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

データ移行調査および実践編

製作者
担当企業名:
株式会社アシスト
株式会社富士通ソーシアルサイエンスラボラトリ

改訂履歴

版	改訂日	変更内容
1.0	2013/04/22	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（PostgreSQL 以外の DBMS）からのデータ移行に関しての、手順や留意事項を記載しています。

■資料内の記述について

本資料では、ETL に従い、次のようなフローによるデータ移行を記載していきます。「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。なお、本資料では有償製品の ETL ツール利用による自動移行に関しての記載はせず、各工程における検討事項や基本的なツールによるコマンド例示の記載を行います。

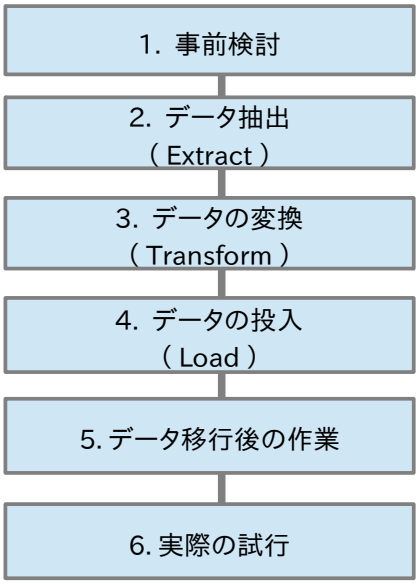

図 1: 本資料の流れ

ETL については下図のような概念となります。

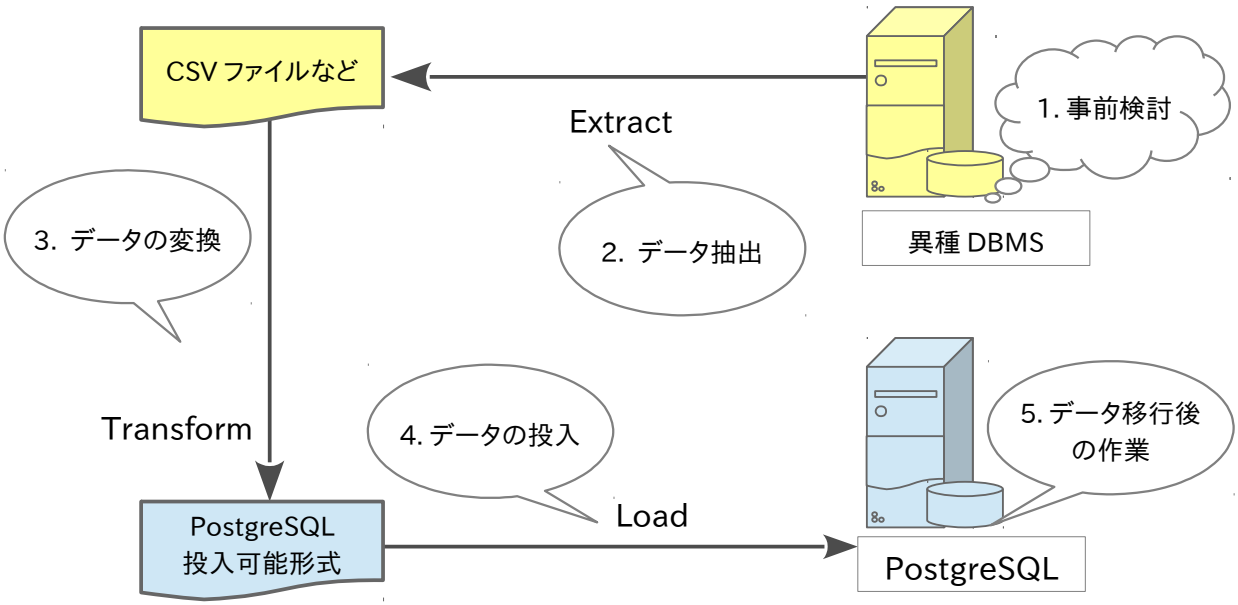

図 2: ETL の概念と各章のマッピング

■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 ではない、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 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。

■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	9.2.2
2	検証対象オペレーティングシステム	CentOS	intel CPU 6.2 FINAL 64bit
3	異種 DBMS 動作オペレーティングシステム （Microsoft SQL Server）	Windows	intel CPU 7 SP1 32bit

表 3: 異種 DBMS 一覧

No.	DBMS	バージョン
1	Oracle Database	11g R2
2	Microsoft SQL Server	2008 R2 Express

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)	9.2.2	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	Ora2Pg (おら・つー・ぴーじー)	10.1	GNU General Public License, version3	http://ora2pg.darold.net/	Oracle Database から、PostgreSQL へ DB のデータスキーマやデータ情報を移行するツール。Oracle Database から PostgreSQL へ互換性のあるスキーマのみであれば直接移行ができる。 本資料では Ora2Pg で対応できないようなデータ型に関して手動での変換も考慮するため、中間ファイル抽出ツールとして利用する。
5	Oracle Instant Client Basic	11.20.3.0 x86_64 (RPM 利用)	Oracle Technology Network Development and Distibution License	http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html	OCI 等クライアント基礎パッケージ。DBD::Oracle 依存プログラム。
6	Oracle Instant Client SQL*Plus				SQL*Plus の実行をするためのパッケージ。DBD::Oracle 依存プログラム。
7	Oracle Instant Client SDK (devel)				ドライバをビルドするためのパッケージ。DBD::Oracle 依存プログラム。
8	Oracle Instant Client JDBC				Java プログラムが Oracle Database に接続するためのドライバ。JdbcRunner が Oracle Database に接続するために必要。
9	perl (ぱーる)	5.10 (CentOS RPM を利用)	Artistic License 2.0 または、GNU General Public License, version 1 以上	http://www.perl.org/	DBD::Oracle 実装言語。
10	ExtUtils-MakeMaker	6.55 (CentOS RPM を利用)		http://search.cpan.org/dist/ExtUtils-MakeMaker/	perl モジュールのビルドに利用。DBD::Oracle 依存プログラム。
11	perl DBI (でーびーあい)	1.609 (CentOS RPM を利用)		http://search.cpan.org/~timb/DBI/	perl 共通データベース操作インタフェース。DBD::Oracle 依存プログラム。
12	Compress::Zlib (こんぷれす・じーりぶ)	2.020 (CentOS RPM を利用)		http://search.cpan.org/~pmqs/IO-Compress-	zlib 圧縮のための perl モジュール。Ora2Pg 依存プログラム。

				2.060/lib/Compress/Zlib.pm	
13	DBD::Oracle （でーびーでー・おらくる）	1.56		http://search.cpan.org/dist/DBD-Oracle/	OracleDatabase へ Perl プログラムが接続するためのドライバモジュール。Ora2Pg 依存プログラム。
14	DBD::Pg （でーびーでー・ぴーじー）	2.19.3		http://search.cpan.org/dist/DBD-Pg/	PostgreSQL へ Perl プログラムが接続するためのドライバモジュール。Ora2Pg 依存プログラム。
15	nkf （えぬけーえふ）	2.0.8 (CentOS RPM を利用)	The zlib/libpng License	http://sourceforge.jp/projects/nkf/	ネットワーク漢字コードフィルタ。本資料では巨大なファイルの文字コード変換等に用いる。
16	iconv （あいこんぶ）	2.12 (CentOS glibc 同梱版を利用)	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、巨大なファイルの文字コード変換等を行う際に利用する。
17	dos2unix （どす・つー・ゆにっくす）	3.1 (CentOS RPM を利用)	The BSD 2-Clause License	http://sourceforge.jp/projects/sfnet_dos2unix/	改行コードを変換するためのユーティリィプログラム。 本資料では、iconv コマンドと併用する。
18	unix2dos （ゆにっくす・つー・どす）	2.2 (CentOS RPM を利用)			
19	JdbcRunner （じぇいでーびーしー・らんなー）	1.2	The BSD 3-Clause License	http://hp.vector.co.jp/authors/VA052413/jdbcrunner/	複数の RDBMS のベンチマークが可能なプログラム。本資料では、移行対象データベースのサンプルとして本ツールが作成するデータベースを用いる。

目次

1. 事前検討

データ移行の実施に際して、DBMS の仕様や動作環境など、事前に検討すべき項目を記載します。

1.1. DBMS の動作環境と移行手順の確認

本資料のデータ移行手順を踏むために、移行元の異種 DBMS や移行先の PostgreSQL から次のような情報を確認する必要があります。

表 1.1.1: 動作環境と移行手順の確認

No.	確認項目	確認観点	関連章
1	DBMS 定量制限	各 DBMS 実装の DBMS 定量制限を事前に確認する必要があります。 本資料では、PostgreSQL、Oracle Database および Microsoft SQL Server について記述します。	1 章
2	データ型の互換性	移行元と移行先のデータ型が、双方に存在するか、格納範囲は同様か等確認する必要があります。 本資料では、PostgreSQL、Oracle Database および Microsoft SQL Server について記述します。	1 章
3	移行対象オブジェクトの確認	移行対象オブジェクトについて確認する手段を例示します。 本資料では、Oracle Database について記載します。	1 章
4	移行対象テーブルの制約と索引について	制約や索引を定義したまま投入すると、速度が劣化する可能性があります。 本資料では、事前に制約を確認し、後に定義する方法を検討します。	1 章
5	DBMS のデータ抽出方法	移行元の異種 DBMS 実装のデータ抽出方法を事前に確認する必要があります。 本資料では、Oracle Database および Microsoft SQL Server について記述します。	2 章
6	エンコーディング	変換や投入を行うクライアントおよびサーバの文字エンコーディングにより、投入時の自動変換挙動が異なる場合があります。PostgreSQL の文字エンコーディングについて記載します。	3 章
7	テキストファイルの文字コード変換	中間ファイルをテキストデータとして出力する際に文字コードを確認します。投入先の PostgreSQL と異なる場合は適切に文字コードを変換する必要があります。	3 章
8	外字の利用の有無	外字として設定している文字が、移行元の異種 DBMS に存在する場合は、移行先の PostgreSQL で正しく利用できる方法を検討する必要があります。	3 章
9	PostgreSQL のデータ投入方法	PostgreSQL が利用できる投入ファイルの形式や、投入方法について事前に確認し決定する必要があります。 ※本資料では触れませんが、抽出したデータのサイズが大きい場合は、データの一次格納方法や、搬送方法なども重要になります。	3 章 4 章
10	データ移行後のデータ確認方法	データ移行完了後に、移行元 DB より適切に移行ができたか、確認する必要があります。	5 章
11	データ移行後に実施すべき処理	PostgreSQL 利用時に、データ移行後に実施すべき保守コマンド等を記載します。	5 章
12	移行手順の確立	実際にコマンド等実行し、手順の確認をします。 本資料の 1～5 章の手順を任意の DB へ適用し、結果を記載します。	6 章

1.2. DBMS 定量制限

PostgreSQL および Oracle Database、Microsoft SQL Server の定量制限には次のような違いがあります。このため、各 DBMS の最大値に近いサイズにて運用をしているような DB を持つ場合は、DB の分割配備が必要になる等、システム仕様に影響を与える可能性があります。

表 1.2.1: DBMS 定量制限

No.	諸元	PostgreSQL	Oracle Database	Microsoft SQL Server
1	DB サイズ	記憶域に依る	不明	524,272TB
2	1 インスタンス あたりの DB 数	-	OS 依存	32,767
3	DB あたりの ファイル数	OS 依存	65,533	32,767
4	データファイル 最大サイズ	1GB（標準）	4,194,303 * DB_BLOCK_SIZE (DB_BLOCK_SIZE デフォルト値 :8192 バイト)	16TB
5	データベースごとの テーブル数	OS 依存	1,000（OS 依存）	記憶域に依る
6	テーブルごとの 行数	記憶域に依る（32TB）	記憶域に依る	記憶域に依る
7	行ごとの 制限サイズ	記憶域に依る	不明	記憶域に依る
8	SQL 命令文字列 最大長	文字列長：1GB スタック：2MB（標準）	制限なし	65,536 * パケットサイズ （標準パケットサイズは 4KB）
9	データベース名 最大長	63 バイト（超過分切り捨て）	8 バイト	128 文字 （ログファイル名なしの場合は 123 文字）
10	テーブル名 最大長	63 バイト（超過分切り捨て）	30 バイト	128 文字
11	カラム名 最大長	63 バイト（超過分切り捨て）	30 バイト	128 文字
12	その他 オブジェクト名 最大長	SQL の引数: 63 バイト （/src/include/ pg_config_manual.h に定義）	dblink 名:128 バイト	ユーザ名:128 文字

1.3. DBMS 固有の実装

PostgreSQL および Oracle Database、Microsoft SQL Server にはそれぞれ固有の実装があります。本資料ではレコードの移行手順を主題としています。SQL や実装された関数等の各実装の違いについては WG2 の関連文書をご参照ください。

表 1.3.1: 各実装の調査と WG2 関連文書

No.	調査観点	文書名
1	アプリケーション接続方法の違い	アプリケーション移行調査編
2	アプリケーション改修アプローチと試行	アプリケーション移行実践作業
3	SQL 実装の違い	SQL 移行調査編
4	ストアドプロシージャの違い	ストアドプロシージャ移行調査編
5	組み込み関数の違い	組み込み関数移行調査編
6	スキーマの違い	スキーマ移行調査編
7	取りうる構成の違い	システム構成調査編

1.4. 利用中のデータ型の互換性

PostgreSQL には多くのデータ型が定義されており、おおよそ移行元の DBMS で利用していたデータ型の代替となるデータ型が存在しています。しかし、一部代替対象の無いデータ型や、同名でも移行元と同様の利用ができないデータ型も存在します。このため、移行先の PostgreSQL へテーブルを新規に定義する際に、単純な型名の変更以外に検討すべき事項が存在する場合があります。

本節では、Oracle Database および Microsoft SQL Server の実装が異なるデータ型を PostgreSQL 側で定義する際の、一部の制限や留意点について記述します。

1.4.1. 空文字の扱い

PostgreSQL と異種 DBMS では、「''(空文字) 」の扱いが異なる場合があります。下記表に、「''(空文字) 」の扱いの違いをまとめます。

表 1.4.1: 空文字の扱い

No.	PostgreSQL	Oracle Database	Microsoft SQL Server
1	''(空文字)は長さ 0 の文字列として処理	''(空文字)は NULL として処理	''(空文字)は長さ 0 の文字列として処理

PostgreSQL は NULL と「''(空文字)」は別物として扱います。しかし、Oracle Database では同じものとして扱いますので、データを抽出する際には、注意が必要です。

1.4.2. Oracle Database

Oracle Database については、WG2 関連文書「スキーマ差分調査」をご確認ください。

1.4.3. Microsoft SQL Server

Microsoft SQL Server のデータ型の多くは、PostgreSQL にも同等のデータ型が存在していますが、次のデータ型については、注意が必要です（PostgreSQL と Microsoft SQL Server の各データ型の対比は別紙「組み込みデータ型対応表（SQLServer-PostgreSQL）」を参照してください）。

a. 数値型

数値型は PostgreSQL 側にほぼ同等の数値型が存在します（1 バイトの数値表現のみありません）。ただし、Microsoft SQL Server では、自動採番は「IDENTITY 属性」を数値型で付与することで定義しますが、PostgreSQL では自動採番のためのデータ型を定義する必要があります。以下の表のように数値のバイトレンジを合わせて、適切な自動採番のデータ型に変更してください。

表 1.4.3-a: 自動採番のデータ型の定義変更（Microsoft SQL Server から PostgreSQL）

No.	IDENTITY 属性が付与された Microsoft SQL Server の数値型	PostgreSQL の自動採番型
1	tinyint	small/serial2
2	smallint	
3	int (integer)	searial
4	bigint	bigserial/serial8

b. 文字列型

文字列型は PostgreSQL 側にほぼ同じ型が存在します。ただし、Microsoft SQL Server では、UNICODE 文字列（UCS2）を格納するデータ型である、「nchar(n)/nvarchar(n)/ntext」が定義されていますが PostgreSQL では特に文字エンコーディングを明示的に指定する文字列型がないため注意が必要です。このため、これらの文字列型を定義している場合は、移行先の PostgreSQL データベースを作成する際に、明示的に文字エンコーディングを UNICODE(UTF-16)と指定し作成する必要があります。

また、UNICODE とは異なる文字エンコーディングで「char(n)/varcahar(n)/text」型を定義、混在利用している場合は、PostgreSQL にはバイナリデータ「bytea(n)」として文字列を格納する、DB クラスタを分割して別の DB として管理する（文字エンコーディングが DB 毎に付与されるため）、など工夫をする必要があります。

c. バイナリデータ型

バイナリデータは PostgreSQL 側にほぼ同等の型が存在します。ただし「image」型に関しては、PostgreSQL に相当の型が無い（2 ギガバイトまで許容するバイナリデータ型が無い）ため、bytea 型を超える大容量のバイナリデータの場合は、「ラージオブジェクト」で定義して格納を行います。「ラージオブジェクト」を定義して利用する場合、アプリケーションは PostgreSQLAPI を用いて制御するように修正する必要があります（libpq や JDBC ドライバで API は提供されています）。

d. 日付・時刻型

日付や時刻を取り扱う型は PostgreSQL 側にほぼ同等の型が存在しますが、PostgreSQL では残念ながら 100 ナノ秒が扱えません。このため、100 ナノ秒を必要とするアプリケーションは別のデータ型（文字列型やバイナリデータ型など）で定義し、データベースから値をロードする際に、時刻の形式に変換してやるなどアプリケーションの実装を改造する必要が出てきてしまいます。

表 1.4.3-d: 精度が落ちる時刻型

No.	Microsoft SQL Server 日付・時刻型	PostgreSQL 日付・時刻型（100 ナノ秒は落ちる）
1	datetime2/datetime2(7)	timestamp(p) without time zone
2	datetimeoffset/datetimeoffset(7)	timestamp(p) with time zone
3	time/time(7)	time(p) without time zone

e. その他

「xml」型は同名の型が PostgreSQL に存在しますが、XML 操作 API として Microsoft SQL Server が提供している「Transact-SQL」は PostgreSQL には存在しないため、PostgreSQL が提供している XML 操作関数を用いるなど、アプリケーションの実装を改造する必要があります。

「uniqueidentifier」型は、同義に定義されている「UUID」型が存在するため問題ありません。

f. Microsoft SQL Server 固有のデータ型

Microsoft SQL Server の内部アドレス等を保有するデータ型や、Microsoft SQL Server 上でのユーザ定義のデータ型等はそのまま移行することはできません。別途アプリケーション実装の見直しや、PostgreSQL の作成規約に従った、ユーザ定義型を作成する必要があります。

表 1.4.3-f: Microsft SQL Server 固有のデータ型

No.	固有のデータ型	型名
1	ストアドプロシージャ用の結果等保持用	cursor
2	階層内の位置 テーブル階層構造走査時の保持用	hierarchyid
3	ユーザ定義（共通言語ランタイム）	CLR
4	ユーザー定義関数の列、 パラメータ、変数、および戻り値	sql_variant
5	行のセットの一時的な格納場所	table

1.5. 操作ユーザの確認

データベースを操作するユーザ定義は、複数のシステムによる操作やオペレータ操作のために、利用スコープに合わせて複数定義されている場合があります。データベースおよび連携するシステムの仕様書を確認し、移行先システムにおいても適切にユーザを定義してください。

また、データベース移行を契機に不要なユーザ定義が無いか、見直しを行うことも有効です。

1.5.1. Oracle Database の操作ユーザの確認

移行対象ユーザの権限情報は、システムの仕様書に明記されていることが望ましいですが、仕様書の入手が難しい場合や不明な場合には、稼働中の移行元 DBMS から操作ユーザの諸情報を取得することを検討します。以下に Oracle Database における操作ユーザ「SCOTT」に関しての各情報出力の例を示します。

PostgreSQL では、ユーザに相当するものはロールと呼ばれますが、本資料では、Oracle Database に合わせて PostgreSQL においてもユーザと記載します。なお、PostgreSQL のユーザ権限は、CREATEDB（データベース作成）、CREATEROLE（ロールの作成）、LOGIN（データベースへ接続）、REPLICATION（レプリケーションで使用）、および最高位の SUPERUSER のみですので、Oracle Database におけるユーザのシステム権限とのマッピングが難しいですが、仕様書や設計書等を参照し定義を行ってください。

実行例: システム権限の出力

```
SQL> select PRIVILEGE from DBA_SYS_PRIVS where GRANTEE='SCOTT';

PRIVILEGE
----------------------------------------
CREATE VIEW
UNLIMITED TABLESPACE
```

実行例: ロールの出力

```
SQL> select GRANTED_ROLE from DBA_ROLE_PRIVS where GRANTEE='SCOTT';

GRANTED_ROLE
--------------------
RESOURCE
CONNECT
```

実行例: ロールに割り当てられているシステム権限の出力

```
SQL> select privilege from DBA_SYS_PRIVS where GRANTEE='RESOURCE';

PRIVILEGE
--------------------
CREATE TRIGGER
CREATE SEQUENCE
CREATE TYPE
CREATE PROCEDURE
CREATE CLUSTER
CREATE OPERATOR
CREATE INDEXTYPE
CREATE TABLE
```

なお、Oracle Database ではユーザを作成することで、ユーザ名と同じスキーマ名が自動的に割り当てられますが、PostgreSQL ではユーザを作成してもスキーマは作成されません。必要に応じてスキーマを作成してください。スキーマを作成しないままオブジェクトを作成することも可能ですが、その場合デフォルトで存在する「public」スキーマのオブジェクトとして作成されます。

本資料では「SCOTT」スキーマを作成し、Oracle Database のオブジェクトは「SCOTT」スキーマのオブジェクトとして移行します。

実行例: PostgreSQL におけるユーザとスキーマの作成

```
$ psql -U postgres
postgres=# create role scott with login password 'tiger';
$ psql -U postgres -d tpcc
postgres=# create schema AUTHORIZATION scott;
```

1.6. 移行対象のオブジェクトの確認

移行対象のオブジェクト情報を確認します。移行対象のオブジェクト情報を移行先のオブジェクト情報と比較し、移行が正しく行われたかを判断します。

1.6.1. Oracle Database のオブジェクト情報の確認

Oracle Database では、以下の実行例のようにユーザ毎にオブジェクトを確認することができます。

実行例: 移行対象のテーブル行数を確認

```
SQL> select table_name,num_rows from user_tables;

TABLE_NAME                     NUM_ROWS
------------------------------ ----------
HISTORY                        120000
ITEM                           100000
STOCK                          400000
ORDERS                         120000

(省略)
```

1.7. 移行対象のテーブル定義時の制約および索引の定義

PostgreSQL へデータ投入する際に、索引属性が定義されていると、データ投入時の挿入速度が劣化してしまいます。また、制約条件（外部キー等）が定義されていると、単純なデータ投入でも成功しません。本資料では、独自の移行プログラムを実装せずにデータ移行を検討していくため、制約および索引についてはデータ投入完了後に定義することにし、記述しています。

1.7.1. Oracle Database のテーブル制約確認

Oracle Database では、以下の例のようにテーブルの制約を確認することができます。

実行例: 移行対象のテーブルの制約を確認

```
SQL> select table_name,constraint_name from user_constraints;

TABLE_NAME                     CONSTRAINT_NAME
------------------------------ ------------------------------
NEW_ORDERS                      NEW_ORDERS_FK1
ORDER_LINE                     ORDER_LINE_FK1
ORDER_LINE                     ORDER_LINE_FK2
STOCK                         STOCK_FK2
HISTORY                        HISTORY_FK2
ORDERS                         ORDERS_FK1
CUSTOMER                          CUSTOMER_FK1
HISTORY                        HISTORY_FK1
DISTRICT                       DISTRICT_FK1
STOCK                         STOCK_FK1
EMP                           FK_DEPTNO
(省略)
```

1.7.2. Oracle Database のテーブル索引確認

Oracle Database では、以下の例のように索引を確認することができます。

実行例: 移行対象のテーブルの索引を確認

```
SQL> select TABLE_NAME,INDEX_NAME from user_indexes;

TABLE_NAME                     INDEX_NAME
------------------------------ ------------------------------
WAREHOUSE                       WAREHOUSE_PK
STOCK                         STOCK_PK
ORDER_LINE                        ORDER_LINE_PK
ORDERS                         ORDERS_PK
ORDERS                         ORDERS_IX1
NEW_ORDERS                       NEW_ORDERS_PK
ITEM                          ITEM_PK
EMP                           PK_EMP
DISTRICT                          DISTRICT_PK
DEPT                           PK_DEPT
DATA_ELT_TEST2                    SYS_IL0000074908C00003$$

TABLE_NAME                     INDEX_NAME
------------------------------ ------------------------------
DATA_ELT_TEST1                    SYS_IL0000074912C00007$$
DATA_ELT_TEST1                    SYS_IL0000074912C00006$$
CUSTOMER                       CUSTOMER_PK
CUSTOMER                       CUSTOMER_IX1
(省略)
```

2. データの抽出(Extract)

本章では、データ抽出（Extract）に関しての注意事項や、具体的な異種 DBMS からのデータ抽出方法を示します。また、抽出時に異種 DBMS の型情報の差分を吸収する必要があるため、DBMS の関数等を用いて出力表現の変換（Transform）が必要な場合があります。

2.1. データ抽出時のフォーマット

PostgreSQL と異種 DBMS では、そのデータを収めた型の情報が異なる場合があり、異種 DBMS から PostgreSQL へ直接のデータ移行が難しい場合があります。この場合はデータの変換、広く知られるフォーマットとして CSV(カンマ・セパレーテッド・バリュー）等の中間フォーマットが必要な場合があります。

注意点として、出力する際のセパレータ識別子（カンマ）や文字列の開始・終了文字（多くはダブルクォーテーション「"」やシングルクォーテーション「'」）をデータと混在しないように、取り扱う必要があります。

また、次章にて詳細に述べますが、抽出元の異種 DBMS と投入先の PostgreSQL の抽出・投入コマンドの仕様および取扱い文字コードを意識して、CSV ファイルの文字コードに注意を払う必要があります。

2.2. データ型と出力時の表現

異種 DBMS との型の不整合を吸収するため、いくつか表現を修正して出力すべき項目があります。具体的に示すと下表にあるような型情報となります。

表 2.2.1: データ型別の出力時の注意点

No.	型名	注意点
1	数値型	基本的にそのまま出力して問題ありません。
2	文字列型	「"」（ダブルクォーテーション）で囲う必要がある場合があります。 文字列中に CSV 区切り文字「,」（カンマ）が含まれている際に必要になります。
3	日付・時刻型	年月日時分秒の出力形式は DBMS によって文字列表現が異なります。例えば、Oracle Database の場合は、TO_CHAR 関数等を用いて、整形する必要があります。
4	バイナリデータ型	CSV 出力は、テキストコードで出力するため、文字として出力できない制御コードなどは出力できません。

2.2.1. PostgreSQL の timestamp 型への有効な入力

日付時刻情報を保持するためによく使用される timestamp 型について、入力可能な日付時刻を下記表に記載します。日付と時刻の後に、オプションでタイムゾーン、AD（西暦）もしくは BC（紀元前）を入力することができます。AD/BC を時間帯の前に付ける方法もありますが、これは推奨される順序ではありません。

表 2.2.2: timestamp 型に入力可能な書式

No.	有効な入力書式	入力例
1	YYYY-MM-DD HH:MM:SS	1999-01-08 04:05:06
2	YYYY-MM-DD HH:MM:SS タイムゾーン（時刻）	1999-01-08 04:05:06 -8:00
3	MONTH DAY HH:MM:SS YYYY タイムゾーン（略称）	January 8 04:05:06 1999 PST

2.2.2. Oracle Databaseで日付・時刻型の出力形式を変換する関数

a. TO_CHAR関数（日付型）

```
TO_CHAR（日付[,書式][,NLS パラメータ]）
```

日付を書式で指定した文字型に変換して戻します。書式は日付書式を参照します（省略可）。NLS パラメータで月と日の名前など、戻される言語を指定します。省略すると、現在のセッションのデフォルトパラメータを使用します。

```
SQL> SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD DAY HH24:MI:SS') FROM DUAL;

TO_CHAR(SYSDATE,'YYYY-MM-DDDAYHH24:MI:SS')
-----------------------------------------------------------
2013-03-18 MONDAY    12:30:48
```

b. TO_DATE関数

```
TO_DATE（文字列[,書式][,NLS パラメータ]）
```

書式で指定した文字列を日付型に変換して戻します。書式は日付書式を参照します（省略可）。NLS パラメータで月と日の名前など、戻される言語を指定します。省略すると、現在のセッションのデフォルトパラメータを使用します。

```
SQL> SELECT TO_DATE('20041022115640', 'YYYYMMDDHH24MISS') FROM DUAL;

TO_DATE('
---------
22-OCT-04
```

(*1) 上記の出力例の「*TO_DATE(')*」は実際にこのように出力されます。

日付書式は、日付を文字列に変換するTO_CHAR関数や、文字列を日付に変換するTO_DATE関数で利用します。以下の日付書式を指定することができます。

表 2.2.3:日付書式

No.	要素	意味
1	YYYY	4桁で表した西暦
2	YYY,YY,Y	それぞれ年の下3桁、2桁、1桁　(西暦)
3	MM	月（01～12）
4	MONTH	月の名前
5	MON	月の省略英語表記（JAN～DEC）
6	DD	日 (1～31)
7	DAY	曜日 (日曜日～土曜日)
8	DY	省略した曜日 (日、月、火、水、木、金、土)
9	D	週における曜日 (日=1、月=2、火=3、水=4、木=5、金=6、土=7)
10	HH,HH12	時間 (12時間制表記)
11	HH24	時間 (24時間制表記)
12	MI	分 (0～59)
13	SS	秒 (0～59)

表 2.2.4:日付書式要素の接尾語

No.	要素	意味	要素の説明	例の結果
1	TH	序数	DDTH	12TH
2	SP	フルスペルで表した数	DDSP	TWELVE
3	SPTH または THSP	フルスペルで表した序数	DDSPTH	TWELVETH

2.2.3. Oracle の NULL を含む列を抽出する関数

Oracle Database か NULL を含むデータを抽出する際に、以下の関数をよく使用します。

a. NVL 関数

```
NVL(式,値)
```

式が NULL の場合、値を戻します。式が NULL 以外であれば、式を戻します。

b. NVL2 関数

```
NVL2(式,値 1,値 2)
```

式が NULL 以外であれば、値 1 を戻します。式が NULL の場合、値 2 を戻します。

2.3. 各 DBMS の CSV ファイル出力方法

Oracle Database、Microsoft SQL Server および PostgreSQL の各実装から、CSV ファイルとしてデータを抽出する方法を記載します。

2.3.1. Oracle Database

1. SPOOL コマンドを使用し、データを抽出する方法

SPOOL コマンドは、SQL*Plus で実行した問い合わせの結果をファイルに格納することができます。
また、SPOOL コマンドを実行する前に、SET コマンドで以下の項目を設定することをお勧めします。

表 *2.3.1: SET*コマンド

No.	設定項目	概要
1	heading	レポートへの列ヘッダの出力を制御します。 ON を指定すると、列ヘッダがレポートに出力されます。OFF を指定すると、列ヘッダが出力されなくなります。
2	feedback	問合せによって *n* 個以上のレコードを選択した場合に、問合せから戻されるレコード数を表示します。ON または OFF によって、この表示をオンまたはオフにできます。フィードバックを ON に設定すると、*n* が 1 に設定されます。フィードバックを 0 に設定することは、OFF に設定するのと同じです。
3	echo	コマンドが実行されるときに、START コマンドによりスクリプト内の各コマンドを表示するかどうかを制御します。ON を指定すると、画面にコマンドが表示されます。OFF を指定すると、非表示になります。
4	termout	スクリプトから実行するコマンド出力の表示を制御します。OFF を指定すると非表示になるため、出力を画面に表示せずにファイルへスプールできます。ON を指定すると、画面に出力が表示されます。 TERMOUT OFF は、対話方式で入力するコマンド、またはオペレーティング・システムから SQL*Plus にリダイレクトするコマンドによる出力には影響しません。
5	linesize	SQL*Plus が新しい行を開始する前に、1 行に表示する文字の合計数を設定します。
6	pagesize	各ページの行数を設定します。PAGESIZE を 0 に設定すると、ヘッダ、ページ・ブレーク、タイトル、初期空白行およびその他の書式設定情報をすべて非表示にできます。
7	trimspool	SQL*Plus で、それぞれのスプール行の終わりに後続の空白を入れるかどうかを指定します。ON を指定すると、各行の終わりの空白が削除されます。OFF を指定すると、SQL*Plus で後続空白を表示できます。TRIMSPOOL ON は、端末出力には影響を与えません。
8	colsep	選択された列の間に出力するテキストを設定します。

実行例: SPOOL コマンドを用いて、item テーブルのデータを抽出

```
SQL> set heading off
SQL> set feedback off
SQL> set echo off
SQL> set termout off
SQL> set linesize 1000
SQL> set pagesize 0
SQL> set trimspool on
SQL> set colsep ','
SQL> spool /tmp/item.csv
SQL> select  i_id,
SQL> i_im_id,
SQL> '"' || i_name ||'",'||
SQL> i_price,
SQL> '"' || i_data || '"'
SQL> from item;
(省略)
SQL> spool off
```

2. Ora2Pg を使用し、データを抽出する方法

Ora2Pg は Perl で記述されたデータ移行ツールです。Ora2Pg は、Oracle Database からのデータのエクスポートと、PostgreSQL に合わせたデータ型へ変更が可能です。具体的には、Oracle Database からテーブルの定義とデータを読み取り、PostgreSQL のデータ型に変更した後、テキスト形式で出力します。Ora2Pg によるデータ型の変換内容は、下表を参照してください。テキスト形式は、PostgreSQL の pg_dump と同じように、CREATE TABLE 文や INSERT 文を使ってデータを投入する形となります。

また、同時に両方のデータベースに接続して、データ型を変換しつつ投入を行うモードもありますが、意図した通りに動作しない場合もありますので、一度テキストファイルに出力する方法をお勧めします。

表 2.3.2: Ora2Pg の型変換

No.	Oracle Database データ型	Ora2Pg による変換（PostgreSQL）
文字型		
1	char(n)	char(n)
2	nchar(n)	char(n)
3	varchar(n)	varchar(n)
4	nvarchar(n)	varchar(n)
5	varchar2(n)	varchar(n)
6	nvarchar2(n)	varchar(n)
7	long	text
8	clob	text
数値型		
9	number	numeric
10	float	float8
日付型		
11	date	timestamp
バイナリ型		
12	long raw	bytea
13	blob	bytea

（1）インストール

Ora2Pg をインストールする前に以下の環境を整備する必要があります。

a. Oracle Instant Client をインストール（Perl5 からの接続に利用）

b. Perl 5 の実行環境

c. Ora2Pg が依存する Perl モジュールのインストール

環境の整備完了後にインストールを実施します。Ora2Pg プロジェクトのサイト[1]から Perl モジュールをダウンロードします。

実行例:

```
$ bzip2 -dc ora2pg-10.1.tar.bz2 | tar xvf -
$ cd ora2pg-10.1
$ perl Makefile.PL
$ make
$ su
パスワード:
# make install
```

Ora2Pg の詳細なインストールについての詳細は、別紙「付帯ツールのインストール手順」を参考にしてください。

（2）Ora2Pg の実行方法

Ora2Pg には、幾つかの実行方法があります。

a. /etc/ora2pg/ora2pg.conf による実行

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は、

・実行コマンドがシンプル

・Perl を意識する必要が無い

となります。

ora2pg.conf では、次に示す項目の設定が可能です。詳細は、Ora2Pg のプロジェクトサイト、もしくは、サンプルファイルである ora2pg.conf.dist のコメントをご確認ください。

1 Ora2Pg : http://ora2pg.darold.net/

表 2.3.3: ora2pg.conf の主な設定項目

No.	設定項目		説明
1	ORACLE_DSN		接続先 Oracle Database のホスト名と SID
2	ORACLE_USER		接続先 Oracle Database のユーザ名
3	ORACLE_PWD		接続先 Oracle Database のパスワード
4	TYPE 出力対象 オブジェクト設定	オブジェクト定義抽出	TABLE
5			PACKAGE
6			VIEW
7			GRANT
8			SEQUENCE
9			TRIGGER
10			FUNCTION
11			PROCEDURE
12			TABLESPACE
13			TYPE
14		データ抽出	DATA（INSERT 文を生成）
15			COPY（COPY 文を生成）
16		表示動作	SHOW_SCHEMA（利用可能なスキーマをリスト表示）
17			SHOW_TABLE（利用可能なテーブルをリスト表示）
18			SHOW_COLUMN（利用可能な表列をリスト表示）
19			SHOW_ENCODING（ENCODING を表示）
20	ALLOW		抽出対象のテーブルを半角空白セパレータで列挙 （デフォルト動作はすべてのテーブルが抽出対象）
21	SCHEMA		抽出対象のスキーマ名を指定

以下の実行例では、「TYPE」に「SHOW_SCHEMA」を指定して、ユーザの一覧を表示しています。

設定例: 変更箇所抜粋

```
ORACLE_DSN      dbi:Oracle:host=接続ホスト;sid=接続 SID
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: スキーマー覧の出力

```
$ ora2pg
Trying to connect to database: dbi:Oracle:host=接続ホスト;sid=接続 SID
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 Database のスキーマー覧が表示されれば成功です。

b. 任意パスの設定ファイルによる実行

a. と同様の書式（ora2pg.conf）のファイルを ora2pg コマンドに引数として与え、処理を行います。

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の名称が付与できます。実行するには、ora2pg コマンドに-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
```

この方式のメリットは、

・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

となります。

c. 設定ファイルとコマンドオプションの併用

Ora2Pg には以下のようなコマンドオプションがあります。コマンドオプションは”-h”オプションで出力させることができます。これらは設定ファイルの内容に優先して適用され、設定ファイルの値を一部上書きして利用することができます。

実行例：コマンドラインオプションの出力

```
$ ora2pg -h

Usage: ora2pg [-dhpqsv] [--option value]

   -a | --allow str  : coma separated list of objects to allow from export.
                       Can be used with SHOW_COLUMN too.
   -b | --basedir dir: Used to set the default output directory, where files
                       resulting from exports will be stored.
   -c | --conf file  : Used to set an alternate configuration file than the
                       default /etc/ora2pg/ora2pg.conf.
   -d | --debug      : Enable verbose output.
   -e | --exclude str: coma separated list of objects to exclude from export.
                       Can be used with SHOW_COLUMN too.
   -h | --help       : Print this short help.
   -i | --input file : File containing Oracle PL/SQL code to convert with
                       no Oracle database connection initiated.
   -l | --log file   : Used to set a log file. Default is stdout.
   -n | --namespace schema : Used to set the Oracle schema to extract from.
   -o | --out file   : Used to set the path to the output file where SQL will
                       be written. Default: output.sql in running directory.
（省略）

See full documentation at http://ora2pg.darold.net/ for more help or see
manpage with 'man ora2pg'
```

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に指定）

```
$ ora2pg -n SCOTT
```

この方式のメリットは a.や b.と併用することにより、

・設定ファイルの一部を流用することができる

となります。

d. Perl スクリプト（設定ファイルを使用しない）

Ora2Pg は Perl モジュールのため、Perl による独自の抽出・移行のためのスクリプトを実装することができます。抽出したスキーマやデータによる、独自の中間ファイル作成や、DBD::Pg を用いた PostgreSQL への自動投入などの実装が可能です。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を全てスクリプト中に指定する必要があります。

以下は、Ora2Pg の export_schema 関数を利用したスクリプト例です。

スクリプト例: export_schema の利用

```
#!/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 $schema=new Ora2Pg (
   datasource=>'dbi:Oracle:host=接続ホスト;sid=base;port=接続ポート',
   user=>'system',
   password=>'oracle',
   schema=>$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 スクリプトの実行（上記のスクリプトは exprot_schema_scott.pl としている）

```
$ perl export_schema_scott.pl
```

この方式のメリットは、

・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

となります。

2.3.2. Microsoft SQL Server（bcp ユーティリティ）

Microsoft SQL Server には「bcp ユーティリティ[2]」という名称の、レコードを一括エクスポート/インポートするための便利なコマンドが存在します。bcp ユーティリティはコマンドラインで利用します。以下のように、bcp ユーティリティの out 命令を用いることで、指定のデータベース中のテーブルからレコードを文字列型（-c オプション）として表現することができます。CSV 形式（以下の区切り文字は「,」を-t オプションで明示。デフォルトはタブ）でレコードを出力することができます。

```
C:¥> bcp データーベース.スキーマ.テーブル out ファイル名 -c -t "," -S サーバ名 -T
```

上記コマンドによる出力は以下の通りです。コピーされた行数が出力されるため、抽出が正しく実行されたか、確認する項目の 1 つとなります。次のコマンド例では、以下のテーブルのレコードを出力しています。

表 2.3.4: bcp コマンドによる CSV ファイル出力で用いたテーブル"NUMBERS_TEST"

No.	データ型	カラム名
1	tinyint	A
2	smallint	B
3	int	C
4	bigint	D
5	decimal(38,10)	E

実行例: bcp コマンドによる CSV ファイル出力

```
C:¥> bcp ETL_TEST.dbo.NUMBERS_TEST out data.csv -c -t "," -S localhost¥SQLEXPRESS -T

コピーを開始しています...

2 行コピーされました。
ネットワーク パケット サイズ (バイト): 4096
クロック タイム (ミリ秒) 合計     : 1      平均 : (2000.00 行/秒)

C:¥> type data.csv
0,-32768,-2147483648,-9223372036854775808,-99999999999999999999.9999999999
255,32767,2147483647,9223372036854775807,99999999999999999999.9999999998
```

また、bcp ユーティリティは、任意のクエリを実行し、その結果を出力することもできます。以下は、参照の結果を出力する例です。特定のカラムのみを移行対象とする場合に利用すると便利です。

```
C:¥> bcp "SELECT カラム名... FROM [データーベース.スキーマ.テーブル]" queryout
    （改行せずに実際は記述します）ファイル名 -c -t ',' -S サーバ名 -T
```

2　bcp ユーティリティ（SQL Server 2008 R2）: http://msdn.microsoft.com/ja-jp/library/ms162802%28v=sql.105%29.aspx

2.3.3. PostgreSQL（COPY）

PostgreSQL ではレコードのエクスポート方法として、COPY コマンドが存在します。
また、本資料では異種 DBMS 間のデータ移行を想定していますが、インポート対象も PostgreSQL の場合は、スキーマ情報の出力やデータ圧縮も可能な pg_dump の利用を検討してください。
COPY コマンドの凡例は以下の通りです。ファイルが出力された場合は行数が表示されるので、簡単な確認項目として用いることができます。

```
=# COPY (SQL command) TO 'target file' WITH (options);
COPY lines
```

上記コマンドの実行例を以下に記載します。

実行例: COPY コマンドによる CSV ファイル出力

```
=# COPY (SELECT * FROM test_tbl) TO '/tmp/test.csv' WITH (FORMAT csv, ENCODING 'UTF8');
COPY 100
```

WITH 句の後に指定可能なオプションについてはオンラインマニュアルを参照して下さい。[3]

3 COPY コマンド：http://www.postgresql.jp/document/9.2/html/sql-copy.html

3. データの整形(Transform)

本章では、抽出した中間情報の文字エンコーディングと PostgreSQL の文字エンコーディングが異なる場合の注意事項と、PostgreSQL における文字エンコーディング変換の方法について示します。文字エンコーディングの変換は ETL において、変換（Transform）にあたります。本章では、データベースに格納する文字コードを「データベースエンコーディング」クライアント環境の文字コードを「クライアントエンコーディング」と呼びます。

3.1. PostgreSQL で利用できる文字エンコーディング（文字セット）サポート

PostgreSQL で利用できる主な文字エンコーディングを下記の表に記載します。

データベースエンコーディングとクライアントエンコーディングが異なる場合は、自動エンコーディング変換を行うことで、文字化けを回避しています。自動エンコーディング変換については、次の節に記載します。

表 3.1.1: 定義できる文字エンコーディングの組み合わせ

No.	データベースエンコーディング	クライアントエンコーディング
1	UTF8	すべてのエンコーディング
2	EUC_JP	EUC_JP、MULE_INTERNAL、SJIS、UTF8
3	EUC_CN	EUC_CN、MULE_INTERNAL、UTF8
4	EUC_KR	EUC_KR、MULE_INTERNAL、UTF8
5	EUC_TW	EUC_TW、BIG5、MULE_INTERNAL、UTF8
6	ISO_8859_5	ISO_8859_5、KOI8、MULE_INTERNAL、UTF8、WIN866、WIN1251
7	ISO_8859_6	ISO_8859_6、UTF8
8	ISO_8859_7	ISO_8859_7、UTF8
9	ISO_8859_8	ISO_8859_8、UTF8
10	JOHAB	JOHAB、UTF8
11	KOI8	KOI8、ISO_8859_5、MULE_INTERNAL、UTF8、WIN866、WIN1251
12	LATIN1	LATIN1、MULE_INTERNAL、UTF8
13	LATIN2	LATIN2、MULE_INTERNAL、UTF8、WIN1250
14	LATIN3	LATIN3、MULE_INTERNAL、UTF8
15	LATIN4	LATIN4、MULE_INTERNAL、UTF8
16	LATIN5	LATIN5、UTF8
17	LATIN6	LATIN6、UTF8
18	LATIN7	LATIN7、UTF8
19	LATIN8	LATIN8、UTF8
20	LATIN9	LATIN9、UTF8
21	LATIN10	LATIN10、UTF8
22	MULE_INTERNAL	MULE_INTERNAL、BIG5、EUC_CN、EUC_JP、EUC_KR、EUC_TW、ISO_8859_5、KOI8、LATIN1toLATIN4、SJIS、WIN866、WIN1250、WIN1251
23	SQL_ASCII	すべて(自動エンコーディングされません)
24	UTF8	すべて
25	WIN866	WIN866、ISO_8859_5、KOI8、MULE_INTERNAL、UTF8、WIN1251
26	WIN874	WIN874、UTF8
27	WIN1250	WIN1250、LATIN2、MULE_INTERNAL、UTF8
28	WIN1251	WIN1251、ISO_8859_5、KOI8、MULE_INTERNAL、UTF8、WIN866
29	WIN1252	WIN1252、UTF8
30	WIN1253	WIN1253、UTF8
31	WIN1254	WIN1254、UTF8
32	WIN1255	WIN1255、UTF8
33	WIN1256	WIN1256、UTF8
34	WIN1257	WIN1257、UTF8
35	WIN1258	WIN1258、UTF8

SJIS のようにデータベースエンコーディングとして設定することができないエンコーディングも存在するため、注意が必要です。設定可能な文字エンコーディング方式一覧は PostgreSQL のオンラインマニュアルに記載されていますので、各バージョンに合わせてご確認下さい。

参考として、下記に SJIS をデータベースエンコーディングに指定して使用しようとした際のエラーを記載します。

```
$ createdb -E SJIS -T template0 sjis
createdb: database creation failed: ERROR:  SJIS is not a valid encoding name
```

3.2. 自動エンコーディング変換

PostgreSQL では「データベースエンコーディング」と「クライアントエンコーディング」が異なる場合、自動的にエンコーディング方式の変換を行なうことでエンコーディング・ミスマッチを回避しています。「データベースエンコーディング」と「クライアントエンコーディング」が同一の場合は、自動エンコーディング変換を行いません。

上記を「①値の取得」と「②値の入力」に分けて以下に図示し、注意すべき点を整理します。以下図で、記載している「CtoD」とは、クライアントエンコーディングをデータベースエンコーディングに変換する処理として用いています。「DtoC」はその逆の処理を指します。

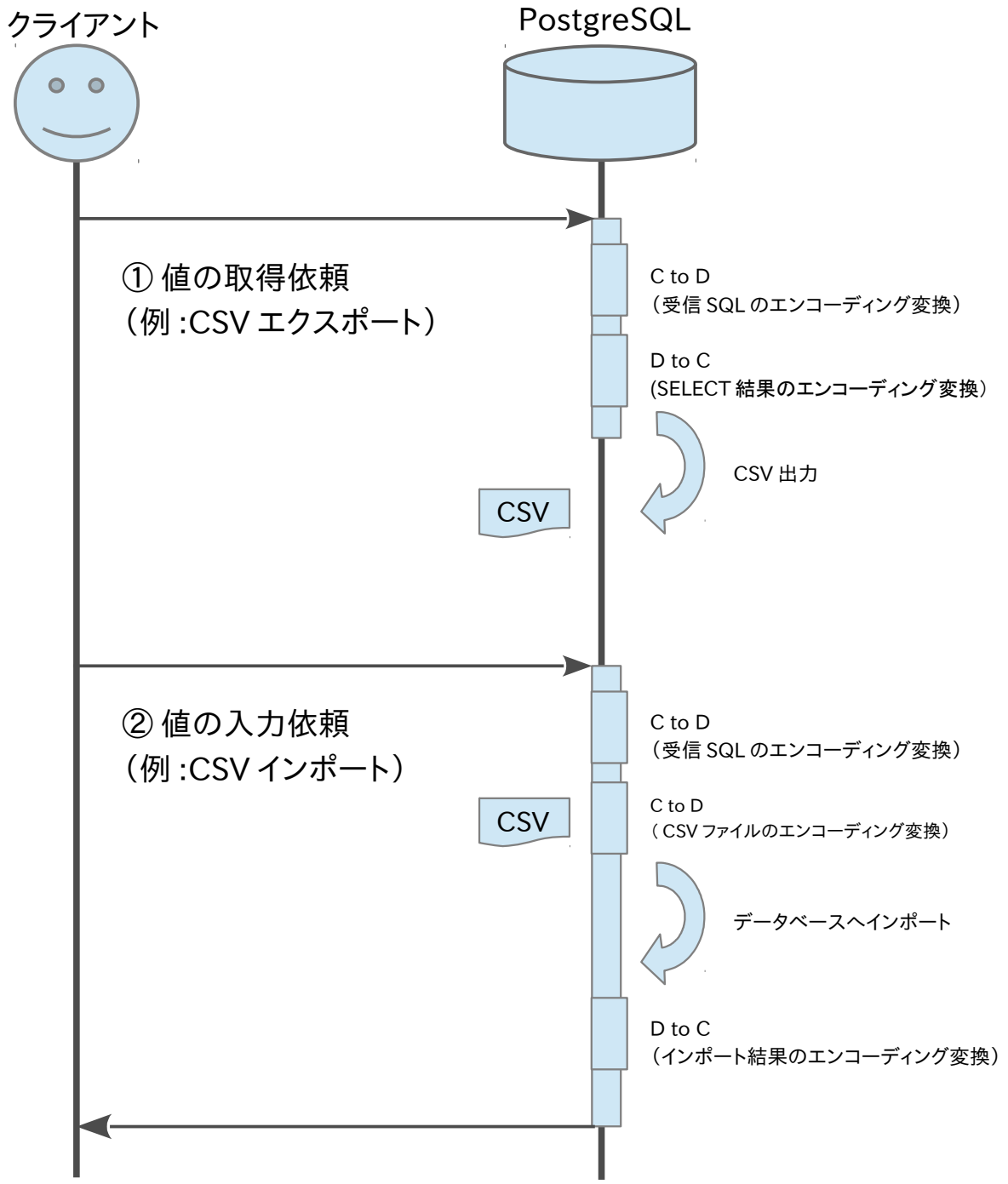

図 3.2.1: 自動エンコーディング変換概要

3.3. PostgreSQL における文字エンコーディングの確認・指定方法

3.3.1. クライアントエンコーディング設定方法

PostgreSQL のクライアントエンコーディング設定は下表の 4 つの設定方法があります。複数の設定により指定がなされていた場合は、「No.」が小さいものが優先されます。

表 3.3.1: クライアントエンコーディングの指定方法（上位 No.が優先される）

No.	方法	設定例（コマンドオペレーション、psql での例）
1	SET コマンドによる設定	=# SET client_encoding TO '*文字エンコーディング*';
2	クライアント環境変数 PGCLIENTENCODING	$ export PGCLIENTENCODING=*文字エンコーディング*
3	クライアント環境変数 LANG	$ export LANG=*文字エンコーディング*
4	postgresql.conf "client_encoding"	=# client_encoding =*文字エンコーディング*

Java などのプログラムからアクセスする場合は、別途接続ドライバの設定ファイルによる指定方法や API リファレンスを確認してください。

3.3.2. クライアントエンコーディング確認方法

接続セッションのクライアントエンコーディングは、SHOW コマンドで確認できます。

実行例: クライアントエンコーディングの確認

```
=# SHOW client_encoding ;
 client_encoding
-----------------
 UTF8
(1 row)
```

3.3.3. データベースエンコーディング設定方法

PostgreSQL のデータベースエンコーディングはデータベースクラスタ初期化時（initdb 実行時）に指定した文字エンコーディングが利用されます。

実行例: UTF8 指定によるデータベースクラスタ初期化

```
$  initdb --encoding=UTF8
```

initdb で指定した文字エンコーディングは、initdb で作成される「template0」および「template1」のデータベースエンコーディングとして登録されます。データベース作成時に別途指定しない限りは「template1」のデータベースエンコーディングがデフォルトとして利用されます。

このデフォルトのデータベースエンコーディングを変更するには、データベース作成時にテンプレートとして「template0」を指定し、任意の文字エンコーディングを指定します。

実行例: データベースエンコーディングの EUC_JP への変更

```
postgres=# CREATE DATABASE testDB TEMPLATE template0 ENCODING 'EUC_JP';
```

3.3.4. データベースエンコーディング確認方法

以下の実行例では、psql を用いて全てのデータベースのエンコーディングを出力しています。

実行例: データベースエンコーディングの確認

```
$ psql -l
                  List of databases
   Name    | Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+---------+----------+---------+-------+-----------------------
 template0 | postgres | UTF8     | C       | C     | =c/postgres  postgres=CTc/postgres
 template1 | postgres | UTF8     | C       | C     | =c/postgres  postgres=CTc/postgres
 test      | postgres | EUC_JP   | C       | C     |
```

3.4. PostgreSQL の自動エンコーディング変換を行う関数

デフォルトで定義されている関数が文字コードマッピングファイルを参照し、自動エンコーディング変換を行っています。自動エンコーディング変換を行う関数は initdb を実行した際に登録されています。

表 3.4.1: 文字エンコーディング関数名

No.	変換元エンコーディング	変換先エンコーディング	関数名
1	sjis	utf8	sjis_to_utf8
2	utf8	sjis	utf8_to_sjis

自動エンコーディング変換に使用されている関数名は、システムカタログの pg_conversion で確認することができます。

実行例: pg_conversion 参照による確認

```
=# SELECT conname,pg_encoding_to_char(conforencoding) AS for_encording,pg_encoding_to_char(contoencoding)
AS to_encording ,conproc FROM pg_conversion;

(一部抜粋)

-[ RECORD 1 ]-+-------------------------------
connam        | sjis_to_utf8   ① エンコーディング変換名
for_encording | SJIS           ②変換元エンコーディング
to_encording  | UTF8           ③変換先エンコーディング
conproc       | sjis_to_utf8   ④ 変換を行う関数名
```

initdb で登録される関数のソースファイルは、postgresql-9.2.2/src/backend/utils/mb/conversion_procs のサブディレクトリ内に存在します。例えば、SJIS から UTF8、UTF8 から SJIS の自動エンコーディング変換を行う関数は、utf8_and_sjis サブディレクトリ内の、utf8_and_sjis.c に定義されています。

3.4.1. 文字コードマッピングファイル

自動エンコーディング変換を行う関数は、文字コードマッピングファイルを元に変換を行っています。
例として SJIS から UTF8 に変換するための文字コードマッピングファイル「sjis_to_utf8.map」を以下に記載します。
下記のように、配列で変換のルールを定義しています。

文字エンコーディングファイル（抜粋）:

```
static pg_local_to_utf LUmapSJIS[ 7787 ] = {
(省略)
{0x82a0, 0xe38182},    //SJIS の「あ」を UTF8 の「あ」に変換
{0x82a1, 0xe38183},    //SJIS の「ぁ」を UTF8 の「ぁ」に変換
{0x82a2, 0xe38184},    //SJIS の「い」を UTF8 の「い」に変換
{0x82a3, 0xe38185},    //SJIS の「ぃ」を UTF8 の「ぃ」に変換
{0x82a4, 0xe38186},    //SJIS の「う」を UTF8 の「う」に変換
(省略)
};
```

文字コードマッピングファイルは「postgresql-9.2.2/src/backend/utils/mb」配下に存在します。

例 Unicode の場合:

```
postgresql-9.2.2/src/backend/utils/mb/Unicode
```

SJIS から UTF8 の文字コードマッピングファイルは、「sjis_to_utf8.map」、UTF8 から SJIS の文字コードマッピングファイルは、「utf8_to_sjis.map」となります。
その他のエンコーディング変換に関しても、同様に文字コードマッピングファイルが存在します。
例えば、UTF8 から EUC_JP に変換する場合は、utf8_to_euc_jp.map を確認してください。

3.5. 外字の扱い

PostgreSQL において、クライアントエンコーディングとデータベースエンコーディングが異なる環境で外字を使用する場合、外字のマッピングを定義し、マッピングを再登録する必要があります。クライアントエンコーディングとデータベースエンコーディングが同一の場合は、自動エンコーディング変換を行わないため、外字のマッピング定義、再登録は不要です。

外字を使用し、自動エンコーディング変換を利用する場合は、PostgreSQL のモジュールの再コンパイル行い、再登録をする必要があります。このため、使用する外字を洗い出しておく必要があります。エンドユーザが自由に外字を追加する運用はできないため注意が必要です。

3.5.1. 外字

ここでは、PostgreSQL の文字セットにあらかじめ登録されていない文字を、外字と呼びます。Windows の外字エディタなどを用いてユーザが独自に字形を作成した文字に、文字コードを割り当て利用します。どんな字形であっても登録できますが、その外字を登録したコンピュータでしか正しく扱うことができない、という点に注意してください。

3.5.2. 文字コードマッピングファイルへの外字登録

PostgreSQL に外字を登録する手順概要は下記の通りです。

(1) 文字コードマッピングファイルに文字コードの定義を追加
(2) マッピングファイルを使用している関数を再コンパイル・インストール
(3) 自動エンコーディング変換を行う関数として登録

本節では、以下の外字の登録例を記載します。

表 3.5.1: 外字登録例

No.	エンコーディング項目	値
1	データベースエンコーディング	UTF8
2	クライアントエンコーディング	SJIS
3	登録したい外字	脇
4	UTF8 で外字に割り当てる外字コード	0xee8080
5	SJIS で外字に割り当てる文字コード	0xf040

（1）文字コードマッピングファイルに文字コードの定義を追加

「postgresql-9.2.2/src/backend/utils/mb/Unicode」配下に存在する SJIS から UTF8 への文字コードマッピングファイル「sjis_to_utf8.map」と UTF8 から SJIS への文字コードマッピングファイル「utf8_to_sjis.map」を編集します。

a. sjis_to_utf8.map の修正

1. 追加する外字の数に応じて、配列数を増やします。
2. SJIS の文字コード「0xf040」を、UTF8 の文字コード「0xee8080」にマッピングします。追加する際に、変換元のエンコーディングを必ず 16 進数の昇順に記載しなければなりません。

```
static pg_local_to_utf LUmapSJIS[ 7788 ] = {
(省略)
{0xeefc, 0xefbc82},
{0xf040, 0xee8080},
{0xfa40, 0xe285b0},
{0xfa41, 0xe285b1},
{0xfa42, 0xe285b2},
(省略)
:
```

b. utf8_to_sjis.map の修正

1. 追加する外字の数に応じて、配列数を増やします。
2. UTF8 の文字コード「0xee8080」を、SJIS の文字コード「0xf040」にマッピングします。追加する際に、変換元のエンコーディングを必ず 16 進数の昇順に記載しなければなりません。

```
static pg_utf_to_local ULmapSJIS[ 7399 ] = {
(省略)
{0xe9bea0, 0xea9e},
{0xee8080, 0xf040},
{0xefa4a9, 0xfae0},
{0xefa79c, 0xfbe9},
{0xefa88e, 0xfa90},
(省略)
```

3.5.3. 文字コードマッピングファイルを使用している関数を再コンパイル

initdb 時に登録された関数 utf8_to_sjis と、sjis_to_utf8 を再コンパイルし、文字コードマッピングファイルの定義の再読み込みを行います。関数 utf8_to_sjis と sjis_to_utf8 は、同一ファイルの中で定義されています[4]。

```
$ cd postgresql-9.2.2/src/backend/utils/mb/conversion_procs/utf8_and_sjis/
$ make clean
$ make
$ su
# make install
```

コンパイルし、インストールすると、/usr/local/pgsql/lib/utf8_and_sjis.so が入れ替わります。

3.5.4. 自動エンコーディング変換を行う関数として登録

自動エンコーディング変換を行う関数として、CREATE CONVERSION で登録をします。
まずは、デフォルトで作成されているエンコーディング変換の定義を削除します。

```
=# DROP CONVERSION sjis_to_utf8;
=# DROP CONVERSION utf8_to_sjis;
```

次に再コンパイルして作成した関数を、エンコーディング変換を行う関数として登録します。

```
=# CREATE DEFAULT CONVERSION sjis_to_utf8 FOR 'SJIS' TO 'UTF8' from sjis_to_utf8;
=# CREATE DEFAULT CONVERSION utf8_to_sjis FOR 'UTF8' TO 'SJIS' from utf8_to_sjis;
```

CREATE CONVERSION コマンド

エンコーディング変換を設定するための SQL コマンドです。書式は以下の通りです。

```
=# CREATE [ =DEFAULT ] CONVERSION name FOR source_encoding TO dest_encoding FROM
function_name
```

下記に自動エンコーディング変換定義の各パラメータの概要を記載します。

表 3.5.2:自動エンコーディング変換定義のパラメータ

No.	パラメータ	概要
1	DEFAULT	定義したエンコーディング変換をデフォルトに指定します。1 つのスキーマ内で、エンコーディングの組み合わせ1つにつき、1つの変換をデフォルトとして指定できます。
2	name	エンコーディング変換の名前を示します。
3	source_encoding	変換元のエンコーディング名です。
4	dest_encoding	変換先のエンコーディング名です。
5	function_name	この関数は、エンコーディング変換実行の際に使用されます。

4　マッピングファイルを使用している関数: postgresql-9.2.2/src/backend/utils/mb/conversion_procs/utf8_and_sjis/utf8_and_sjis.c

3.6. ロケールの指定

PostgreSQL でロケールの設定を行うと、データベース内での文字列処理、日付や通貨の表示、メッセージの言語などを変更することができます。

3.6.1. ロケール指定の種類

ロケール関連項目は下表の通り複数ありますが、基本的にデータベース・クラスタ作成時に指定したロケール（initdb の--locale オプション）がすべてに適用されます。

ロケール関連項目のうち、「LC_COLLATE」と「LC_CTYPE」はデータベース・クラスタまたはデータベース作成時しか設定できません。データベース作成後は変更できないので注意してください。

表 3.6.1: ロケールの種類

No.	ロケール関連項目	概要
1	LC_COLLATE	文字列の並び換え順の設定
2	LC_CTYPE	文字の分類の設定
3	LC_MESSAGES	メッセージの言語の設定
4	LC_MONETARY	通貨の書式
5	LC_NUMERIC	数字の書式
6	LC_TIME	日付と時刻の書式

ロケールは指定しないことも可能です。明示的にロケールを指定すること（ja_JP.UTF8 など）で、辞書順でのソート処理が可能になるなど利点もありますが、実行速度が遅くなるなどの問題点もある[5]ため、ロケールはシステムの仕様上、特別に求めない限り設定しないでください。

3.6.2. ロケール指定方法

a. データベース・クラスタ作成時 (initdb)

ロケールを指定する場合は、以下のように initdb を実行します。日本語の場合は [言語] = ja、[地域] = JP と設定します。

```
$ initdb --encoding=UTF8 --locale=ja_JP.UTF8
```

ロケールを指定しない場合は、以下のように実行します（基本的にこちらのロケールなしでの構築になります）。

```
$ initdb --no-locale
または、
$ initdb --locale=C
```

b. データベース作成時（CREATE DATABASE）

データベース作成時に initdb で指定したロケール以外を指定したい場合は以下のようにデータベースを作成します。

```
=# CREATE DATABASE データベース LC_COLLATE 'ロケール' LC_CTYPE 'ロケール' TEMPLATE
template0;
```

※LC_COLLATE と LC_CTYPE 以外は postgresql.conf の修正と再読み込みで設定可能です。

3.6.3. ロケールの確認方法

ロケールの確認方法は以下の通りです。

```
=# SELECT name,setting,context,sourcefile FROM pg_settings WHERE name LIKE 'lc%';
```

5 ロケールのサポート：http://www.postgresql.jp/document/9.2/html/locale.html

3.7. ファイルの文字コードの変換

PostgreSQL へ投入するテキストファイルが PostgreSQL に対応していない場合や、pg_bulkload[6]を用いてデータ投入を行う場合、文字エンコーディングの変換を行う必要があります。本資料では、文字エンコーディングの変換は「nkf」および「iconv」と「dos2unix」を用います。なお、ファイルサイズが小さい場合などは任意のエディタを用い変換しても問題ありません。

3.7.1. nkf の利用

nkf は日本製ソフトウェアのため、文字エンコーディング変換や確認がある程度柔軟に実行できます。

a. ファイルの文字エンコーディング確認（-g）

テキストファイルの文字エンコーディングが不明な場合や nkf による変換が正しく成功しているか確認する場合、次のように実行します。

```
$ nkf -g 確認したいテキストファイル
Shift_JIS (CR)
```

b. ファイルの文字エンコーディング変換（UTF-8/LF）

テキストファイルの文字エンコーディングを「UTF-8」、改行コードを「LF」に変換する際は、次のように実行します。他の文字コードおよび改行コードについては下表を参考に記載してください。

```
$ nkf -w -Lu 変換したいテキストファイル > 新規テキストファイル名
$ nkf -g 新規テキストファイル名
UTF-8 (LF)
```

ファイルを上書きして変換する場合は次のように「--overwrite」オプションを付与して、実行してください。

```
$ nkf -w -Lu –overwrite 変換したいテキストファイル
$ nkf -g 変換したいテキストファイル
UTF-8 (LF)
```

上記以外のコマンドオプションについては、man 等で確認してください。

6 pg_bulkload は処理高速化のため、ロード中に文字エンコーディングの自動変換を行いません

表 3.7.1: 文字エンコーディングと付与する変換時のオプション

No.	文字エンコーディング	オプション
1	UTF-8 (BOM なし)	-w/-w80
2	UTF-8 (BOM 有)	-w8
3	UTF-16 (big endian BOM なし)	-w16/-16B0
4	UTF-16 (big endian BOM 有)	-w16B
5	UTF-16 (little endian BOM なし)	-w16L0
6	UTF-16 (little endian BOM 有)	-w16L
7	EUC-JP	-e
8	Shift-JIS (CP932)	-s
9	JIS (ISO-2022-JP)	-j

表 3.7.2: 改行コードと付与する変換時のオプション

No.	改行コード (主な OS)	オプション
1	LF (UNIX)	-Lu/-d
2	CRLF (Windows)	-Lw/-c
3	CR (Mac)	-Lm

3.7.2. iconv の利用

iconv は文字エンコーディングを変換するコマンドです。利用できる文字エンコーディングは日本語に限りません。ここでは iconv コマンドを利用した例を以下に示します。

a. 利用可能な文字エンコーディングの確認

変換したい文字エンコーディング（変換元、変換先）が存在するか確認する場合、次のように入力します。

```
$ iconv --list
437, 500, 500V1, 850, 851, 852, 855, 856, 857, 860, 861, 862, 863, 864, 865,
:
（省略）
```

grep コマンドと併用すると便利です。

```
$ iconv --list | grep UTF-8
ISO-10646/UTF-8/
UTF-8//
```

b. ファイルの文字エンコーディング変換（Shift_JIS/CRLF から UTF-8/LF）

文字エンコーディングが「Shift_JIS」、改行コードが「CRLF」のテキストファイルから、「UTF-8」、改行コードを「LF」のテキストファイルに変換する際は、次のように実行します。改行コードを「CRLF」にしたい場合は、unix2dos を利用してください。

```
$ iconv -f Shift_JIS -t UTF-8 変換したいファイル | dos2unix > 新規テキストファイル名
$ nkf -g 新規テキストファイル名
UTF-8 (LF)
```

4. データの投入(Load)

本章では、抽出（Extract）し、任意の整形（Transform）を行った中間データを利用して PostgreSQL へデータを投入する方法（Load）について記載します。

4.1. COPY

PostgreSQL でサポートされる SQL コマンドの COPY 文を使用して、次のように CSV ファイルからテーブルへデータを投入します。

```
$ psql -U postgres testdb
testdb=# COPY test_table FROM '/tmp/test1.csv' WITH DELIMITER ',';
COPY 10
```

(*1)既にデータが存在する場合は追加で投入します。データ投入の重複を防ぐため、投入前には TRUNCATE を実行するなどの対応をお勧めします。
(*2)COPY FROM 文は、PostgreSQL スーパユーザで実行する必要があります。

CSV ファイル内の任意の文字列を NULL 値として格納する場合の例を以下に記載します。

```
$ psql -U postgres testdb
testdb=# COPY test_table FROM '/tmp/test1.csv' with (NULL 'null_string', format CSV);
COPY 10
```

4.2. psql からファイル指定で実行

Ora2Pg を用いて INSERT 文などの形式でデータの抽出を行った場合は、psql ユーティリティにてファイル指定で実行します。

```
$ psql -U postgres testdb -f test1.sql
```

(*1)既にデータが存在する場合は追加で投入します。データ投入の重複を防ぐため、投入前には TRUNCATE を実行するなどの対応をお勧めします。

4.3. pg_bulkload

pg_bulkload は、大量のデータを高速に投入するためのツールです。データベース制約のチェックの有無や、エラーデータをスキップして投入を継続するか否かを制御でき、入力データに応じた柔軟なデータ移行が可能です。たとえば、あるデータベースに格納されている情報を別のデータベースへ移送するような状況では、データの整合性は既に確認済みですので、細かなチェックは省き、高速にデータをロードできます。一方、別ツールで出力したデータの整合性が疑わしい場合には、制約をチェックしながら投入できます。

pgFoundry のサイト[7]から、ダウンロードしてインストールする必要があります。インストールは PostgreSQL をビルド可能な環境で行う必要があります。

```
$ cd pg_bulkload
$ make USE_PGXS=1
$ su
$ make USE_PGXS=1 install
```

また、PostgreSQL のバージョンによって、DB への登録方法が異なります。

実行例: PostgreSQL 9.1 以下の場合

```
$ postgresql start
$ psql -f $PGSHARE/contrib/pg_bulkload.sql database_name
```

実行例: PostgreSQL 9.2 以上の場合

```
$ psql database_name
database_name=# CREATE EXTENSION pg_bulkload;
```

1. 制御ファイルを作成

ロード対象のテーブル名、入力ファイルのパスなどを指定します。

制御ファイルサンプル

```
#
# sample_csv.ctl -- Control file to load CSV input data
#
#    Copyright (c) 2007-2011, NIPPON TELEGRAPH AND TELEPHONE CORPORATION
#
TABLE = test                    # [<schema_name>.]table_name
INPUT = /home/postgres/input_data_file.csv  # Input data location (absolute path)
TYPE = CSV                      # Input file type(CSV | BINARY | FIXED | FUNCTION )
WRITER = DIRECT                     # Road (DIRECT | BUFFERED | BINARY | PARALLEL )
QUOTE = "¥""                    # Quoting character
ESCAPE = ¥                      # Escape character for Quoting
DELIMITER = ","                  # Delimiter
ENCODING = UTF8                     # DB Encode
CHECK_CONSTRAINTS = YES             # Constraintcheck(YES | NO)
```

2. $PGDATA/pg_bulkload ディレクトリの存在を確認

$PGDATA/pg_bulkload ディレクトリにはロードステータスファイルが作成されます。本ディレクトリが存在しない場合は、作成する必要があります。

7 pg_bulkload : http://pgbulkload.projects.pgfoundry.org/

3. 制御ファイルを引数としてコマンドを実行

実行例

```
$ pg_bulkload sample_csv.ctl
NOTICE: BULK LOAD START
NOTICE: BULK LOAD END
        0 Rows skipped.
        8 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.
```

(*1)すでにデータが存在する場合は追加で投入します。データ投入の重複を防ぐため、投入前には TRUNCATE を実行するなどの対応をお勧めします。
(*2)pg_bulkload は、PostgreSQL のスーパユーザで実行する必要があります。
(*3)pg_bulkload は高速化のためにロード中に文字コードの変換を行いません。pg_bulkload を利用するユーザは、ロード対象のデータベースの文字コードに合わせて、事前に入力ファイルを作成しておく必要があります。例えば、エンコーディングを EUC_JP としてデータベースを作成した場合、入力ファイルのエンコーディングも EUC_JP で作成してください。

また、pg_bulkload では、下記のコマンドライン引数を指定できます。

a) ロードオプション

```
-i INPUT
--input=INPUT
--infile=INPUT
   入力データソースを指定します。
-O OUTPUT
--output=OUTPUT
   データの出力先を指定します。
-l LOGFILE
--logfile=LOGFILE
   ロード処理の結果を記録するログファイルのパスを指定します。
-P PARSE_BADFILE
--parse-badfile=PARSE_BADFILE
   入力データのパース時に見つかった不良レコードを記録する BAD ファイルのパスを指定します。
-u DUPLICATE_BADFILE
--duplicate-badfile=DUPLICATE_BADFILE
   インデックスメンテナンス処理で見つかった一意制約違反の不良レコードを記録する BAD ファイルのパスを指定します。
-o "key=val"
--option="key=val"
   制御ファイルで指定可能な設定項目を指定します。
```

b) 接続オプション

-d DBNAME
--dbname=DBNAME
　接続するデータベース名を指定します。データベース名が指定されていない場合、PGDATABASE 環境変数からデータベース名を
　読み取ります。この変数も設定されていない場合は、接続時に指定したユーザ名が使用されます。
-h HOSTNAME
--host=HOSTNAME
　サーバが稼働しているマシンのホスト名を指定します。ホスト名がスラッシュから始まる場合、Unix ドメインソケット用の
　ディレクトリとして使用されます。
-p PORT
--port=PORT
　サーバが接続を監視する TCP ポートもしくは Unix ドメインソケットファイルの拡張子を指定します。
-U USERNAME
--username=USERNAME
　接続するユーザ名を指定します。
-W
--password
　データベースに接続する前に、pg_bulkload は強制的にパスワード入力を促します。サーバがパスワード認証を
　要求する場合 pg_bulkload は自動的にパスワード入力を促しますので、これが重要になることはありません。
　しかし、pg_bulkload は、サーバにパスワードが必要かどうかを判断するための接続試行を無駄に行います。
　こうした余計な接続試行を防ぐために -W の入力が有意となる場合もあります。

c) 一般オプション

-e
--echo
　サーバに送信する SQL を表示します
-E LEVEL
--elevel = LEVEL
　ログ出力レベルを設定します。DEBUG, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, PANIC から選択します。
　デフォルトは INFO です
--help
　ヘルプを表示し、終了します
--version
　バージョン情報を出力し、終了します

5. データ移行後の作業

データ移行後に確認すべき項目や、作業すべき手順について記載します。

5.1. データロード時のエラー等の確認

COPY コマンド、pg_bulkload で PostgreSQL データベースにデータのロードを実行した場合は、エラーメッセージの有無を確認します。エラーが表示された場合は、その問題点を確認し、対処してから再度データのロードを実行します。

COPY コマンドによるロード時にエラーが発生した場合は、その処理が全てロールバックされますが、pg_bulkload は直接ファイルへ書き込みを行うため、ロードが完了している行についてはロールバックは実行されません。このため、pg_bulkload によるデータのロードを再実行する際は、一旦テーブルを TRUNCATE してから実行するよう注意が必要です。

5.2. 制約・索引の作成

PostgreSQL へデータを投入する際、索引や制約が定義されていると、データの挿入速度が劣化してしまいます。また、制約条件（外部キー等）が定義されていると、単純なデータの投入でも失敗する可能性があります。このため、データ投入完了後、索引および制約を作成します。

実行例：制約の作成（主キー）

```
=# ALTER TABLE warehouse ADD CONSTRAINT warehouse_pk PRIMARY KEY (w_id);
```

実行例：制約の作成（外部キー）

```
=# ALTER TABLE district ADD CONSTRAINT district_fk1 FOREIGN KEY (d_w_id) REFERENCES warehouse (w_id);
```

実行例：索引の作成

```
=# CREATE INDEX customer_ix1 ON customer (c_w_id, c_d_id, c_last);
```

5.3. 移行対象オブジェクト数および各オブジェクトの行数の確認

データロード後の PostgreSQL データベースのオブジェクト数と各オブジェクトの行数が、移行元データベースの状態と一致するかどうか確認します。

実行例: ロード結果の確認

```
/* ユーザ毎に所有するテーブルの確認 */
=# ¥d

List of relations
 Schema |       Name        |   Type   | Owner
--------+-------------------+----------+--------
 scott  | emp               | table    | scott
 scott  | dept              | table    | scott

/* 各テーブルの行数を確認 */
=#select count(*) from emp;

  COUNT(*)
----------
    602000

/* インデックスを確認 */
=# select schemaname,relname,indexrelname from pg_stat_user_indexes;
 schemaname | relname |  indexrelname
------------+---------+-----------------
 scott      | emp     | empno_pkey
 scott      | dept    | dept_no

/* 外部キーを確認*/
=# ¥d+ scott.history;

(省略)
Foreign-key constraints:
    "history_fk1" FOREIGN KEY (h_w_id, h_d_id) REFERENCES scott.district(d_w_id, d_id)
    "history_fk2" FOREIGN KEY (h_c_w_id, h_c_d_id, h_c_id) REFERENCES scott.customer(c_w_i
d, c_d_id, c_id)
Has OIDs: no
```

5.4. ユーザのシステム権限およびオブジェクト権限の確認

データロード後の PostgreSQL データベースユーザのシステム権限やオブジェクト権限が、移行元データベースの状態と一致するかどうか確認します。

実行例: システム権限、オブジェクト権限の確認

```
/* SCOTT ユーザに付与されているロールの確認 */

=# select rolsuper,rolinherit,rolcreaterole,rolcreatedb,rolcatupdate,rolcanlogin from pg_roles where rolname='scott';
 rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcatupdate | rolcanlogin
----------+------------+---------------+-------------+--------------+-------------
 f        | t          | f             | f           | f            | t
(1 row)

/* オブジェクト権限の確認 */

=# ¥z
                            Access privileges
 Schema |       Name        |   Type   |   Access privileges   | Column access privileges
--------+-------------------+----------+-----------------------+--------------------------
 scott  | emp               | table    | scott=arwdDxt/scott+ |
        |                   |          | test=r/scott         |
```

5.5. VACUUM と ANALYZE の実行

不要タプルが使用する領域の回収および統計情報の更新を行います。VACUUM については FULL を指定し、未使用領域を解放します。

実行例: VACUUM と ANALYZE の実行

```
=# VACUUM(FULL, ANALYZE);
VACUUM
```

5.6. テーブルサイズの確認

データロード後、初期状態のテーブルサイズを取得します。以下の実行例を、投入した全てのテーブルに対して実行します。

異種 DBMS と PostgreSQL は同じデータでも格納方式やデータ長の違いなどから異なるサイズとなる場合があるため、事前に異種 DBMS のサイズを取得していたとしても、本項で取得した PostgreSQL データベースサイズとの比較による移行成功確認はできません。しかし、本項の作業は、PostgreSQL の初期状態のサイズ確認として有益です。

実行例：scott スキーマの warehouse テーブルサイズを取得

```
=# select pg_relation_size('scott.warehouse');
 pg_relation_size
------------------
             8192
```

5.7. アプリケーションテスト

実際にアプリケーションから接続して一連の処理を行い、想定通りの動きをするかどうか、文字化けが発生していないか等を確認します。

6. 実際の試行

本章では、これまで記載したデータ移行フローに従い、ある試験データの移行を実際に行った結果を示します。なお、本章で実施した試行コマンドについては別紙「6 章移行手順」をご確認ください。

6.1. 試行環境と試行データ

異種 DBMS(Oracle Database)を準備し、PostgreSQL へデータの移行を実施します。

6.1.1. 試行環境（プラットフォーム等）

本試行は、次の環境で実施しました。

表 6.1.1: 試行環境

No.	環境名	実装	バージョン情報
1	プラットフォーム	VMware vSphere	4.1
2	OS	CentOS	6.2 final x86_64
3	CPU	2 個	intel Xeon(R) CPU　E5-2650 相当
4	RAM	2GB	
5	移行元 DBMS	Oracle Database	11gR2
6	移行先 DBMS	PostgreSQL	9.2.2
7	利用抽出ツール	Ora2Pg	10.1
8	利用登録ツール	pg_bulkload	3.1.2
9	利用アプリケーション	JdbcRunner	1.2

6.1.2. 移行元、移行先 DBMS の環境設定値

移行元 DBMS、移行先 DBMS の環境設定値は次のように定義されています。

表 6.1.2: 移行先、移行元データベース情報

No.	環境名	値	備考
1	Oracle Database SID	orcl	
2	Oracle Database 移行対象ユーザ名	scott	
3	Oracle Database 移行対象 ユーザパスワード	tiger	
4	Oracle Database システムユーザ名	system	
5	Oracle Database 移行対象 システムユーザパスワード	oracle	
6	Oracle Database 文字コード	UTF-8	
7	移行先 PostgreSQL の DB 名	tpcc	
8	PostgreSQL スーパユーザ名	postgres	
9	PostgreSQL スーパユーザパスワード	postgres	
10	移行先 PostgreSQL 文字コード	UTF-8	本試行では日本語が記載されたデータが存在していません

6.1.3. JdbcRunner について

本章では、移行対象のデータ作成と、移行後のアプリケーションテストを行うために、JdbcRunner[8]と付属のテーブル構造の 1 つである「Tiny TPC-C[9]」を利用しています。以下のような利点があるため、JdbcRunner を採用しました。

・PostgreSQL と異種 DBMS に対してデータのロードとベンチマークの実行が可能である。

また、以下のような利点があるため、移行元データベースのモデルとして Tiny TPC-C を利用しました。

・TinyTPC-C は、TPC－C[10]に基づいており、データ構造やアプリケーションの挙動が良く知られている。

本試行では、データ構造の妥当性及び、アプリケーションの動作保証を JdbcRunner に担ってもらうことで、移行結果の情報収集にのみ専念することができました。

8　JdbcRunner : http://hp.vector.co.jp/authors/VA052413/jdbcrunner/
9　JdbcRunner「テストキット Tiny TPC-C」: http://hp.vector.co.jp/authors/VA052413/jdbcrunner/manual_ja/tpc-c.html
10　TPC-C : http://www.tpc.org/tpcc/

6.1.4. 移行対象テーブル

本試行で利用するテーブルを下表に示します。OracleDatabase および PostgreSQL のデータ型を列挙しています。

表 6.1.3: warehouse テーブル

No.	カラム名	Oracle Database データ型	PostgreSQL データ型
1	w_id	NUMBER(6, 0)	INTEGER
2	w_name	VARCHAR2(10)	VARCHAR(10)
3	w_street_1	VARCHAR2(20)	VARCHAR(20)
4	w_street_2	VARCHAR2(20)	VARCHAR(20)
5	w_city	VARCHAR2(20)	VARCHAR(20)
6	w_state	CHAR(2)	CHAR(2)
7	w_zip	CHAR(9)	CHAR(9)
8	w_tax	NUMBER(4, 4)	DECIMAL(4, 4)
9	w_ytd	NUMBER(12, 2)	DECIMAL(12, 2)

表 6.1.4: district テーブル

No.	カラム名	Oracle Database データ型	PostgreSQL データ型
1	d_id	NUMBER(2, 0)	INTEGER
2	d_w_id	NUMBER(6, 0)	INTEGER
3	d_name	VARCHAR2(10)	VARCHAR(10)
4	d_street_1	VARCHAR2(20)	VARCHAR(20)
5	d_street_2	VARCHAR2(20)	VARCHAR(20)
6	d_city	VARCHAR2(20)	VARCHAR(20)
7	d_state	CHAR(2)	CHAR(2)
8	d_zip	CHAR(9)	CHAR(9)
9	d_tax	NUMBER(4, 4)	DECIMAL(4, 4)
10	d_ytd	NUMBER(12, 2)	DECIMAL(12, 2)
11	d_next_o_id	NUMBER(8, 0)	INTEGER

表 6.1.5: customer テーブル

No.	カラム名	Oracle Database データ型	PostgreSQL データ型
1	c_id	NUMBER(5, 0)	INTEGER
2	c_d_id	NUMBER(2, 0)	INTEGER
3	c_w_id	NUMBER(6, 0)	INTEGER
4	c_first	VARCHAR2(16)	VARCHAR(16)
5	c_middle	CHAR(2)	CHAR(2)
6	c_last	VARCHAR2(16)	VARCHAR(16)
7	c_street_1	VARCHAR2(20)	VARCHAR(20)
8	c_street_2	VARCHAR2(20)	VARCHAR(20)
9	c_city	VARCHAR2(20)	VARCHAR(20)
10	c_state	CHAR(2)	CHAR(2)
11	c_zip	CHAR(9)	CHAR(9)
12	c_phone	CHAR(16)	CHAR(16)
13	c_since	DATE	TIMESTAMP
14	c_credit	CHAR(2)	CHAR(2)
15	c_credit_lim	NUMBER(12, 2)	DECIMAL(12, 2)
16	c_discount	NUMBER(4, 4)	DECIMAL(4, 4)
17	c_balance	NUMBER(12, 2)	DECIMAL(12, 2)
18	c_ytd_payment	NUMBER(12, 2)	DECIMAL(12, 2)
19	c_payment_cnt	NUMBER(4, 0)	DECIMAL(4, 0)
20	c_delivery_cnt	NUMBER(4, 0)	DECIMAL(4, 0)
21	c_data	VARCHAR2(500)	VARCHAR(500)

表 6.1.6: history テーブル

No.	カラム名	Oracle Database データ型	PostgreSQL データ型
1	h_c_id	NUMBER(5, 0)	INTEGER
2	h_c_d_id	NUMBER(2, 0)	INTEGER
3	h_c_w_id	NUMBER(6, 0)	INTEGER
4	h_d_id	NUMBER(2, 0)	INTEGER
5	h_w_id	NUMBER(6, 0)	INTEGER
6	h_date	DATE	TIMESTAMP
7	h_amount	NUMBER(6, 2)	DECIMAL(6, 2)
8	h_data	VARCHAR2(24)	VARCHAR(24)

表 6.1.7: Item テーブル

No.	カラム名	Oracle Database データ型	PostgreSQL データ型
1	i_id	NUMBER(6, 0)	INTEGER
2	i_im_id	NUMBER(6, 0)	INTEGER
3	i_name	VARCHAR2(24)	VARCHAR(24)
4	i_price	NUMBER(5, 2)	DECIMAL(5, 2)
5	i_data	VARCHAR2(50)	VARCHAR(50)

表 6.1.8: stock テーブル

No.	カラム名	Oracle Database データ型	PostgreSQL データ型
1	s_i_id	NUMBER(6, 0)	INTEGER
2	s_w_id	NUMBER(6, 0)	INTEGER
3	s_quantity	NUMBER(4, 0)	DECIMAL(4, 0)
4	s_dist_01	CHAR(24)	CHAR(24)
5	s_dist_02	CHAR(24)	CHAR(24)
6	s_dist_03	CHAR(24)	CHAR(24)
7	s_dist_04	CHAR(24)	CHAR(24)
8	s_dist_05	CHAR(24)	CHAR(24)
9	s_dist_06	CHAR(24)	CHAR(24)
10	s_dist_07	CHAR(24)	CHAR(24)
11	s_dist_08	CHAR(24)	CHAR(24)
12	s_dist_09	CHAR(24)	CHAR(24)
13	s_dist_10	CHAR(24)	CHAR(24)
14	s_ytd	NUMBER(8, 0)	DECIMAL(8, 0)
15	s_order_cnt	NUMBER(4, 0)	DECIMAL(4, 0)
16	s_remote_cnt	NUMBER(4, 0)	DECIMAL(4, 0)
17	s_data	VARCHAR2(50)	VARCHAR(50)

表 6.1.9: orders テーブル

No.	カラム名	Oracle Database データ型	PostgreSQL データ型
1	o_id	NUMBER(8, 0)	INTEGER
2	o_d_id	NUMBER(2, 0)	INTEGER
3	o_w_id	NUMBER(6, 0)	INTEGER
4	o_c_id	NUMBER(5, 0)	INTEGER
5	o_entry_d	DATE	TIMESTAMP
6	o_carrier_id	NUMBER(2, 0)	INTEGER
7	o_ol_cnt	NUMBER(2, 0)	DECIMAL(2, 0)
8	o_all_local	NUMBER(1, 0)	DECIMAL(1, 0)

表 6.1.10: new_orders テーブル

No.	カラム名	Oracle Database データ型	PostgreSQL データ型
1	no_o_id	NUMBER(8, 0)	INTEGER
2	no_d_id	NUMBER(2, 0)	INTEGER
3	no_w_id	NUMBER(6, 0)	INTEGER

表 6.1.11: order_line テーブル

No.	カラム名	Oracle Database データ型	PostgreSQL データ型
1	ol_o_id	NUMBER(8, 0)	INTEGER
2	ol_d_id	NUMBER(2, 0)	INTEGER
3	ol_w_id	NUMBER(6, 0)	INTEGER
4	ol_number	NUMBER(2, 0)	INTEGER
5	ol_i_id	NUMBER(6, 0)	INTEGER
6	ol_supply_w_id	NUMBER(6, 0)	INTEGER
7	ol_delivery_d	DATE	TIMESTAMP
8	ol_quantity	NUMBER(2, 0)	DECIMAL(2, 0)
9	ol_amount	NUMBER(6, 2)	DECIMAL(6, 2)
10	ol_dist_info	CHAR(24)	CHAR(24)

上記の 9 つのテーブルを JdbcRunner のイニシャライズスクリプト（tpcc_load.js）を用いて登録しています。登録する際に利用する scale_factor（ベンチマーク計測時のデータ量生成因子）を今回の試行では値を”4”として利用しました。各テーブルのレコード件数とテーブルサイズは下表のようになります。

表 6.1.12: テーブルのレコード件数（OracleDatabase）

No.	テーブル名	レコード件数
1	warehouse	4
2	district	40
3	customer	120,000
4	history	120,000
5	item	100,000
6	stock	400,000
7	orders	120,000
8	new_orders	36,000
9	order_line	1,199,845

6.1.5. データ移行の成功判定

本試行では、JdbcRunner のベンチマーク走行が可能であった場合に、データ移行が完了したと判定しています。

実行例: JdbcRunner の実行

```
$ export CLASSPATH=Jdbcrunner を展開したディレクトリ/jdbcrunner-1.2.jar
$ java JR jdbcrunnerを展開したディレクトリ/scripts/tpcc.js
```

6.2. データ抽出時の結果

6.2.1. SPOOL コマンド/Ora2Pg による出力時間

本試行では、SPOOL コマンドによる CSV 出力と、Ora2Pg による COPY 文形式の出力を行いました。その際に、time コマンドによる計測を実施しています。結果は次の通りです。

表 6.2.1: 抽出方法と抽出時間

No.	抽出方法	抽出時間
1	SPOOL（スクリプトによる抽出）	42.368 秒
2	Ora2Pg（TYPE COPY による抽出）	2 分 29.831 秒

6.3. データ投入時の結果

6.3.1. COPY コマンド/Ora2Pg 出力の投入（psql）/pg_bulkload による投入総時間

本試行では、SPOOL コマンドによる CSV 出力と、Ora2Pg による COPY 文形式の出力結果の投入（いずれも COPY）と、pg_bulkload による投入方法の 3 つの投入手法について試行しました。

表 6.3.1: 投入方法と投入時間

No.	投入方法	投入時間
1	COPY（SPOOL 出力）	20.560 秒
2	psql -f（Ora2Pg 出力）	13.410 秒
3	pg_bulkload（SPOOL 出力）	15.120 秒

pg_bulkload については、テーブル単位での実行となり、各テーブルごとに制御ファイルを用意する必要があるため、小規模なテーブルについてはロード前の準備時間とロード時間を合算すると COPY の方が短時間で実行できると考えられます。

6.3.2. COPY コマンドと pg_bulkload のデータ投入の結果詳細

本試行では、移行対象テーブル毎に COPY コマンド、pg_bulkload によるデータ投入を行いました。また、その際に、time コマンドから処理にかかる所要時間を計測しています。結果は次に示す通りです。今回の検証では pg_bulkload の方が COPY コマンドよりも約 1.5 倍処理速度が速いことが確認できました。

表 6.3.2: データ投入所要時間

No.	テーブル名	レコード件数	COPY コマンド	pg_bulkload
1	warehouse	4	0.011 秒	0.157 秒
2	district	40	0.012 秒	0.103 秒
3	customer	120,000	3.49 秒	1.954 秒
4	history	120,000	0.601 秒	0.581 秒
5	item	100,000	0.623 秒	0.596 秒
6	stock	400,000	7.502 秒	3.853 秒
7	orders	120,000	0.601 秒	0.540 秒
8	new_orders	36,000	0.07 秒	0.158 秒
9	order_line	1,199,845	7.65 秒	5.468 秒
-	実行時間総計	-	**20.56 秒**	**13.41 秒**

6.3.3. COPY コマンドと pg_bulkload による NULL 置換について

移行対象「order_line」テーブルには、一部 NULL 値の DATE 情報「ol_delivery」が存在しています。通常の Oracle Database の SPOOL コマンドでの出力結果である空文字列は COPY コマンドにおいて文字リテラルと判断され、投入することができません。

本試行では、DATE 型の NULL 値を、SPOOL コマンド実行時に「1900/01/01」として出力するように明示し、投入時に COPY コマンドの NULL 値置換オプションに「1900/01/01」を指定して、NULL として投入を行いました。しかし、抽出時に、「1900/01/01」を「"」で囲んだ場合、pg_bulkload では「1900/01/01」を NULL 文字に置換することができませんでした。pg_bulkload の開発チームに確認したところ、「"」で囲んだ文字列を NULL 置換するには、クォート文字を「"」以外に設定しなければならないとの回答を得ました。

本ドキュメントでは、Oracle Database から抽出する際に文字列を、「"」で囲んでいるため、クォート文字を変更することは好ましくありません。そのため、「ol_delivery」を「"」付与せずに抽出する方法を試行しました。結果は下表のとおりです。

SPOOL コマンドでは「"」を省略した際に、下表の No.2 のように空白が入ってしまいますが、No.3 のように空白を削除することも可能です。よって、No.3 の手法が SPOOL 抽出、pg_bulkload 投入においては最も望ましい方法と考えられます。

表 6.3.3: データ投入時の NULL 値置換設定（アンダーバー「_」は実際には空白）

No.	SPOOL による NULL 抽出時のフォーマット	COPY コマンドの設定	pg_bulkload の設定
1	,"1900/01/01",	NULL '1900/01/01 '	QUOTE=~ NULL=""1900/01/01""
2	,1900/01/01____,	NULL '1900/01/01_____'	NULL="1900/01/01_____"
3	,1900/01/01,	NULL '1900/01/01'	NULL="1900/01/01"

実行例:COPY コマンドによる NULL 文字置換設定（アンダーバー「_」は実際には空白）

```
=# copy scott.order_line from '/tmp/order_line.csv' with (NULL '1900/01/01_____',format CSV);
```

設定抜粋: pg_bulkllload の制御ファイルの NULL 文字置換設定（アンダーバー「_」は実際には空白）

```
NULL="1900/01/01_____"
```

6.4. 移行後の作業

データ移行後の作業として以下を行い、データ投入が完了していることを確認しました。

6.4.1. データ件数の確認

COUNT(*)にてデータ投入後の各テーブルの行数が、移行元データの行数と一致しているか確認しました。

表 6.4.1: データ投入後のテーブルのレコード件数（PostgreSQL）

No.	テーブル名	レコード件数	移行元データと同じ件数か
1	warehouse	4	○
2	district	40	○
3	customer	120,000	○
4	history	120,000	○
5	item	100,000	○
6	stock	400,000	○
7	orders	120,000	○
8	new_orders	36,000	○
9	order_line	1,199,845	○

6.4.2. 制約・索引の作成

PostgreSQL へデータを投入する際に、索引、制約が定義されていると、データの挿入速度が劣化してしまいます。また、制約条件（外部キー等）が定義されていると、単純なデータの投入でも失敗する可能性があります。そのため、データの投入が完了した後、制約および索引を作成しました。

データ投入後、問題なく制約・索引を定義することができました。

表 6.4.2: 制約・索引の定義に要した時間

No.	プライマリキー・索引の定義に要した時間
1	13.548 秒

移行元データで各テーブルに定義されているインデックスや制約を作成し、「¥di 」、「¥d+ テーブル名」の結果から移行元データと相違がないことを確認しました。

表 6.4.3: テーブルに定義されている索引・制約の確認

No.	テーブル名	索引・制約	結果
1	warehouse	warehouse_pk	○
2	district	district_pk	○
3	district	district_fk1	○
4	customer	customer_pk	○
5	customer	customer_fk1	○
6	history	item_pk	○
7	history	history_fk1	○
8	history	history_fk2	○
9	item	stock_pk	○
10	stock	new_orders_pk	○
11	stock	stock_fk1	○
12	stock	stock_fk2	○
13	orders	order_line_pk	○
14	orders	orders_fk1	○
15	new_orders	customer_ix1	○
16	new_orders	new_orders_fk1	○
17	order_line	orders_ix1	○
18	order_line	order_line_fk1	○
19	order_line	order_line_fk2	○

6.4.3. VACUUM および ANALYZE の実行

VACUUM FULL と ANALYZE を実行しました。処理にかかる時間は約 15.2 秒でした。

表 6.4.4: VACUUM FULL および ANALYZE に要した時間

No.	VACUUM FULL および ANALYZE に要した時間
1	15.2 秒

6.4.4. データ移行後のテーブルサイズ

テーブルサイズは次のようになりました。

表 6.4.5: 移行後のテーブルのレコード件数とサイズ

No.	テーブル名	レコード件数	PostgreSQL テーブルサイズ（KB）
1	warehouse	4	8
2	district	40	8
3	customer	120,000	74,153
4	history	120,000	10,616
5	item	100,000	10,641
6	stock	400,000	141,893
7	orders	120,000	8,192
8	new_orders	36,000	1,597
9	order_line	1,199,845	121,348

6.4.5. アプリケーションテスト

JdbcRunner を使用してアプリケーションテストを行い、データ移行後のデータベースが問題なく動作することを確認しました。

6.5. まとめ

今回は、Oracle Database および PostgreSQL の両者に対応しているアプリケーションを用いたため、容易に移行を完了することができたと考えられます。しかし、Oracle Database および PostgreSQL に関して両ソフトウェアへの知見が必要である箇所があり、移行対象のツールや、各データ型の取り扱いについて横断的に理解する必要がありました。

また、データ投入の結果から、COPY コマンドより pg_bulkload の方が処理速度が速いことが確認できました。大規模データを投入する際には、更に処理速度の違いが顕著になると考えられます。しかし、pg_bulkload の場合、テーブル単位でのデータ投入しかできず、テーブル単位で制御ファイルを作成する必要があるため、テーブル数が多い場合には、全てのデータを pg_bulkload で移行するのは現実的ではありません。そのような場合、大規模テーブルは pg_bulkload、その他は COPY コマンドで移行するなど、ツールを使い分けることで効率が上がると思われます。

7. 別紙一覧

・別紙 01: 付帯ツールのインストール手順
・別紙 02: 6 章移行手順
・別紙 03: 組み込みデータ型対応表（SQLServer-PostgreSQL）

おわりに

本資料では、異種 DBMS から PostgreSQL へのデータ移行について、手順の確立と注意点の列挙、および実際のデータ移行の試行結果について記載しました。異種 DBMS の要件によっては、本資料記載の手法をそのまま適用することが難しい場合もあると思いますが、これから PostgreSQL を実際に業務システムへ適用してみようという方への具体的な作業イメージの想起となれば幸いです。

また、下表に記載された課題について本資料では記載やフォローが不足しています。実際のデータ移行に本資料を利用して臨む場合には留意してください。

表 1: 本資料に記載していない課題

No.	課題名	概要・留意点
1	Oracle Database および MicrosoftSQLServer の CSV 出力時の文字形式	本資料では作成時間の都合上、検証しきれず全てのデータ型について記載していません。各 DBMS からデータを抽出する行う際に、COPY/pg_bulkload で PostgreSQL へ投入可能な形式にする必要があります。
2	Oracle Database および MicrosoftSQLServer の ユーザおよびロールと PostgreSQL ロールとのマッピング	本資料では作成時間の都合上、検証しきれず各ロール・権限について、意味のマッピングについて言及されていません。 PostgreSQL 操作ユーザの振る舞いを移行元のデータベースと同義にしたいが、仕様書等が存在しない場合は、機械的にマッピングできるように各 DBMS のロール・権限について調査し、定義する必要があります。
3	移行時の PostgreSQL の設定	移行は、運用時と異なり、PostgreSQL の設定を移行用に設定して速やかに投入が行えるよう努めるべきです。本資料は移行時のみ利用するパラメータ設定に関して触れていません。 チェックポイント・セグメントの設定や、WAL 出力の抑制等を行うことで、データ投入時間を短縮することできます。

著者

版	所属企業・団体名	部署名	氏名
データ移行調査および実践編 第 1 版 （2012 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
			永田 まゆみ
	株式会社 富士通ソーシアル サイエンスラボラトリ	公共ビジネス本部	青木 俊彦
			杉山 貴洋
			小山田 政紀