )
     values( p_name, p_device, p_listening, p_ignore_tracking, p_source );
   else
      if p_source = 'tracking' then
     update ml_listening
   set device_name = p_device,
      listening = p_listening,
      ignore_tracking = p_ignore_tracking,
      source = p_source
   where name = p_name and ignore_tracking = 'n';
 else
     update ml_listening
   set device_name = p_device,
      listening = p_listening,
      ignore_tracking = p_ignore_tracking,
      source = p_source
   where name = p_name;
 end if;
   end if;
end;
```

Exodus変換後：SQL Anywhere WATCOM SQL

```
CREATE OR REPLACE PROCEDURE mluser1.ml_set_listening
(
   IN p_name VARCHAR(75) ,
   IN p_device VARCHAR(75) ,
   IN p_listening VARCHAR(75) ,
   IN p_ignore_tracking VARCHAR(75) ,
   IN p_source VARCHAR(75)
)
SQL SECURITY DEFINER
BEGIN
   DECLARE DBMTK_CALLER_ID INT; SET DBMTK_CALLER_ID = @@PROCID;

   DECLARE d VARCHAR(255);
   BEGIN
      SELECT
         device_name
      INTO
         d
      FROM
         ml_listening
      WHERE name = p_name;

   EXCEPTION /* RESOLVE: EXCEPTION handler   Details: Manual adjustments required to convert to SA
EXCEPTION syntax; see Exodus User Guide. */
      WHEN NO_DATA_FOUND THEN  /* RESOLVE: Manually add DECLARE...EXCEPTION (SA syntax)
for ORACLE 'NO_DATA_FOUND' condition (see SA documentation for corresponding SQLSTATE value) */
      SET d = NULL;
   END;

   IF d IS NULL
   THEN
      INSERT INTO ml_listening ( name, device_name, listening, ignore_tracking, source )
      VALUES( p_name, p_device, p_listening, p_ignore_tracking, p_source );
   ELSE

      IF p_source = 'tracking'
      THEN
         UPDATE ml_listening
            SET device_name = p_device,
            listening = p_listening,
            ignore_tracking = p_ignore_tracking,
            source = p_source
         WHERE name = p_name
         AND ignore_tracking = 'n';
      ELSE

         UPDATE ml_listening
            SET device_name = p_device,
            listening = p_listening,
            ignore_tracking = p_ignore_tracking,
            source = p_source
         WHERE name = p_name;
      ENDIF;
   ENDIF;
END;
```

移行コスト見積もり

移行元DBMSに接続し、コスト見積もり機能を使用することで、移行対象のオブジェクト数、Exodusで変換可能なオブジェクト数、変換できないオブジェクト数等を見積もり、移行にかかるコストの超概算見積もりが可能です。

```
----- Schema-based Migration Cost Estimate -----------------

Estimated migration cost for schema:   49 days   Cost breakdown(1 day=8 hrs)
   Schema Identifiers as Reserved Words : 235             1 day     (15m/unit)
   Regular Tables : 1428                                 36 days    (4h/unit)
   Nested Tables : 1                                      4 hours   (4h/unit)
   Object Tables : 2                                      1 day     (4h/unit)
   Index Types : 5                                       25 minutes (5m/unit)
   Views : 54                                             4 hours   (10m/unit)
   Columns with Oracle BFILE datatype : 1                 1 day     (1d/unit)
   Columns with Oracle INTERVAL/TIME ZONE datatype : 12   4 days    (3h/unit)
   Composite User-Defined Datatypes : 21                  2 days    (1h/unit)
   Collection User-Defined Datatypes : 7                  2 days    (2h/unit)
   Local Synonyms : 48                                    2 hours   (5m/unit)
   Remote Synonyms : 2                                   30 minutes (15m/unit)
   Other Object Types : 3                                 6 hours   (2h/unit)
```

「何にどのくらいかかるか」（例：1つのテーブルの移行にどのくらいの工数を要するか）はExodus上で事前定義が必要です。

```
----- SQL-based Migration Cost Estimate --------------------

Total estimated migration cost for all 755 files:   51 days    (1 day=8 hrs)
Conversion statistics for all files:                Cost breakdown(1 day=8 hrs)
   Datatype length exceeds target DBMS limit : 4     20 minutes (5m/unit)
   Syntax construct not supported in SQLA : 1         15 minutes (15m/unit)
   Feature not suppt'd in SQLA SQL Functions : 1276    9 days    (1h/unit)
   Delimited identifiers in SQLA : 469                 5 hours   (5m/unit)
   Performance feature not supported : 29             5 hours   (15m/unit)
   Review semantics in source/target DBMS : 2856      3 days    (15m/unit)
   Oracle Standard Package call not suppt'd : 36      3 days    (1h/unit)
   Mismatch formal/actual parameters : 481            1 day     (10m/unit)
   Mismatch nr of variables/expressions : 24          1 hour    (5m/unit)
   Owner/Scope of identifier unresolved : 568         4 days    (30m/unit)
   Identifier not found : 956                         4 days    (30m/unit)
   %TYPE declaration : 4949                           0         Fully converted
   Autonomous transaction : 2                         4 hours   (2h/unit)
   ANSI outer join syntax recommended : 257           1 day     (10m/unit)
   Dynamically generated SQL : 178                    3 hours   (5m/unit)
   Database link : 15                                 3 hours   (15m/unit)
[…]
```

```
Package declaration : 194                          0          Fully converted
DBMS_OUTPUT.PUT_LINE : 5178                        0          Fully converted
Execute Immediate : 178                            0          Fully converted
Standard Package call : 16                         0          Fully converted
Procedure call : 3527                              0          Fully converted
NULL statement : 209                               0          Fully converted
Oracle hint : 407                                  0          Fully converted
Oracle pragma : 5                                  1 hour     (15m/unit)
   Additional manual conversion required
SELECT-INTO : 2027                                 0          Fully converted
Built-in function : 6446                           0          Fully converted
Procedure declaration : 822                        0          Fully converted
Function declaration : 1821                        0          Fully converted
Variable declaration : 7804                        0          Fully converted
Variable assignment : 36123                        0          Fully converted
RETURN : 7092                                      0          Fully converted
END of block : 10924                               0          Fully converted
IF-THEN : 5584                                     0          Fully converted
ELSE (IF-THEN) : 1150                              0          Fully converted
ELSIF (IF-THEN) : 310                              0          Fully converted
[…]
```

```
SYSDATE,SYSTIMESTAMP,etc. : 5342                0          Fully converted
SQLCODE,SQLERRM : 2856                          0          Fully converted
RAISE_APPLICATION_ERROR : 805                   0          Fully converted
Boolean constant : 595                          0          Fully converted
Boolean condition : 276                         0          Fully converted
SQL Function call : 3776                        0          Fully converted
DUAL table : 389                                0          Fully converted
Outer join : 255                                1 day      (10m/unit)
   Manual conversion recommended
WHILE loop : 58                                 0          Fully converted
Loop control : 379                              0          Fully converted
EXCEPTION handler : 6230                       23 days     (2h/unit)
   Manual conversion to ASE required
Sequence CURRVAL,NEXTVAL : 304                  0          (0/unit)
Transaction COMMIT : 1366                       0          Fully converted
Transaction ROLLBACK : 2176                     0          Fully converted
Transaction SAVEPOINT : 10                      0          Fully converted
MINUS operator : 13                             6 hours    (30m/unit)
   Manual conversion required
INSERT statement : 1112                         0          Fully converted
[…]
```

```
   UPDATE statement : 953                          0         Fully converted
   DELETE statement : 565                          0         Fully converted
   SELECT statement : 2858                         0         Fully converted


----- Total Migration Cost Estimate ------------------------

Based on SQL conversion :   51 days

----- Conversion totals over all 755 files: ---------------------------------
Totals: 212627 lines processed (in 755 files) in 4040 seconds (52 lines/sec)
Conversion coverage: 84%
Total issues found, still to be resolved: 10794 (in 18 categories)
-----------------------------------------------------------------------------
```

移行工数は51人日と見積もり

「何にどのくらいかかるか」（例：1つのテーブルの移行にどのくらいの工数を要するか）はExodus上で事前定義が必要です。

（移行対象）DBオブジェクトのうち84%がExodusで変換可能

Exodusのサポート状況

- **Exodusは現在以下をサポートしています。**
  - 動作環境：Windows 7~10
    - Cドライブに5GB以上の空きスペース
    - 上記条件で動作するが、開発としては快適な動作のために以下のスペック以上のHWを推奨
      - マルチコアCPU（2コア以上）
      - 8GB以上のRAM
    - Cygwin / JRE 環境が必要です。

  - 移行元DBMS： Oracle, Microsoft SQL Server and IBM DB2 UDB
  - 移行先DBMS：SAP Adaptive Server Enterprise, SAP IQ, SAP SQL Anywhere
    (Sybase ASE/Sybase IQ/SQL Anywhere)
    - Oracle/DB2からの変換精度は80%~90%、SQL Serverからは90~95%

  - 移行先DBMSとしてのSAP HANA、及び移行元としての他DBMSのサポートは現在計画中/開発中です。

Exodusの注意点

SAP Exodus DBMS migration toolは

- **SAPの”製品”ではありません。**
- **SAPパートナー契約が必要です。**
- **無料です。**
- **全てのSQL/SQLオブジェクトの移行を保証するものではありません。**
- **移行の全てを行えるものではありません。**
  - スキーマ移行には対応していません。 (SAP PowerDesignerを御使用下さい。)
  - データ移行には対応していません。 (ETLツールをご利用下さい。)
  - クライアントアプリケーションや管理アプリケーションの移行はサポートしていません。

SQL Anywhereは SQL Centralで可能

- **アプリケーション開発ライフサイクル管理ツール、その機能はありません。**

付録：SQL Anywhere 関連情報

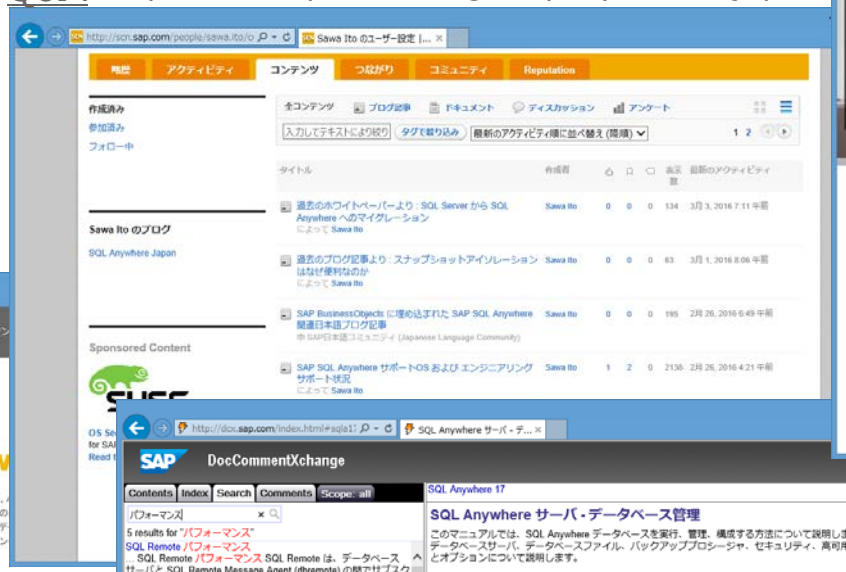
技術書

【目次】

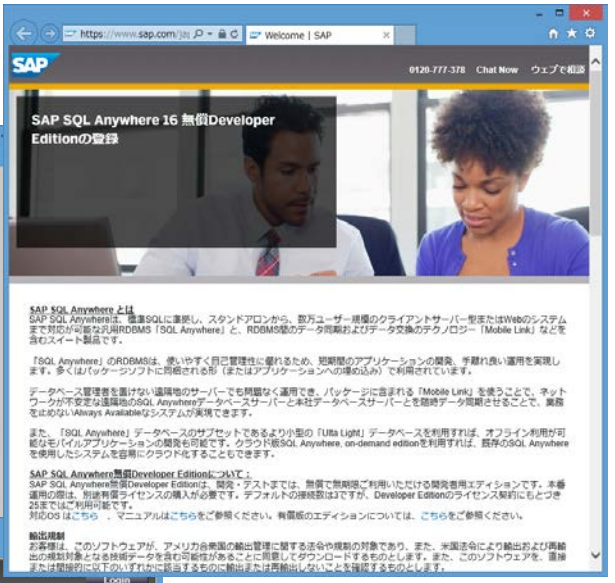
15年以上もの間蓄積された技術情報が掲載された日本語SQL Anywhere専用サイト

http://www.sqlanywhere.jp/

SAP コミュニティーネットワーク上の豊富な日本語技術情報と日本語によるQ&A http://scn.sap.com/blogs/sqlanywhere_japan



使いやすい日本語オンラインマニュアルhttp://dcx.sap.com/index.html



気軽に試せる無期限無償Developer Edition

http://www.sqlanywhere.jp/d

Additional：
プロシージャ・ユーザー定義関数の移行概要

プロシジャ・ユーザー定義関数

プロシジャやユーザー定義関数はそれ自体で何らかの（複雑な）処理を行う「ロジック」となります。SQLのような1文ではなく、複合文であり、それ自体がフローを持っていることに注意が必要です。

- プロシジャ内でSQLが使用されている場合、そのSQLは前章の例に従い、SQL Anywhere用に変換する必要があります。
- SQL AnywhereはTransact SQLも使用可能です。SQL Serverからの移行であれば構文上そのまま動くことが期待できます。

ここではいくつかの変換例を交え、ポイントを解説します。

（移行セミナーではより詳しい内容も解説しますのでそちらへの御参加も御検討下さい。）

SAPではパートナー様の移行支援の為に、SAP Exodusというプロシジャやユーザー定義関数の移行支援ツール・変換ツールを用意しています。こちらの使用も御検討下さい。

データ型名の違いと互換性

プロシジャやユーザー定義関数では入出力でデータ型を意識する必要があります。

OracleとSQL Anywhereでは一部のデータ型名に互換性がないものが存在します。代表的なものを以下に示します。型名が違うのみで、データ型には互換性があります。

Oracle データ型	SQL Anywhere データ型	備考
VARCHAR2[(n)]	VARCHAR(n)	可変長文字列型
NUMBER(1), NUMBER(2)	TINYINT	符号なし整数値型（１バイト）
NUMBER(1) ～ NUMBER(4)	SMALLINT	整数値型（2バイト）
NUMBER(5) ～ NUMBER(9)	INTEGER	整数値型（4バイト）
NUMBER(10) ～ NUMBER(18)	BIGINT	整数値型（8バイト）
DATE	DATETIME	日付時刻型
DATE	DATE	日付型（時刻がデフォルトの場合）

プロシジャ

Oracle

```
CREATE OR REPLACE
PROCEDURE inout_sample
(
  no IN OUT NUMBER
)
IS
BEGIN
  no := no + 100;
END;
/
```

SQL Anywhere

```
CREATE OR REPLACE
PROCEDURE inout_sample
(
  INOUT no INT
)
BEGIN
  SET no = no + 100;
END;
```

ユーザー定義関数

Oracle

```
CREATE OR REPLACE FUNCTION
triangle
(
  base IN NUMBER DEFAULT 10,
  height IN NUMBER DEFAULT 20
)
RETURN NUMBER
IS
BEGIN
  RETURN((base * height)/2);
END;
/
```

SQL Anywhere

```
CREATE OR REPLACE FUNCTION
triangle
(
  base INT DEFAULT 10,
  IN height INT DEFAULT 20
)
RETURNS INT
BEGIN
  RETURN((base * height)/2);
END;
```

パッケージ

パッケージはSQL Anywhereに相当する機能が存在しないためパッケージを使用しない形で作り変えが必要です。

Oracle

```
CREATE OR REPLACE PACKAGE pkg1 AS
 PROCEDURE proc1 (arg1 char(2) )
 FUNCTION func1 (arg1 char(2) )
END pkg1;
CREATE OR REPLACE PACKAGE BODY pkg1 AS
    PROCEDURE proc1 (arg1 char(2) ) IS
    BEGIN
      ・・・・
    END
CREATE OR REPLACE PACKAGE BODY pkg1 AS
    FUNCTION func1 (arg1 char(2))  IS
    BEGIN
      ・・・・
    END
```

SQL Anywhere

```
CREATE OR REPLACE PROCEDURE pkg1_proc1
@arg1 char(2)
AS
  BEGIN
    ・・・
  END
CREATE OR REPLACE FUNCTION pkg1_func1
(arg1 char(2) )
RETURNS int
  BEGIN
    ・・・
  END
```

複数パッケージが存在し、同一名称のプロシジャやユーザー定義関数がある場合を考慮し、<元のパッケージ名>_<プロシジャや関数名>とするのが常套手段です。また、パッケージ内で宣言してパッケージ内で共用する変数が存在する場合はそれはセッション変数とする必要がありますのでそちらも同じように名前を変更して対応します。

その他

- 変数宣言

  Oracle

  ```
  DECLARE var1 CHAR(2) : = '01';
  ```

  SQL Anywhere

  ```
  DECLARE var1 CHAR(2) = '01’;
  ```

- 代入

  Oracle

  ```
  var1 : = '01';
  ```

  SQL Anywhere

  ```
  SET var1 = '01’;
  ```

- セッション内（接続内）で有効な変数の宣言

  Oracle

  パッケージを作成する。パッケージ内で宣言した変数はそのパッケージ内でセッションの間永続する。

  SQL Anywhere

  ```
  CREATE VARIABLE con_var INT;
  SET con_var = 10;
  //DROP VARIABLEするか、切断するまでそのセッション内でcon_varは永続する。
  ```

- IF-THEN-(ELSEIF)-ELSE

Oracle

```
IF var = 10 THEN
   DBMS_OUTPUT.PUT_LINE('値は 10');
 ELSIF var = 20 THEN
   DBMS_OUTPUT.PUT_LINE('値は 20');
 ELSE
   DBMS_OUTPUT.PUT_LINE('値はそれ以外');
 END IF;
```

SQL Anywhere

```
IF var = 10 THEN
   print '値は 10';
 ELSEIF var = 20 THEN
      print '値は 20';
 ELSE
      print '値はそれ以外';
END IF;
```

- GOTO

Oracle

```
GOTO lavel;
   DBMS_OUTPUT.PUT_LINE('処理1');
   DBMS_OUTPUT.PUT_LINE('処理2');
<<lavel>>
   DBMS_OUTPUT.PUT_LINE('ラベル処理');
```

SQL Anywhere

GOTO的な機能はないので、GOTOではない形に処理を変更する必要がある。
＊変更が困難な場合は応相談

- ループ処理

Oracle

```
LOOP
   EXIT WHEN c_count = 3;
   DBMS_OUTPUT.PUT_LINE('OK');
   c_count := c_count +1;
 END LOOP;
```

```
FOR c_count IN 1..3 LOOP
   DBMS_OUTPUT.PUT_LINE('OK');
 END LOOP;
```

SQL Anywhere

```
lbl:
 LOOP
     IF c_count = 3 THEN LEAVE lbl;
     END IF;
     MESSAGE 'OK' TO CLIENT;
     SET c_count = c_count + 1;
 END LOOP lbl;
```

```
SET c_count = 1;
WHILE i <= 3 LOOP
  MESSAGE 'OK' TO CLIENT;
  SET c_count = c_count + 1;
END LOOP;
```

同義は存在するが、完全一致は存在しない。なお、SQL AnywhereではFOR文はカーソルループの為の文であり、それ以外では使用できないことに注意。

- カーソルループ処理

Oracle

```
DECLARE
  CURSOR emp_cur IS SELECT empno, ename
  FROM emp WHERE deptno = 10;
BEGIN
  FOR emp_rec IN emp_cur LOOP
    DBMS_OUTPUT.PUT_LINE(emp_rec.empno || '   '
                         || emp_rec.ename);
  END LOOP;
END;
```

SQL Anywhere

```
BEGIN
  FOR cursor_loop AS emp_cur CURSOR FOR
    SELECT empno,ename FROM emp
    WHERE deptno = 10
  DO
    MESSAGE empno || '   ' || ename TO CLIENT;
  END FOR;
END
```

FORでカーソルのSQLを宣言するのが特徴

- パラメーター付きカーソルループ処理

Oracle

```
DECLARE
  CURSOR emp_cur(d_no NUMBER) IS
    SELECT empno, ename FROM emp
    WHERE deptno = d_no;
  d_var NUMBER;
BEGIN
  d_var := &DEPTNO;
  FOR emp_rec IN emp_cur(d_var) LOOP
    DBMS_OUTPUT.PUT_LINE(emp_rec.empno || '   ' ||
emp_rec.ename);
  END LOOP;
END;
```

SQL Anywhere

```
SET @qry='SELECT empno,ename FROM emp
    WHERE deptno = ';
@qry=@qry||string(d_var)

BEGIN
  FOR cursor_loop AS emp_cur CURSOR FOR @qry
  DO
    MESSAGE empno || '   ' || ename TO CLIENT;
  END FOR;
END
```

パラメーター付きカーソルの機能はSQL Anywhereに無いので、上記の様に動的SQLで代用するなどで置き換えが必要。

- エラーハンドリング

Oracle

```
DECLARE
  e_empno NUMBER;

BEGIN
  SELECT empno INTO e_empno FROM emp;
EXCEPTION
  WHEN too_many_rows THEN
    DBMS_OUTPUT.PUT_LINE('TOO_MANY_ROWS!');
  WHEN no_data_found THEN
    DBMS_OUTPUT.PUT_LINE('NO_DATA_FOUND!');
END;
```

SQL Anywhere

```
BEGIN
  DECLARE NO_DATA_FOUND EXCEPTION FOR SQLSTATE
'02000';
  DECLARE TOO_MANY_ROWS EXCEPTION FOR SQLSTATE
'21000';
  DECLARE e_empno INT;
  DECLARE msg VARCHAR(50);
  SELECT empno INTO e_empno FROM emp;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    MESSAGE 'NO_DATA_FOUND!' TO CLIENT;
  WHEN TOO_MANY_ROWS THEN
    MESSAGE 'TOO_MANY_ROWS!' TO CLIENT;
  WHEN OTHERS THEN
    MESSAGE 'その他の例外発生' TO CLIENT;
END;
```

Oracleはいくつかのエラーがシステム定義されており、それを利用してハンドリングするが、SQL Anywhereでは事前に定義が必要。

Thank you

Contact information:

F name L name
Title
Address
Phone number

F name L name
Title
Address
Phone number