PGECons
PostgreSQL Enterprise Consortium

PostgreSQL エンタープライズ・コンソーシアム 技術部会 WG#2

データ移行・文字コード変換編

異種 DBMS から PostgreSQL への移行

製作者
担当企業名
株式会社アシスト
NECソリューションイノベータ株式会社

改訂履歴

版	改訂日	変更内容
1.0	2014/04/17	新規作成

はじめに

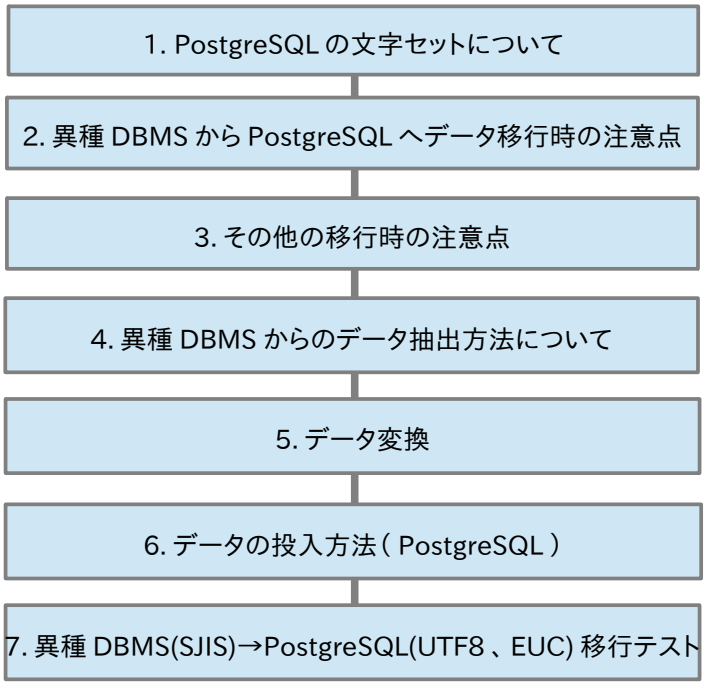
■本資料の概要と目的

本資料では、既存の異種 DBMS から PostgreSQL へのデータ移行に関して、文字コード変換について事前に判断するための参考資料として利用することを想定しています。

■資料内の記述について

本資料では、移行元の異種 DBMS として Oracle Database、Microsoft SQL Server、DB2 を想定し、各異種 DBMS から PostgreSQL へデータ移行する際の文字コード変換について、手順や留意事項を記載します。「はじめに」以降の各章に関しては、次のフローの項番名称に従い、記述されます。

図 1: 本資料の流れ



ETL については下図のような概念となり、本資料では「5.データ変換」に注力して記載しています。

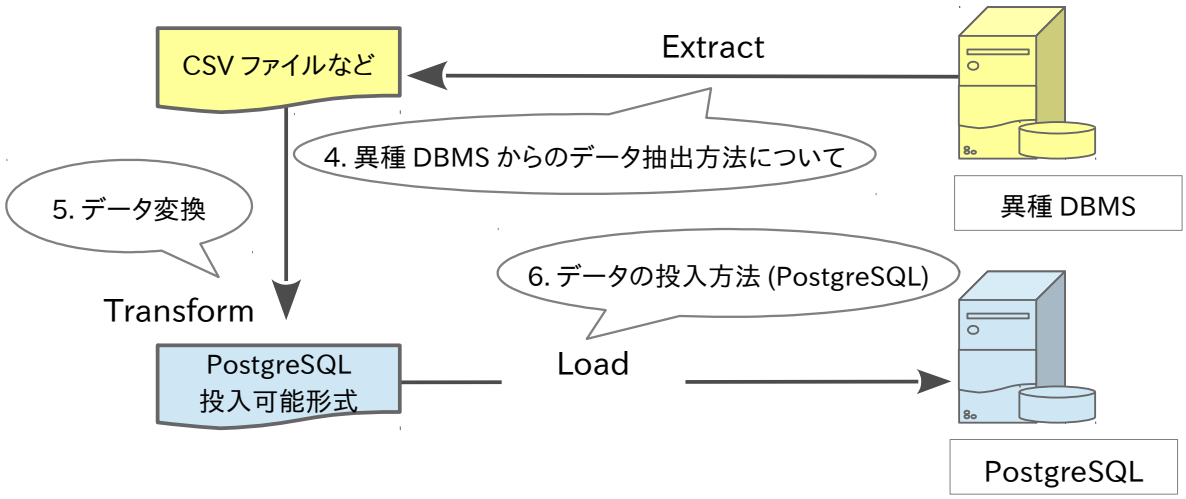


図 2: ETL の概念と各章のマッピング

■本資料で扱う用語の定義

本資料では、曖昧な意味としてとらえることができる用語を用いますが、次のような意味で記載しています。

表 1: 用語定義

No.	用語	意味
1	DBMS （でーびーえむえす）	データベース管理システムを指します。ここでは、PostgreSQL および異種 DBMS の総称として利用します。
2	異種 DBMS	PostgreSQL 以外のデータベース管理システムを指します。 本資料では、Oracle Database と Microsoft SQL Server、DB2 が該当します。
3	ETL （いーてぃーえる）	移行元から、データを抽出（Extract）、投入可能な形式に変換（Transform）、移行対象のデータベースに投入（Load）することの接頭文字を合わせて呼称する、一連の移行処理を指します。
4	文字コード	本資料では、任意の文字を定義するために用いられる 16 進数を指します。
5	文字セット	本資料では、DBMS が利用できる文字集合として呼称します。
6	文字エンコーディング	本資料では、DBMS が解釈できる文字集合をコンピュータ上の符号（文字コード）に対応させる文字符号化方式と定義します。
7	オブジェクト	PostgreSQL が取り扱うデータ型や DBMS 変数等の総称として用います。
8	Oracle	データベース管理システムの Oracle Databae を指します。
9	SQL Server	データベース管理システムの Microsoft SQL Server を指します

■本資料で扱うソフトウェア

本資料では、次のソフトウェアを用いてコマンド例証等を挙げています。以下ソフトウェアバージョンと異なるソフトウェアを用いる場合や、特別な設定等を入れて実行する場合は、実行結果を記載する章で利用するソフトウェアや設定について紹介します。

表 2: 動作環境

No.	環境名	実装	バージョン
1	検証対象 DBMS	PostgreSQL	9.3.3
2	移行元オペレーティングシステム	Windows	Windows 2012 Standard 64bit
3	移行先オペレーティングシステム	Linux	RedHat Enterprise Linux 6.5 64bit

表 3: 異種 DBMS 一覧

No.	環境名	バージョン	入手元（URI）
1	Oracle Database	12c Standard Edition	http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html
2	Microsoft SQL Server	2012 Express Edition	http://www.microsoft.com/ja-jp/download/details.aspx?id=29062
3	DB2	Express-C v10.5	http://www-06.ibm.com/software/jp/data/db2/express-c/download.html

表 4: コマンド・ツール一覧

No.	コマンド・ツール	バージョン	ライセンス	入手元（URI）	概要
1	COPY （こぴー）	- (PostgreSQL 標準 SQL コマンド)	The PostgreSQL Lincense	（PostgreSQL 同梱）	PostgreSQL においてファイルとテーブル間のデータをコピーするためのコマンド。 本資料では、データが列挙されているファイルからテーブルへデータを挿入する際に利用する。
2	psql （ぴー・えす・きゅー・える）	9.3.3	The PostgreSQL Lincense	（PostgreSQL 同梱）	PostgreSQL に同梱されている、ターミナル型フロントエンド。ファイルから入力を読み込むことも可能である。
3	pg_bulkload （ぴーじー・ばるくろーど）	3.1	The BSD 3-Clause License	http://pgbulkload.projects.pgfoundry.org/	PostgreSQL 向けの高速データ投入ツール。一般的に COPY より高速に動作する。
4	nkf （えぬけーえふ）	2.0.8	The zlib/libpng License	http://sourceforge.jp/projects/nkf/	ネットワーク漢字コードフィルタ。本資料では文字コード変換に用いる。
5	iconv （あいこんぶ）	2.12	GNU General Public License, version 3	http://www.gnu.org/software/libiconv/	International Codeset Conversion Library。異なる文字コード間の相互変換を行う。 本資料では、iconv コマンドを用いて、文字コード変換を行う際に利用する。
6	spool （すぷーる）	12.1.0.1.0	Oracle Technology Network Development and Distibution License	（Oracle Database に同梱）	問合せの結果をファイルに格納する場合に用いる。
7	bcp （びー・しー・ぴー）	2012	Microsoft EULA	（Microsoft SQL Server に同梱）	データベースとファイルの間でデータの一括コピーするときに用いる。
8	DB2 EXPORT （でーびーつー えくすぽーと）	10.5	IPLA	（DB2 に同梱）	データベースからファイルにデータをエクスポートする場合に用いる。

目次

1. PostgreSQL の文字セットについて

PostgreSQL の文字セットサポートにより、シングルバイト文字や EUC（Extended Unix Code）、UTF8 などのマルチバイト文字を含む、各種文字セットでテキストを保存することができます。

代表的な日本語の文字エンコーディングは以下の3つで、PostgreSQL でサポートされる文字セットは*表 1.1* に記載しています。

- SJIS
- EUC
- UTF8

PostgreSQL は SJIS の文字セットをサポートしていません。しかし、サーバ・クライアント間の文字セット変換により、SJIS のデータを扱うことができます。

デフォルトの文字セットは、initdb を使用した PostgreSQL データベースクラスタの初期化時に決定されます。また、データベース作成時にデフォルト以外の文字セットを作成することも可能です。

1.1. サポートされる文字セット

PostgreSQL で使用できる文字セットを表 1.1 に示します。

表 1.1: PostgreSQL の文字セット

名前	説明	言語	サーバ[1]	バイト数/文字	別名
BIG5	Big Five	繁体字	いいえ	1-2	WIN950、WIndows950
EUC_CN	Extended UNIX Code-C	簡体字	はい	1-3	
EUC_JP	Extended UNIX Code-JP	日本語	はい	1-3	
EUC_JIS_2004	Extended UNIX Code-JP、JIS X 0213	日本語	はい	1-3	
EUC_KR	Extended UNIX Code-KR	韓国語	はい	1-3	
EUC_TW	Extended UNIX Code-TW	繁体字、台湾語	はい	1-3	
GB18030	National Standard	中国語	いいえ	1-2	
GBK	Extended National Standard	簡体字	いいえ	1-2	WIN936、Windows936
ISO_8859_5	ISO 8859-5、ECMA 113	ラテン/キリル	はい	1	
ISO_8859_6	ISO 8859-6、ECMA 114	ラテン/アラビア語	はい	1	
ISO_8859_7	ISO 8859-7、ECMA 118	ラテン/ギリシャ語	はい	1	
ISO_8859_8	ISO 8859-8、ECMA 121	ラテン/ヘブライ語	はい	1	
JOHAB	JOHAB	韓国語（ハングル）	いいえ	1-3	
KOI8R	KOI8-R	キリル文字（ロシア）	はい	1	KOI8
KOI8U	KOI8-U	キリル文字（ウクライナ）	はい	1	
LATIN1	ISO 8859-1、ECMA 94	西ヨーロッパ	はい	1	ISO88591
LATIN2	ISO 8859-2、ECMA 94	中央ヨーロッパ	はい	1	ISO88592
LATIN3	ISO 8859-3、ECMA 94	南ヨーロッパ	はい	1	ISO88593
LATIN4	ISO 8859-4、ECMA 94	北ヨーロッパ	はい	1	ISO88594
LATIN5	ISO 8859-9、ECMA 128	トルコ	はい	1	ISO88599
LATIN6	ISO 8859-10、ECMA 144	北欧	はい	1	ISO885910
LATIN7	ISO 8859-13	バルト語派	はい	1	ISO885913
LATIN8	ISO 8859-14	ケルト	はい	1	ISO885914

1 サーバサイドエンコーディングとしての使用をサポート

名前	説明	言語	サーバ	バイト数/文字	別名
LATIN9	ISO 8859-15	LATIN1 でヨーロッパと訛りを含む	はい	1	ISO885915
LATIN10	ISO 8859-16、ASRO SR 14111	ルーマニア	はい	1	ISO885916
MULE_INTERNAL	Mule 内部コード	多言語 Emacs	はい	1-4	
SJIS	Shift JIS	日本語	いいえ	1-2	Mskanji、ShiftJIS、WIN932、Windows932
SHIFT_JIS_2004	Shift JIS、JIS X 0213	日本語	いいえ	1-2	
SQL_ASCII	未指定（テキストを参照）	何でも	はい	1	
UHC	統合ハングルコード	韓国語	いいえ	1-2	WIN949、Windows949
UTF8	Unicode、8 ビット	すべて	はい	1-4	Unicode
WIN866	Windows CP866	キリル文字	はい	1	ALT
WIN874	Windows CP874	タイ語	はい	1	
WIN1250	Windows CP1250	中央ヨーロッパ	はい	1	
WIN1251	Windows CP1251	キリル文字	はい	1	WIN
WIN1252	Windows CP1252	西ヨーロッパ	はい	1	
WIN1253	Windows CP1253	ギリシャ	はい	1	
WIN1254	Windows CP1254	トルコ	はい	1	
WIN1255	Windows CP1255	ヘブライ	はい	1	
WIN1256	Windows CP1256	アラビア語	はい	1	
WIN1257	Windows CP1257	バルト語派	はい	1	
WIN1258	Windows CP1258	ベトナム語	はい	1	ABC、TCVN、TCVN5712、VSCII

1.2. サーバ・クライアント間の自動文字セット変換

PostgreSQL では、データベース内に格納している文字セットとは異なる文字セットをクライアント側で使用する場合、自動的に変換する機能を実装しています。

自動文字セット変換を有効にするためには、クライアントでどのような文字セットを使用させたいかを以下のような方法で PostgreSQL に指定します。

- ¥encoding コマンド(psql)
- postgresql.conf の client_encoding 定義
- SET client_encoding TO(SQLL コマンド)
- クライアントに環境変数 PGCLIENTENCODING を定義
- クライアントに環境変数 LANG を定義

この機能を使用することによって、アプリケーションはデータベース内に格納されている文字セットを意識することなく、開発することができます。

変換情報は pg_conversion システムカタログに格納されています。

PostgreSQL では*表 1.2* に示すような組み合わせで変換が可能です。新しい変換を作成するには SQL コマンドの CREATE CONVERSION を使用します。

表 1.2: クライアント・サーバ文字セット変換

サーバ文字セット	利用可能なクライアント文字セット
BIG5	サーバの符号化方式としてはサポートしていません。[2]
EUC_CN	EUC_CN、 MULE_INTERNAL、 UTF8

2　サーバサイドエンコーディングとしての使用をサポートしていないため、事前に定義された変換はありません。

サーバ文字セット	利用可能なクライアント文字セット
EUC_JP	EUC_JP、 MULE_INTERNAL、 SJIS、 UTF8
EUC_KR	EUC_KR、 MULE_INTERNAL、 UTF8
EUC_TW	EUC_TW、 BIG5、 MULE_INTERNAL、 UTF8
GB18030	サーバの符号化方式としてはサポートしていません。
GBK	サーバの符号化方式としてはサポートしていません。
ISO_8859_5	ISO_8859_5、 KOI8、 MULE_INTERNAL、 UTF8、 WIN866、 WIN1251
ISO_8859_6	ISO_8859_6、 UTF8
ISO_8859_7	ISO_8859_7、 UTF8
ISO_8859_8	ISO_8859_8、 UTF8
JOHAB	JOHAB、 UTF8
KOI8R	KOI8R、 ISO_8859_5、 MULE_INTERNAL、 UTF8、 WIN866、 WIN1251
KOI8U	KOI8U、 UTF8
LATIN1	LATIN1、 MULE_INTERNAL、 UTF8
LATIN2	LATIN2、 MULE_INTERNAL、 UTF8、 WIN1250
LATIN3	LATIN3、 MULE_INTERNAL、 UTF8
LATIN4	LATIN4、 MULE_INTERNAL、 UTF8
LATIN5	LATIN5、 UTF8
LATIN6	LATIN6、 UTF8
LATIN7	LATIN7、 UTF8
LATIN8	LATIN8、 UTF8
LATIN9	LATIN9、 UTF8
LATIN10	LATIN10、 UTF8
MULE_INTERNAL	MULE_INTERNAL、 BIG5、 EUC_CN、 EUC_JP、 EUC_KR、 EUC_TW、 ISO_8859_5、 KOI8R、 LATIN1 to LATIN4、 SJIS、 WIN866、 WIN1250、 WIN1251
SJIS	サーバの符号化方式としてはサポートしていません。
SQL_ASCII	どれでも （変換されません）
UHC	サーバの符号化方式としてはサポートしていません。
UTF8	すべてサポートされています。
WIN866	WIN866、 ISO_8859_5、 KOI8R、 MULE_INTERNAL、 UTF8、 WIN1251
WIN874	WIN874、 UTF8
WIN1250	WIN1250、 LATIN2、 MULE_INTERNAL、 UTF8
WIN1251	WIN1251、 ISO_8859_5、 KOI8R、 MULE_INTERNAL、 UTF8、 WIN866
WIN1252	WIN1252、 UTF8
WIN1253	WIN1253、 UTF8
WIN1254	WIN1254、 UTF8
WIN1255	WIN1255、 UTF8
WIN1256	WIN1256、 UTF8
WIN1257	WIN1257、 UTF8
WIN1258	WIN1258、 UTF8

2. 異種 DBMS から PostrgeSQL へデータ移行時の注意点

以下に異種 DBMS から PostgreSQL にデータを移行する際に注意が必要な点について記載します。

2.1. 文字コードの違いによる注意点

SJIS（Shift_JIS）は、文字の 1 バイト目を見るだけで漢字か 1 バイト文字(半角英数字)かが分かります。等幅フォントで表示した場合に、画面上の桁数とバイト数が一致するなどの特長から、Microsoft 社の MS-DOS や Windows、Apple 社の Mac OS など、パソコンの標準文字コードとして広く普及しました。現在多くのパソコン上のファイル内で日本語を表すために使用されている文字コードです。

EUC（Extended Unix Code)は、UNIX 上でよく使用される文字コードの符号化方式で、複数バイトの文字を扱う文字コードの枠組みです。日本語だけでなく複数バイト言語の各国の文字コードが規定されています。日本語の EUC コードを特に「EUC-JP」や「日本語 EUC」と呼びます。

EUC-JP（Extended UNIX Code Packed Format for Japanese）は UNIX 上で日本語の文字を扱う場合に使用される文字コードの符号化方式のひとつです。UNIX 以外の OS 上で使われることもあります。

UTF8（Unicode Transformation Format8)は、UCS-2 や UCS-4(Unicode)で定義される文字集合を用いて記述された文字列をバイト列(数値の列)に変換する方式の一つです。UTF8 では 1 文字を 1～6 バイトの可変長の数値(バイト列)に変換するようになっていますが、現在定義されている Unicode 文字を UTF8 で表現した場合、最長で 4 バイトのバイト列に変換されます。

2.2. ダブルバイト文字の格納データサイズの違い

SJIS や EUC ではダブルバイト文字(漢字やひらがななど)は 2 バイトを必要としますが、UTF8 では 3 バイトを必要とします。そのため、SJIS や EUC でデータを格納しているデータベースと UTF8 のデータベース間でデータを移行する場合は、格納されるデータサイズの違いに注意し、テーブルサイズを見積もる必要があります。

2.3. 文字化けに関する注意点

文字エンコーディングや文字セットの異なる DBMS 間でのデータ移行では、移行先で文字化けが発生する可能性があります。そのため、文字エンコーディングだけでなく適切な文字セットに変換する必要があります。

例えば SJIS から EUC の環境にデータを移行する際には以下のような変換が考えられます。

・SJIS⇔EUC-JP
・SJIS-win⇔eucJP-win

2.4. OS の改行コードの違いに関する注意点

データベースサーバーのプラットフォームが Windows と Liunx および UNIX では改行コードが異なります。

改行コードは、それぞれ以下の通りです。

・Windows … CRLF（キャリッジリターン+ラインフィード）
・Linux/UNIX … LF（ラインフィード）、typo()

PostgreSQL にデータを投入する際、改行コードが異なるとエラーが発生するため注意が必要です。そのため、Windows と Linux/UNIX 間でのデータ移行の際は、移行先へデータを投入する際に改行コードが入っている場合は移行先の改行コードに変更する必要があります。

3. その他の移行時の注意点

3.1. 空文字の取り扱いについて

DBMS の仕様動作として、空文字を NULL として扱うケースと、文字として扱うケースがあります。
例えば Oracle は空文字を NULL として扱いますが、PostgreSQL は文字として扱います。そのため、Oracle と PostgreSQL 間でデータ移行をする際は、以下のように空文字の取り扱いについて考慮する必要があります。

1. NULL でも空文字でもない、第3の文字として挿入
2. NULL として挿入
3. NULL は空文字として挿入

NULL として挿入する場合は、setNull や setString 関数などを使用して データ登録のアプリを改修する方法や、検査制約(空文字を許さない)を定義するという方法が考えられます。

検査制約(空文字を許さない)定義例:LENGTH 関数を用いた場合

```
=# create table test(a integer,b text check (length(b) > 0));
CREATE TABLE
=# insert into test values(1,'');
ERROR:  new row for relation "test" violates check constraint "test_b_check"
DETAIL:  Failing row contains (1, ).
```

4. 異種 DBMS からのデータ抽出方法について

4.1. Oracle のデータ抽出方法

以下に Oracle Database のデータを抽出する方法を記載します。

4.1.1. spool コマンドによる CSV ファイルへのデータ抽出

Oracle の Scott ユーザの emp 表を例にします。
emp 表は以下のように定義されています。

```
SQL> desc emp
 名前                              NULL?    タイプ
 --------------------------------- -------- ----
 EMPNO                             NOT NULL NUMBER(4)
 ENAME                                      VARCHAR2(10
 JOB                                        VARCHAR2(9)
 MGR                                        NUMBER(4)
 HIREDATE                                   DATE
 SAL                                        NUMBER(7,2)
 COMM                                       NUMBER(7,2)
 DEPTNO                                     NUMBER(2)
```

CSV ファイルを作成する為に以下を実行します。

```
SQL> set echo off
SQL> set heading off
SQL> set termout off
SQL> set pause off
SQL> set pagesize 0
SQL> set linesize 80
SQL> set feedback off
SQL> spool emp.csv
SQL> select empno || ',' || ename || ',' ||
  2  job || ',' || sal from emp;
7369,SMITH,CLERK,800
7499,ALLEN,SALESMAN,1600
7521,WARD,SALESMAN,1250
7566,JONES,MANAGER,2975
7654,MARTIN,SALESMAN,1250
7698,BLAKE,MANAGER,2850
7782,CLARK,MANAGER,2450
7788,SCOTT,ANALYST,3000
7839,KING,PRESIDENT,5000
7844,TURNER,SALESMAN,1500
7876,ADAMS,CLERK,1100
7900,JAMES,CLERK,950
7902,FORD,ANALYST,3000
7934,MILLER,CLERK,1300
SQL> spool off
```

以上の操作により emp.csv という名前で CSV ファイルが作成されます。

4.1.2. ora2pg を使用したデータ抽出

ora2pg は Perl モジュールのデータ移行ツールです。使用する際は別途インストールが必要になります。
ora2pg には、3 種類の実行方法があります。

1)/etc/ora2pg/ora2pg.conf を使用

ora2pg の設定ファイルである/etc/ora2pg/ora2pg.conf を作成します。
インストール直後には/etc/ora2pg/ディレクトリに ora2pg.conf.dist ファイルが存在します。これを
コピーし、ora2pg.conf を作成します。
各種の設定ファイルを事前に用意して、実行前にファイルを置き換える運用になります。
この方式のメリット
・実行コマンドがシンプル
・Perl を意識する必要が無い

ora2pg.conf に以下の項目の設定を行います。

設定項目	説明
ORACLE_DSN	接続先 Oracle Database のホスト名と SID
ORACLE_USER	接続先 Oracle Database のユーザ名
ORACLE_PWD	接続先 Oracle Database のパスワード
TYPE	出力対象のオブジェクト定義抽出、データ抽出、または表示動作を指定 ●オブジェクト定義抽出 TABLE PACKAGE VIEW GRANT SEQUENCE TRIGGER FUNCTION PROCEDURE TABLESPACE TYPE ●データ抽出 DATA : INSERT 文生成 COPY : COPY 文生成 ●表示動作 SHOW_SCHEMA : データベース内で使用可能なスキーマのリストを表示 SHOW_TABLE : 使用可能なテーブルのリストを表示 SHOW_COLUMN : 使用可能な表列のリストを表示 SHOW_ENCODING :データベースの ENCODING を表示

例えば TYPE に SHOW_SCHEMA を指定して、ユーザの一覧を表示します。
設定例

```
ORACLE_DSN      dbi:Oracle:host=XXX.XXX.XXX.XXX;sid=base
ORACLE_USER     system
RACLE_PWD       oracle
TYPE                SHOW_SCHEMA
```

ora2pg コマンドを実行します。この時、ora2pg は/etc/ora2pg/ora2pg.conf を参照します。
実行例

```
$ ora2pg
Trying to connect to database: dbi:Oracle:host=XXX.XXX.XXX.XXX;sid=base
Isolation level: SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
Auto detecting Oracle character set and the corresponding PostgreSQL client encoding to use.
        Using Oracle character set: AMERICAN_AMERICA.JA16SJISTILDE.
        Using PostgreSQL client encoding: EUC_JIS_2004.
Force Oracle to compile schema before code extraction
Showing all schema...
SCHEMA APPQOSSYS
SCHEMA SCOTT
```

Oracle のスキーマの一覧が表示されれば成功です。

2)任意のパスの設定ファイルを使用

この方式のメリット

・設定ファイルの置き換え替えが不要

・任意のディレクトリに配置可能(/etc 配下に配置する必要が無い)

・Perl を意識する必要が無い

設定ファイルは任意のディレクトリに配置可能です。またファイル名も任意です。

ora2pg 実行時に-c オプションで設定ファイルを指定します。

実行例

カレントディレクトリに ora2pg_show_schema.conf を作成し、-c パラメータでパスを指定

```
$ cp /etc/ora2pg/ora2pg.conf  ora2pg_show_schema.conf
$ ora2pg -c ora2pg_show_schema.conf
```

3)設定ファイルとコマンドオプションの併用

Ora2pg には以下のコマンドオプションがあります。これらは設定ファイルの内容に優先します。

```
Usage: ora2pg [-dhpv] [--option value]

    -d | --debug      : Enable verbose output.
    -h | --help       : Print this short help.
    -v | --version    : Show Ora2Pg Version and exit.
    -c | --conf file  : Used to set an alternate configuration file than the
                        default /etc/or2pg/ora2pg.conf.
    -l | --log file   : Used to set a log file. Default is stdout.
    -o | --out file   : Used to set the path to the output file where SQL will
                        be written. Default: output.sql in running directory.
    -t | --type export: Used to set the export type. It will override the one
                        given in the configuration file (TYPE).
    -p | --plsql      : Enable PLSQL to PLPSQL code conversion.
    -s | --source     : Allow to set the Oracle DBI datasource.
    -u | --user       : Used to set the Oracle database connection user.
    -w | --password   : Used to set the password of the Oracle database user.
    -n | --namespace schema : Used to set the Oracle schema to extract from.
```

実行例

対象ユーザを SCOTT に指定(その他のパラメータは設定ファイルの指定)

```
$ ora2pg -n SCOTT
```

4)Perl スクリプト(設定ファイルを使用しない)

Perl スクリプトを作成し、ora2pg の export_schema を直接実行します。

スクリプト作成例

```
#!/usr/bin/perl
BEGIN {
  $ENV{ORACLE_HOME}='/opt/app/oracle/product/11.2.0/dbhome_1';
  $ENV{NLS_LANG} = 'JAPANESE_JAPAN.JA16SJIS';
}

use strict;
use Ora2Pg;

my $user= 'SCOTT;
my $schema=new Ora2Pg (
    datasource=>'dbi:Oracle:host=XXX.XXX.XXX.XXX;sid=base;port=1521',
    user=>'system',
    password=>'oracle',
    schema=>$user,
    type=>'TABLE',
```

```
    pg_numeric_type=>1,
    debug=>0
);

$schema->export_schema('/home/postgres/mig/sql/schema_'.$user.'.sql');
exit(0);
```

実行例

Perl スクリプトを exprot_schema_scott.pl とする

```
$ perl export_schema_scott.pl
```

この方式では、/etc/ora2pg/ora2pg.conf を参照しません。そのため必要なパラメータはスクリプト中に全て指定する必要があります。

この方式のメリット

・環境変数も設定可能

・必要なパラメータがコンパクトにまとまっており理解しやすい

・Perl スクリプトに機能を組み込むことが可能

4.2. SQL Server のデータ抽出方法

以下に Microsoft SQL Server のデータを抽出する方法を紹介します。

4.2.1. テーブルサイズの確認

例として、SQL Server 上に、EMP 表、DEPT表を作成しました。

Microsoft SQL Server Management Studio 上でテーブルを選択し、「右クリック⇒デザイン」でテーブル定義を確認します。

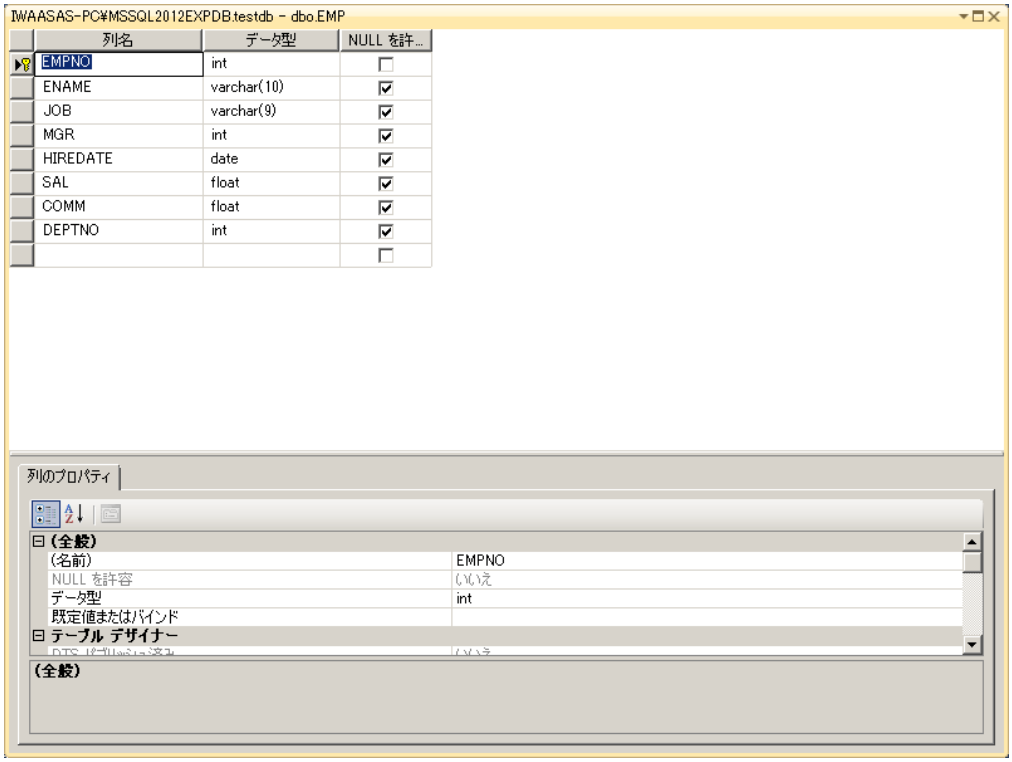


図 4.1: EMP 表定義

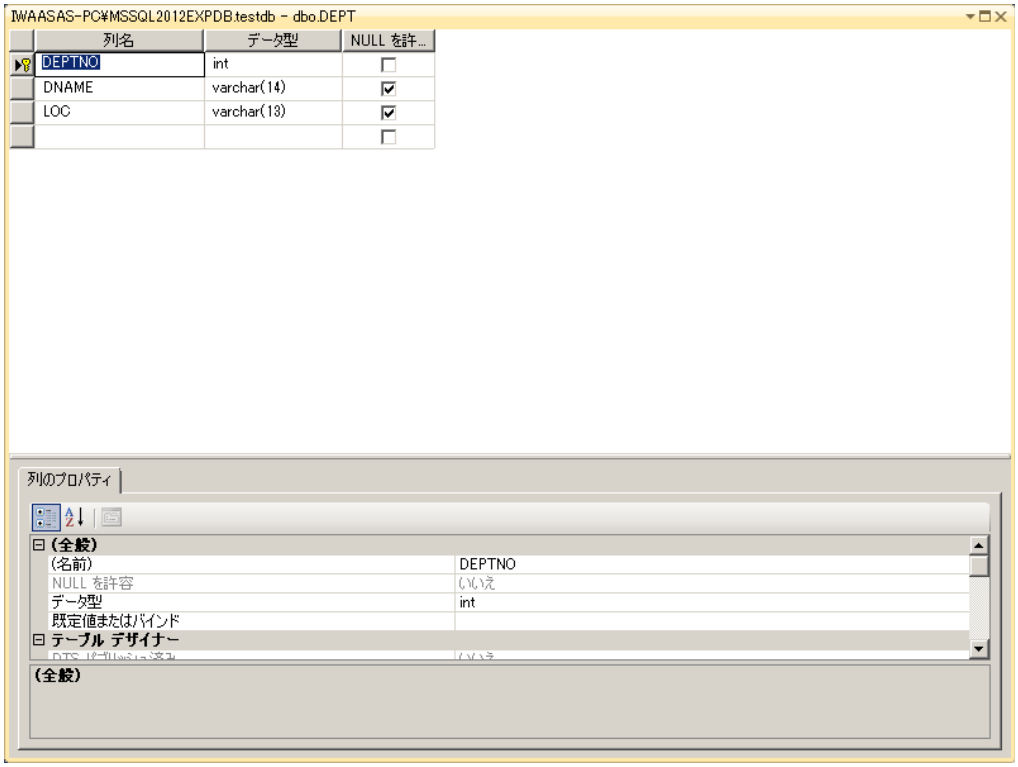


図 4.2: DEPT 表定義

テーブルにデータを格納し、Transact-SQL でレコード件数及びデータサイズを確認します。
サイズ確認のため、sp_MStablespace パッケージを使用しています。

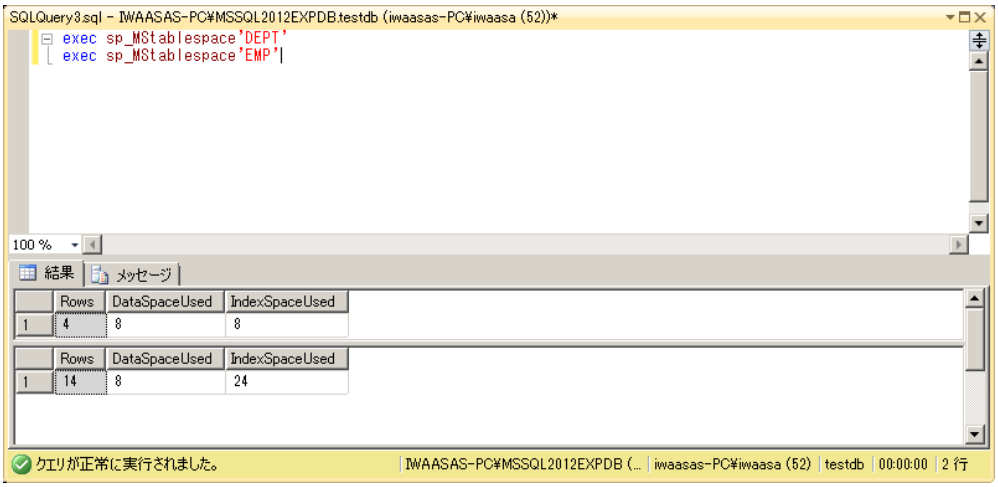


図 4.3: 規模確認

4.2.2. データの抽出

複数のテーブルからデータを抽出する場合、bcp[3]コマンドや SSIS[4]等を使用します。bcp コマンドや SSIS については、マイクロソフトのサイトで使用方法を確認してください。以下の例では、Microsoft SQL Server Management Studio からデータのエクスポートを行っています。作業手順を、DEPT 表の出力を例として紹介します。

3 http://msdn.microsoft.com/ja-jp/library/ms162802.aspx
4 http://technet.microsoft.com/ja-jp/library/ms141134.aspx

① データベースのタスクからエクスポートを選択

② 出力するデータソースを選択

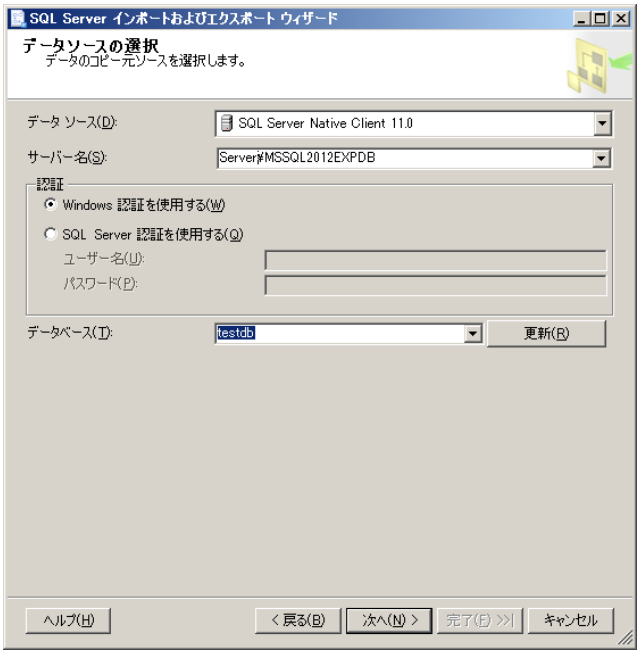


図 4.4: DB 指定

③ 出力形式を選択

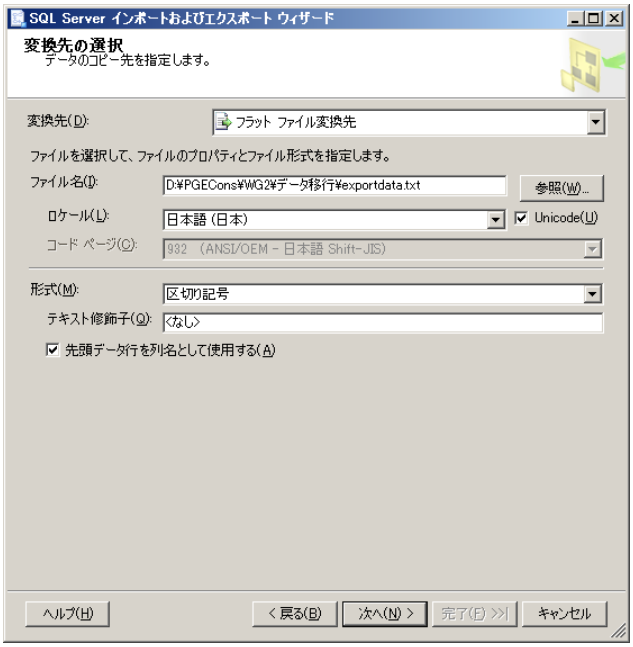


図 4.5: 出力先指定

④ 出力するオブジェクトの種別（テーブル／クエリ）を選択

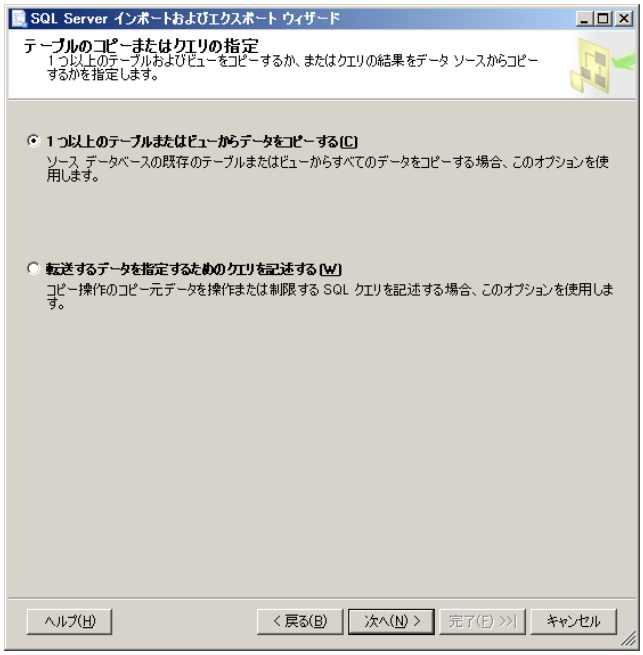


図 4.6: エクスポート対象種別選択

⑤　テーブルと出力形式を選択

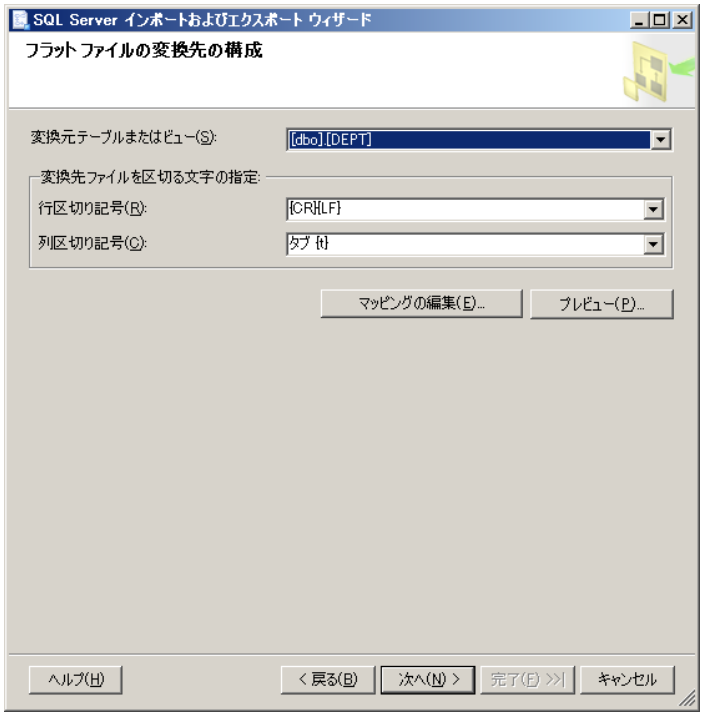


図 4.7: エクスポート対象選択

⑥　実行

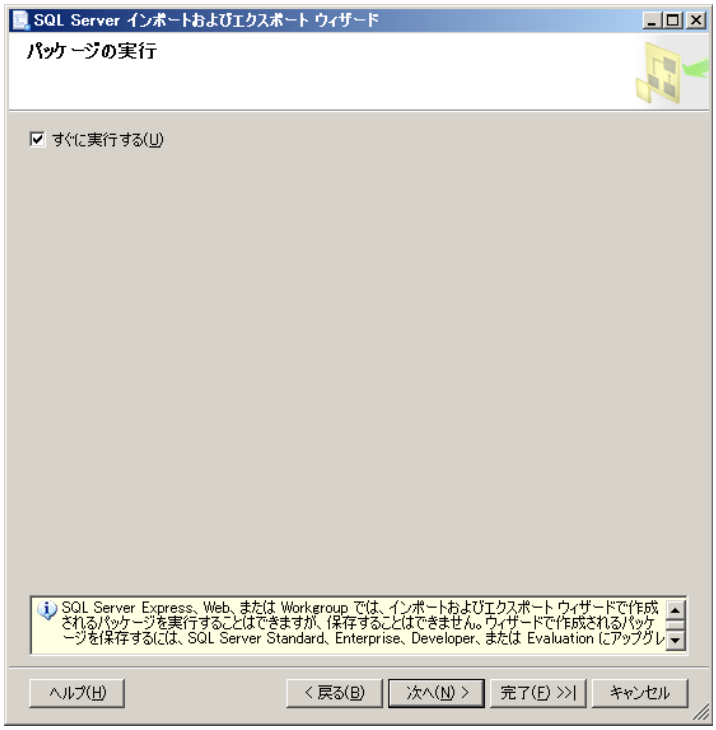


図 4.8: 実行

上記手順により、先頭行にカラム名のレコードを含むタブ区切りのデータ出力が行われます。

4.3. 大量データの場合のデータ連携

データ件数やデータサイズが非常に大きい場合、移行作業に非常に時間がかかるうえ、既存システムへの影響も非常に大きくなる可能性があり、移行作業における要件に応じて、移行の手段を選択する必要があります。

1. ワンポイントでの一括移行

   移行元のシステム停止から新システムの開始までに十分な移行期間を確保することができ、データ移行に必要なリソースを確保できる場合には、移行作業に入ってからデータの抽出、加工、投入で問題ありません。

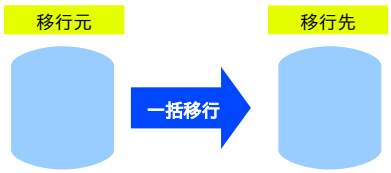


図 4.9: 一括移行

2. 一部データの事前移行

   移行すべきデータを選別し、更新される可能性が無いデータは事前に移行し、更新がかかるデータの移行を移行作業期間中に行います。記事など蓄積型のデータは後で更新されることが少ないと考えられ、移行作業中に移行するデータ量を削減することにより作業を短期間で完了させることができます。

   ただし、事前にデータを抽出する際に既存システムへの負荷がかかる可能性があるため、性能への影響を考慮する必要があります。また、事前に移行するデータと移行作業期間中のデータが明確に分かれているか、何らかのチェックポイントを設定して未反映のデータが判断できるようにしておく必要があるため、移行作業における設計が重要となります。

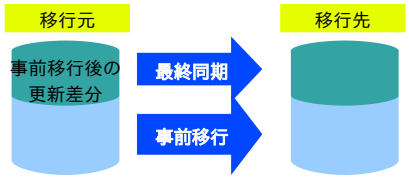


図 4.10: 事前移行+最終同期

3. データ連携

移行元システムの DB と PostgreSQL を使用する新システムを並行運用したり、更新データを随時新システムに反映することにより、データの移行期間を最小限にすることができます。

移行元システムの DB からデータを抽出、加工し PostgreSQL に投入するため、移行元のシステムにかかる負荷が課題となる可能性があるほか、ETL／データ連携ツールや専用アプリケーションを構築する必要があるため、移行費用が大きくなる可能性があります。

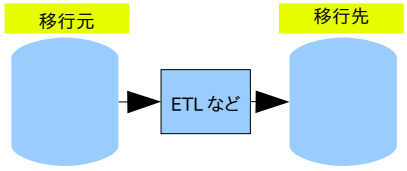


図 4.11: データ連携

5. データの変換

既存システムの DB からデータを移行する際、抽出したデータをそのまま移行することができない場合があります。本章では、データ変換が必要なケースや変換方法について紹介します。

5.1. 事前検討・確認

5.1.1. データ移行に関する各種検討および確認

以下のケースでは、データ移行時にデータの変換が発生する可能性があります。

① 定義の変更

システムの変更に対応した定義の変更や、DBMS の仕様の差異により止む無く定義を変更している場合、移行作業においてデータの形式等を変換する必要があります。

② 値に対する意味の違い

0 バイト文字列（DBMS により NULL 値と同等とみなす）など、格納する値の意味が DBMS により異なる場合があります。アプリケーション移行時の仕様調整を行い、必要に応じてデータの変換を行う必要があります。

③ 属性ごとの格納可能なデータサイズや範囲の違い

文字列のサイズの指定方法の違い（バイトまたは文字数）や、同じデータ型でも格納可能な限界値に差異がある可能性があります。また、数値の丸めや切り捨てに関する仕様の違いにより、移行前後の検索結果に差異が発生する可能性があるため、事前の確認が必要です。

④ データ量の変動

データベースへの格納文字コードを変更した場合、データ量が増減する可能性があります。既存システムの SJIS や EUC_JP で格納された DB を UNICODE に移行する場合、データ量が大きく変わる可能性があります。

⑤ 各 DBMS に固有な問題

移行元 DB に固有な問題により、データ移行時に文字化け等の問題が発生する可能性があります。

5.1.2. 移行元（Oracle、SQL Server、DB2）

移行元の DBMS により、選択可能な格納文字コードが異なります。
日本国内で使用される主な文字コードは、以下の通りです。

表 5.1: DBMS 別の格納文字コード

DBMS	選択可能な格納文字コード		備考
Oracle Database	SJIS	JA16SJIS	
		JA16SJISTILDE (9i R2 以降)	チルダ文字化け対応
	EUC_JP	JA16EUC	
		JA16EUCTILDE (9i R2 以降)	チルダ文字化け対応
	UNICODE	AL32UTF8	
Microsoft SQL Server	Japanese_CI_AS	SJIS (ms932)	char、varchar
		UNICODE	nchar、nvarchar
DB2	UNICODE	UTF8	既定値。
	EUC_JP	IBM-eucJP,eucJP	OS、テリトリー別に指定可能なコードセットが異なる。
	SJIS	SJIS,IBM-943	OS、テリトリー別に指定可能なコードセットが異なる。

5.1.3. 移行先の文字コード

移行先である PostgreSQL は、マルチバイト文字を格納する可能性がある場合、以下の何れかの文字コードを選択します。住所、名前など JIS2004 で追加された文字が使用される可能性がある場合、UTF8 を選択することを推奨します。

- EUC_JP
- UTF8

5.1.4. 移行元 DB の文字コードの特性

文字コード変換のにおいて良く知られている問題に、”～”の文字化けの問題があります。
“～”の文字化けは以下の条件で発生します。

- Oracle DB のコードセットが JA16SJIS
- Java、ODBC などの API を使用し、UNICODE 環境の DB クライアントを使用して文字コードを抽出
- 抽出したデータを SJIS（MS932）に変換

クライアントアプリケーション内部の文字コードが UNICODE の場合、UNICODE の“～（U+FF5E）”は Oracle 格納時に SJIS のコード“0x8160”に変換されています。これを同様に UNICODE 環境のアプリケーションで読み出したとき、Oracle は同じ文字を UNICODE の”U+310C”に変換してしまうため、入力した文字とは異なるコードになってしまいます。また、UNICODE の”U+310C”に対応する SJIS の文字が存在しないため、SJIS 環境のクライアントに出力しようとすると文字化けが発生します。

コードセットが JA16SJIS の Oracle データベースからデータをエクスポートする際、”～”を含むデータが存在する場合には UNICODE に変換せず、SJIS のまま抽出して PostgreSQL へのインポート時に UNICODE に変換する必要があります。

5.1.5. データ移行に使用するツール

データ移行作業で使用する主なツールを紹介します。

表 5.2: データ移行時に使用するツール

作業項目	ツール	備考
抽出	Oracle Database：Ora2Pg、sqlplus Microsoft SQL Server：bcp コマンド DB2：CLP（EXPORT コマンド）	製品に含まれるツールの一例であり、挙がっているツールがすべてではありません。
変換	nkf、iconv	
投入	psql、pg_bulkload	

5.1.6. 変換のタイミング

コード変換は、以下の何れかのタイミングで実施することが考えられます。

- 移行元での変換／抽出
- 抽出データに対する変換
- 移行先でのデータ投入時の変換

いずれの方法を使用するかは、移行作業にかかわる各種の要件から検討します。

- 移行環境の制約
  データ抽出時のファイル容量や移行元／移行先サーバ環境の性能により、移行元での変換・抽出が難しい場合が考えられます。また、サーバの集約や仮想環境への移行により、移行先のサーバスペックが不十分な可能性もあります。
- 移行作業の段取り
  文字コード変換を行わない方が抽出、投入時間を短縮することができます。アプリケーション移行等の都合によりデータの抽出／投入作業が連続して行うことができない場合、無変換で抽出し、抽出データの変換を別途行った上で投入する方法が選択できます。
- 移行元 DBMS に固有な問題
  抽出／投入時も文字コードの変換により問題が発生することが事前に判明している場合、抽出したデータに対して変換やコードの書き換えを行い、変更後のデータを投入します。

6. データの投入方法（PostgreSQL）

移行元システムの DB から抽出したデータを PostgreSQL に投入する方法として、以下が考えられます。

- PostgreSQL 標準機能（COPY 文）
- 高速ローダ（pg_bulkload）

6.1. PostgreSQL 標準機能（COPY 文）

psql コマンドなど PostgreSQL の SQL 文を実行する手段を利用し、COPY 文によるデータの投入を行います。

COPY 文に対して標準入力もしくはファイルにより投入するデータを渡す必要がありますが、ファイルを入力データとする場合にはファイルが DB サーバから直接参照可能な場所に配置されている必要があります。また、入力するデータが CSV、カンマ区切り、タブ区切り等のいずれの形式かを指定する必要があります。

データを格納するテーブルの定義または TRUNCATE を実行したトランザクションの中で COPY 文を実行した場合、WAL の出力が抑制され、データ投入の応答時間が短縮する可能性があります。

COPY コマンドの入力データ形式や文法については、PostgreSQL のマニュアル[5]をご覧ください。

6.2. 高速ローダ（pg_bulkload）

PostgreSQL の標準機能ではなく、独立した OSS として公開[6]されているツールです。

CSV、固定長バイナリ形式のデータを入力し、データを高速に投入します。定義変更への対応や実行中に発生したエラーレコードの抽出や参照制約チェックのスキップなど、移行時に必要となる様々な機能を提供しています。

pg_bulkload の設定やコマンドの記述方法については、pg_bulkload のドキュメント[7]をご覧ください。

5 http://www.postgresql.jp/document/current/html/sql-copy.html
6 http://pgfoundry.org/projects/pgbulkload
7 http://pgbulkload.projects.pgfoundry.org/pg_bulkload-ja.html

7. 異種 DBMS（SJIS）→PostgreSQL（UTF8、EUC）移行テスト

本章では、ある試験データの移行を実際に行った結果を示します。

7.1. 検証環境

異種 DBMS から PostgreSQL にデータ移行する際に使用した検証環境を紹介します。

7.1.1. ハードウェア

表 7.1: ハードウェア

サーバ	Express5800/R120d-1E(4C/E5-2407) × 2
スペック	メインメモリ：32GB、内蔵 HDD：SATA 1TB×2、ネットワークカード：6port
その他	L2-SW(24port 程度) × 1

7.1.2. ソフトウェア

表 7.2: ソフトウェア

ソフトウェア名	エディション・バージョン	用途
Oracle Database	Standard Edition 12c	移行元 DB
Microsoft SQL Server	2012 Express Edition	移行元 DB
DB2	Express-C for Windows 64-bit	移行元 DB
PostgreSQL	9.3.3	移行先 DB
pg_bulkload	3.1	移行ツール

7.1.3. テストデータの基礎情報

表 7.3: 基礎情報

項目	内容
入手元	郵便番号データ[8]
データ件数	123,584 件（件数を増やす場合には 5 倍、10 倍にして使用）

8 http://www.post.japanpost.jp/zipcode/dl/oogaki-zip.html

7.1.4. テストデータのレコード形式

表 7.4: レコード形式

項番	内容	属性	サイズ	備考
1	全国地方公共団体コード	数字	5	char
2	旧郵便番号	数字	5	char
3	郵便番号	数字	7	char
4	都道府県名	文字列(カナ)	可変長	text
5	市区町村名	文字列(カナ)	可変長	text
6	町域名	文字列(カナ)	可変長	text
7	都道府県名	文字列(漢字)	可変長	text
8	市区町村名	文字列(漢字)	可変長	text
9	町域名	文字列(漢字)	可変長	text
10	複数の郵便番号を持つ町域フラグ	char	1	1 or 0
11	小字毎に番地を持つ町域フラグ	char	1	1 or 0
12	丁目の有無フラグ	char	1	1 or 0
13	複数町域にまたがる番号フラグ	char	1	1 or 0
14	更新の有無フラグ	char	1	0:変更なし、1:変更あり、2:廃止
15	変更理由	char	1	0:変更なし、1:市政・区政・町政・分区・政令指定都市施行、2:住居表示の実施、3:区画整理、4:郵便区調整等、5:訂正、6:廃止

Oracle、SQL Server、DB2 共に、以下のテーブル定義を実行し、データベースのテーブルを作成しました。

テストデータのテーブル定義

```
CREATE TABLE japanese_postal_code(
    regions              char(5),
    old_code             char(5),
    new_code             char(7),
    prefectures_yomi     character varying(1024),
    municipality_yomi    character varying(1024),
    area_yomi            character varying(1024),
    prefectures          character varying(1024),
    municipality         character varying(1024),
    area                 character varying(1024),
    area_overlap         char(1),
    partition            char(1),
    partition2           char(1),
    code_overlap         char(1),
    update_flag          char(1),
    update_reason        char(1)
    );


CREATE INDEX japanese_postal_code_regions_idx ON japanese_postal_code(regions);
CREATE INDEX japanese_postal_code_old_code_idx ON japanese_postal_code(old_code);
CREATE INDEX japanese_postal_code_new_code_idx ON japanese_postal_code(new_code);
```

データロード時にインデックス定義が存在する場合と存在しない場合の応答時間を比較するため、インデックスを定義しています。また、pg_bulkload による並列インデックス生成の効果を確認するために複数のインデックスを定

義しています。

7.2. 検証環境の構成

表 7.5: 検証環境の構成

	移行元	移行先	備考
OS	Windows 2012 Standard	Redhat Enterprise Linux 6.5	Windows、Linux 共に 64Bit 環境
DB	DB2 10.5	PostgreSQL 9.3	
	Microsoft SQL Server 2012 Express Edition		
	Oracle 12c Standard Edition		
HDD	内蔵 HDD(RAID0)	内蔵 HDD(RAID0)	可用性は不要と判断し、RAID0 にしています。

7.2.1. 移行元移行先 DBMS の文字コード設定値

表 7.6: 移行元移行先データベース文字コード情報

環境名	値	備考
Oracle Database 文字コード	SJIS	
Microsoft SQL Server 文字コード	Japanese_CI_AS	
DB2 文字コード	SJIS,IBM-943	
移行先 PostgreSQL 文字コード	UTF8	

7.3. 評価項目

評価項目として、各 DBMS 環境においてデータの抽出、データ変換、PostgreSQL へのデータ格納（変換あり、無し、インデックスあり、無し）、そしてシナリオ評価を実施しました。

表 7.7: 評価項目

項番	大項目	中項目	内容	レコード件数
	エクスポート			
7.4.1		Oracle Database エクスポート	SJIS→SJIS の CSV ファイル出力	12 万/60 万/120 万
			SJIS→UNICODE の CSV ファイル出力	12 万/60 万/120 万
7.4.2		Microsoft SQL Server エクスポート	Japanese_CI_AS→SJIS の CSV ファイル出力	12 万/60 万/120 万
			Japanese_CI_AS→UNICODE の CSV ファイル出力	12 万/60 万/120 万
7.4.3		DB2 エクスポート	SJIS,IBM-943→SJIS の CSV ファイル出力	12 万/60 万/120 万
			SJIS,IBM-943→UNICODE の CSV ファイル出力	12 万/60 万/120 万
	データ変換			
7.5		nkf	SJIS→UTF8	12 万/60 万/120 万
			UTF16LE→UTF8	12 万/60 万/120 万
		iconv	SJIS→UTF8	12 万/60 万/120 万
			UTF16LE→UTF8	12 万/60 万/120 万
	インポート			
7.6.1	インデックス定義無し	COPY(変換あり)	SJIS→UTF8 のデータロード・WAL あり	12 万/60 万/120 万
		COPY(変換あり)	SJIS→UTF8 のデータロード・WAL 無し	12 万/60 万/120 万
		pg_bulkload(変換あり)	SJIS→UTF8 のデータロード	12 万/60 万/120 万
		COPY(変換無し)	UTF8→UTF8 のデータロード・WAL あり	12 万/60 万/120 万
		COPY(変換無し)	UTF8→UTF8 のデータロード・WAL 無し	12 万/60 万/120 万
		pg_bulkload(変換無し)	UTF8→UTF8 のデータロード	12 万/60 万/120 万
	インデックス定義あり（3 インデックス）	COPY(変換あり)	SJIS→UTF8 のデータロード	12 万/60 万/120 万
		NOWALCOPY(変換あり)	SJIS→UTF8 のデータロード	12 万/60 万/120 万
		pg_bulkload(変換あり)	SJIS→UTF8 のデータロード	12 万/60 万/120 万
		COPY(変換無し)	UTF8→UTF8 のデータロード	12 万/60 万/120 万
		NOWALCOPY(変換無し)	UTF8→UTF8 のデータロード	12 万/60 万/120 万
		pg_bulkload(変換無し)	UTF8→UTF8 のデータロード	12 万/60 万/120 万
	シナリオ評価			
7.7		エラーデータ対応 (pg_bulkload)	不正データを格納し、正常データのみロード(エラーデータは別ファイルに格納)	12 万
		エラーデータ復旧	別ファイルに出力したエラーデータを直接編集し、エラーデータのみロード	

7.4. データ抽出の結果（異種 DBMS）

本試行では、異種 DBMS の各エクスポート処理について、SJIS の CSV ファイルと UNICODE の CSV ファイル形式で 12 万レコード、60 万レコード、120 万レコードの各 3 パターンの出力を行いました。各処理の実行スクリプトと抽出時間の結果は以下の通りです。

7.4.1. Oracle からのデータ抽出

Oracle からの抽出処理の実行スクリプトは以下の通りです。

DIRECTORY、PROCEDURE 作成 DDL

```
CREATE DIRECTORY res_sjis_out_dir AS 'C:¥Users¥Administrator¥Desktop¥result¥oracle2¥sjis';
CREATE DIRECTORY res_unicode_out_dir AS 'C:¥Users¥Administrator¥Desktop¥result¥oracle2¥unicode';
GRANT READ ON DIRECTORY res_sjis_out_dir TO DBA;
GRANT WRITE ON DIRECTORY res_sjis_out_dir TO DBA;
GRANT READ ON DIRECTORY res_unicode_out_dir TO DBA;
GRANT WRITE ON DIRECTORY res_unicode_out_dir TO DBA;
CREATE OR REPLACE PROCEDURE postal_out(dirname in varchar2,filename in varchar2,tblname in varchar2)
IS
        csvout   UTL_FILE.FILE_TYPE;
        type curtype is ref cursor;
        cur curtype;
        rec japanese_postal_code%rowtype;
BEGIN
        csvout := UTL_FILE.FOPEN(dirname,filename,'W',10240);
        OPEN cur for 'SELECT * from '||tblname;
        LOOP
                FETCH cur INTO rec;
                exit when cur%notfound;
                UTL_FILE.PUT_LINE(csvout,
                        rec.regions                             ||',"'||
                        rec.old_code                    ||'",'||
                        rec.new_code                    ||'",'||
                        rec.prefectures_yomi            ||'",'||
                        rec.municipality_yomi           ||'",'||
                        rec.area_yomi                   ||'",'||
                        rec.prefectures                 ||'",'||
                        rec.municipality                ||'",'||
                        rec.area                        ||'",'||
                        rec.area_overlap                ||','||
                        rec.partition                   ||','||
                        rec.partition2                  ||','||
                        rec.code_overlap                ||','||
                        rec.update_flag                 ||','||
                        rec.update_reason
                                );
        END LOOP;
        UTL_FILE.FCLOSE(csvout);
END;
/
CREATE OR REPLACE PROCEDURE postal_out_uni(dirname in varchar2,filename in varchar2,tblname in
varchar2)
IS
        csvout   UTL_FILE.FILE_TYPE;
        type curtype is ref cursor;
        cur curtype;
        rec japanese_postal_code%rowtype;
BEGIN
        csvout := UTL_FILE.FOPEN_NCHAR(dirname,filename,'W',10240);
        OPEN cur for 'SELECT * from '||tblname;
        LOOP
                FETCH cur INTO rec;
                exit when cur%notfound;
```

```
                UTL_FILE.PUT_LINE_NCHAR(csvout,
                        rec.regions                             ||',"'||
                        rec.old_code                    ||'",""'||
                        rec.new_code                    ||'",""'||
                        rec.prefectures_yomi            ||'",""'||
                        rec.municipality_yomi           ||'",""'||
                        rec.area_yomi                   ||'",""'||
                        rec.prefectures                 ||'",""'||
                        rec.municipality                ||'",""'||
                        rec.area                        ||'",'||
                        rec.area_overlap                ||','||
                        rec.partition                   ||','||
                        rec.partition2                  ||','||
                        rec.code_overlap                ||','||
                        rec.update_flag                 ||','||
                        rec.update_reason
                                );
        END LOOP;
        UTL_FILE.FCLOSE(csvout);
END;
/
quit
```

実行 SQL(SJIS ファイル変換)

```
execute postal_out('&1','&2','&3');
quit
```

実行 SQL(UTF8 ファイル変換)

```
execute postal_out_uni('&1','&2','&3');
quit
```

spool 実行(SJIS ファイル変換:12 万レコード、別途 60 万レコード、120 万レコードを実行)

```
set echo off
set heading off
set termout off
set pause off
set pagesize 0
set linesize 10240
set trimspool on
set feedback off
spool C:¥Users¥Administrator¥Desktop¥result¥oracle¥sjis¥export.csv
select
        regions                         ||',"'||
        old_code                ||'","'||
        new_code                ||'","'||
        prefectures_yomi        ||'","'||
        municipality_yomi       ||'","'||
        area_yomi                       ||'","'||
        prefectures                     ||'","'||
        municipality                    ||'","'||
        area                            ||'",'||
        area_overlap                    ||','||
        partition                       ||','||
        partition2                      ||','||
        code_overlap                    ||','||
        update_flag                     ||','||
        update_reason
from
        japanese_postal_code
;
spool off
```

```
;
quit
```

spool 実行（UTF8 ファイル変換:12 万レコード、別途 60 万レコード、120 万レコードを実行）

```
set echo off
set heading off
set termout off
set pause off
set pagesize 0
set linesize 10240
set trimspool on
set feedback off
spool C:¥Users¥Administrator¥Desktop¥result¥oracle¥unicode¥export.csv
select
        regions                    ||',"'||
        old_code          ||'",'||
        new_code          ||'",'||
        prefectures_yomi  ||'",'||
        municipality_yomi ||'",'||
        area_yomi                  ||'",'||
        prefectures                ||'",'||
        municipality               ||'",'||
        area                       ||'",'||
        area_overlap               ||','||
        partition                  ||','||
        partition2                 ||','||
        code_overlap               ||','||
        update_flag                ||','||
        update_reason
from
        japanese_postal_code
;
spool off
;
quit
```

実行スクリプト（SJIS ファイル変換）

```
@echo off
echo SJIS-EXPORT
echo START:%time%
sqlplus system/Qwer1234 @C:¥Users¥Administrator¥Desktop¥result¥oracle2¥exec.sql "RES_SJIS_OUT_DIR"
"export.csv" "japanese_postal_code"
echo END:%time%

echo SJIS-EXPORT-MIDDLE
echo START:%time%
sqlplus system/Qwer1234 @C:¥Users¥Administrator¥Desktop¥result¥oracle2¥exec.sql "RES_SJIS_OUT_DIR"
"export_mid.csv" "japanese_postal_code_middle"
echo END:%time%

echo SJIS-EXPORT-LARGE
echo START:%time%
sqlplus system/Qwer1234 @C:¥Users¥Administrator¥Desktop¥result¥oracle2¥exec.sql "RES_SJIS_OUT_DIR"
"export_lar.csv" "japanese_postal_code_large"
echo END:%time%

echo TERMINATED
```

実行スクリプト（UTF8 ファイル変換）

```
@echo off
```

```
echo UTF-EXPORT
echo START:%time%
sqlplus system/Qwer1234 @C:¥Users¥Administrator¥Desktop¥result¥oracle2¥exec_uni.sql
"RES_UNICODE_OUT_DIR" "export.csv" "japanese_postal_code"
echo END:%time%

echo UTF-EXPORT-MIDDLE
echo START:%time%
sqlplus system/Qwer1234 @C:¥Users¥Administrator¥Desktop¥result¥oracle2¥exec_uni.sql
"RES_UNICODE_OUT_DIR" "export_mid.csv" "japanese_postal_code_middle"
echo END:%time%

echo UTF-EXPORT-LARGE
echo START:%time%
sqlplus system/Qwer1234 @C:¥Users¥Administrator¥Desktop¥result¥oracle2¥exec_uni.sql
"RES_UNICODE_OUT_DIR" "export_lar.csv" "japanese_postal_code_large"
echo END:%time%

echo TERMINATED
```

7.4.2. SQL Server からのデータ抽出

SQL Server からの抽出処理の実行スクリプトは以下の通りです。

設定ファイル（SJIS ファイル変換）

```
11.0
15
1       SQLCHAR         0       5       ",¥""           1       regions                 Japanese_CI_AS
2       SQLCHAR         0       5       "¥",¥""         2       old_code                Japanese_CI_AS
3       SQLCHAR         0       7       "¥",¥""         3       new_code                Japanese_CI_AS
4       SQLCHAR         0       1024    "¥",¥""         4       prefectures_yomi        Japanese_CI_AS
5       SQLCHAR         0       1024    "¥",¥""         5       municipality_yomi       Japanese_CI_AS
6       SQLCHAR         0       1024    "¥",¥""         6       area_yomi               Japanese_CI_AS
7       SQLCHAR         0       1024    "¥",¥""         7       prefectures             Japanese_CI_AS
8       SQLCHAR         0       1024    "¥",¥""         8       municipality            Japanese_CI_AS
9       SQLCHAR         0       1024    "¥","           9       area                    Japanese_CI_AS
10      SQLCHAR         0       1       ","             10      area_overlap            Japanese_CI_AS
11      SQLCHAR         0       1       ","             11      partition               Japanese_CI_AS
12      SQLCHAR         0       1       ","             12      partition2              Japanese_CI_AS
13      SQLCHAR         0       1       ","             13      code_overlap            Japanese_CI_AS
14      SQLCHAR         0       1       ","             14      update_flag             Japanese_CI_AS
15      SQLCHAR         0       1       "¥r¥n"          15      update_reason           Japanese_CI_AS
```

設定ファイル（UTF16 ファイル変換）

```
11.0
15
1       SQLNCHAR        0       10      ",¥0¥"¥0"       1       regions                 Japanese_CI_AS
2       SQLNCHAR        0       10      "¥"¥0,¥0¥"¥0"   2       old_code                Japanese_CI_AS
3       SQLNCHAR        0       14      "¥"¥0,¥0¥"¥0"   3       new_code                Japanese_CI_AS
4       SQLNCHAR        0       2048    "¥"¥0,¥0¥"¥0"   4       prefectures_yomi        Japanese_CI_AS
5       SQLNCHAR        0       2048    "¥"¥0,¥0¥"¥0"   5       municipality_yomi       Japanese_CI_AS
6       SQLNCHAR        0       2048    "¥"¥0,¥0¥"¥0"   6       area_yomi               Japanese_CI_AS
7       SQLNCHAR        0       2048    "¥"¥0,¥0¥"¥0"   7       prefectures             Japanese_CI_AS
8       SQLNCHAR        0       2048    "¥"¥0,¥0¥"¥0"   8       municipality            Japanese_CI_AS
9       SQLNCHAR        0       2048    "¥"¥0,¥0"       9       area                    Japanese_CI_AS
10      SQLNCHAR        0       2       ",¥0"           10      area_overlap            Japanese_CI_AS
11      SQLNCHAR        0       2       ",¥0"           11      partition               Japanese_CI_AS
12      SQLNCHAR        0       2       ",¥0"           12      partition2              Japanese_CI_AS
13      SQLNCHAR        0       2       ",¥0"           13      code_overlap            Japanese_CI_AS
14      SQLNCHAR        0       2       ",¥0"           14      update_flag             Japanese_CI_AS
15      SQLNCHAR        0       2       "¥r¥0¥n¥0"      15      update_reason           Japanese_CI_AS
```

実行スクリプト（SJIS ファイル変換）

```
@echo off
echo SJIS-EXPORT
echo START:%time%
bcp japanese_postal_code out C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥sjis¥export.csv -S DBMS-
PGSQL-WN¥SQLEXPRESS -d db -T -f C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥postal_sjis.fmt
echo END:%time%

echo SJIS-EXPORT-MIDDLE
echo START:%time%
bcp japanese_postal_code_middle out
C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥sjis¥export_mid.csv -S DBMS-PGSQL-WN¥SQLEXPRESS -d db
-T -f C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥postal_sjis.fmt
echo END:%time%

echo SJIS-EXPORT-LARGE
echo START:%time%
bcp japanese_postal_code_large out
C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥sjis¥export_lar.csv -S DBMS-PGSQL-WN¥SQLEXPRESS -d db
```

```
-T -f C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥postal_sjis.fmt
echo END:%time%

echo TERMINATED
```

実行スクリプト(UTF16 ファイル変換)

```
@echo off
db2set DB2CODEPAGE=1208
db2 terminate
db2 connect to db
echo UTF8-EXPORT
echo START:%time%
bcp japanese_postal_code out C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥unicode¥export.csv -S
DBMS-PGSQL-WN¥SQLEXPRESS -d db -T -f C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥postal_utf.fmt
echo END:%time%

echo UTF8-EXPORT-MIDDLE
echo START:%time%
bcp japanese_postal_code_middle out
C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥unicode¥export_mid.csv -S DBMS-PGSQL-WN¥SQLEXPRESS -d
db -T -f C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥postal_utf.fmt
echo END:%time%

echo UTF8-EXPORT-LARGE
echo START:%time%
bcp japanese_postal_code_large out
C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥unicode¥export_lar.csv -S DBMS-PGSQL-WN¥SQLEXPRESS -d
db -T -f C:¥Users¥Administrator¥Desktop¥result¥sqlserver¥postal_utf.fmt
echo END:%time%

echo TERMINATED
```

7.4.3. DB2 からのデータ抽出

DB2 からの抽出処理の実行スクリプトは以下の通りです。

実行スクリプト(SJIS ファイル変換)

```
@echo off
echo SJIS-EXPORT
echo START:%time%
db2 export to 'C:¥¥Users¥¥Administrator¥¥Desktop¥¥result¥¥db2¥¥sjis¥¥export.csv' of del select * from
japanese_postal_code
echo END:%time%

echo SJIS-EXPORT-MIDDLE
echo START:%time%
db2 export to 'C:¥¥Users¥¥Administrator¥¥Desktop¥¥result¥¥db2¥¥sjis¥¥export_mid.csv' of del select *
from japanese_postal_code_middle
echo END:%time%

echo SJIS-EXPORT-LARGE
echo START:%time%
db2 export to 'C:¥¥Users¥¥Administrator¥¥Desktop¥¥result¥¥db2¥¥sjis¥¥export_lar.csv' of del select *
from japanese_postal_code_large
echo END:%time%

echo TERMINATED
```

実行スクリプト(UTF8 ファイル変換)

```
@echo off
db2set DB2CODEPAGE=1208
db2 terminate
db2 connect to db
echo UTF8-EXPORT
echo START:%time%
db2 export to 'C:¥¥Users¥¥Administrator¥¥Desktop¥¥result¥¥db2¥¥unicode¥¥export.csv' of del select *
from japanese_postal_code
echo END:%time%

echo UTF8-EXPORT-MIDDLE
echo START:%time%
db2 export to 'C:¥¥Users¥¥Administrator¥¥Desktop¥¥result¥¥db2¥¥unicode¥¥export_mid.csv' of del select
* from japanese_postal_code_middle
echo END:%time%

echo UTF8-EXPORT-LARGE
echo START:%time%
db2 export to 'C:¥¥Users¥¥Administrator¥¥Desktop¥¥result¥¥db2¥¥unicode¥¥export_lar.csv' of del select
* from japanese_postal_code_large
echo END:%time%

echo TERMINATED
```

7.4.4. データ抽出での考察

本試行のデータ抽出で気づいた点を紹介します。

・SQL Server の bcp コマンドは、UTF-8 に出力できない
  - SJIS はそのまま出力できる
  - SJIS 範囲外の文字を使用している場合は、UTF16 で出力し、UTF8 への文字コード変換を行った上で、投入が必要

・ツールや手法により形式が異なる
  - CSV
  - カンマ区切り（ダブルクオートで囲まれない）
  - カンマ区切り（全列データがダブルクオートで囲まれる）

・SQL*Plus の性能
  - spool でファイル出力を行うと、他の DBMS で採用した方法より圧倒的に遅い

7.4.5. 抽出データ確認

本試行では、異種 DBMS の各エクスポート処理について、抽出されたデータサイズの確認を行いました。抽出データのサイズの結果は以下の通りです。

表 7.8: 抽出データサイズ（kb）

処理	内容	レコード件数			
		12 万	60 万	120 万	
Oracle Database エクスポート	SJIS→SJIS の CSV ファイル出力	11,932	59,656	119,311	
	SJIS→UNICODE の CSV ファイル出力	18,376	91,876	183,751	
Microsoft SQL Server エクスポート	Japanese_CI_AS→SJIS の CSV ファイル出力	11,932	59,656	119,311	
	Japanese_CI_AS→UNICODE の CSV ファイル出力	21,133	105,662	211,323	※[9]
DB2 エクスポート	SJIS,IBM-943→SJIS の CSV ファイル出力	13,621	68,104	136,208	
	SJIS,IBM-943→UNICODE の CSV ファイル出力	20,065	100,324	200,647	※[10]

9　SQL Server の UNICODE エクスポートは、UTF8 での出力は不可。UTF16 での出力のため、多少サイズが大きい。
10　DB2 のエクスポートは、指定列のみをダブルクォートで括ることができないため、他のデータよりサイズが大きい。

7.5. 文字コード変換

本試行では、文字コード変換を nkf と iconv で、SJIS から UTF8、UTF16LE から UTF8 への文字コード変換を試行しました。各処理の実行コマンドと変換時間は以下の通りです。

nkf コマンド実行（SJIS ファイル変換）

```
nkf -S -X -w 入力ファイル > 出力ファイル
```

nkf コマンド実行（UTF16 ファイル変換）

```
nkf -W16L -w 入力ファイル > 出力ファイル
```

iconv コマンド実行（SJIS ファイル変換）

```
iconv -f CP932 -t UTF-8 入力ファイル -o 出力ファイル
```

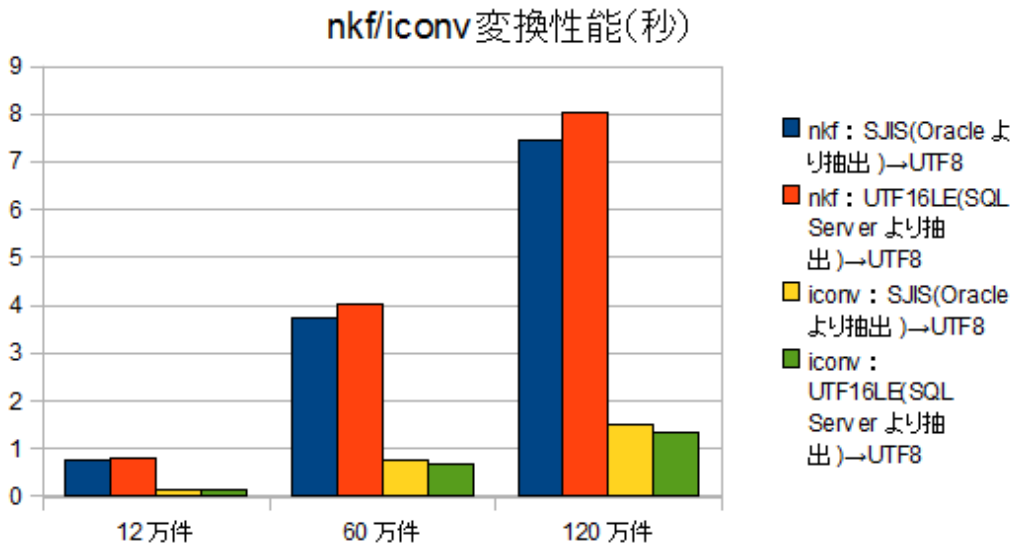
iconv コマンド実行（UTF16 ファイル変換）

```
iconv -f UTF16LE -t UTF-8 入力ファイル -o 出力ファイル
```

表 7.9: 文字コード変換と処理時間(秒)

処理（コマンド）	変換内容	レコード件数		
		12万	60万	120万
nkf	SJIS(Oracleより抽出)→UTF8	0.75	3.723	7.453
	UTF16LE(SQL Serverより抽出)→UTF8	0.806	4.008	8.022
iconv	SJIS(Oracleより抽出)→UTF8	0.152	0.756	1.502
	UTF16LE(SQL Serverより抽出)→UTF8	0.139	0.682	1.359

図 7.1: nkf/iconv 変換性能(秒)



7.6. データ投入の結果(PostgreSQL)

本試行では、データ投入をCOPY文形式(変換ありと変換なし)と、pg_bulkload(変換ありと変換なし)による投入方法の2つの投入手法について試行しました。さらに、インデックス定義無しと3インデック定義有りの場合で、試行しました。

7.6.1. インポートコマンド

試行したインポートコマンドと設定ファイルは以下の通りです。

COPYコマンド実行(WALあり)

```
TRUNCATE japanese_postal_code;
BEGIN;
¥timing on
COPY japanese_postal_code from '/var/lib/pgsql/9.3/import_data/sjis/export.csv' with csv;
¥timing off
END;
```

COPY コマンド実行（WAL 無し）

```
BEGIN;
TRUNCATE japanese_postal_code;
¥timing on
COPY japanese_postal_code from '/var/lib/pgsql/9.3/import_data/sjis/export.csv' with csv;
¥timing off
END;
```

pg_bulkload コマンド

```
pg_bulkload -i /var/lib/pgsql/9.3/import_data/sjis/export.csv -o "ENCODING=sjis"
/var/lib/pgsql/9.3/import_data/bulk.ctl
```

pg_bulkload 設定ファイル

```
 TYPE=CSV
WRITER=DIRECT
TABLE=japanese_postal_code
CHECK_CONSTRAINTS=NO
PARSE_ERRORS=-1
DUPLICATE_ERRORS=-1
LOGFILE=/tmp/bulkload.out
PARSE_BADFILE=/tmp/bulk_parse.bad
DUPLICATE_BADFILE=/tmp/bulk_dup.bad
MULTI_PROCESS=YES
```

7.6.2. インポート実行結果ファイル

試行したインポートの実行結果ファイルは以下の通りです。

COPY コマンド実行（WAL あり）

```
TRUNCATE japanese_postal_code;
BEGIN;
COPY japanese_postal_code  from '/var/lib/pgsql/9.3/import_data/sjis/export.csv' with csv;
Time: 877.352 ms
END;
```

COPY コマンド実行（WAL 無し）

```
BEGIN;
TRUNCATE japanese_postal_code;
COPY japanese_postal_code  from '/var/lib/pgsql/9.3/import_data/sjis/export.csv' with csv;
Time: 771.358 ms
END;
```

pg_bulkload コマンド

```
NOTICE: BULK LOAD START
NOTICE: BULK LOAD END
         0 Rows skipped.
         123584 Rows successfully loaded.
         0 Rows not loaded due to parse errors.
         0 Rows not loaded due to duplicate errors.
         0 Rows replaced with new rows.
```

表 7.10: 投入方法と投入時間(秒)

	処理	内容	レコード件数		
			12 万	60 万	120 万
インデックス定義無し	COPY(データ投入時に文字コード変換)	SJIS(Oracle より抽出)→UTF8 のデータロード・WAL あり	0.877	4.308	9.027
	COPY(データ投入時に文字コード変換)	SJIS(Oracle より抽出)→UTF8 のデータロード・WAL 無し	0.771	3.791	7.544
	pg_bulkload(データ投入時に文字コード変換)	SJIS(Oracle より抽出)→UTF8 のデータロード	1.773	8.529	16.971
	COPY(事前に文字コード変換済)	UTF8(Oracle より抽出)→UTF8 のデータロード・WAL あり	0.663	3.281	6.671
	COPY(事前に文字コード変換済)	UTF8(Oracle より抽出)→UTF8 のデータロード・WAL 無し	0.58	2.769	5.583
	pg_bulkload(事前に文字コード変換済)	UTF8(Oracle より抽出)→UTF8 のデータロード	0.469	2.06	4.055
インデックス定義あり（3 インデックス）	COPY(データ投入時に文字コード変換)	SJIS(Oracle より抽出)→UTF8 のデータロード・WAL あり	2.094	11.437	24.34
	COPY(データ投入時に文字コード変換)	SJIS(Oracle より抽出)→UTF8 のデータロード・WAL 無し	1.953	10.984	23.107
	pg_bulkload(データ投入時に文字コード変換)	SJIS(Oracle より抽出)→UTF8 のデータロード	2.155	10.952	21.012
	COPY(事前に文字コード変換済)	UTF8(Oracle より抽出)→UTF8 のデータロード・WAL あり	1.801	10.632	22.214
	COPY(事前に文字コード変換済)	UTF8(Oracle より抽出)→UTF8 のデータロード・WAL 無し	1.738	10.19	22.611
	pg_bulkload(事前に文字コード変換済)	UTF8(Oracle より抽出)→UTF8 のデータロード	0.883	6.714	14.391

図 7.2: インデック定義なしの環境へのインポート結果(秒)

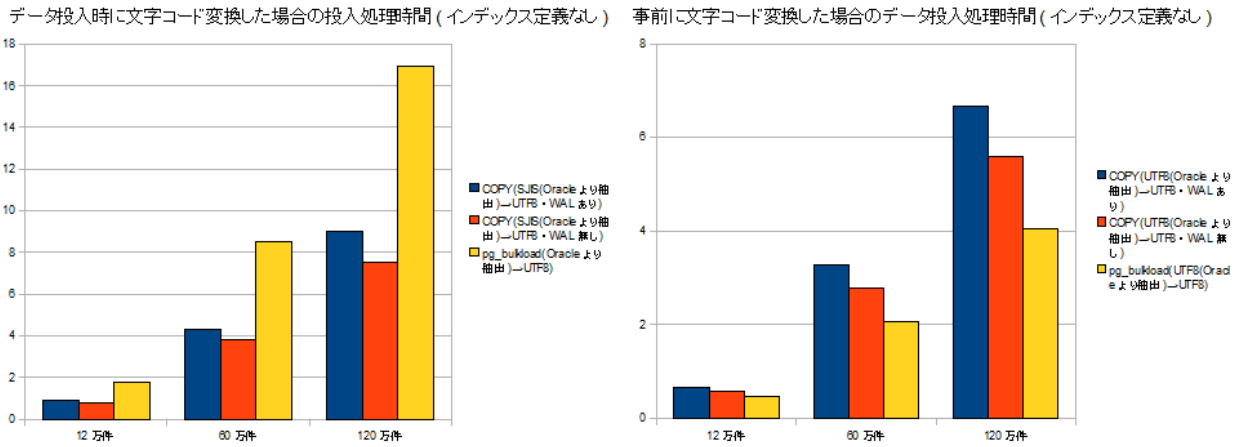
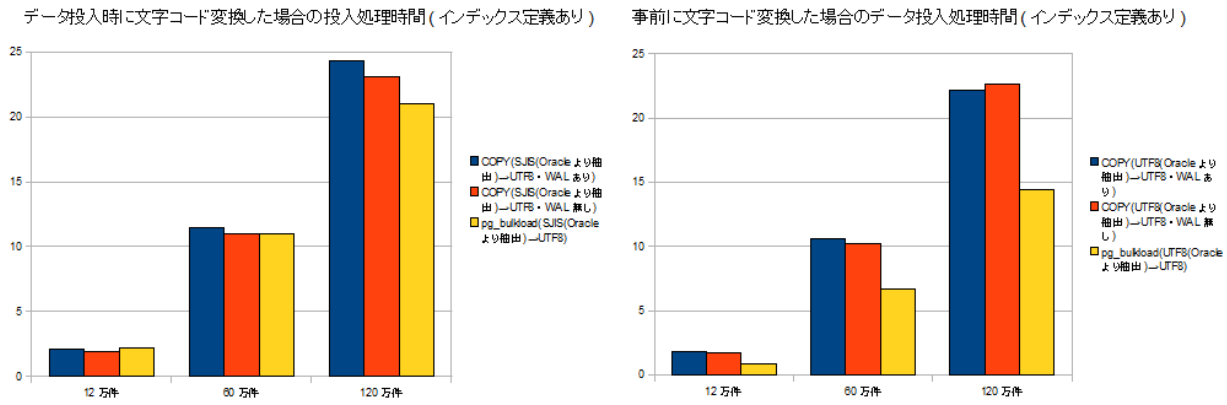


図 7.3: 3インデックスを定義した環境へのインポート結果（秒）



7.7. シナリオ検証

シナリオ検証では、一部のデータがエラー(重複キー)となり移行できなかったことを想定した作業の検証を行っています。

データ移行作業で COPY 文を使用した場合、不正データを検知した段階でトランザクションが異常終了し、ロード処理全体がロールバックされてしまいます。移行作業では大量データをロードする可能性があり、データの修正およびロード処理の再実行による時間のロスが移行作業の失敗につながる可能性があります。

本検証では pg_bulkload を使用し、不正データを除外してロードを完了した後でデータの修正およびロードを行うシナリオ検証を行っています。

シナリオ検証スクリプト

```
#! /bin/bash

export LANG=C

date
echo -n "current directory is ...";pwd
echo  "ls (cwd)... ";ls -l
echo  "ls (cwd/bad)...";ls -1 bad


psql -q -c "¥l"
psql -q -c "¥dt+"
psql -q -c "¥d japanese_postal_code"
psql -q -c "truncate japanese_postal_code;"

for code in sjis
do
echo "**********************"
echo "CLIENTENCODING : $code"
echo "**********************"

cd bad

for file in `ls -1Sr|grep export`
do
 echo
 echo "######  FILE NAME : $file ##########"
 echo -n "$file size : "
 du -b $file
 echo -n "$file lines : "
 wc -l $file
 echo -n "$file encoding : "
 nkf -g $file
 file $file

 echo "-----------------------------------------"

 echo -n "bulkload  ready..."
 sleep 2
 echo "go."
 psql -c "truncate japanese_postal_code;" >/dev/null
 echo -n "COUNT:"
 psql -qt -c "select count(*) from japanese_postal_code;"
 echo -n "WAL LOCATION:"
 psql -qt -c "select pg_current_xlog_location();"

 time pg_bulkload -i /var/lib/pgsql/9.3/import_data/bad/$file -o "ENCODING=$code"
/var/lib/pgsql/9.3/import_data/bulk.ctl

 echo -n "WAL LOCATION:"
```

```
 psql -qt -c "select pg_current_xlog_location();"
 echo -n "COUNT:"
 psql -qt -c "select count(*) from japanese_postal_code;"
 psql -q -c "¥dt+"

 echo "------------------------------------------"

 echo "##########################################"
 echo
done

cd ../
done

date
echo "TERMINATED."
```

pg_bulkload 設定ファイル

```
TYPE=CSV
WRITER=DIRECT
TABLE=japanese_postal_code
CHECK_CONSTRAINTS=NO
PARSE_ERRORS=-1
DUPLICATE_ERRORS=-1
LOGFILE=/tmp/bulkload.out
PARSE_BADFILE=/tmp/bulk_parse.bad
DUPLICATE_BADFILE=/tmp/bulk_dup.bad
MULTI_PROCESS=YES
```

スクリプト実行ログ

```
Mon Mar 24 13:57:50 JST 2014
current directory is .../var/lib/pgsql/9.3/import_data
ls (cwd)...
total 40
drwxr-xr-x 2 postgres postgres 4096 Mar 24 13:47 bad
-rw-r--r-- 1 root     root      222 Mar 20 15:34 bulk.ctl
-rwxr-xr-x 1 root     root     1356 Mar 20 15:34 bulk.sh
-rwxr-xr-x 1 root     root     1348 Mar 24 13:37 bulk_bad.sh
-rwxr-xr-x 1 postgres postgres 2173 Mar 20 14:27 copy.sh
drwxr-xr-x 3 postgres postgres 4096 Mar 20 14:58 db2
drwxr-xr-x 3 postgres postgres 4096 Mar 20 16:33 oracle
drwxr-xr-x 2 postgres postgres 4096 Mar 20 13:48 sjis
drwxr-xr-x 3 postgres postgres 4096 Mar 20 14:58 sqlserver
drwxr-xr-x 2 postgres postgres 4096 Mar 20 13:48 utf8
ls (cwd/bad)...
export.csv
                             List of databases
   Name    |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+----------+----------+---------+-------+-----------------------
 postgres  | postgres | UTF8     | C       | C     |
 template0 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
 template1 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
(3 rows)

                            List of relations
 Schema |         Name          | Type  |  Owner   | Size  | Description
--------+-----------------------+-------+----------+-------+-------------
 public | japanese_postal_code | table | postgres | 20 MB |
(1 row)
```

```
          Table "public.japanese_postal_code"
      Column       |          Type          | Modifiers
-------------------+------------------------+-----------
 regions           | character(5)           | not null
 old_code          | character(5)           |
 new_code          | character(7)           | not null
 prefectures_yomi  | character varying(1024) |
 municipality_yomi | character varying(1024) |
 area_yomi         | character varying(1024) | not null
 prefectures       | character varying(1024) |
 municipality      | character varying(1024) |
 area              | character varying(1024) | not null
 area_overlap      | character(1)           |
 partition         | character(1)           |
 partition2        | character(1)           |
 code_overlap      | character(1)           |
 update_flag       | character(1)           |
 update_reason     | character(1)           |
Indexes:
    "japanese_postal_code_pkey" PRIMARY KEY, btree (regions, new_code, area_yomi, area)
    # ★(2) 重複を許さないプライマリキーの定義

**********************
CLIENTENCODING : sjis
**********************

######  FILE NAME : export.csv ##########  # ★(3) 重複データの入ったデータファイル
export.csv size : 12217539  export.csv
export.csv lines : 123585 export.csv
export.csv encoding : Shift_JIS (CR)
export.csv: Non-ISO extended-ASCII text, with CRLF, NEL line terminators
-----------------------------------------
bulkload  ready...go.
COUNT:     0         # ★(1) PostgreSQL のテーブルを TRUNCATE

WAL LOCATION: 1/D5D35CF8

NOTICE: BULK LOAD START
NOTICE: BULK LOAD END
    0 Rows skipped.
    123584 Rows successfully loaded.
    0 Rows not loaded due to parse errors.
    1 Rows not loaded due to duplicate errors.       # ★(4)エラー発生
    0 Rows replaced with new rows.
WARNING: some rows were not loaded due to errors.

real    0m2.111s
user    0m0.003s
sys 0m0.003s
WAL LOCATION: 1/D5D37FD0

COUNT: 123584

                          List of relations
 Schema |         Name          | Type  |  Owner   | Size  | Description
--------+-----------------------+-------+----------+-------+-------------
 public | japanese_postal_code | table | postgres | 20 MB |
(1 row)

-----------------------------------------
#########################################
```

```
Mon Mar 24 13:57:55 JST 2014
TERMINATED.
```

エラーデータ復旧

```
[root@dbms-pgsql-rhel import_data]# cat /tmp/bauulk_dup.bad   # ★(5)エラーになったコードの確認
47382,90718,9071801,ｵｷﾅﾜｹﾝ,ﾔｴﾔﾏｸﾞﾝﾖﾅｸﾞﾆﾁﾖｳ,ﾖﾅｸﾞﾆ,沖縄県,八重山郡与那国町,与那国,0,0,0,0,0,0


[root@dbms-pgsql-rhel import_data]# sed -i /tmp/bulk_dup.bad -e "s/9071801/9999999/g"  # ★(6)レコードの修正
[root@dbms-pgsql-rhel import_data]# cat /tmp/bulk_dup.bad
47382,90718,9999999,ｵｷﾅﾜｹﾝ,ﾔｴﾔﾏｸﾞﾝﾖﾅｸﾞﾆﾁﾖｳ,ﾖﾅｸﾞﾆ,沖縄県,八重山郡与那国町,与那国,0,0,0,0,0,0   # ★(6)修正後レコードの確認

[root@dbms-pgsql-rhel import_data]# psql -c "copy japanese_postal_code from '/tmp/bulk_dup.bad' with csv"  # ★(6)COPY成功を確認
COPY 1
[root@dbms-pgsql-rhel import_data]# psql -c "select count(*) from japanese_postal_code"  # ★(6)COPY成功を確認
 count
--------
 123585
(1 行)

[root@dbms-pgsql-rhel import_data]# psql -c "select count(*) from japanese_postal_code where new_code='9999999'"  # ★(6)COPY成功を確認
 regions | old_code | new_code | prefectures_yomi | municipality_yomi | area_yomi | prefectures | municipality   |  area  | area_overlap | parti
tion | partition2 | code_overlap | update_flag | update_reason
---------+----------+----------+------------------+-------------------+-----------+-------------+-----------------+--------+--------------+------
-----+------------+--------------+-------------+---------------
 47382   | 90718    | 9999999  | ｵｷﾅﾜｹﾝ           | ﾔｴﾔﾏｸﾞﾝﾖﾅｸﾞﾆﾁﾖｳ   | ﾖﾅｸﾞﾆ     | 沖縄県      | 八重山郡与那国町 | 与那国 | 0            | 0
     | 0          | 0            | 0           | 0
(1 行)

[root@dbms-pgsql-rhel import_data]#
```

8. まとめ

本資料では、異なる文字コードのデータベースから PostgreSQL への移行に関して、注意点や必要な移行手順を洗い出し、実際にデータ移行検証を実施してその妥当性および、移行に要する時間を確認しました。

検証の結果から、文字コード変換を含むデータ移行方法として P12～P25 に記載しているデータ抽出、コード変換、データ投入手順に問題ないことが確認できました。
データ抽出時の注意点としては、抽出するツールによりデータ出力の形式が異なることや、Oracle の SQL*Plus においては spool でデータ抽出を行うと他 DBMS のエクスポートに比べて時間がかかることが確認できました。

文字コード変換においては、nkf と iconv コマンドの処理時間を比較すると、iconv コマンドの方が早いことが確認でき、データ投入検証においては、データ投入時に文字コード変換しながらデータ投入したケースより、iconv コマンドなどで事前に文字コード変換した後、データ投入をする方が処理速度が早いことが確認できました。
また、通常のデータ移行でも同様の事が言えますが、インデックス定義がある場合は無い場合と比較して、データ投入処理速度が遅いため、インデックス定義が無い状態でデータ投入することを推奨します。

以上のことから、文字コードの違う異種 DBMS からデータを移行する場合、抽出したデータを iconv コマンドで文字コード変換した後、インデックス定義が無い状態で、データ投入を実施する手順が、一番処理効率が良いと言えます。
また、COPY 文を使用した場合、エラーが発生するとトランザクションが異常終了し、ロード処理全体がロールバックされてしまうため。大量データを投入する場合には pg_bulkload を使用することを推奨します。

著者

版	所属企業･団体名	部署名	氏名
データ移行･文字コード変換編 第 1 版 （2013 年度 WG2）	株式会社アシスト	データベース技術本部	佐々木 美江
			中嶋 辰也
	NECソリューションイノベータ株式会社		岩浅 晃郎