PostgreSQL エンタープライズ・コンソーシアム 技術部会 WG#2

試験編

異種 DBMS から PostgreSQL への移行

製作者
担当企業名
日本電信電話株式会社
NECソリューションイノベータ株式会社
富士通株式会社
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	2015/04/23	新規作成
1.1	2015/05/26	改訂日や誤字等、細かい点を修正

はじめに

■本資料の概要と目的

本資料では任意のアプリケーションが利用する DBMS を異種 DBMS から PostgreSQL に移行した際に実施する試験について記載します。そのため、本資料は異種 DBMS から PostgreSQL への移行が完了していることを前提に記載しています。異種 DBMS から PostgreSQL への移行手法に関しては、2012、2013 年度に公開した下表の WG2 文書(以降「過去の WG2 文書」と呼称)を参照して下さい。

本資料では過去の WG2 文書に記載した手法で移行したデータベースやアプリケーション等の移行結果の妥当性を確認する試験を記載しています。試験方法については下表の「本資料での試験方法記載箇所」を参照して下さい。

また、異種 DBMS を PostgreSQL に移行した際に必要になると考える一般的な試験について可能な限り記載しておりますが、本資料に記載できてない範囲があることに注意して下さい。

表 1: 過去の WG2 文書と本資料との対応

No.	WG2 文書名	WG2 文書の概要	本資料での試験方法記載箇所
1	DB 移行フレームワーク編	異種 DBMS からの移行とは具体的に何を行うのかを紹介します。 DBMS の移行作業において一般的に発生すると考えられる作業工程を定義し、各工程における検討結果をベースとして移行可否判断の手がかりとなる情報を提供します。	- (記載対象外)
2	システム構成調査編	DBMS の代表的なシステム構成とその特徴を挙げ、PostgreSQL 移行時に採用可能な構成を紹介します。	- (記載対象外)
3	異種 DB 間連携調査編	異種 DBMS で稼動する既存システムとの連携を想定し、異種 DBMS と PostgreSQL の連携について、実現方法や移行前後における機能差などを紹介します。	- (記載対象外)
4	スキーマ移行調査編	PostgreSQL へスキーマを移行する際に注意すべき点を調査し、異種 DBMS と PostgreSQL 間における DDL 仕様の相違点や書き換えが必要な DDL の変換方法を紹介します。	3 スキーマ移行結果確認試験
5	データ移行調査および実践編	異種 DBMS から PostgreSQL へデータの移行するために必要となるデータ抽出(Extract)、変換(Transform)、および PostgreSQL への書き出し(Load)を中心に紹介します。また、本文書には実際に DB 移行作業を実施したレポートが含まれます。 2013 年度の成果物として文字コード変換をともなうマルチバイトコードのデータ移行を追加しました。	4 データ移行結果試験
6	ストアドプロシージャ移行調査編	異種 DBMS と PostgreSQL のストアドプロシージャの仕様の差異から、PostgreSQL へストアドプロシージャを移行する際に注意すべき点について紹介します。 2013 年度の成果物として、Microsoft SQL Server、DB2 からのストアドプロシージャ移行に関する調査結果を追加しました。	- (本年度は記載を見送っています)
7	アプリケーション移行調査編	DB 移行の際、DBMS 接続用ドライバやエラーハンドリング、トランザクション制御方法の違いなどアプリケーション側で意識すべき内容について紹介します。	5 アプリケーション移行確認試験 下記に関する試験については、 記載していません 1. DBMS 接続用ドライバ 2. エラーハンドリング
8	アプリケーション移行実践編	Oracle を利用するオープンソースのソフトウェアをテーマとして PostgreSQL への DB マイグレーション作業を実際に行い、作業のポイントや移行作業の負荷などを紹介します。	- (記載対象外)
9	SQL 移行調査編	異種 DBMS と PostgreSQL が対応している SQL の	5 アプリケーション移行確認試験

		差異および、異種 DBMS から PostgreSQL への SQL 文の書き換え方針について、DML を中心に紹介します。	
10	組み込み関数移行調査編	PostgreSQL の関数の互換性を調査し、DBMS 毎の組み込み関数実装の有無や書き換えが必要な組み込み関数の変換方法を紹介します。	－ （本年度は記載を見送っています）
11	チューニング編	異種 RDBMS からの移行の際、移行元システムで定義されていた性能要件や性能関連の状態、チューニングをポイントを確認して PostgreSQL 構築に反映したり、PostgreSQL 移行時に注意が必要なポイントを紹介します。	6 性能試験
12	バージョンアップ編	PostgreSQL は毎年メジャーバージョンアップを行っており、新たに追加された機能を利用するには適切な手順でバージョンアップを行う必要があります。バージョンアップ編では、現在利用中の PostgreSQL をバージョンアップする際に使用するツールや操作方法をご紹介します。	－ （記載対象外）

■資料内の記述について

本資料で記載されている試験および手順などが読者の方々のプロジェクトおいて必要十分であることを保証することはできません。また、本資料には記載されていない試験も必要になると考えております。

しかし、多くの項目は各社の実機検証または実プロジェクトの経験に基づいて記載されていますので、お役に立てる情報が記載されていると考えています。

■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 2: 用語定義

No.	用語	意味
1	DBMS	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 が該当します。
3	Oracle	データベース管理システムの Oracle Database を指します。
4	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。下表の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 3: 動作環境

No.	環境名	実装	バージョン
1	検証対象 DBMS	PostgreSQL	9.4.0
2	移行元オペレーティングシステム	Linux	CentOS 6.5
3	移行先オペレーティングシステム	Linux	CentOS 6.5

表 4: 異種 DBMS 一覧

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

表 5: 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。
2	psql (ぴー・えす・きゅー・える)	9.4.0	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	Ora2Pg (おら・つー・ぴーじー)	13.0	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。
4	perl (ぱーる)	5.10.1 (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。
5	Compress::Zlib (こんぷれす・じーりぶ)	2.021 (CentOS RPM を利用)		http://search.cpan.org/~pmqs/IO-Compress-2.060/lib/Compress/Zlib.pm	zlib 圧縮のための perl モジュール。Ora2Pg 依存プログラム。
6	DBI (でーびーあい)	1.609 (CentOS RPM を利用)		http://search.cpan.org/~timb/DBI/DBI.pm	DBMS へ Perl プログラムが接続するためにドライバモジュール。
7	DBD::Oracle (でーびーでー・おらくる)	1.74		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。
8	DBD::Pg (でーびーでー・ぴーじー)	2.15.1-4 (CentOS RPM を利用)		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。
9	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。本資料ではファイルの文字コード変換等に用いる。
10	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、ファイルの文字コード変換等を行う際に利用する。
11	Selenium (せれにうむ)	2.31.0	Apache 2.0 license	https://github.com/SeleniumHQ/selenium	ブラウザに表示される要素を操作し、取得して想定されうる状態になっているかを試験するためのツール。本資料では、DBMS 移行後のアプリケーション試験ツールとして利用する。
12	infoScoop (いんふぉ・すくーぷ)	3.4.0	The BSD 3-Clause License	https://www.infoscoop.org/ja/	柔軟、軽快、直感的に使えるフリーの企業情報ポータル(EIP). 1 万人以上の環境での稼働実績を持つ企業情報ポータル。 本資料では、移行対象アプリ

					ケーションとして本アプリケーションを利用する。

目次

1. 試験概要

本書に記載する試験を下表に記載します。詳細は各章を参照して下さい。

表 1.1: 試験概要

No.	試験	試験対象	確認観点	本資料での記載章
1	スキーマ移行結果 確認試験	・ スキーマ ・ テーブル ・ カラム ・ インデックス ・ ビュー ・ シリアルオブジェクト	・オブジェクトの名前 ・カラム属性 ・制約条件等	3 スキーマ移行結果確認 試験
2	データ移行結果 確認試験	・ データ	・ テーブル数 ・ データ件数 ・ データサイズ ・ データ内容 ・ 外字登録	4 データ移行結果試験
3	アプリケーション 移行結果確認試験	・ SQL 文 ・ 業務バッチ ・ 画面 ・ OR マッパーを利用してい る場合の DB 依存定義	・SQL 文の実行結果の妥当性 ・業務バッチの実行結果の妥当性 ・画面操作・遷移の妥当性	5 アプリケーション移行確 認試験
4	性能試験	・ アプリケーションの性能	・DBMS 移行前の性能要件を満た すか否か	6 性能試験 詳細は「チューニング編」 を参照

2. 試験中の設定

試験時には、OS やアプリケーションサーバ、PostgreSQL 等のログ情報が重要になります。
参考に試験時に必要になるログ取得設定方法を一部を下表に記載します。
試験用のログ設定は、性能に影響を及ぼす可能性があるため、運用開始時には設計した設定値に戻すことを忘れないようにしてください。

表 2.1: 試験中の設定

No.	カテゴリ	参考設定	概要
1	PostgreSQL 設定	log_statement = all	本設定を行うことで、PostgreSQL に対して実行された全ての SQL がログへ出力されます。 このまま運用しますと、性能劣化やディスク領域のひっ迫を引き起こす可能性がありますので注意してください。
2		log_line_prefix = '[%m][%d][%h][%u][%e][%p] '	ログの接頭情報を付与します。参考設定値として挙げている文字列には、以下の情報が補完されます。 %m：時刻（ミリ秒） %d：データベース名 %h：ホスト名 %u：ユーザ名 %e：SQL の状態（戻り値については以下を参照してください） http://www.postgresql.jp/document/current/html/errcodes-appendix.html %p：PID
3	操作ログ	–	リモートから SSH クライアントを利用して操作を行う場合は、当該ソフトウェアの機能を利用してコマンドの履歴を保存するようにしてください。サーバ内の端末で直接操作する場合は、script コマンドで代替が可能です。
4	OS 情報（性能、ネットワークなど）	sar, netstat など	基本的に性能試験時のみ取得しますが、アプリケーションとの結合試験時などは常時取得しておくと問題発生時への対応が迅速に行えます。 仮想環境上で PostgreSQL を動作させる場合は、仮想インスタンス内のコマンド（sar など）ではなく、仮想ホスト（VMware ESX など）の性能情報表示機能を利用してください。仮想インスタンス側では正確な情報が取得できない可能性があります。

3. スキーマ移行結果確認試験

本章では、スキーマのオブジェクト（以降スキーマオブジェクトと呼称）移行に際して行う単体試験手順を記載します。

3.1. 目的

定義の移行作業を過去の WG2 文書「スキーマ移行調査編」に従い実施した後、スキーマ定義移行が正常に完了していることを確認します。

3.2. 前提条件

本章で紹介する試験は、スキーマ移行作業が完了し、下表の状態であることを前提としています。

表 3.1: 前提条件

No.	前提条件	理由
1	移行の対象とするスキーマオブジェクトの選定	試験に際し、移行が必要なスキーマオブジェクトが選定されている必要があります。異種 DBMS 上に一時的に定義した不要な表やインデックスなどが存在する場合、これらの移行および評価作業は不要な作業となるため、移行の必要性を確認し対象を絞り込む必要があります。
2	DDL の変換作業（PostgreSQL への定義が正常に完了）	単体試験は、スキーマ定義の移行作業中に発生するエラーその他の問題は解消した上で、変換作業の正常性を確認します。
3	PostgreSQL の未対応オブジェクトの移行方針決定	PostgreSQL では非対応のインデックスなど存在しない機能に関しては、移行前後の比較を行うことができません。差分として抽出された結果が正しいかどうかを個別に確認するため、移行方針が決定している必要があります。
4	DBMS の仕様の差異による変更方針決定	文字列長の指定（バイト数または桁数）など PostgreSQL と異種 RDBMS 間で仕様が異なる機能については、移行前後の比較を行うことができません。差分として抽出された結果が正しいかどうかを個別に確認するため、移行方針が決定している必要があります。
5	未対応、仕様差異による変更対象オブジェクトの選定	差分の確認を行うには、PostgreSQL が未対応の機能や仕様の際があるオブジェクトが明確になっている必要があります。
6	移行時にシステムの改造やそれに伴うスキーマの変更が明確になっている	RDBMS の移行と合わせて仕様を変更している場合、単体試験では差異として現れるため、どのオブジェクトがどのように変更されているのかを明確にしておく必要があります。

3.3. 試験対象と観点

定義移行の単体試験では、異種 DBMS および PostgreSQL のシステムカタログに登録されたスキーマオブジェクトを採取し、移行前後の定義が同義である、もしくは仕様変更した箇所が正しく変更されていることを確認します。

本章では下表のスキーマオブジェクトの変換結果に関する単体試験方法を紹介します。

表 3.2: 試験観点

No.	オブジェクト	試験観点
1	スキーマ	同名のスキーマが定義されていること。
2	テーブル	同名のスキーマに同名のテーブルが定義されていること。
3	カラム	同名のスキーマ、テーブルに同名のカラムが定義されていること。 カラムの属性が移行仕様通りの属性で定義されていること。
4	インデックス	同名のスキーマ、インデックスが定義されていること。 インデックスが同じスキーマ、テーブルに対して定義されていること。
5	ビュー	同名のスキーマ、ビューが、同名のスキーマ、テーブルに対して定義されていること。
6	シリアルオブジェクト	同名のスキーマ、シリアルオブジェクトが定義されていること。 初期値、加算の単位、上限到達後の動作が同じであること。
7	制約条件	各種制約条件が定義されていること。 制約条件名で移行を確認しますが、NOT NULL や プライマリキー制約を除き、ロジックを含む定義自体は比較対象とはしません。

本章では、以下の項目については、スキーマ移行の単体評価範囲外とします。

表 3.3: 評価対象外オブジェクト

No.	オブジェクト	除外理由
1	関数	関数定義やストアドプロシージャ定義はロジックを含み DBMS 間の差異が大きいため、定型的な移行ができないオブェクトです。移行結果確認の方法が大きく異なります。ロジックの移行評価方法は別途検討するものとし、本章の説明対象外とします。
2	ストアドプロシージャ	
3	パッケージ	Oracle のパッケージ定義自体はスキーマ定義に移行しますが、パッケージ定義に含まれる変数定義等は移行評価の対象外とします。これは PostgreSQL には存在しない機能であり、個別に実装、評価する必要があります。
4	パーティション	パーティション定義は不完全ながら継承、トリガに移行可能です。しかし、データを振り分けるトリガ関数はロジックを含み、定義の有無や簡単な確認で試験を行うことができないため、本章の説明対象外とします。
5	DB利用者	利用者、利用権の定義は DBMS ごとに考え方が異なるため、本章の説明対象外とします。
6	利用権	
7	その他定義	ユーザ定義属性など、DBMS に新規に追加したオブジェクトは個別に移行方法を検討する必要があり、本章の説明対象外とします。

3.4. 試験手順

試験は以下の手順で実施します。

① 異種 DBMS からスキーマオブジェクト抽出
② 移行先 DB（PostgreSQL）からスキーマオブジェクト抽出
③ 定義情報の差分抽出
④ 定義差異と移行時の変更仕様の突合せ

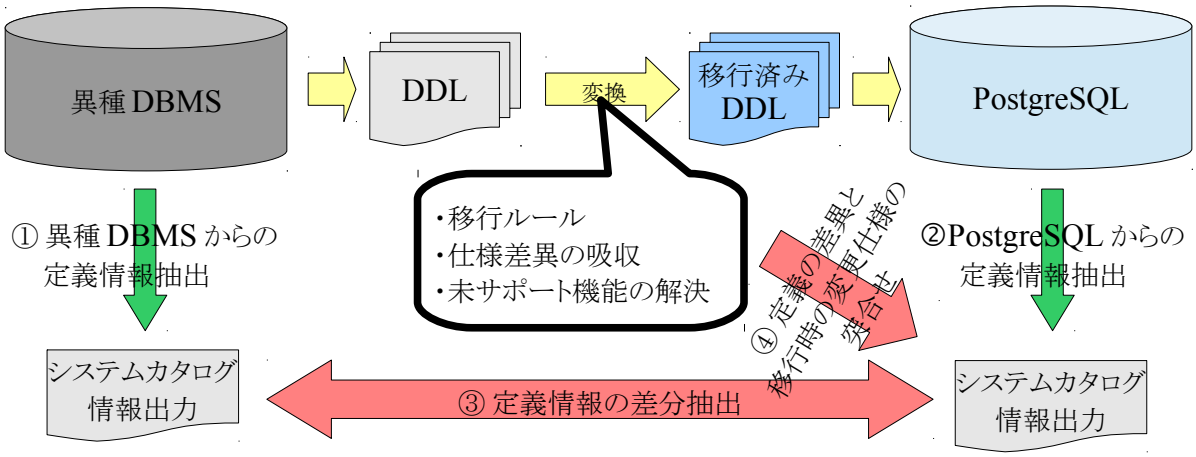


図 3.1: 試験概要図

表 3.4: 試験手順詳細

No.	試験手順	作業内容
①	異種 DBMS からスキーマオブジェクト抽出	移行前後の DB の仕様に合わせて定義文は変更されます。 比較を行うため、定義情報の抽出では定義文を DB からダンプするのではなく、カタログ情報から定義の名称や属性、桁数などの定義情報を取得します。
②	移行先 DB（PostgreSQL）からスキーマオブジェクト抽出	
③	定義情報の差分抽出	抽出した定義情報を比較します。差分が発生している（定義上の差異がある）ことに対し、移行仕様との突合せを行います。
④	定義の差異と移行時の変更仕様の突合せ	機能差異に対する変更方針との突合せを行います。

定義の差異は、移行方針に従った変更（システムの改造によるスキーマ変更を含む）か、移行時のミスによる問題の何れかと考えられます。スキーマ移行の単体評価においては、仕様に無い定義の変更や移行漏れを「不具合」とします。差分の抽出を行った時点で、定義の状態は以下のパターンに分けられます。

表 3.5: 試験結果と定義移行結果の判定

No.	試験結果	状態	判定
1	定義が一致	移行ルールに従い、定義移行が正しく行われた	正常
2	定義が不一致	PostgreSQL の定義が移行前の定義を包含	正常
		未サポート機能や DB の仕様差異による修正を行った	正常
		移行前後の定義の不一致	不具合

3.5. 異種 DBMS からの定義情報の抽出

1. Oracle からの定義情報の抽出
   Oracle のカタログ情報から、移行前後に比較を行うための定義情報を抽出します。
   定義情報の抽出に使用するカタログテーブルと定義情報収集用のクエリを例示します。

表 3.6: 定義情報の抽出（Oracle）

No.	カタログテーブル名	抽出情報（概要）	抽出情報（詳細）	抽出方法例
1	ALL_TABLES	テーブル情報の収集	オーナ名 テーブル名	SELECT OWNER,TABLE_NAME FROM ALL_TABLES WHERE OWNER='オーナ名'
2	ALL_VIEWS	ビュー情報の収集	オーナ名 ビュー名	SELECT OWNER,VIEW_NAME FROM ALL_VIEWS WHERE OWNER='オーナ名'
3	ALL_TAB_COLUMNS	列情報の収集	オーナ名 テーブル名 カラム名 データ型 バイト長 精度 NULL制約 デフォルト値	SELECT OWNER,TABLE_NAME,COLUMN_NAME,DATA_TYPE,DATA_LENGTH,DATA_PRECISION,DATA_SCALE,NULLABLE,DATA_DEFAULT FROM ALL_TAB_COLUMNS WHERE OWNER='オーナ名'
4	ALL_CONSTRAINTS	制約定義	オーナ名 テーブル名 制約定義名 制約定義のタイプ	SELECT OWNER,TABLE_NAME,CONSTRAINT_NAME,CONSTRAINT_TYPE FROM ALL_CONSTRAINTS WHERE OWNER='オーナ名'
5	ALL_INDEXES	インデックス定義	テーブルオーナ名 テーブル名 インデックスオーナ名 インデックス名 インデックスのタイプ 空き領域の割合	SELECT TABLE_OWNER,TABLE_NAME,OWNER,INDEX_NAME,INDEX_TYPE,PCT_FREE FROM ALL_INDEXES WHERE OWNER='オーナ名'
6	ALL_IND_COLUMNS	インデックスの列情報	インデックスオーナ名 インデックス名 列名 データの並び順	SELECT INDEX_OWNER,INDEX_NAME,COLUMN_NAME,DESCEND FROM ALL_IND_COLUMNS WHERE OWNER='オーナ名'
7	ALL_SOURCE	プロシージャ,ファンクション	オブジェクト名	SELECT OWNER,NAME FROM ALL_SOURCE WHERE TYPE IN ('PROCEDURE','FUNCTION') AND OWNER='オーナ名'
8	ALL_SEQUENCES	シーケンス	シーケンスのオーナ名 シーケンス名 最小値（初期値） 最大値 増分 最大値到達後の動作	SELECT SEQUENCE_OWNER,SEQUENCE_NAME,MIN_VALUE,MAX_VALUE,INCREMENT_BY,CYCLE_FLAG FROM ALL_SEQUENCES WHERE SEQUENCE_OWNER='オーナ名'

2. Microsoft SQL Server からの定義情報の抽出

Microsoft SQL Server の information schema から、移行前後に比較を行うための定義情報を抽出します。定義情報の抽出に使用するシステムカタログは、以下のとおりです。

表 3.7: 定義情報の抽出（SQL Server）

No.	システムカタログ名	抽出情報（概要）	抽出情報（詳細）	抽出方法例
1	INFORMATION_SCHEMA.TABLES	テーブル情報の収集	スキーマ名 テーブル名	SELECT TABLE_SCHEMA,TABLE_NAME FROM INFORMATION_SCHEMA WHERE TABLE_TYPE = 'BASE TABLE'
2	INFORMATION_SCHEMA.COLUMNS	ビュー情報の収集	スキーマ名 ビュー名	SELECT TABLE_SCHEMA,TABLE_NAME FROM INFORMATION_SCHEMA WHERE TABLE_TYPE = 'VIEW'
3	INFORMATION_SCHEMA.COLUMNS	列情報の収集	スキーマ名 テーブル名 カラム名 データ型 バイト長 精度 NULL制約 デフォルト値	SELECT TABLE_SCHEMA,TABLE_NAME,COLUMN_NAME,DATA_TYPE,CHARACTER_OCTET_LENGTH,NUMERIC_PRECISION,NUMERIC_SCALE,IS_NULLABLE,COLUMN_DEFAULT FROM INFORMATION_SCEMA.COLUMNS
4	INFORMATION_SCHEMA.TABLE_CONSTRAINTS	制約定義	スキーマ名 テーブル名 制約定義名 タイプ (CHECK,UNIQUE,PRIMARY KEY,FOREGIN KEY)	SELECT TABLE_SCHEMA,TABLE_NAME,CONSTRAINT_NAME,CONSTRAINT_TYPE FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
5	INFORMATION_SCHEMA.KEY_COLUMN_USAGE	インデックス定義	制約のスキーマ名 制約名 スキーマ名 テーブル名 インデックス名 （インデックスのタイプが収集できない）	SELECT CONSTRAINT_SCHEMA,CONSTRAINT_NAME,TABLE_SCHEMA,TABLE_NAME,CONSTRAINT_NAME FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
6	INFORMATION_SCHEMA.KEY_COLUMN_USAGE	インデックスの列情報	スキーマ名 テーブル インデックス名 列名 （データの並び順が収集できない）	SELECT TABLE_SCHEMA,TABLE_NAME,CONSTRAINT_NAME,COLUMN_NAME,ORDINAL_POSITION FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
7	INFORMATION_SCHEMA.ROUTINES	プロシージャ、ファンクション、 （トリガ）	オブジェクト名	SELECT ROUTINE_SCHEMA,ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES WHERE ROUTINE_TYPE = 'PROCEDURE' SELECT ROUTINE_SCHEMA,ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES WHERE ROUTINE_TYPE = 'FUNCTION'
8	sys.sequences	シーケンス	シーケンス名 スキーマ ID 最小値（初期値） 最大値 増分 最大値到達後の動作	SELECT NAME,SCHEMA_ID,START_VALUE,MAXIMUM_VALUE,INCREMENT,IS_CYCLING FROM sys.sequences

3.6. 移行先 DBMS（PostgreSQL）からの定義情報抽出

PostgreSQL の情報スキーマから、移行前後に比較を行うための定義情報を抽出します。
定義情報の抽出に使用するシステムカタログは、以下のとおりです。

表 3.8: 定義情報の抽出（PostgreSQL）

No.	システムカタログ名	抽出情報（概要）	抽出情報（詳細）	抽出方法例
1	INFORMATION_SCHEMA.TABLES	テーブル情報の収集	スキーマ名 テーブル名	SELECT TABLE_SCHEMA,TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_SCHEMA NOT IN ('information_schema','pg_catalog')
2	INFORMATION_SCHEMA.TABLES	ビュー情報の収集	スキーマ名 ビュー名	SELECT TABLE_SCHEMA,TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'VIEW' AND TABLE_SCHEMA NOT IN ('information_schema','pg_catalog')
3	INFORMATION_SCHEMA.COLUMNS	列情報の収集	スキーマ名 テーブル名 カラム名 データ型 バイト長 精度 NULL制約 デフォルト値	SELECT TABLE_SCHEMA,TABLE_NAME,COLUMN_NAME,DATA_TYPE,CHARACTER_OCTET_LENGTH,NUMERIC_PRECISION,NUMERIC_SCALE,IS_NULLABLE,COLUMN_DEFAULT FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA NOT IN ('information_schema','pg_catalog')
4	INFORMATION_SCHEMA.TABLE_CONSTRAINTS	制約定義 インデックス定義	スキーマ名 テーブル名 制約定義名 制約定義のタイプ（CHECK,UNIQUE,PRIMARY KEY,FOREGIN KEY）	SELECT TABLE_SCHEMA,TABLE_NAME,CONSTRAINT_NAME,CONSTRAINT_TYPE FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS WHERE TABLE_SCHEMA NOT IN ('information_schema','pg_catalog')
5	INFORMATION_SCHEMA.KEY_COLUMN_USAGE	インデックスの列情報	スキーマ名 テーブル名 制約のスキーマ名 制約名 列名 （列の並び順が取得できない）	SELECT TABLE_SCHEMA,TABLE_NAME,CONSTRAINT_SCHEMA,CONSTRAINT_NAME,COLUMN_NAME,ORDINAL_POSITION FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE TABLE_SCHEMA NOT IN ('information_schema','pg_catalog')
6	INFORMATION_SCHEMA.ROUTINES	プロシージャ ファンクション	オブジェクト名	SELECT ROUTINE_SCHEMA,ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES WHERE ROUTINE_TYPE IN ('PROCEDURE','FUNCTION') AND TABLE_SCHEMA NOT IN ('information_schema','pg_catalog')
7	INFORMATION_SCHEMA.SEQUENCES	シーケンス	シーケンス名 最小値 最大値 増分 最大値到達後の動作	SELECT SEQUENCE_SCHEMA,SEQUENCE_NAME,MINIMUM_VALUE,MAXIMUM_VALUE,INCREMENT,CYCLE_OPTION FROM INFORMATION_SCHEMA.SEQUENCES

3.7. 定義情報の差分抽出

前項で抽出した定義情報をファイル出力し、定義の差分情報を抽出します。

Oracle からのテーブルの定義情報出力実行例を示します。以下の例では、テーブルのオーナは"INFOSCOOP"です。

```
# cat get_tableinfo.sql
SPOOL OFF
SET COLSEP ,
SET HEAD OFF
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0
SET LINESIZE 1000
SET TRIMSPOOL ON
SPOOL oracle_tables.csv
SELECT OWNER,TABLE_NAME FROM ALL_TABLES WHERE OWNER='INFOSCOOP' ;
SPOOL OFF
# sqlplus -s infoscoop/infoscoop
@get_tableinfo.sql
現在はスプールしていません。
INFOSCOOP                      ,IS_SESSIONS
INFOSCOOP                      ,IS_PREFERENCES
INFOSCOOP                      ,IS_TABS
INFOSCOOP                      ,IS_WIDGETS
INFOSCOOP                      ,IS_USERPREFS
INFOSCOOP                      ,IS_CACHES
INFOSCOOP                      ,IS_RSSCACHES
INFOSCOOP                      ,IS_MENUCACHES
INFOSCOOP                      ,IS_LOGS
INFOSCOOP                      ,IS_WIDGETCONFS
INFOSCOOP                      ,IS_KEYWORDS
INFOSCOOP                      ,IS_MENUS
INFOSCOOP                      ,IS_MENUS_TEMP
INFOSCOOP                      ,IS_SEARCHENGINES
INFOSCOOP                      ,IS_GADGETS
INFOSCOOP                      ,IS_GADGET_ICONS
INFOSCOOP                      ,IS_PROPERTIES
INFOSCOOP                      ,IS_PROXYCONFS
INFOSCOOP                      ,IS_I18N
INFOSCOOP                      ,IS_I18NLASTMODIFIED
INFOSCOOP                      ,IS_I18NLOCALES
INFOSCOOP                      ,IS_TABLAYOUTS
INFOSCOOP                      ,IS_PORTALLAYOUTS
INFOSCOOP                      ,IS_ADMINROLES
INFOSCOOP                      ,IS_PORTALADMINS
INFOSCOOP                      ,IS_FORBIDDENURLS
INFOSCOOP                      ,IS_AUTHCREDENTIALS
INFOSCOOP                      ,IS_HOLIDAYS
INFOSCOOP                      ,IS_ACCESSLOGS
INFOSCOOP                      ,IS_MESSAGES
INFOSCOOP                      ,IS_SYSTEMMESSAGES
INFOSCOOP                      ,IS_ACCOUNTS
INFOSCOOP                      ,IS_OAUTH_CONSUMERS
INFOSCOOP                      ,IS_OAUTH_GADGET_URLS
INFOSCOOP                      ,IS_OAUTH_TOKENS
INFOSCOOP                      ,IS_OAUTH2_TOKENS
INFOSCOOP                      ,IS_OAUTH_CERTIFICATE
EXIT
```

PostgreSQL からの情報収集例は以下の通りです。

COPY文を使用し、情報スキーマの情報を出力しています。

```
infoscoop=# copy (SELECT TABLE_SCHEMA,TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE
TABLE' AND TABLE_SCHEMA NOT IN ('information_schema','pg_catalog')) to '/home/postgres/postgres_table¥
.csv' with csv ;
COPY 37
infoscoop-# ¥q
[postgres@rock02 ~]$ cat postgres_table.csv
public,is_caches
public,is_tablayouts
public,is_holidays
public,is_i18n
public,is_menus_temp
public,is_keywords
public,is_gadget_icons
public,is_searchengines
public,is_oauth_certificate
public,is_forbiddenurls
public,is_i18nlastmodified
public,is_accesslogs
public,is_menucaches
public,is_i18nlocales
public,is_properties
public,is_menus
public,is_oauth_gadget_urls
public,is_oauth2_tokens
public,is_widgets
public,is_systemmessages
public,is_preferences
public,is_accounts
public,is_messages
public,is_rsscaches
public,is_sessions
public,is_logs
public,is_userprefs
public,is_gadgets
public,is_widgetconfs
public,is_authcredentials
public,is_portallayouts
public,is_tabs
public,is_proxyconfs
public,is_oauth_tokens
public,is_portaladmins
public,is_oauth_consumers
public,is_adminroles
```

移行前後の DB から抽出したデータをテキストとして出力し比較することも可能ですが、各 DBMS のカタログ情報の差異や各種オブジェクト名の既定文字（大小文字）、出力形式の差異など、比較前にさまざまな加工を行わなければ、比較することができません。

ここでは、定義のメタ情報を比較するため、抽出した定義情報を DB に格納し、定義情報の形式変換および比較を行う方法を紹介します。定義情報の格納用テーブル定義を以下に例示します。比較するオブジェクトや定義情報の抽出用クエリに合わせて、テーブル定義を選択、変更してください。

表 3.9: テーブル名比較用テーブル

No.	cmp_table		
	列名	属性	説明
1	table_schema	text	スキーマ名
2	table_name	text	テーブル名

表 3.10: ビュー比較用テーブル

No.	cmp_view		
	列名	属性	説明
1	table_schema	text	スキーマ名
2	table_name	text	ビューの名前

表 3.11: カラム比較用テーブル

No.	cmp_column		
	列名	属性	説明
1	table_schema	text	スキーマ名
2	table_name	text	テーブル名
3	column_name	text	カラム名
4	data_type	text	型
5	character_octet_length	integer	列長
6	numeric_precision	integer	数値（numeric）の最大精度
7	numeric_scale	integer	数値（numeric）の位取り
8	is_nullable	boolean	NULL 値の格納可否
9	column_default	text	デフォルト値

表 3.12: インデックス比較用テーブル

No.	cmp_index		
	列名	属性	説明
1	table_schema	text	スキーマ名
2	table_name	text	テーブル名
3	constraint_schema	text	制約条件のスキーマ名
4	constraint_name	text	制約条件名

表 3.13: インデックスカラム比較用テーブル

No.	cmp_indexcol		
	列名	属性	説明
1	table_schema	text	スキーマ名
2	table_name	text	テーブル名
3	constraint_schema	text	制約条件のスキーマ名
4	constraint_name	text	制約条件名
5	column_name	text	列名
6	ordinal_position	integer	定義位置

表 3.14: プロシージャ比較テーブル

No.	cmp_proc		
	列名	属性	説明
1	routine_schema	text	ストアドプロシージャ、関数、トリガのスキーマ名
2	routine_name	text	ストアドプロシージャ、関数、トリガ名

表 3.15: シーケンス比較用テーブル

No.	cmp_trigger		
	列名	属性	説明
1	sequence_schema	text	シーケンスのスキーマ名
2	sequence_name	text	シーケンス名
3	minimum_value	bigint	最大値
4	maximum_value	bigint	最小値
5	increment	bigint	増分
6	cycle_option	boolean	最大値到達後の周回可否

3.8. 定義情報の差異確認

上記テーブルに移行前後の定義情報を格納し、差異を確認します。今回は前提として Ora2Pg が行う属性変換ルールに従うものとします。Ora2Pg の変換ルールは以下のとおりです。

表 3.16: 変換ルール

	Oracle			PostgreSQL
	type	scale	precision	type
文字列	CHAR			char
	NCAHR			char
	VARCHAR			varchar
	VARCHAR2			varchar
	NVARCHAR			char
	LONG			text
数値	SMALLINT			smallint
	INTEGER			integer
	BINARY_INTEGER			integer
	PLS_INTEGER			integer
	REAL			real
	DECIMAL			decimal
	DOUBLE PRECISION			double precision
	BINARY_FLOAT			double precision
	BINARY_DOUBLE			double precision
	NUMBER	0	5未満	SMALLINT
	NUMBER	0	9以下	INTEGER
	NUMBER	0	9より大きい	BIGINT
	NUMBER	0以外	6以下	real
	NUMBER	0以外	6より大きい	double
バイナリ	LONG RAW			bytea
	CLOB			text
	BLOB			bytea
	BFILE			bytea
	BOOLEAN			boolean
日時	DATE			timestamp
	TIMESTAMP WITH TIME ZONE			timestamp with time zone
	TIMESTAMP WITH LOCAL TIME ZONE			timestamp with time zone
	INTERVAL			interval
その他	XMLTYPE			xml

この変換ルールは一例であり、マイグレーションを行う際に決定した個々の変換ルールに従って比較を行ってください。

上記の変換ルールに従い変換結果確認を行うためのクエリの実行例を以下に示します。カタログ登録情報の差異を吸収するためにオブジェクト名を全て大文字に変換し、比較を行います。このため、大小文字による差異の確認ができない点には注意が必要です。

DB への定義情報の格納および比較方法を例示します。ここでは、抽出したテーブル一覧とカラム情報の比較を行っています。

定義情報格納用テーブルの作成

```
# psql -c "CREATE TABLE cmp_table_oracle (table_schema text,table_name text)" DB名
# psql -c "CREATE TABLE cmp_table_postgres (table_schema text,table_name text)" DB名
# psql -c "CREATE TABLE cmp_column_oracle_wk (table_schema text,table_name text,column_name
text,data_type text,character_octet_length integer,numeric_precision text,numeric_scale text,is_nullable
boolean,column_default text)" DB名
# psql -c "CREATE TABLE cmp_column_oracle (table_schema text,table_name text,column_name text,data_type
text,character_octet_length integer,numeric_precision integer,numeric_scale integer,is_nullable
boolean,column_default text)" DB名
# psql -c "CREATE TABLE cmp_column_postgres (table_schema text,table_name text,column_name text,data_type
text,character_octet_length integer,numeric_precision integer,numeric_scale integer,is_nullable
boolean,column_default text)" DB名
```

定義データのロード

Oracle の定義情報はワークに書き出し、加工(空白のトリム、数字から数値への変換)を行います。

```
# psql -c "COPY cmp_table_oracle FROM 'Oracleテーブル定義情報ファイル名' with csv" DB名
# psql -c "COPY cmp_table_postgres FROM 'PostgreSQLテーブル定義情報ファイル名' with csv" DB名
# psql -c "COPY cmp_column_oracle_wk FROM 'Oracleテーブル定義情報ファイル名' with csv" DB名
# psql -c "INSERT INTO cmp_column_oracle SELECT trim(both ' ' from table_schema),trim(both ' ' from
table_name),trim(both ' ' from column_name),trim(both ' ' from data_type), character_octet_length, case
when octet_length(trim(both ' ' from numeric_precision)) = 0 then 0 else (trim(both ' ' from
numeric_precision))::integer end,case when octet_length(trim(both ' ' from numeric_scale)) = 0 then 0 else
(trim(both ' ' from numeric_scale))::integer end,is_nullable,trim(both ' ' from column_default) from
cmp_column_oracle_wk" DB名
# psql -c "COPY cmp_table_postgres FROM 'PostgreSQLテーブル定義情報ファイル名' with csv" DB名
```

テーブル定義情報の比較
同一名のテーブルが双方に存在する場合は正常(OK)と判定しています。

```
$ psql db
psql (9.4.1)
Type "help" for help.

db=# select t1.table_name as oracle,t2.table_name as postgres,
db-#  case when t1.table_name is null then 'NG'
db-#       when t2.table_name is null then 'NG'
db-#       else 'OK'
db-#  end
db-#  from cmp_table_oracle t1 full outer join cmp_table_postgres t2
db-#       on (t1.table_name = upper(t2.table_name))
db-#  order by 1 ;
        oracle        |       postgres       | case
----------------------+----------------------+------
 IS_ACCESSLOGS        | is_accesslogs        | OK
 IS_ACCOUNTS          | is_accounts          | OK
 IS_ADMINROLES        | is_adminroles        | OK
 IS_AUTHCREDENTIALS   | is_authcredentials   | OK
 IS_CACHES            | is_caches            | OK
 IS_FORBIDDENURLS     | is_forbiddenurls     | OK
 IS_GADGETS           | is_gadgets           | OK
 IS_GADGET_ICONS      | is_gadget_icons      | OK
 IS_HOLIDAYS          | is_holidays          | OK
 IS_I18N              | is_i18n              | OK
 :
 IS_PROXYCONFS        | is_proxyconfs        | OK
 IS_RSSCACHES         | is_rsscaches         | OK
 IS_SEARCHENGINES     | is_searchengines     | OK
 IS_SESSIONS          | is_sessions          | OK
 IS_SYSTEMMESSAGES    | is_systemmessages    | OK
 IS_TABLAYOUTS        | is_tablayouts        | OK
 IS_TABS              | is_tabs              | OK
 IS_USERPREFS         | is_userprefs         | OK
 IS_WIDGETCONFS       | is_widgetconfs       | OK
 IS_WIDGETS           | is_widgets           | OK
(37 rows)
```

カラム定義情報の比較
同一名のカラムが双方に存在し、想定通りの属性になっている場合には正常(OK)としています。

```
$ psql db
psql (9.4.1)
Type "help" for help.

db=# select t1.table_name as oracle_tbl,t1.column_name as oracle_col,
db-#        t2.table_name as postgres_tbl,t2.column_name as postgres_col,
db-#  case when t1.column_name is null then 'NG'
db-#       when t2.column_name is null then 'NG'
db-#       when (t1.data_type = 'VARCHAR2' and t2.data_type = 'character varying' and
db(#             t1.character_octet_length = t2.character_octet_length) then 'OK'
db-#       when (t1.data_type = 'CLOB' and t2.data_type = 'text')       then 'OK'
db-#       when (t1.data_type = 'BLOB' and t2.data_type = 'bytea')      then 'OK'
db-#       when (t1.data_type = 'TIMESTAMP(6)' and
db(#             t2.data_type = 'timestamp without time zone')          then 'OK'
db-#       when (t1.data_type = 'NUMBER' and
db(#             t1.numeric_scale = 0 and t1.numeric_precision < 5 and
db(#             t2.data_type = 'smallint')                             then 'OK'
db-#       when (t1.data_type = 'NUMBER' and
db(#             t1.numeric_scale = 0 and t1.numeric_precision < 10 and
db(#             t2.data_type = 'integer')                              then 'OK'
db-#       when (t1.data_type = 'NUMBER' and
db(#             t1.numeric_scale = 0 and t1.numeric_precision > 9 and
db(#             t2.data_type = 'bigint')                               then 'OK'
db-#       when (t1.data_type = 'NUMBER' and
db(#             t1.character_octet_length = 22 and
db(#             t1.numeric_scale = 0 and t1.numeric_precision = 0 and
db(#             t2.data_type = 'bigint')                               then 'OK'
db-#       else 'NG'
db-#  end
db-#  from cmp_column_oracle t1 full outer join cmp_column_postgres t2
db-#       on (t1.table_name = upper(t2.table_name) and
db(#           t1.column_name = upper(t2.column_name))
db-#  order by 1,2 ;
     oracle_tbl      |      oracle_col      |     postgres_tbl     |     postgres_col     | case
---------------------+----------------------+----------------------+----------------------+------
 IS_ACCESSLOGS       | DATE                 | is_accesslogs        | date                 | OK
 IS_ACCESSLOGS       | ID                   | is_accesslogs        | id                   | OK
 IS_ACCESSLOGS       | UID                  | is_accesslogs        | uid                  | OK
  :
 IS_WIDGETS          | TYPE                 | is_widgets           | type                 | OK
 IS_WIDGETS          | UID                  | is_widgets           | uid                  | OK
 IS_WIDGETS          | WIDGETID             | is_widgets           | widgetid             | OK
(186 rows)
```

3.9. 定義差異と移行時の変更仕様の突合せ

定義に差異のあったオブジェクトに関して、移行時の変更仕様に合致するものであるかどうかを確認します。

マイグレーションの際に予定していた仕様の変更や、DBMS の機能差異を吸収するために実施した定義変更に該当しない定義上の差異は、本評価では不具合と判断します。

不具合には以下のケースが考えられます。不具合の内容に応じて対策を講じてください。

表 3.17: 不具合の内容と対処

No.	不具合の内容	状況および対処
1	仕様差異の反映ミス	移行に際し、設計時に想定していなかった DBMS の仕様差異対応の仕様書変更漏れや、変更仕様の適用時にミスが発生した可能性があります。 仕様書への反映もしくは適用ミスの修正を行います。
2	移行漏れ	移行作業中に、何らかのミスやエラーにより移行作業が完了していない可能性があります。 必要なオブジェクトを確認し、移行が必要な場合には定義の移行を行います。

3.10. 評価における注意

本章で提示した検証方法は、単純な構文の比較では移行結果の確認が難しい各種定義の移行結果比較を行うため、抽象化した定義情報に比較により移行結果の確認を行うものです。「3.3 試験対象と観点」に挙げたオブジェクトについては、個々に評価方法の検討を行ってください。

3.11. 移行検証

infoScoop 上での評価は別紙掲載。

4. データ移行結果試験

本章では、異種 DBMS から PostgreSQL へのデータ移行後に実施する試験について記載します。

4.1. 目的

過去の WG2 文書「データ移行調査および実践編」に従い実施した後、異種 DBMS から PostgreSQL へのデータ移行が正常に完了していることを確認します。

4.2. 前提条件

本章で紹介する試験は、下記を前提とします。

表 4.1: 前提条件

No.	前提条件	理由
1	異種 DBMS から PostgreSQL への DDL 変換・移行作業の完了	PostgreSQL へのスキーマ移行が完了していることが前提となります。
2	PostgreSQL へのデータ移行の完了	異種 DBMS と PostgreSQL のデータの比較作業を行うため、異種 DBMS から PostgreSQL へデータを移行し終えていることが前提となります。

4.3. 試験項目

データ移行結果試験として、下表の試験項目が考えられます。試験項目により難易度が大きく異なるので、必要な試験項目を選択し実施して下さい。以降に各試験項目の実施手順を記載します。

表 4.2: データ移行試験項目

No.	試験項目	概要	難易度
1	テーブル数の確認	異種 DBMS に定義されたテーブル数と、PostgreSQL に定義されたテーブル数が同一であることを確認します。本試験項目で試験対象とするテーブルを決定します。	低
2	データ件数の確認	異種 DBMS に格納されている各テーブルのデータ件数と、PostgreSQL に格納されている各テーブルのデータ件数が同一であることを確認します。	低
3	データサイズの確認	異種 DBMS に格納されたデータ件数およびデータの種別等から、PostgreSQL 移行後のデータサイズの想定値を算出します。PostgreSQL の実際のデータサイズと計算より算出した想定値が近似であることを確認します。	高
4	データ内容の確認	異種 DBMS に格納されたデータと PostgreSQL に格納されたデータが同一であることを確認します。本資料では、異種 DBMS から出力した CSV ファイルと、PostgreSQL から出力した CSV ファイルの内容が同一であることを確認します。	高
5	外字登録の確認	異種 DBMS に登録された外字が PostgreSQL にも正しく登録されていることを確認します。	低

4.4. テーブル数の確認試験

PostgreSQL および異種 DBMS に定義されたテーブル数の確認方法を記載しています。

下記手順で取得したテーブル数を比較し、異種 DBMS および PostgreSQL に定義されたテーブル数が同一であることを確認して下さい。またパーティショニングを利用していた場合、テーブル数が同一にならない可能性があります。
パーティショニングテーブルを利用していた時の注意点を下記に記載しています。

4.4.1. PostgreSQL

PostgreSQL のテーブル数は、稼働統計情報ビュー(pg_stat_user_tables)を参照することで、取得可能です。
ただし、下記 SQL で得られた結果にはパーティショニングの子テーブルの数も含まれるため、注意が必要です。

```
 =# SELECT COUNT(*) FROM pg_stat_user_tables;
 count
-------
    14
(1 行)
```

パーティショニングが存在する場合は異種 DBMS と PostgreSQL の仕様の違いに注意してください。PostgreSQL のパーティショニングは「図 4.1 PostgreSQL のパーティショニング」のように、親テーブルの定義を継承して子テーブルを作成していきます。親テーブルも子テーブルも実体があります。

テーブル数を取得する SQL によっては、異種 DBMS と差異が発生する可能性があります。

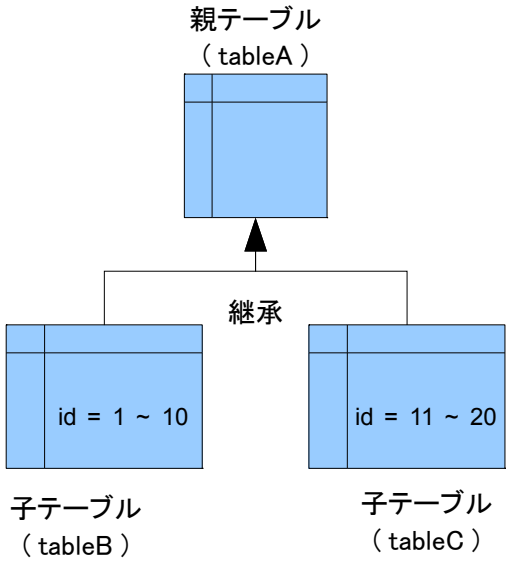


図 4.1: PostgreSQL のパーティショニング

上述の通り、PostgreSQL では tableA が実体として存在しますので、テーブル数を取得する方法によっては異種 DBMS よりも多く取得される場合があります。また、データ件数を取得する場合も注意が必要です。

tableA, tableB, tableC をテーブル一覧として取得し、それぞれのテーブルについて件数を取得した場合、tableA が 20 件(id=1～20)、tableB が 10 件(id=1～10)、tableC が 10 件(id=11～20)と、重複して取得されてしまいますので注意してください。

パーティショニングの子テーブルを除いたテーブル数は、下記 SQL で取得することが可能です。
ただし、下記 SQL で取得するテーブル数は、継承を利用していないテーブルの合計数であるため、パーティショニングの子テーブル以外で継承を利用している場合には、注意が必要です。

```
 =# SELECT COUNT(*) FROM pg_stat_user_tables WHERE relid not IN (SELECT inhrelid FROM pg_inherits);
 count
-------
     44
(1 row)
```

4.4.2. Oracle

Oracle のテーブル数は、下記 SQL で取得可能です。

```
SQL> SELECT COUNT(*) FROM user_tables;
  COUNT(*)
----------
        44
```

4.4.3. SQL Server

SQL Server のテーブル数は、下記 SQL で取得可能です。

```
> sqlcmd -S localhost¥SQLServer -d testdb -U sa -P hogehoge -Q
 "SELECT COUNT(*) FROM sysobjects WHERE xtype = 'U' "

-----------
         44

(1 行処理されました)
```

4.5. データ件数の確認試験

異種 DBMS に格納されているデータ件数と、PostgreSQL に格納されているデータ件数が同一であることを確認します。
データ件数の比較はテーブル毎に実施し、各テーブルのデータ件数は、COUNT(*)で取得します。
COUNT(*)は、PostgreSQL でも Oracle、SQL Server でも実行可能です。

テーブル名、データ件数の形式で結果を出力することで、各テーブルのデータ件数が比較しやすい形になります。
下記にテーブル名、データ件数の形式で結果を出力する方法を記載します。

4.5.1. PostgreSQL

テーブル名、データ件数は下記の処理で取得します。

① テーブル名、データ件数を取得する SQL の作成
② テーブル名、データ件数を取得する SQL の実行

① テーブル名、データ件数を取得する SQL の作成
下記 SQL を実行するとテーブル、データ件数を取得する SQL が得られます。

```
=#  WITH temp AS
(
 SELECT
  'SELECT '''||relname||''' AS table_name,count(*) AS count FROM '||relname as query,
  CASE WHEN lead(relname) over (ORDER BY relname) IS NOT NULL then ' UNION all' ELSE '' END AS
connect_query
 FROM
   pg_stat_user_tables
  )
SELECT
  query || connect_query AS table_count_query
FROM
  temp;
```

実行結果①
上記 SQL を実行すると下記のような結果が得られます。

```
                              table_count_query
-------------------------------------------------------------------------------
 SELECT 't1' AS table_name,count(*) AS count FROM t1 UNION all
 SELECT 't1_201301' AS table_name,count(*) AS count FROM t1_201301 UNION all
 SELECT 't2' AS table_name,count(*) AS count FROM t2 UNION all
 SELECT 'test' AS table_name,count(*) AS count FROM test
(4 rows)
```

② テーブル名、データ件数を取得する SQL の実行
実行結果①で得た SQL を実行することで、テーブル名とデータ取得することができます。
ただし、①で得た SQL を実行すると、全てのテーブル対してシーケンシャルスキャンが実行されるため、
実行する際には、ディスク I/O の負荷に注意して下さい。

```
=#  SELECT 't1' AS table_name,count(*) AS count FROM t1 UNION all
    SELECT 't1_201301' AS table_name,count(*) AS count FROM t1_201301 UNION all
    SELECT 't2' AS table_name,count(*) AS count FROM t2 UNION all
    SELECT 'test' AS table_name,count(*) AS count FROM test;
```

実行結果②
上記 SQL を実行すると PosgreSQL のテーブル、データ件数を得ることができます。

```
 table_name | count
------------+--------
 t1         |     999
 t1_201301  |     999
 t2         |     120
 test       |  100000
(4 rows)
```

また、稼働統計情報の pg_stat_all_tables ビューや pg_stat_user_tables ビューを用いて、各テーブルの有効行数の推定値を確認することができます。ただし、表示される値は推定値ですので、データ件数確認試験時には利用しないで下さい。下記 SQL は、各テーブルの大まかなデータ件数を把握したい場合に有用です。

```
=# SELECT relname AS table_name,n_live_tup FROM pg_stat_user_tables ORDER BY relname;
 table_name | n_live_tup
------------+------------
 t1         |        999
 t1_201301  |        999
 t1_201302  |        120
 test       |     100000
(4 rows)
```

4.5.2. Oracle

Oracle のテーブル名、データ件数は下記 SQL[1]で取得可能です。

```
SQL>
select
   table_name,
   to_number(
     extractvalue(
       xmltype(
    dbms_xmlgen.getxml('select count(*) c from '||table_name))
       ,'/ROWSET/ROW/C')) count
 from user_tables
 WHERE TABLE_NAME NOT LIKE 'BIN$%'
   and (iot_type != 'IOT_OVERFLOW' or iot_type is null)
order by table_name;
```

1 http://www.oracle.co.jp/forum/thread.jspa?threadID=35001130

実行結果
上記SQLを実行するとテーブル、データ件数を得ることができます。

```
TABLE_NAME                          COUNT
------------------------------ ----------
DATABASECHANGELOG                      22
DATABASECHANGELOGLOCK                   1
IS_ACCESSLOGS                           3
IS_ACCOUNTS                            11
IS_ADMINROLES                           2
IS_AUTHCREDENTIALS                      0
IS_CACHES                               1
IS_FORBIDDENURLS                        2
IS_GADGETS                             45
IS_GADGET_ICONS                         6
IS_HOLIDAYS                             1
[省略]
44行が選択されました。
```

4.5.3. SQL Server

SQL Serverのテーブル名、データ件数は下記SQLで取得可能です。

```
> select distinct a.name, b.row_count from sys.tables a left join sys.dm_db_partition_stats b on
a.object_id = b.object_id where b.index_id<=1 order by a.name
```

実行結果
上記SQLを実行するとPosgreSQLのテーブル、データ件数を得ることができます。

```
name                                                     row_count
----------------------------------------------------------------------------------------------------------
--------------------------- --------------------
testtbl
                                        1
testtbl2
                                        10
TestTbl3
                                        1

(3 行処理されました)
```

4.6. データサイズの確認試験

異種 DBMS に格納されたデータ件数およびデータの種別等から、PostgreSQL 移行後のデータサイズの想定値を算出します。PostgreSQL の実際のデータサイズと計算より算出した想定値が近似であることを確認します。

下記を参照し、PostgreSQL に DBMS を移行した場合のデータサイズの想定値を算出して下さい。

- 4.6.1 テーブルサイズ見積もり方法
- 4.6.2 TOAST テーブルサイズ見積もり方法

実際のテーブルサイズの取得方法は下記を参照して下さい。

- 4.6.3 テーブルサイズ取得方法

4.6.1. テーブルサイズ見積もり方法

PostgreSQL の各テーブルのサイズの予測値は PostgreSQL のデータブロック（デフォルト 8192 バイト）に対象のテーブルのレコードがいくつ入るかを計算した後、全レコードを格納するための必要ブロック数を乗算して算出します。

まず、1 レコードのサイズ算出は以下の様に行います。

```
レコードサイズ = ブロック内部ポインタ + レコードヘッダ + NULL ビットマップ + パディング + OID + フィールド領域
```

図 4.2: レコードサイズ算出方法

上に記載されている因子のサイズは以下の通りです。

表 4.3: レコードサイズ関連因子

No.	因子	サイズ
1	ブロック内部ポインタ	4 バイト
2	レコードヘッダ	23 バイト
3	NULL ビットマップ	計算により算出
4	パディング	計算により算出
5	OID	4 バイト
6	フィールド領域	計算により算出

NULL ビットマップは NULL を許可する列が一つでもある場合は、全列に対して 1 バイトずつ必要です。

```
len = （列数 + 7)/8
NULL ビットマップ = len を len 以上の最も小さい 4 の倍数になるように切り上げ
```

図 4.3: NULL ビットマップ算出方法

パディングはレコードヘッダと NULL ビットマップの合計が 4 の倍数にならない場合に、切り上げるために用います。

フィールド領域は PostgreSQL のドキュメント[2]に従って合算をしてください。ただし、合算値が 8 の倍数にならない場合は、最も小さい 8 の倍数に切り上げてください。

1 レコードあたりのサイズを求めることができましたので、次に、1 ブロックに何レコード格納できるかを計算します。計算結果の端数は切り捨てです。

PostgreSQL9.3 から、チェックサム機能が実装されブロックヘッダの内容が変わりましたが、既存の構造体の中で、同サイズのメンバが置換されたのでブロックヘッダのサイズに変更はありません。

```
1 ブロックに格納できる行数 = （ブロックサイズ - ブロックヘッダ）/ レコードサイズ
```

図 4.4: 1 ブロックに格納できる行数算出方法

上に記載されている因子のサイズは以下の通りです。

2 https://www.postgresql.jp/document/current/html/datatype.html

表 4.4: ブロックサイズ関連因子

No.	因子	サイズ
1	ブロックサイズ	8192 バイト
2	ブロックヘッダ	24 バイト

次に、全レコードを格納するのに必要なブロック数と、テーブルサイズを計算します。計算結果の端数は切り上げです。

```
全レコードを格納するのに必要なブロック数 = 全レコード数 / 1 ブロックに格納できる行数
テーブルサイズ = 全レコードを格納するのに必要なブロック数 * ブロックサイズ
```

図 4.5: テーブルサイズ算出方法

4.6.2. TOAST テーブルサイズ見積もり方法

PotgreSQL のブロックのサイズはデフォルトで 8192 バイトであるため、1 行が 8192 バイトを超えるようなデータを、通常のブロックには格納することができませんでした。

PostgreSQL 7.1 より TOAST (The Oversized-Attribute Storage Technique)と呼ばれる仕組みを取り入れ 8192 バイト以上のデータを格納できるようにしています。

TOAST とは、行内の大きなデータを複数に分割すると同時に圧縮し、複数のブロックにデータを保存する仕組みです。データの圧縮には LZ 系の圧縮アルゴリズムが利用されています。[3]

小さなデータを圧縮してもあまり効果がないため、TOAST_TUPLE_THRESHOLD バイト(デフォルト 2 キロバイト)を超える時にのみ実行されます。分割されたデータが TOAST_TUPLE_TARGET バイト(デフォルト 2 キロバイト)より小さくなるか縮小ができなくなるまで、圧縮し外部ブロックへの保存を行います。

TOAST を使用するためにはテーブルに可変長の列を持つ必要があります。テーブルが可変長の列を持つ場合のみ外部ブロックを保持する TOAST テーブルを持ちます。TOAST テーブルは可変長の列を含むテーブルが作成されると自動的に作成されます。

下記に任意のテーブルに作成された TOAST テーブルおよび TOAST インデックスを確認する SQL を記載します。TOAST インデックスとは TOAST テーブルに付与されたインデックスです。TOAST テーブルのに格納されたデータの検索性能向上のために自動作成されます。TOAST インデックスは、PostgreSQL の内部処理で利用されるものです。

```
=# SELECT relname, relpages
FROM pg_class,
     (SELECT reltoastrelid
      FROM pg_class
      WHERE relname = 'テーブル名') AS ss
WHERE oid = ss.reltoastrelid OR
      oid = (SELECT indexrelid
             FROM pg_index
             WHERE indrelid = ss.reltoastrelid)
ORDER BY relname;
        relname        | relpages
-----------------------+----------
 pg_toast_25085        |        0  --TOAST テーブル
 pg_toast_25085_index  |        1  --TOAST インデックス
(2 rows)
```

図 4.6: 任意のテーブルに作成された TOAST テーブルおよび TOAST インデックスの確認

psql の¥d を用いて、TOAST テーブルの構造を確認することが可能です。

TOAST テーブルは pg_toast スキーマに格納されているため、テーブル名の前に pg_toast を付与して下さい。

3 src/backend/utils/adt/pg_lzcompress.c

```
=# ¥d  pg_toast.pg_toast_25085
TOAST table "pg_toast.pg_toast_25085"
   Column   |  Type
------------+---------
 chunk_id   | oid
 chunk_seq  | integer
 chunk_data | bytea
```

図 4.7: TOAST テーブルの構造確認

上記 SQL の結果の通り、TOAST テーブルは下表の列を持ちます。

表 4.5: TOAST テーブルの列構成

No.	列名	データ型
1	chunk_id	OID
2	chunk_seq	INTEGER
3	chunk_data	BYTEA

TOAST テーブルにはデータが BYTEA 型に変換された後、「チャンク（chunk）」と呼ばれる単位に分割されて格納されてます。1 つのチャンクの大きさはデフォルトで 1996 バイトです。1 ブロックのサイズが 8192 バイトであるため、1 ブロックに最大 4 チャンクを格納可能です。

TOAST テーブルのサイズ予測値は 1 データを格納するためにチャンク（通常 1996 バイト）がいくつ必要かを計算した後、全データを格納するために必要なチャンクする数を算出します。

必要なチャンク数を格納するために必要ブロック数を算出します。

まず、1 レコードのサイズ算出は以下の様に行います。

```
TOAST テーブルレコードサイズ = ブロック内部ポインタ + OID + データ領域 × 圧縮率
```

図 4.8: レコードサイズ算出方法

上に記載されている因子のサイズは以下の通りです。

表 4.6: レコードサイズ関連因子

No.	因子	サイズ
1	ブロック内部ポインタ	4 バイト
2	OID	4 バイト
3	データ領域	平均データ長(可変長列の平均データ長を確認する必要がある)
4	圧縮率	実機で確認の必要あり

1 レコードあたりのサイズを求めることができましたので、次に 1 レコードを格納するために必要なチャンク数を計算します。計算結果の端数は切り捨てです。

```
1 行を格納するために必要なチャンク数 = TOAST テーブルレコードサイズ ÷ チャンクサイズ
```

図 4.9: 1 行を格納するために必要なチャンク算出方法

上に記載されている因子のサイズは以下の通りです。

表 4.7: 必要チャンク数関連因子

No.	因子	サイズ
1	チャンクサイズ	1996 バイト

次に、全レコードを格納するのに必要なチャンク数を計算し、チャンクを格納するために必要なブロック数を計算します。計算結果の端数は切り上げです。

テーブル格納に必要なチャンク数 = 1行を格納するために必要なチャンク数 * 行数
チャンク格納に必要なブロック数 = テーブル格納に必要なチャンク数 / 1ブロックに格納できるチャンク数

図 4.10: テーブルサイズ算出方法

上に記載されている因子のサイズは以下の通りです。

表 4.8: ブロックサイズ関連因子

No.	因子	値
1	1ブロックに格納できるチャンク数	4
2	行数	テーブルの行数

TOAST テーブルを持つ元テーブルのサイズは、フィールド領域以外は、TOAST テーブルを用いない場合と同じ構図となっています。そのため「4.6.1 テーブルサイズ見積もり方法」に記載した方法で算出可能です。
フィールド領域は下表のような構成となっています。

表 4.9: フィールド構造

No.	フィールド明	データ長	保持情報
1	va_header	4バイト	フラグ＋データ長
2	va_rawsize	4バイト	圧縮前のサイズ
3	va_extsize	4バイト	TOAST に格納されているサイズ
4	va_valueid	4バイト	chunk_id
5	va_toastrelid	4バイト	TOAST テーブルの ID

4.6.3. テーブルサイズ取得方法

PostgreSQL の各テーブルサイズは下記 SQL で確認することが可能です。
実際のテーブルサイズと見積もったテーブルサイズが近似であることを確認して下さい。

```
 =# SELECT
  relname AS tablename,
  pg_relation_size(relid) as bytes
FROM
  pg_stat_user_tables
ORDER BY tablename;
 tablename |  bytes
-----------+----------
 t1        |        0
 t1_201301 |        0
 t1_201302 |        0
 test      | 37240832
(4 rows)
```

図 4.11: PostgreSQL テーブルサイズ取得方法

4.6.4. テーブルサイズ算出例

例として、以下の様なテーブルサイズ算出を行います。テーブル名は emp とし、データ件数は id カラムの投入データに記載している通り、10000 件です。

表 4.10: テーブルサイズ算出例

No.	カラム名	データ型	制約	投入データ
1	id	integer	not null	generate_size(1,10000)
2	name	text	not null	'postgres_taro'
3	dept	integer	not null	10
4	join_date	date	not null	'2000-1-1'

まず、レコードサイズを求めます。レコードサイズは前述の通り以下のように求められます。

```
レコードサイズ = ブロック内部ポインタ + レコードヘッダ + NULL ビットマップ + パディング + OID + フィールド領域
```

レコードサイズの動的部分は NULL ビットマップ、パディング、フィールド領域の 3 つです。今回の例では、全てのカラムに not null 制約を付与しているため、NULL ビットマップは不要です。

パディングについては、レコードヘッダ（23 バイト）と NULL ビットマップ（0 バイト）の和が 27 バイトであるため、4 の倍数である 28 に切り上げるために 1 バイト必要です。

フィールド領域は、まずカラムの合計サイズをマニュアルに従って算出すると、id = 4, name = 14, dept = 4, join_date = 8 なので、合計 30 バイトです。フィールド領域は 8 バイト単位で確保されますので、32 バイトまで切り上げます。従って、レコードサイズは以下の通り 60 バイトです。

```
レコードサイズ = 4 + 23 + 0 + 1 + 0 + 32
              =60
```

次に、このレコードが 1 ブロックに何レコード格納可能かを求めます。以下の計算式に値を代入します。
計算結果は小数点以下を切り捨てです。

```
1 ブロックに格納できる行数 = （ブロックサイズ - ブロックヘッダ）/ レコードサイズ
                          = (8192 - 24) / 60
                          = 136.1333333・・・・・・
                          ≒ 136
```

最後に、全レコードを格納するのに必要なブロック数を基にテーブルサイズを算出します。全レコードを格納するのに必要なブロック数は小数点以下繰り上げです。

```
全レコードを格納するのに必要なブロック数 = 全レコード数 / 1 ブロックに格納できる行数
                                    = 10000 / 136
                                    = 73.52941・・・・
                                    ≒ 74
```

テーブルサイズは以下の通り、606,208（バイト）と予測できます。

```
テーブルサイズ = 全レコードを格納するのに必要なブロック数 * ブロックサイズ
              = 74 * 8192
              = 606208
```

以下のように実際のテーブルサイズを確認すると、予測結果と同値であることがわかります。

```
=# ¥d emp
       Table "public.emp"
  Column   |  Type   | Modifiers
-----------+---------+-----------
 id        | integer | not null
 name      | text    | not null
 dept      | integer | not null
 join_date | date    | not null

=# select count(*) from emp;
 count
-------
 10000
(1 row)

=# SELECT
-#   relname AS tablename,
-#   pg_relation_size(relid) as bytes
-# FROM
-#   pg_stat_user_tables
-# ORDER BY tablename;
 tablename |  bytes
-----------+---------
 emp       |  606208
(1 row)
```

4.7. データ内容の確認試験

異種 DBMS に格納されたデータと PostgreSQL に格納されたデータが同一であることを確認します。

本資料では、異種 DBMS から出力した CSV ファイルと、PostgreSQL から出力した CSV ファイルの内容が同一であることを確認します。下図にデータ内容確認試験の概念図を記載します。

各工程の詳細な説明については次章以降を参照ください。

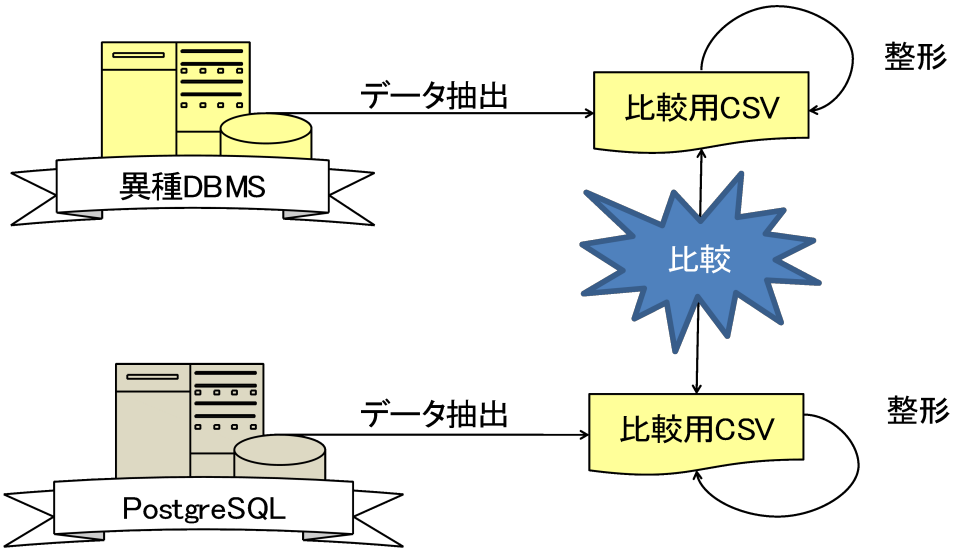


図 4.12: データ移行結果確認試験の流れ

4.7.1. データ抽出

試験を行うにあたり、CSV を利用した試験が前提となりますので、各 DBMS からの CSV 出力方法を以下に記載します。可能であれば、データ移行時のデータ移行漏れを検出するため、移行のデータ抽出手順とは異なる手順でデータ抽出を実施して下さい。(Oracle からのデータ移行時に Ora2Pg を用いた場合は、試験時に SPOOL コマンドを利用する等。)

4.7.2. Oracle

Oracle からデータを抽出する手法として、SPOOL コマンドと Ora2Pg が挙げれらます。それぞれの操作コマンド例は以下を参照して下さい。

表 4.11: Oracle からのデータ抽出方法

No.	項目	URL
1	SPOOL	https://www.pgecons.org/downloads/53
2		https://www.pgecons.org/downloads/54
3	Ora2Pg	https://www.pgecons.org/downloads/54

ただし、Ora2Pg を用いてデータを抽出する場合、PostgreSQL に適した形に整形されている場合がありますので、試験時には注意して下さい。(Ora2Pg は試験時のデータ抽出方法としては推奨しません。)

SPOOL を利用した CSV 抽出の例を以下に記載します。

以下例では emp テーブルから output.csv という名前の CSV ファイルを出力しています。

```
SQL> set echo off
SQL> set linesize 32767
SQL> set trimspool on
SQL> set pagesize 0
SQL> set heading off
SQL> set feedback off
SQL> spool output.csv
SQL> select col1 ||','|| col2 ||','||...  coln from emp;
SQL> spool off
```

上記 SPOOL 例で利用しているシステム変数の説明は以下を参照してください。

表 4.12: spool コマンドのシステム変数

No.	オプション	説明
1	echo	入力されたコマンドを出力するか設定するシステム変数です。off を指定することでコマンドを出力しないようになります。
2	linesize	1 行の長さを設定するシステム変数です。csv を出力する場合は、途中で改行されないように最大値を指定しておくことが望ましいです。(32bitOS では 32767)
3	trimspool	行末移行の空白をカットするか設定するシステム変数です。on を指定することで行末移行の空白を出力しなくなります。
4	pagesize	1 ページの行数を設定するシステム変数です。短縮形は pages。0 を指定することで全行を 1 ページとして出力します。
5	heading	ヘッダを出力するか設定するシステム変数です。off を指定することでヘッダを出力しなくなります。
6	feedback	問合せ結果レコード件数表示の有無を設定するシステム変数です。off を指定することで結果の出力を行わないようになります。

Oracle から CSV ファイルを抽出する際の注意点について以下に記載します。

表 4.13: Oracle からデータを抽出する際の注意点

No.	項目	概要	対処
1	SELECT 文の記述方法	SPOOL を使用した CSV 出力を行う際、「SELECT * ~」で出力してしまうと固定長の CSV ファイルができてしまいます。	セパレータをシステム変数で指定するのではなくクエリ内に埋め込むことで対処(「‖','‖」等)します。
2	linesize の制限	SPOOL では 1 行の出力文字列長がシステム変数 linesize の上限を超過する長さである場合に改行が発生してしまうため、CSV として正しく出力できません。	SPOOL 以外の手段で抽出するか、あるいはハッシュ値等の固定長データに置き換えて出力することで対処します。 ※ BLOB 型, CLOB 型のデータを含むテーブルを出力する際は linesize 上限を超過する可能性が高いため、ハッシュ値変換を推奨します。

上記注意点(また後述する整形時注意点)を踏まえた、Oracle から CSV を抽出する参考スクリプトを次頁に記載します。使用する際は下記仕様が存在することを留意ください。

1. テーブルリストの作成
CSV として出力するテーブル名のリスト(改行区切り)を「/tmp/table_list.txt」に作成しておく必要があります。
以下 SQL を実行することにより作成可能です。

```
# sqlplus -s [スキーマ名]/[パスワード]@[ホスト名]:[ポート番号]/[SID] <<EOF
> SET FEEDBACK OFF
> SET HEADING OFF
> SET PAGESIZE 0
> SET TRIMOUT ON
> SET TRIMSPOOL ON
> SPOOL /tmp/table_list.txt
> SELECT TABLE_NAME FROM USER_TABLES;
> SPOOL OFF
> EXIT
> EOF
```

2. ハッシュ値比較
BLOB 型や CLOB 型などの linesize を超過する可能性のあるデータをハッシュ値に置き換えています。またハッシュ値の算出に DBMS_CRYPTO パッケージ(Hash()関数)を使用しているため、以下 SQL 文を実行しパッケージの実行権限を付与しておく必要があります。

```
SQL> grant execute on DBMS_CRYPTO to [スキーマ名];
```

```
#!/bin/bash
#以下権限は手動で付与
#grant execute on DBMS_CRYPTO to [スキーマ名];
mkdir -p /tmp/ora_table_define/
mkdir -p /tmp/ora_csv/
READFILE=/tmp/table_list.txt

#カラム名、定義確認
while read line; do
  sqlplus -s system/[パスワード]@[ホスト名]:[ポート番号]/[インスタンス名] <<EOF
    SPOOL "/tmp/ora_table_define/$line.define.csv"
    SET LINESIZE 32767
    SET TRIMSPOOL ON
    SET PAGESIZE 0
    SET HEADING OFF
    SET NEWPAGE NONE
    SET FEEDBACK OFF
    SELECT COLUMN_NAME||','|| DATA_TYPE FROM DBA_TAB_COLUMNS where TABLE_NAME = '$line';
    exit;
EOF
done < $READFILE

#CSV出力クエリ生成
rm -f /tmp/createcsv.sql
echo "SET LINESIZE 32767" > /tmp/createcsv.sql
echo "SET TRIMSPOOL ON" >> /tmp/createcsv.sql
echo "SET PAGESIZE 0" >> /tmp/createcsv.sql
echo "SET HEADING OFF" >> /tmp/createcsv.sql
echo "SET NEWPAGE NONE" >> /tmp/createcsv.sql
echo "SET FEEDBACK OFF" >> /tmp/createcsv.sql
echo "SET LONG 1000000" >> /tmp/createcsv.sql
echo "SET LONGC 1000000" >> /tmp/createcsv.sql

while read line; do
  flag=0
  sql="SELECT "
  for def in `cat /tmp/ora_table_define/$line.define.csv`
  do
    if test $flag -ne 0 ; then
      sql="${sql} ||"¥'",'¥'"|| "
    fi
    arr=( `echo $def | tr -s ',' ' '`)
    if test "${arr[1]}" = "BLOB" || test "${arr[1]}" = "CLOB" ; then
      sql="${sql} rawtohex(DBMS_CRYPTO.Hash(¥"${arr[0]}¥",2))"
    else
      sql="${sql} ¥"${arr[0]}¥""
    fi
    flag=1
  done
  sql="${sql} FROM $line;"
  echo "$line" >> /tmp/output_table_list.txt
  echo "spool /tmp/ora_csv/"$line".csv" >> /tmp/createcsv.sql
  echo $sql >> /tmp/createcsv.sql
done < $READFILE

#CSV出力クエリ実行
sqlplus -s [スキーマ名]/[パスワード]@[ホスト名]:[ポート番号]/[インスタンス名] <<EOF
  @/tmp/createcsv.sql
  exit;
EOF
exit 0
```

4.7.3. SQL Server

SQL Server からデータを抽出する方法として、bcp ユーティリティと Microsoft SQL Server Management Studio が挙げられます。それぞれの操作、コマンド例はそれぞれ以下を参照してください。

表 4.14: SQL Server からのデータ抽出方法

No.	項目	URL
1	bcp ユーティリティ	https://www.pgecons.org/downloads/54
2	Microsoft SQL Server Management Studio	https://www.pgecons.org/downloads/54

Microsoft SQL Server Management Studio は GUI ツールですので容易に扱えますが、bcp ユーティリティの方が SQL の結果を CSV 化できるなど、柔軟な対応が取れますので、後々のデータ整形の必要性を考慮して、bcp ユーティリティの利用を推奨します。

bcp コマンドの例を以下に記載します。

```
> bcp.exe "SELECT * FROM [[DB 名]].[[スキーマ名]].[[テーブル名]];" queryout [出力 CSV ファイルパス] -c -t[区切り文字] -S [サーバ名]¥[インスタンス名] -T
```

上記の bcp コマンドで利用しているオプションの説明を以下に記載します。

No.	オプション	説明
1	queryout	SQL コマンドの結果を CSV ファイルに出力します。上述の例だと、"SELECT * FROM [[DB 名]].[[スキーマ名]].[[テーブル名]];"です。
2	-c	このオプションをつけない場合、カラム毎に対話式のデータ出力方法となるため、コマンド処理に適しません。
3	-t	区切り文字を指定します。-t と区切り文字の間にスペースは不要です。
4	-S	サーバ名とインスタンス名を指定します。
5	-T	Windows 認証等の統合認証を利用します。-U [ユーザ名]と、-P [パスワード]でも認証可能です。

4.7.4. PostgreSQL

PostgreSQL からは COPY コマンドを利用すると CSV としてデータを出力可能です。COPY コマンドの例を以下に記載します。以下の例では psql の-c オプションを利用して COPY コマンドを実行しています。この理由としては、テーブル数が多い場合はスクリプトで自動化することになると思いますが、psql 等の OS コマンドでラップされていた方が引数等を柔軟に変更しやすいため下記例としました。

注意点としては、下記例では FORMAT オプションが WITH の中で指定されていないため、フォーマットは CSV ではなく、デフォルトのテキストフォーマットで出力されます。この理由は、CSV フォーマットで出力すると空白や区切り文字を含むデータを自動で引用符で括ってくれるのですが、この引用符が比較の際に邪魔になってしまう場合があるためです。データを比較試験目的ではなく、簡易バックアップや、(無いとは思いますが)PostgreSQL から異種 DBMS へのデータ移行といった再利用可能な形で出力したい場合は、CSV フォーマットを選択されることを推奨します。

```
$ psql -d [データベース名] -U [ユーザ名] -p [ポート番号] -c "COPY (SELECT * FROM [テーブル名]) TO '[出力 CSV ファイルパス]' WITH (NULL '[NILL 文字列]', DELIMITER '[区切り文字]');"
```

No.	コマンド	オプション	説明
1	psql	-d	データベース名を指定します。
2		-U	ユーザ名を指定します。
3		-p	ポート番号を指定します。
4		-c	実行する SQL を指定します。
5		-T	Windows 認証等の統合認証を利用します。-U [ユーザ名]と、-P [パスワード]でも認証可能です。
6	COPY	TO	出力 CSV のファイルパスを指定します。
7		WITH	COPY の詳細オプションを指定します。
8		NULL	NULL の代替文字列を指定します。
9		DELIMITER	CSV の区切り文字を指定します。

出力するテーブル名のリストから自動で CSV ファイルを出力する参考スクリプトを以下に記載します。
以下手法であれば、抽出時に実施すべき整形処理も記載することが可能です。(以下スクリプトでは Oracle から抽出した CSV と比較する際に必要となる整形処理(後述)が実行されています)
使用する際は下記仕様が存在することを留意ください。

1. テーブルリストの作成
CSV として出力するテーブル名のリスト(改行区切り)を「/tmp/table_list.txt」に作成しておく必要があります。
以下 SQL を実行することにより作成可能です。

```
$ psql [データベース名] -c "COPY (SELECT relname FROM pg_stat_user_tables) TO '/tmp/table_list.txt';"
```

2. ハッシュ値比較
Oracle の出力結果と比較するため、bytea 型や text 型などのデータを md5()関数によりハッシュ値に置き換えています。また Oracle はハッシュ値が大文字で出力されるため、PostgreSQL 側では upper()関数により大文字化しています。

3. タイムスタンプ値の整形
Oracle と PostgreSQL では日付型のミリ秒表示の形式が異なるため(詳細は後述)、to_char()関数を使用し、出力フォーマットを Oracle に合わせています。

```
#!/bin/bash
user=$1
dbname=$2
rm -rf /tmp/pg_table_define/
rm -rf /tmp/pg_csv/
mkdir -p /tmp/pg_table_define/
mkdir -p /tmp/pg_csv/
READFILE=/tmp/table_list.txt
#カラム名、定義情報出力
while read line; do
  psql -U $user $dbname -c "COPY (SELECT a.attname, format_type(a.atttypid, a.atttypmod) FROM pg_attribute a
LEFT JOIN pg_attrdef d ON a.attrelid = d.adrelid AND a.attnum = d.adnum WHERE a.attrelid = '$line'::regclass
AND a.attnum > 0 AND NOT a.attisdropped ORDER BY a.attnum) TO '/tmp/pg_table_define/$line.define.csv' WITH
DELIMITER ','"
done < $READFILE

#CSV 出力クエリ生成
rm -f /tmp/createcsv.sql
while read line; do
```

```
  flag=0
  sql="COPY (SELECT "
  for def in `cat /tmp/pg_table_define/$line.define.csv | sed -e 's/ //g'`
  do
    if test $flag -ne 0 ; then
      sql="${sql} ,"
    fi
    arr=( `echo $def | tr -s ',' ' '`)
    if test "${arr[1]}" = "bytea" || test "${arr[1]}" = "text" ; then
      sql="${sql} upper(md5(¥"${arr[0]}¥"))"
    elif test "${arr[1]}" = "timestampwithouttimezone" ; then
      sql="${sql} to_char(¥"${arr[0]}¥", 'YY-MM-DD HH24:MI:SS.US')"
    else
      sql="${sql} ¥"${arr[0]}¥""
      #sql="${sql} ¥"$1¥""
    fi
    flag=1
  done
  sql="${sql} FROM $line) TO '/tmp/pg_csv/$line.csv' WITH DELIMITER ',';"
  echo $sql >> /tmp/createcsv.sql
done < $READFILE

#CSV出力クエリ実行
psql -U $user $dbname -c "¥i /tmp/createcsv.sql"
exit 0
```

4.7.5. CSV ファイルの整形

データベース毎に出力するデータの順番や文字コードが異なる場合があるので、CSV ファイルの整形が必要な場合があります。前章で記載した通り、CSV ファイルの整形は抽出時のオプション等によっても可能です。

PostgreSQL から出力した CSV ファイルは試験対象であるため基本的には整形せずに可能な限り異種 DBMS から出力した CSV を整形します。ただし、異種 DBMS から出力した CSV の整形が困難な場合は、PostgreSQL の CSV ファイルを整形します。

例えば、Oracle Database の SPOOL コマンドでは NULL は空文字として CSV ファイルに出力されます。

Oracle Database から出力した CSV ファイルを PostgreSQL の CSV ファイルと比較する場合、Oracle Database から出力した CSV ファイル内の空文字を全て NULL(¥N)に置換する必要があります。CSV ファイル内の全ての空文字を NULL(¥N)に置換する処理は影響が大きいため、PostgreSQL から出力した NULL(¥N)を空文字に変換する処理が必要になる場合があります。上記のような場合は、変換処理の影響範囲を十分に検討し、実施して下さい。

CSV ファイルのソート

出力された CSV は DBMS の選択順に依存しているため、データの順番が同一でない場合があります。

この現象を回避するために、SQL 抽出時に ORDER BY 等を用いてソートすることも可能ですが、ソート対象のカラムを判別しなくてはならず、SQL 作成に要する作業時間が大きくなる可能性があるため、OS コマンドでカラムを意識せず全体をソートした方が効率が良い場合もあります。そこで本節では、OS 毎のソートコマンドについて記載します。

Linux 系 OS

Linux 系 OS ではソートを行うコマンドとして sort コマンドがあります。基本的なコマンド実行方法は以下の通りです。

```
$ sort [オプション] [ソート対象ファイル] > [ソート結果ファイル]
```

デフォルトでは、文末までを 1 データとみなし、昇順で出力されますので、適宜オプションを付与して必要なかたちにソートを行ってください。

No.	オプション	説明	対象データ	ソート後
1	-r	逆順(降順)でソートされるようになります。	test3,2 test4,5 test1,4 test5,1 test2,3	test5,1 test4,5 test3,2 test2,3 test1,4
2	-t	区切り文字を指定します。次に記載するフィールド位置と組み合わせると、カラム毎の ORDER BY を疑似できます。	-	-
3	-k	フィールド位置を指定します。フィールド番号は 1 から開始します。 右の例では、「-t “,” -k 1」を付与し、カンマを区切り文字として、二番目の数値列でソートするように指定しています。	test3,2 test4,5 test1,4 test5,1 test2,3	test5,1 test3,2 test2,3 test1,4 test4,5

Windows Server

Windows Server でも Linux 同様 sort コマンドがあり、使い方も同様です。

```
> sort [オプション] [ソート対象ファイル] > [ソート結果ファイル]
```

オプションは逆順ソートが可能ですが、区切り文字は指定できず、ソート位置の指定が行えるのみです。

No.	オプション	説明	対象データ	ソート後
1	/R	逆順(降順)でソートされるようになります。	test3,2 test4,5 test1,4	test5,1 test4,5 test3,2

			test5,1 test2,3	test2,3 test1,4
2	/+N	N 文字目から比較します。 右のテストデータでは、1 カラム目の文字数が同ーですので、/+7 とすることで、、2 カラム目をpソートすることができました。	test3,2 test4,5 test1,4 test5,1 test2,3	test5,1 test3,2 test2,3 test1,4 test4,5

4.7.6. CSV ファイルの文字コード変換

移行元および移行先から出力した CSV ファイル文字コードが異なるとデータ比較処理によって差分として検出される可能性があります。文字コードの違いによって差分が出てしまった場合は、文字コードを変換した上でデータ比較を行い、データが同一であることを確認してください。

Linux 系 OS

一般的には文字コードの変換ツールとして nkf、iconv が利用されます。nkf、iconv のコマンド例はそれぞれ以下を参照して下さい。

表 4.15: 文字コード変換ツール

No.	項目	URL
1	nkf	https://www.pgecons.org/downloads/54
2	iconv	https://www.pgecons.org/downloads/54

Windows Server

Windows でも nkf が利用可能です。

No.	項目	URL
1	nkf	http://www.vector.co.jp/soft/win95/util/se295331.html

ダウンロードすると日本語の使い方が同梱されていますので、詳細はそちらでご確認ください。以下のようなオプションが利用可能です。

No.	オプション	説明
1	--guess	ファイルの文字コードを確認します。
2	-w8	UTF8 へ変換します。
3	-s	Shift-JIS へ変換します。

4.7.7. Oracle Database の CSV 整形方針

Oracle Database の SPOOL コマンドで出力した CSV ファイルと、PostgreSQL の COPY コマンドで出力した CSV ファイルを比較する際の注意点と対処方法を下表に記載します。

表 4.16: CSV ファイル比較時の注意点(Oracle(SPOOL)、PostgreSQL(COPY))

No.	項目	Oracle(SPOOL)	PostgreSQL(COPY)	対処
1	バイトデータ(16 進数)の出力形式	大文字	小文字 値の最初に"¥x"が付与される	Oracle のバイトデータを小文字に変換し、値の最初に"¥x"付与する必要があります。 あるいは PostgreSQL のバイトデータの"¥x"を空文字に置換することで対処可能です。 (置換がバイトデータ以外に適用されないように注意が必要です。)
2	ハッシュ値の出力形式 (ハッシュ値比較を行う場合のみ[4])	大文字 (DBMS_CRYPTO.Hash()関数等を利用)	小文字 (md5()関数等を利用)	Oracle のハッシュ値を小文字化するか、あるいは PostgreSQL のハッシュ値を大文字化することで対処します。[5]
3	メタ文字を含むデータへの引用符の有無	引用符を付与しない	引用符「"」が付与される	全てのカラムに引用符「"」を付与することで対処します。[6]
4	メタ文字のエスケープ	メタ文字のエスケープは行われない	メタ文字には"¥"を付与しエスケープ 例:"→ ¥"	Oracle の出力にエスケープ文字"¥"を付与するか、あるいは PostgreSQL の出力 CSV のエスケープ文字"¥"を空文字に置換します。 例: ¥" → " 文字列中の"¥"が置換されないように注意して下さい。
5	NULL の出力	空文字に変換し出力	NULL(¥N)を出力	PostgreSQL の NULL 文字を空文字に置き換えることで対処します。
6	日付型のミリ秒表示	ミリ秒末尾に 0 を補完 (例:2015/04/13 13:30:55.150)	ミリ秒末尾に 0 は補完されない (例:2015/04/13 13:30:55.15)	Oracle 側の末尾のゼロを削除するか、あるいは PostgreSQL の日付型の末尾にゼロを付与することで対処します。併せて、区切り文字を比較対象のフォーマットに変換することを推奨します。 (デフォルトは「-」ハイフンです。) 抽出時にゼロを付与する例: to_char(″日付型カラム名″, 'YY/MM/DD HH24:MI:SS.MS')
6	行の表示順	データの物理上の位置に依存	データの物理上の位置に依存	ORDER BY などで出力時に整形するか、出力後に OS のファイル内容操作コマンドでソートを行う必要があります。

4 バイトデータの比較を行う場合はハッシュ値を利用することを推奨します。
5 整形による対処が困難な場合、CSV 抽出時に upper 関数(PostgreSQL 側)/lower 関数(Oracle 側)を使用し対処します。
6 CSV 抽出時に引用符を付与することを推奨します。

4.7.8. SQL Server の CSV 整形方針

SQL Server の bcp コマンドで出力した CSV ファイルと、PostgreSQL の COPY コマンドで出力した CSV ファイルを比較する差際の注意点と対処方法を下表に記載します。下表における PostgreSQL の出力例は Windows Server 上での挙動です。

表 4.17: CSV ファイル比較時の注意点(SQL Server(bcp)、PostgreSQL(COPY))

No.	項目	SQL Server(bcp)	PostgreSQL(COPY)	対処
1	NULL の出力	空文字に変換し出力	NULL(¥N)を出力	SQL Server の空文字を NULL に置き換える必要があります。その際、CSV の出力時に PostgreSQL 側で設定した NULL の代替文字に置換します。
2	空文字の出力 (データ内の空文字は除く)	NULL に変換し出力	空文字を出力	SQL Server 側の NULL は本来は空文字ですので、空文字に変換します。
3	改行の出力変換	改行コードを出力	¥n を出力	PostgreSQL の CSV の¥n を改行コードに変換する必要があります。 (SQL Server が側の改行コードは、CSV の行末改行区別がつかないため、PostgreSQL 側の CSV を整形します。)
4	データ内のタブ出力	タブコードを出力	¥t を出力	PostgreSQL から出力した CSV の¥t をタブコードに変換する必要があります。
5	日付型のミリ秒表示	ミリ秒末尾に 0 を補完 (例:2015-04-13 13:30:55.150)	ミリ秒末尾に 0 は補完されない (例:2015-04-13 13:30:55.15)	SQL Server から出力した CSV の末尾の 0 を削除する必要があります。あるいは PostgreSQL の日付型の末尾にゼロを付与することで対処します。
6	行の表示順	データの物理上の位置に依存	データの物理上の位置に依存	ORDER BY などで出力時に整形するか、出力後に OS のファイル内容操作コマンドでソートを行う必要があります。

4.7.9. CSV ファイルの比較

異種 DBMS および PostgreSQL から抽出した CSV ファイルを比較し、同一であることを確認します。

データ内容

出力した CSV のデータが同一であるかを試験します。本節では、OS コマンドでファイル比較を行う方法を記載します。独自のファイル比較ツールをご用意されている場合は、そちらをご利用ください。

本節の例では、以下のようなファイルを利用しました。

```
$ cat testA.txt
1,bbb,ccc
2,bbb,ccc
3,bbb,ccc
4,bbb,ccc
5,bbb,ccc
6,bbb,ccc
7,誤ったデータ 1,ccc
8,bbb,ccc
9,bbb,ccc
10,bbb,ccc
11,bbb,ccc
12,bbb,誤ったデータ 2
```

```
$ cat testB.txt
1,bbb,ccc
2,bbb,ccc
3,bbb,ccc
4,bbb,ccc
5,bbb,ccc
6,bbb,ccc
7,bbb,ccc
8,bbb,ccc
9,bbb,ccc
10,bbb,ccc
11,bbb,ccc
12,bbb,ccc
```

Linux 系 OS

Linux 系 OS ではファイル比較のコマンドとして、diff コマンドがあります。以下に diff コマンドの例を記載します。

```
> diff [比較ファイル A] [比較ファイル B] >> [比較結果ファイル]
```

実際に実行すると以下のように差分が出力されます。

```
$ diff testA.txt  testB.txt > result.txt
$ cat result.txt
7c7
< 7,誤ったデータ 1,ccc
---
> 7,bbb,ccc
12c12
< 12,bbb,誤ったデータ 2
---
> 12,bbb,ccc
```

ファイルが同一であった場合、デフォルトでは何も表示されませんが、diff コマンド実行時に「-s」オプションを付与すると、ファイルが同一でも以下のように結果が出力されます。

```
$ diff -s testA.txt testB.txt >> result.txt
$ cat result.txt
ファイル testA.txt と testB.txt は同一
```

Windows Server

Windows Server ではファイル比較のコマンドとして、fc コマンドがあります。以下に fc コマンドの例を記載します。

```
> fc [比較ファイル A] [比較ファイル B] >> [比較結果ファイル]
```

実際に実行すると以下のように差分が出力されます。

```
> fc testA.txt testB.txt >> result2.txt
> type result2.txt
```

```
ファイル testA.txt と TESTB.TXT を比較しています
***** testA.txt
6,bbb,ccc
7,誤ったデータ 1,ccc
8,bbb,ccc
***** TESTB.TXT
6,bbb,ccc
7,bbb,ccc
8,bbb,ccc
*****

***** testA.txt
11,bbb,ccc
12,bbb,誤ったデータ 2
***** TESTB.TXT
11,bbb,ccc
12,bbb,ccc
*****
aaa,bbb,ccc
aaa,bbb,ccc
aaa,bbb,ccc
*****
```

差分が無いファイルでは、以下のように出力されます。

```
> fc testA.txt testB.txt >> result2.txt
> type result2.txt
ファイル testA.txt と TESTB.TXT を比較しています
FC: 相違点は検出されませんでした
```

4.8. 外字登録の確認

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。
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利用する場合、外字のマッピングを定義し、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が行わないため外字のマッピング定義、登録は不要です。
前提となる外字の以降については、WG2 の過去文書「データ移行調査および実践編」の P34 を参照して下さい。

以降に下表の外字が登録されているか否かを確認する試験を記載します。

表 4.18: 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	朏
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

外字の登録確認試験では下記の 2 パターンで試験を行います。

表 4.19: 外字登録確認試験

No.	試験パターン	確認方法
1	クライアントエンコーディングから データベースエンコーディング への自動変換	データ格納時に利用する自動エンコーディング変換。 INSERT 文等で確認。
2	データベースエンコーディング から クライアントエンコーディングへの自動変換	データ取得時に利用する自動エンコーディング変換。 SELECT 文等で確認

試験パターン 2.「データベースエンコーディング(UTF8) からクライアントエンコーディング(SJIS)への自動変換」を SELECT 文で確認した例を記載します。psql 等のクライアントアプリケーションを利用する場合は、クライアントエンコーディングを正しく指定した状態で実施して下さい。下記は正常に外字が登録されている場合の実行例です。

```
=# SET client_encoding to 'SJIS';
=# SELECT name FROM gaiji_test ;
 name
------
髙
 [省略]
```

下記のようなメッセージが出力された場合、外字が正しく登録されていません。

```
=# SET client_encoding to 'SJIS';
=# SELECT name FROM gaiji_test ;
ERROR:  character with byte sequence 0xee 0x80 0x80 in encoding "UTF8" has no equivalent in encoding
"SJIS"
STATEMENT:  SELECT name FROM gaiji_test ;
ERROR:  character with byte sequence 0xee 0x80 0x80 in encoding "UTF8" has no equivalent in encoding
"SJIS"
```

4.9. 移行検証

infoScoop 上での評価は別紙掲載。

5. アプリケーション移行確認試験

本章では、アプリケーションの移行に際して行う試験手順を記載します。

5.1. 目的

アプリケーションの移行を過去の WG2 文書「アプリケーション移行調査編」、「SQL 移行調査編」に従い実施した後、移行後のアプリケーションと移行前のアプリケーションの動作が変わらないことを確認します。

5.2. 前提条件

本章で紹介する試験は、下表の作業を前提とします。

表 5.1: 前提条件

No.	前提条件	備考
1	試験対象とするアプリケーションの範囲(機能、画面等)が明確になっていること	移行対象のアプリケーションの使われていない機能等は試験の試験対象外とします。
2	異種 DBMS から PostgreSQL へのスキーマ移行が完了していること	_
3	異種 DBMS から PostgreSQL へのデータ移行が完了していること	SQL および画面試験時に異種 DBMS と PostgreSQL の動作に差異がないかを確認するため、異種 DBMS と PostgreSQL には同一のデータが格納されている必要があります。
4	DBMS 仕様の差異によるアプリケーションの変更方針が決定し、変更したアプリケーションが明確になっていること	移行前のアプリケーションと移行後のアプリケーションの試験結果に差が出た場合に確認する必要があります。
5	プロジェクトの状況とユーザへのヒアリング状況を踏まえ、移行したアプリケーションや SQL 文の試験方法、試験範囲について、ユーザと合意していること	新規アプリケーション作成時と同様にアプリケーションの全ての機能を網羅的に試験することが最善です。しかし、プロジェクトの状況(スケジュール等)で網羅的な試験が厳しい場合は、ユーザへのヒアリング状況(試験セットの有無など)を踏まえて、事前にユーザと試験内容について合意する必要があります。 移行前のアプリケーションの試験仕様書や試験セットが既に存在する場合はそれらを利用し結果を確認します。

5.3. 試験項目

アプリケーション移行結果試験として、下表の試験項目が考えられます。

表 5.2: アプリケーション移行試験項目

No.	試験対象	試験観点
1	SQL 文	異種 DBMS と PostgreSQL の場合で SQL 文の処理結果が変わらないことを確認します。また DBMS の仕様差異を埋めるため、SQL 文の修正を行った場合は、修正した SQL の処理結果が想定通りであることを確認します。
2	業務バッチ試験	DBMS 移行前後でバッチの実行結果に差異がないことを確認します。 日時や月次バッチなど、業務の種別ごとに試験を実施する必要があります。
3	画面試験	DBMS 移行前と同様に、画面表示・画面操作が正しく動作することを確認します。 通常、画面試験を補助するツール(Selenium 等)を利用し、実施することを推奨します。

5.3.1. SQL 試験手順

試験は以下の手順で実施します。

① 異種 DBMS に対して SQL を実行し、実行結果を取得
② PostgreSQL に対して SQL を実行、実行結果を取得
③ 手順①、②の実行結果の突合せ

※「実行結果」は、戻り値、影響を受けた行数、結果セットの内容を指します。

SQL 文の実行結果が異なる場合は、移行方針に従った変更（SQL の修正）か、移行時のミスによる問題の何れかと考えられます。SQL の試験においては、仕様に無い SQL の変更や移行漏れを「不具合」とします。
SQL の実行結果の突合せを行った時点で、試験結果の状態は以下のパターンに分けられます。

表 5.3: 試験結果と SQL 試験結果の判定

No.	試験結果	状態	判定
1	結果が一致	SQL の修正が正しく行われた もしくは、SQL の修正は行われていない	正常
2	結果が不一致	移行方針に従った SQL の変更 （もしくはアプリケーションの修正で SQL の実行結果の差分を吸収）	正常
3		予期せぬ SQL 実行結果の不一致	不具合
4	SQL 実行エラー	-	不具合

5.3.2. SQL 実行エラー時の対処

「1.1 サーバログの出力設定」で設定した、PostgreSQL のサーバログからエラー内容を確認します。
SQL 文の実行に失敗した場合の確認観点を下表に記載します。

表 5.4: SQL 文がエラーとなった場合の確認観点

No.	確認観点	概要
1	ユニーク制約違反	当該レコードが存在しなければ INSERT を行い、当該レコードが存在した場合には UPDATE を実施する MERGE 文[7]の書き換え箇所を確認します。 PostgreSQL は UPSERT 文に対応していないため、下記のような修正が行われている場合は「正常」とします。 ・当該レコードの存在有無を SELECT 文で確認し、確認結果に応じて UPDATE 文もしくは INSERT 文を実行するようにアプリケーションのロジックを修正している
2	デッドロック	トランザクション同士のデッドロックが発生していないか確認します。
3	データ型の不一致	PostgreSQL の場合、データ型の暗黙的な型変換を実施しないため、文字型と数値型等の比較でエラーが発生します。暗黙的な型変換を実施する異種 DBMS ではエラーとならない SQL でも、PostgreSQL では型の不一致でエラーが発生します。

ユニーク制約違反

MERGE 文の書き換え箇所（すでに同一の PRIMARY KEY などのユニーク制約を含む列にレコードがある場合は、そのレコードの更新(UPDATE)、無い場合はレコードの挿入(INSERT)を実施する箇所）の確認を実施します。
挿入しようとしたレコードに対して、すでに同じ値のキーを持つレコードが挿入されていた場合、以下のエラーメッセージとなります。

```
ERROR: duplicate key value violates unique constraint "[キー名]"
```

トランザクション同士のデッドロック

また、サーバログを確認し、トランザクション同士のデッドロックが頻発することがないか確認します。以下に、PostgreSQL によりデッドロックを検出した際のメッセージを示します。

7 過去の WG2 文書「SQL 移行調査編」に PostgreSQL の WITH 句を用いた MERGE 文の書き換え方法が記載されています。

```
ERROR:  deadlock detected
        DETAIL:  Process 7005 waits for ShareRowExclusiveLock
                 on relation 25123 of database 25122; blocked by process 7003.
        Process 7003 waits for ShareRowExclusiveLock
                 on relation 25127 of database 25122; blocked by process 7005.
```

データ型の不一致

PostgreSQL の場合、データ型の暗黙的な型変換を実施しないため、文字型と数値型等の比較でエラーが発生します。下記に文字型と数値型の比較を行った場合の Oracle と PostgreSQL の挙動の差異を示します。
下記の test_type テーブルの name 列は文字型です。

PostgreSQL の場合

文字型と数値型を比較した場合、下記のエラーメッセージが出力されます。

```
# SELECT name FROM test_type WHERE name = 1;
ERROR:  operator does not exist: character varying = integer
LINE 1: SELECT name FROM test_type WHERE name = 1;
                                              ^
HINT:  No operator matches the given name and argument type(s). You might need to
add explicit type casts.
```

Oracle の場合

文字型と数値型を比較した場合も、正常に SQL が処理されます。
ただし暗黙的な型変換が実施された場合、インデックスが利用されません。

```
> SELECT name FROM test_type WHERE name = 1;

NA
--
1
```

SQL Server の場合

文字型と数値型を比較した場合も、正常に SQL が処理されます。
ただし暗黙的な型変換が実施された場合、インデックスが利用されません。

```
> select name from test_type where name = 1;
> go
name
----
1
```

5.3.3. 予期せぬ SQL 実行結果の不一致

SQL 実行結果が不一致となった場合の確認観点を下表に記載します。

表 5.5: DBMS の仕様の差異確認

No.	確認観点	概要
1	ソート順序	SQL でのソート結果(ORDER BY)の差異がアプリケーションに影響を及ぼさないこと。
2	精度	桁数や精度を省略してカラム定義を実施し、アプリケーションでの演算を実施している場合に、演算結果に差異がないこと。

3	指数表記	データ型書式設定関数を使用している箇所について、移行前、移行後で SQL による取り出し結果に差異がないこと。
4	アプリケーションと DBMS の文字コードの組み合わせ	アプリケーションの実行環境にコード系が複数存在する場合は、その組み合わせについて確認すること(サーバ側で設定する client_encoding、JDBC の接続 URL の CharSet プロパティなど)
5	トランザクション分離レベル	トランザクション分離レベルに依存した作りのアプリケーションの場合、トランザクション実行中に、別のトランザクションからのデータを参照することでアプリケーションの振る舞いが変化しないこと。
6	トランザクション内での SQL エラー	PostgreSQL の場合は、トランザクション内でエラーが発生すると全てトランザクションをロールバックします。トランザクション内の成功した SQL 文のみ COMMIT するような異種 DBMS の場合と比較すると、トランザクション(SQL)の処理結果が異なります。

ソート順序

SQL 文でのソート結果(ORDER BY)をアプリケーションに渡している箇所(渡した結果をもとに画面要素を生成するなど)を確認します。

Oracle は空文字を NULL として扱います。Oracle へのデータ投入時に空文字は NULL に変換されるため、データ移行の際には空文字を含むデータは存在しませんが、運用後に空文字を PostgreSQL に投入するアプリケーションの場合には、ソート順序に注意が必要です。

5.6: 各 DBMS のソート順序差異(昇順時)

No.	データベース	ソート順序
1	PostgreSQL	「空文字, 非 NULL, NULL」の順で表示される。ただし NULLS FIRST, NULLS LAST(デフォルト)を ORDER BY 句に付与することでソート順序を制御可能。 NULLS FIRST を付与した場合は「NULL, 空文字, 非 NULL」の順で表示。
2	Oracle	「非 NULL, NULL(空文字)」の順で表示される。ただし NULLS FIRST, NULLS LAST(デフォルト)を ORDER BY 句に付与することでソート順序を制御可能。 NULLS FIRST を付与した場合は「NULL(空文字), 非 NULL」の順で表示。
3	SQL Server	「NULL, 空文字, 非 NULL」の順で表示される。

精度

「データベース移行」工程でのスキーマ定義で、DECIMAL や NUMERIC などのデータ型の列を定義し、移行後の PostgreSQL 上で、桁数や精度を省略してカラム定義を実施し、アプリケーションでの演算を実施している場合には、演算結果に差異がないか、確認します。以下に例を示します。

```
# DECIMAL 型の例
# PostgreSQL の場合
=# create table test(test decimal);
=# insert into test values (9.9);
=# select * from test;
 test
------
  9.9
(1 row)

# Oracle の場合
SQL> create table test(test decimal);
SQL> insert into test values (9.9);
SQL> select * from test;

      TEST
----------
       10

# SQL Server の場合
1> create table test(test decimal);
2> go
1> insert into test values (9.9);
2> go

(1 行処理されました)
1> select * from test;
2> go
value
----------------------
                  10
```

```
(1 行処理されました)
```

指数表記

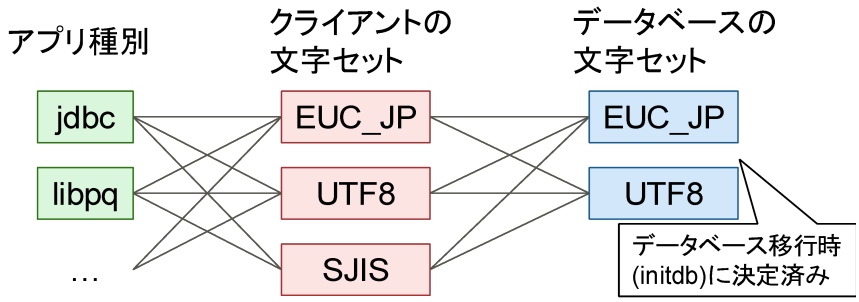
データ型書式設定関数を使用して、数値を指数表記に変換しアプリケーションに渡している箇所は、移行前、移行後で SQL による取り出し結果に差異がないか、確認します。以下に例を示します。

```
## to_char での出力形式 ##

# PostgreSQL の場合

=# select to_CHAR(123.456,'EEEE');
  1e+02

# Oracle の場合

SQL> select to_CHAR(123.456,'EEEE') from dual;
  1E+02
```

このほか、Oracle とのデータ型書式設定関数の仕様の差異が、「PostgreSQL 文書 データ型書式設定関数」にも記載されていますので参照して下さい。

アプリケーションとDBの文字コードの組み合わせ

「データベース移行」工程にて、データベースに格納するデータの文字コードを変更している場合には、テストするSQL文や取り出した結果にエラーや文字化けがないか確認します。また、アプリケーションの実行環境にコード系が複数存在する場合は、その組み合わせについて検証します。(サーバ側で設定するclient_encoding、JDBCの接続URLのCharSetプロパティなど)



トランザクション分離レベル

DBMSではトランザクション同士が相互に影響することにより下表のような不具合が生じる場合があります。

表 5.7: トランザクションの相互作用により発生する不具合

No.	不具合	概要
1	ダーティリード	他のトランザクションが更新した未コミット状態のデータを読み込んでしまう。
2	ファージーリード	トランザクション内で同じデータを2回読み込んだ時、読み込み間隔内に他のトランザクションが該当レコードの更新をコミットしたことで、1回目と2回目の読み込み結果が変わってしまう。
3	ファントムリード	トランザクション内で同一条件のデータ抽出を2回行った時、読み込み間隔内に他のトランザクションがレコードの挿入/削除をコミットしたことで、1回目と2回目の結果のデータ数が変わってしまう。

SQLの標準規格では下表のトランザクションの分離レベルが定義されています。

表 5.8: トランザクション分離レベル

No.	トランザクション分離レベル	概要
1	READ UNCOMMITTED	同時に実行されている他のトランザクションがコミットしていないデータであっても読み込んでしまう。
2	READ COMMITTED	同時に実行されている他のトランザクションがコミットしたデータを読み込んでしまう。
3	REPEATABLE READ	他のトランザクションでコミットされた更新を参照しないことで、同じレコードを繰り返し読んでも常に同じ結果が得られることを保証する。
4	SERIALIZABLE	他のトランザクションの影響を受けない。 トランザクション内で参照したデータを他のトランザクションでは変更することができない。また、参照に使用した条件に一致するデータを挿入することもできなくなる。

各DBMSに実装されたトランザクション分離レベルおよび、各トランザクション分離レベルで発生する不具合を下表に記載します。下表にはSQL標準規格でされたトランザクション分離レベルのみを記載しています。
下表以外のトランザクション分離レベルを実装している異種DBMSも存在します。

表 5.9: 各DBMSのトランザクション分離レベルと発生する不具合

No.	トランザクション分離レベル	Oracle	SQL Server	PostgreSQL	発生する不具合		
					ダーティリード	ファージーリード	ファントムリード
1	READ UNCOMMITTED	-	○	-[8]	発生	発生	発生
2	READ COMMITTED	○ (デフォルト値)	○ (デフォルト値)	○ (デフォルト値)	抑止	発生	発生

8 PostgreSQLでは、READ UNCOMMITTEDはREAD COMMITTEDとして扱われます。

3	REPEATABLE READ	-	○	○	抑止	抑止	発生
4	SERIALIZABLE	○	○	○	抑止	抑止	抑止

移行前後でDBMSのトランザクション分離レベルをそろえた場合でもDBMS毎にトランザクション分離レベルの実装が異なるため、挙動が異なる場合があります。
下表の処理を実行した場合のOracleとPostgreSQLの挙動の差異を例示します。
下記はREAD UNCOMMITTEDを前提に記載しています。

表 5.10: READ UNCOMMITTEDの動作差異確認方法

No.	SESSION①	SESSION②
1	トランザクション開始(BEGIN)	-
2	行をロック(SELECT FOR UPDATE)	-
3	-	トランザクション開始(BEGIN)
4	-	行をロック(SELECT FOR UPDATE)
5	ロックした行を削除(DELETE)	-
6	トランザクション終了(COMMIT)	ロック待ち解放 (PostgreSQLとOracleで表示されるデータが異なる)

表 5.11: 確認に利用するテーブル(testテーブル)

No.	列名	型
1	id	数値型
2	data	文字型

testテーブルには下表のデータが格納しています。

表 5.12: 確認に利用するテーブルに格納したデータ(testテーブル)

No.	id列	data列
1	1	a
2	2	b
3	3	c

PostgreSQLの場合

SESSION1にて行ロック後に削除されたレコードは、SESSION2では処理対象とならず該当のデータは返却されません。

```
SESSION1
 =# BEGIN;
 BEGIN

 =# SELECT id, data FROM test WHERE id = (SELECT MIN(id) FROM test) FOR UPDATE;
  id | data
 ----+------
   1 | a
 (1 行)
```

```
SESSION2
 =# BEGIN;
 BEGIN

 =# SELECT id, data FROM test WHERE id = (SELECT MIN(id) FROM test) FOR
UPDATE;
     (ロック待ち)
```

SESSION1

```
=# DELETE FROM test WHERE id = 1;
DELETE 1

=# COMMIT;

COMMIT
```

SESSION2

（SESSION2 がロック待ちから解放される）

```
 id | data
----+------
(0 行)
```

Oracle の場合

SESSION2 からの見え方が PostgreSQL とは異なり、SESSION1 にて行ロック後に削除した行が SESSION 2 でも削除されているように見えます。

SESSION1

```
> SELECT id, data FROM test WHERE id = (SELECT MIN(id) FROM test) FOR UPDATE;
        ID DATA
 ---------- ----------
         1 a
```

SESSION2

```
> SELECT id, data FROM test WHERE id = (SELECT MIN(id) FROM test) FOR UPDATE;
  （ロック待ち）
```

SESSION1

```
> DELETE FROM test WHERE id = 1;
 1 row deleted.

> COMMIT;
 Commit complete.
```

SESSION2

（SESSION2 がロック待ちから解放される）

```
                ID DATA
          ---------- ----------
                 2 b
```

トランザクション内部でのエラー

Oracle など、成功した SQL 文のみ COMMIT する DBMS の場合と結果が異なることで後続の処理に影響がある場合があります(PostgreSQL の場合は、全てロールバックします)。

また、PostgreSQL でも、ODBC ドライバの文単位ロールバック機能(SQL 文の実施単位で SAVEPOINT を発行する)を有効にすると、文単位で SQL 文を COMMIT することが可能となります。

その他、DBMS による挙動の違いについては、過去の WG2 文書「SQL 移行調査編」を参照してください。

5.4. 業務バッチ試験

業務バッチ試験を実施する際の確認観点を以下に記載します。

本資料における業務バッチとは参照・更新処理を含む複数 SQL 文をデータベースに対して実行するプログラムを指します。埋め込み SQL などのバッチ処理が存在する場合、SQL 文試験としてではなく、バッチ処理単位で試験を行うべきと考えました。

表 5.13: バッチ試験の確認観点

No.	確認項目	概要
1	バッチの実行結果の正しさ	・ 更新処理を含む業務バッチの場合 業務バッチ実行前後のテーブルのデータが異種 DBMS と PostgreSQL で同一であることを確認します。 ・ 参照処理を含む業務バッチの場合 業務バッチで出力された情報(CSV ファイル等)が、異種 DBMS と PostgreSQL で同一であることを確認します。
2	バッチ内で投入される SQL 文が明確になっていること	「5.5 画面操作試験」と同様に、SQL 文の実行ログを参照し意図した SQL 文が実行されていることを確認します。
3	バッチ内で実行される SQL 文が想定される実行計画に基づいて実行されているか確認できること	必要に応じ、EXPLAIN 文、EXPLAIN ANALYZE 文を実行し、意図した実行計画で SQL 文が実行されていることを確認します。

上記観点を確認する際には、バッチ処理によって影響を受けるデータの範囲を確認しておく必要があります。

日時バッチ、月次バッチなどを実施する際には、バッチ処理の対象とするデータの範囲、データ量そのものを、可能な限り実際の業務とそろえるようにします。

対象のデータの範囲やデータ量が少ない場合、実際の業務と試験で SQL 文の実行計画が乖離する可能性があります(データ量が少ないために、本来使用するはずのインデックスを使用しないなど)。

5.5. 画面操作試験

画面操作試験の確認観点を以下に記載します。

1. 画面遷移結果が移行元 DBMS 上で動作していたアプリケーションと同一であること
2. 画面に出力される情報が移行元 DBMS 上で動作していたアプリケーションと同一であること
3. 画面遷移の契機で、移行した SQL 文が想定どおり実行されていること

上記を確認するために、以下試験を実施します。

表 5.14: 画面確認試験の手順

No.	項目	概要
1	移行元 DBMS 上で動作するアプリケーションの画面確認試験の実施	Selenium 等の画面確認試験ツールを使用し、移行元 DBMS 上で動作するアプリケーションに対し画面試験を実施する。
2	PostgreSQL 上で動作するアプリケーションの画面確認試験の実施	Selenium 等の画面確認試験ツールを使用し、移行先である PostgreSQL 上で動作するアプリケーションに対し画面試験を実施する。
3	画面確認試験結果の比較	画面の遷移結果及び画面の出力内容がどちらも同一となることを確認します。

また 3 つ目の観点「画面遷移の契機で、移行した SQL 文が想定どおり実行されていること」を確認するために、以下の確認準備を実施しておく必要があります。

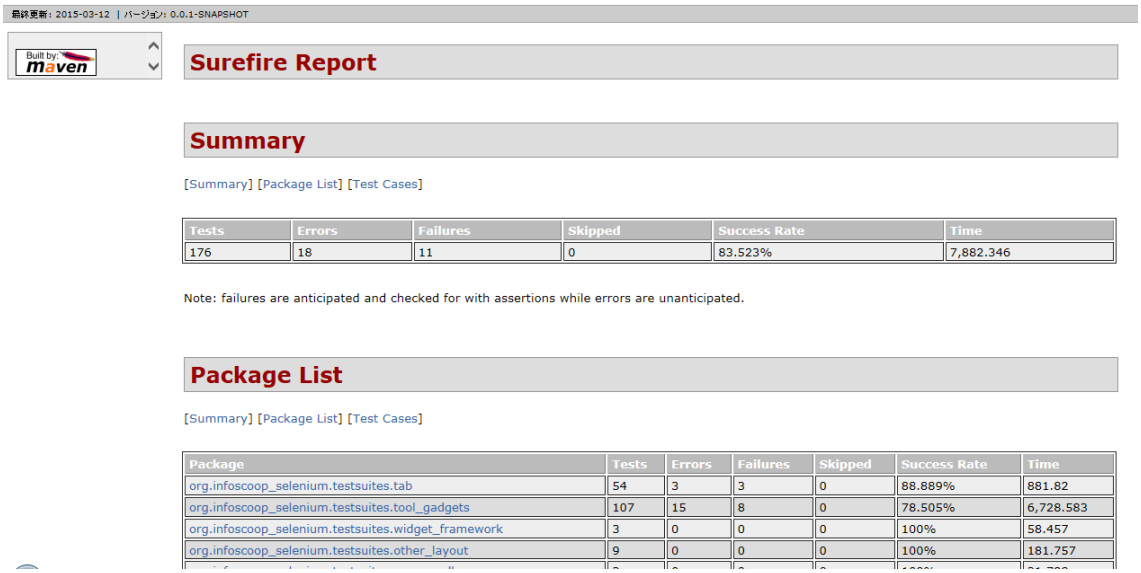
表 5.15: 試験準備

No.	項目	概要
1	試験クライアントと DB サーバの時刻同期	試験クライアント（Selenium 等）による画面を操作（ボタンのクリック、タブのクリックなど）と実行された SQL 文を対応付けるため、、試験クライアントの時刻と DB サーバの時刻が揃っている必要があります。 ※ NTP 等でサーバ間の時刻同期を行うとより確実です。
2	試験クライアント以外からのアクセス遮断	試験中に試験クライアント以外の他のアプリケーションからのアクセスが発生すると、実行した SQL 文の特定と画面遷移を結びつけることが困難となります。そのため影響を受けないようにアクセスを遮断しておく必要があります。
3	実行される SQL 文の確認	画面遷移の契機で実行されるであろう SQL 文を予めソースコードから確認する必要があります。 O/R マッパーを使用したアプリケーションの場合には、試験結果によりアプリケーションソース内での SQL 文の実行箇所を特定する必要がある場合があります。（本ドキュメントでは、Hibernate での SQL 文の投入メソッドの例を「5.6.2O/R マッパーを使用したアプリケーションについて」に記載しています）

5.5.1. 画面操作試験の実施

本資料では画面確認試験ツールとして Selenium を利用した確認方法を紹介します。

以下は Selenium を用いて画面遷移試験を実施した際に出力されるテスト実行結果レポートの例になります。



最終更新: 2015-03-12 | バージョン: 0.0.1-SNAPSHOT

Built by maven

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
176	18	11	0	83.523%	7,882.346

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
org.infoscoop_selenium.testsuites.tab	54	3	3	0	88.889%	881.82
org.infoscoop_selenium.testsuites.tool_gadgets	107	15	8	0	78.505%	6,728.583
org.infoscoop_selenium.testsuites.widget_framework	3	0	0	0	100%	58.457
org.infoscoop_selenium.testsuites.other_layout	9	0	0	0	100%	181.757

Selenium では、テストシナリオに沿って意図した画面遷移がなされることが確認できます。移行前後の DBMS でのテストの実行レポートを比較することで、アプリケーションの振る舞いの差分を抽出できます。
※ Selenium の使用方法については infoScoop 移行手順(別紙)を参照してください。

画面遷移に伴って投入された SQL 文は、サーバログから確認します。

```
$ cat ${PGDATA}/pg_log/postgresql-2015-03-12_141641.csv

...(略)...

2015-03-12 14:16:42.233
JST,"postgres","infoscoop",21337,"127.0.0.1:50991",550116c3.5359,18908,"PARSE",2015-03-12 13:32:03
JST,5/17053,0,LOG,00000,"duration: 0.064 ms  parse <unnamed>: select this_.ID as ID20_0_,
this_.""UID"" as UID2_20_0_, this_.ROLEID as ROLEID20_0_, this_.Roleid as Roleid20_0_ from
public.IS_PORTALADMINS this_ order by this_.ID asc",,,,,,,,,""
2015-03-12 14:16:42.233
JST,"postgres","infoscoop",21337,"127.0.0.1:50991",550116c3.5359,18909,"BIND",2015-03-12 13:32:03
JST,5/17053,0,LOG,00000,"duration: 0.069 ms  bind <unnamed>: select this_.ID as ID20_0_,
this_.""UID"" as UID2_20_0_, this_.ROLEID as ROLEID20_0_, this_.Roleid as Roleid20_0_ from
public.IS_PORTALADMINS this_ order by this_.ID asc",,,,,,,,,""
2015-03-12 14:16:42.233
JST,"postgres","infoscoop",21337,"127.0.0.1:50991",550116c3.5359,18910,"SELECT",2015-03-12 13:32:03
JST,5/17053,0,LOG,00000,"execute <unnamed>: select this_.ID as ID20_0_, this_.""UID"" as UID2_20_0_,
this_.ROLEID as ROLEID20_0_, this_.Roleid as Roleid20_0_ from public.IS_PORTALADMINS this_ order by
this_.ID asc",,,,,,,,,""

...(略)...
```

移行前後の DBMS で、同様の SQL 文が投入されていることを確認します。Oracle での確認方法を以下に示します。

```
$ cat ${Oracle の SQL トレースログの出力先}/[ログファイル名].trc

...(略)...
=====================
PARSING IN CURSOR #140371815759552 len=166 dep=0 uid=50 oct=3 lid=50 tim=1426225319766987
hv=3669853992 ad='6a7dc5c8' sqlid='c7rpj6gdbv1t8'
select this_.ID as ID20_0_, this_."UID" as UID2_20_0_, this_.ROLEID as ROLEID20_0_, this_.Roleid as
Roleid20_0_ from scott.IS_PORTALADMINS this_ order by this_.ID asc
```

```
END OF STMT
PARSE #140371815759552:c=0,e=56,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=543897239,tim=1426225319766986
EXEC #140371815759552:c=0,e=14,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=543897239,tim=1426225319767043
FETCH #140371815759552:c=0,e=50,p=0,cr=2,cu=0,mis=0,r=6,dep=0,og=1,plh=543897239,tim=1426225319767128
STAT #140371815759552 id=1 cnt=6 pid=0 pos=1 obj=20337 op='TABLE ACCESS BY INDEX ROWID
IS_PORTALADMINS (cr=2 pr=0 pw=0 time=53 us cost=2 size=114 card=6)'
STAT #140371815759552 id=2 cnt=6 pid=1 pos=1 obj=20338 op='INDEX FULL SCAN SYS_C007255 (cr=1 pr=0
pw=0 time=32 us cost=1 size=0 card=6)'
CLOSE #140371815759552:c=0,e=4,dep=0,type=1,tim=1426225319767599
=====================
...(略)...
```

5.6. 移行検証

infoScoop 上での移行評価の手順を別紙に掲載しています。

また、本節では、infoScoop 特有の内容として、O/R マッパーを使用したアプリケーションについて、SQL 移行試験の観点で補足情報を記載します。

5.6.1. infoScoop の特徴

まず infoScoop の概要を記載します。infoScoop は、直感的で分かりやすい操作性が特徴のオープンソースの企業情報ポータルです。

言語	Java
コード規模	69Ks (Java+SQL)
対応データベース	MySQL 5.1、Oracle 11g,12c、DB2 9.7,10.1,10.5

表 5.16: infoscoop 概要

5.6.2. O/R マッパーを使用したアプリケーションについて

infoScoop では、DBMS での SQL 文の差異を吸収するために、O/R マッパーを使用しています。

Hibernate などの O/R マッパーを使用したアプリケーションでは、アプリケーション内部で SQL 文を抽象化し、方言(Dialect)ファイルと呼ばれるファイルを元に、各 DBMS に向けて SQL 文を最適化して投入しています。

単体テストのテスト仕様を作成する際に、SQL 文の実行箇所の洗い出しが必要な場合には O/R マッパーによる SQL 文投入メソッドを調査し、SQL 文の投入ソースおよび投入箇所を特定します。

以下に、Hibernate を使用した場合の例を記載します。

クラス名	メソッド名	メソッドの説明/対応する SQL 文
org.hibernate.Session	save	INSERT
	saveOrUpdate	INSERT / UPDATE 文(MERGE 文)
	delete	DELETE 文
	update	UPDATE 文
	get	SELECT 文
org.hibernate.Criteria	list	SELECT 文
org.hibernate.Query	list	SELECT 文
	createQuery	WHERE 句の生成
	createSQLQuery	SQL 文の生成

組み込み関数の書式/引数の違いなどは、各 DBMS 固有の SQL 記述を O/R マッパーにて吸収します。O/R マッパー側で対応している DBMS であれば、SQL 文の移行は必要ありませんが、実行結果やアプリケーションの作りにより移行前の DBMS と動作が異なる場合があります。

Spring などのフレームワークにより、Hibernate の DB アクセスメソッドを呼び出していることがあります。そのため、O/R マッパーのメソッド名だけでなく、フレームワークのドキュメントも参照します。

(例 : org.springframework.orm.hibernate3.HibernateTemplate などのクラス)[9]

9 Spring フレームワークにおける Hibernate 関連クラスの詳細は、以下を参照して下さい。
http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/orm/hibernate3/HibernateTemplate.html

6. 性能試験

本章では、性能試験の手順を記述します。

6.1. 目的

テーブル定義、データおよびアプリケーションを移行後、異種 DMBS で求められた性能要件を PostgreSQL が満たすことを確認します。性能要件は、要件書や設計書から抽出し、実現すべき性能目標として抽出しておく必要があります。

6.2. 前提条件

本章で紹介する試験は以下を作業の前提とします。

表 6.1: 前提条件

No.	前提条件	備考
1	性能要件が明確になっていること	基本的には異種 DBMS に求められた性能要件を満たすことを目標とします。
2	性能試験シナリオが準備されていること	異種 DBMS の性能試験時に利用した試験シナリオを利用します。 試験シナリオない場合は作成して下さい。
3	性能要件をユーザと合意していること	「6.3 試験対象と観点」に記載する観点で、ユーザと合意して下さい。

6.3. 試験の観点

事前に用意した性能試験ナリオを実行した際に下表の観点で結果を確認します。

表 6.2: 試験観点

No.	観点	概要
1	応答時間	・SQL の目標応答時間／応答時間の上限 ・トランザクションの目標応答時間／応答時間の上限 ・単位時間当たりのトランザクション処理件数
2	単位時間あたりの処理件数	・単位時間当たりのトランザクション処理件数
3	消費するハードウェア・OS リソース	・ CPU ・ メモリ ・ ディスク領域 ・ ネットワーク等

6.4. 試験の実行

以下の手順を実施したうえで、「6.2 前提条件」に記載の通り、性能試験シナリオを実行します。
試験の実行により、下記を確認できるようにしておく必要があります。

1. 「6.3 試験の観点」における「目標応答時間/応答時間の上限」を超える SQL 文(時間のかかる SQL 文)を確認する。
2. SQL 文に時間のかかる原因の特定ができるようにしておく。原因の例を以下に示します。
    - インデックスによる絞込みが有効になっているか
    - 共用バッファなどの DB リソースが使用できているか
    - CPU、I/O など OS リソースが使用できているか
    - VACUUM、CHECKPOINT、WAL 書き込みなど DBMS のアーキテクチャによるネックとなっていないか。
        - PostgreSQL の独自アーキテクチャによる負荷を与えた状態でどの程度、性能が劣化するのかを確認することを推奨します。(VACUUM 実行時の性能劣化等)

6.4.1. SQL 文の性能情報の取得

pg_stat_statements を使用し、投入された SQL 文の実行時間、実行回数の情報を取得できるようにしておきます。

- shared_preload_libraries の設定
  DB クラスタ起動時に pg_stat_statements のライブラリが読み込まれるようするため、postgresql.conf を修正します。修正後は、DB クラスタを再起動します。

```
shared_preload_libraries = 'pg_stat_statements'
```

- 性能情報の初期化
  性能測定の実施前に、蓄積された性能情報を初期化しておきます。

```
$ psql -U postgres -c "select pg_stat_statements_reset();"
```

6.5. 試験結果の確認

「6.3 試験の観点」に沿って、要件を満たしていない場合は、性能ボトルネックの抽出、分析、性能チューニングを実施します。過去の WG2 文書「チューニング編」も併せて参照してください。

6.5.1. 使用リソースの確認

- データベースへのアクセス多重度や、データベースから取得するデータ量や大量のレコードを対象にして集計を実施する業務がある場合には、移行前と比較してリソースの使用量に差異が無いか、確認します。
- 特にデータベースの共有バッファの使用率、PostgreSQL 特有の観点として、autovacuum が正しく実行されていることを確認します。以下の SQL 文により、特定のスキーマ、テーブルにおける VACUUM/autovacuum が実施された回数を取得できます。

```
-- VACUUM の実行回数
=# select vacuum_count from pg_stat_user_tables where schemaname = [スキーマ名] and relname =
[テーブル名];
-- autovacuum の実行回数
=# select autovacuum_count from pg_stat_user_tables where schemaname = [スキーマ名] and relname =
[テーブル名];
```

- 共用バッファの使用率
  - pg_stat_database、pg_stat_user_tables、pg_statio_user_tables のシステムカタログの blks_hit, blks_read をもとに算出します。以下に、pg_stat_database を使用した場合の算出方法について記載します。

```
=# SELECT datname, blks_hit, blks_read, round(blks_hit*100/(blks_hit+blks_read), 2) AS hit_rate
from pg_stat_database where datname='[データベース名]' AND blks_read > 0;
```

6.5.2. SQL 文の実行時間の確認

性能測定の実施後に、pg_stat_statement テーブルを参照し、SQL 文の実行情報を取得します。
すべてのユーザが実行した SQL 文が表示するために、PostgreSQL でのスーパーユーザで実行します。
また、pg_stat_statements で計上されている時間は、SQL 処理フェーズの時間であり、構文解析やデータのバインド処理にかかる時間は含まれないことに注意してください。

```
=# ¥x
Expanded display is on.
=# SELECT pg_get_userbyid(userid) AS username, (SELECT datname FROM pg_database WHERE oid =
dbid) AS dbname, substring(query from 0 for 100), calls, total_time, rows FROM
pg_stat_statements order by total_time desc limit 10;
-[ RECORD
1 ]----------------------------------------------------------------------------------------------
---------
username   | postgres
dbname     | infoscoop
substring  | select this_."UID" as UID1_21_0_, this_.SESSIONID as SESSIONID21_0_,
this_.LOGINDATETIME as LOGINDA
calls      | 11
total_time | 196.723
rows       | 0
-[ RECORD
2 ]----------------------------------------------------------------------------------------------
---------
username   | postgres
dbname     | infoscoop
substring  | select distinct widget0_.TYPE as col_0_0_ from public.IS_WIDGETS widget0_ where
widget0_."UID"=$1 a
calls      | 1
```

```
total_time | 41.639
rows       | 4

...(略)...
```

6.5.3. DBMS の振る舞いの確認

以下の 3 つの観点で、必要に応じて DBMS の振る舞いをチューニングします。

- VACUUM
  - autovacuum 動作にかかるリソース(CPU 等)の使用量
    autovacuum に割り当てるメモリ量や(maintenance_work_mem)、割り当てプロセス数(autovacuum_max_workers)により調整します。
  - 無効レコードの増加によるリソース(I/O)の使用量
    データベースへの負荷状況などにより、autovacuum が動作せず無効レコードが増加し、目的のレコードを探す上でのレスポンスが劣化する場合があります。autovacuum の必要性の確認周期(autovacuum_naptime)、vacuum を動作させる上での閾値(autovacuum_vacuum_threshold, autovacuum_vacuum_scale_factor)により調整します。
  - autovacuum の使用頻度
    当該スキーマ、当該テーブルにおける autovacuum の実行回数を確認し、アプリケーションによるテーブルの更新頻度/更新レコード数に対して適切か確認します。以下の SQL 文により、特定のスキーマ、テーブルにおける更新/挿入/削除された行数を取得できます。

```
-- 挿入された行数
=# select n_tup_ins from pg_stat_user_tables where schemaname = [スキーマ名] and relname
= [テーブル名];
-- 更新された行数
=# select n_tup_upd from pg_stat_user_tables where schemaname = [スキーマ名] and relname
= [テーブル名];
-- 削除された行数
=# select n_tup_del from pg_stat_user_tables where schemaname = [スキーマ名] and relname
= [テーブル名];
```

  - VACUUM/autovacuum が実施された回数については「6.5.1 使用リソースの確認」を参照してください。

- CHECKPOINT
  - OS のリソースの利用状況のうち、I/O 負荷が大きい場合には、CHECKPOINT 時の I/O 負荷が原因の可能性があります。
    - checkpoint_segments を大きくし、頻繁に CHECKPOINT が発生しないようにします。頻繁に CHECKPOINT が発生している場合には、以下のメッセージがログに出力されます。ただし、大きく設定すると WAL 書き出しのためのディスク領域が増加する点、データベースのリカバリの際に時間がかかるようになる点に注意が必要です。

```
LOG: checkpoints are occurring too frequently
```

    - checkpoint_completion_target の値を調節し、CHECKPOINT の書き出し負荷を次の CHECKPOINT までの間に分散します。

- WAL の書き込み競合
  - WAL の書き込み処理は直列化されるため、更新業務が多重度の高い状態で実行されると、WAL の書き込みが競合し、特にオンライン業務での単位時間当たりのトランザクション処理量や CPU の使用率が向上しないことがあります。
  - 単位時間当たりの WAL の書き込みサイズは、以下のように計算します。この数値が WAL の格納先ディスクにおける単体での書き込み性能限界に近い場合は、WAL の書き込み競合がネックとなっている可能性があります。

```
--ある時点でのログの位置を算出
=# SELECT 'xlog' as xlog, pg_current_xlog_insert_location() as xlog_insert_loc;

 xlog | xlog_insert_loc
 -----+-----------------
 xlog | 8/2E000020

--次の時点でのログの位置を算出

=# SELECT 'xlog' as xlog, pg_current_xlog_insert_location() as xlog_insert_loc;

 xlog | xlog_insert_loc
 -----+-----------------
 xlog | 8/30625298

--比較
「8/2E000020」と「8/30625298」を比較します。
「/」以降の数字を16進数として引き算し、10進数にします。
30625298(16進数) - 2E000020(16進数) = 39998072(10進数：byte)
が、WALの書き込み量になります。

※「/」より前の数字が繰り上がっている場合は、FFFFFFFFを足しこんでから比較します。
(例)8/30625298 と 9/00001234を比較する場合

00001234 + FFFFFFF - 30625298として計算します。

また下記のSQLでも確認することができます。
=# select '8/30625298'::pg_lsn - '8/2E000020'::pg_lsn;
 ?column?
----------
 39998072
(1 row)
```

7. まとめ

本資料では異種 DBMS を PostgreSQL に移行した際に、移行結果が妥当であるか否かを確認する試験とその手順について記載しました。過去の WG2 文書「アプリケーション移行実践編」に記載されている通り[10]、移行作業の大半は試験が占めると考えます。本年度の活動でも試験内容の検討や試験手順の確立に多くの時間を要しました。
下記に各試験項目を記載した所感を記載しています。

・ スキーマ移行結果確認試験

本資料で記載した試験手順では、異種 DBMS および PostgreSQL に格納されたスキーマオブジェクトの情報を試験用のテーブルに格納し、SQL で比較し試験結果を判定しています。上記確認を行うにはスキーマオブジェクトの各情報がどのシステムカタログに保持されているか等の知見が必要となり、異種 DBMS および PostgreSQL の両ソフトウェアに対する調査が必要でした。

・ データ移行結果試験

異種 DBMS から出力した CSV 形式のデータと、PostgreSQL から出力した CSV 形式のデータの比較を行いデータに差分がないことを確認しています。データの比較であれば短時間で完了すると考えていましたが、異種 DBMS と PostgreSQL でデータの出力形式に差異がありました。また、発生した差異に対する原因調査に想定した以上に時間がかかりました。
DBMS のデータ出力ツールは独自の実装が行われているため、PostgreSQL および異種 DBMS のツールに対する調査が必要となりました。

・ アプリケーション移行確認試験

DBMS 移行前後のアプリケーションの動作が変わらないことを確認する試験を記載しました。
今回、移行対象とした infoScoop のように試験セット(infoScoop の場合、「infoScoop-selenuim」)が提供されている場合は、試験セットを用いて、DBMS 移行前後でアプリケーションの動作が変わらないことを確認することが可能です。
既存の試験セットが存在しない場合は、新規アプリケーション開発時と同様に全ての機能を網羅的に確認することが理想的です。ただし、スケジュール等のプロジェクトの状況により、網羅的な機能試験が難しい場合は、ユーザと試験範囲について事前に合意を得る必要があると考えます。(DBMS を移行する多くのプロジェクトは、コスト削減を目的としているため移行作業にコストをかけられない場合が多いです。)

・ 性能試験

異種 DBMS で求められた性能要件を PostgreSQL が満たすことを確認する試験について記載しました。
異種 DBMS で求められた性能要件を満たすために PostgreSQL において注意すべて挙動(VACUUM 等)をいくつか記載しています。PostgreSQL 独自の動作(VACUUM 等)の実行中に、どの程度の性能劣化があるのかを確認することを推奨します。また、本資料では性能チューニングと関わりの深い性能情報収集(監視)や統計情報のメンテナンスについてはあまり触れませんでした。運用関係の調査、検証を行う PostgreSQL エンタープライズコンソーシアム・WG3 の成果も合わせてご覧ください。

本資料に記載した試験内容で移行ミスや移行漏れを防ぎ、PostgreSQL への移行をより確実に実現できれば幸いです。

10 「アプリケーション実践実践編」の p22 には下記の記載があります。
今回の移行作業では、作業時間の大半が SQL 文の動作確認 – テストフェーズとなっており、全体の 90%以上を占めています。

8. おわりに

本資料では異種 DBMS から PostgreSQL への移行が正しく完了しているか確認を実施する際の観点や確認方法について記載しました。本書を参考に移行試験を行うことにより、基本的なオブジェクトについては確実な移行を実施できるようになったと考えております。

しかし、ストアドプロシージャや関数など、ロジックを含むオブジェクト等の移行難易度の高いオブジェクトの移行試験については本年度の記載は見送らせていただいているため、課題としては依然として残されています。

来年度以降も下記課題の解決、あるいは別の切り口から PostgreSQL への移行を促進するための調査及び検証を継続して行っていきたいと考えております。

本資料をご覧になって WG2 及び PostgreSQL エンタープライズ・コンソーシアムの活動に関心を持たれた方は是非ご参加いただき、共に PostgreSQL の調査検証を進めていきたいと考えております。宜しくお願い致します

表 8.1: 課題一覧

No.	課題	概要
1	組み込み関数移行確認	本資料では記載を見送りました。 ロジックを含むため DBMS 間の差異が大きいと思われるため、移行確認が難しいオブジェクトであると考えています。
2	ストアドプロシージャ移行確認	
3	DB利用者移行確認	DBMS 毎に概念が異なるため移行が難しいと考えられます。まずはどのように置き換えるべきか検討する必要があります。
4	利用権限移行確認	

著者

版	所属企業・団体名	部署名	氏名
試験編 第1版 （2014年度 WG2）	日本電信電話株式会社	オープンソースソフトウェアセンタ	邊見 均
	NECソリューションイノベータ株式会社	PFシステム事業部	岩浅 晃郎
	富士通株式会社	データマネジメント・ミドルウェア事業部	山本 明範
			山本 貢嗣
			榎本 友理枝
	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	杉山 貴洋
			小山田 政紀
			高橋 勝平