OSS-DB

OSS-DB Silver
技術解説無料セミナー
2021/06/29 開催

主題	運用管理（52%）
副題	設定ファイル 【重要度：5】

本日の講師

FUJITSU

富士通株式会社
長尾 正樹

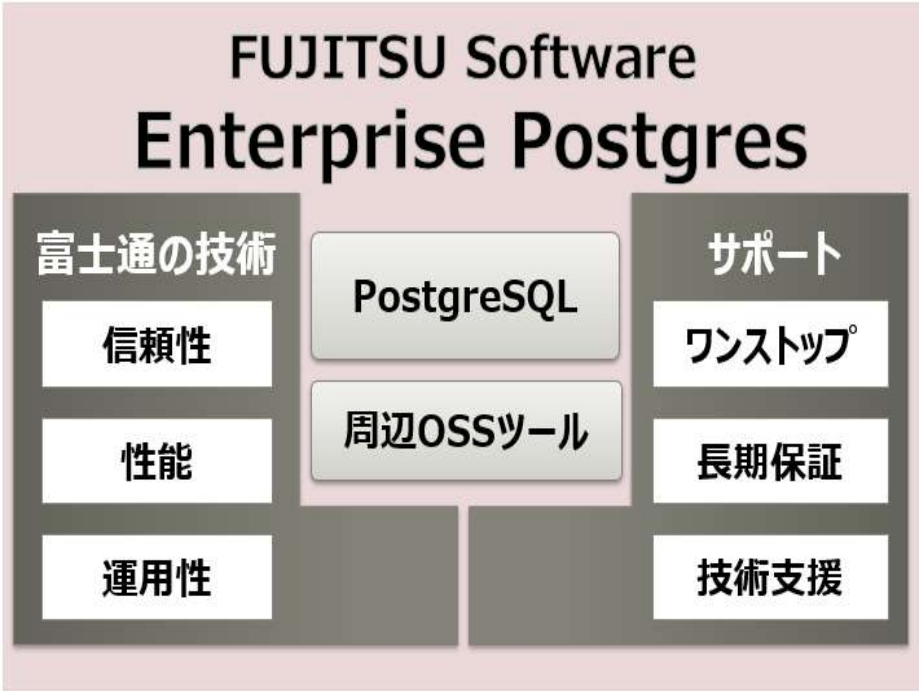
LPI-JAPAN

講師の紹介 (1/2)

■長尾 正樹

#OSS-DB

- 2012年より、富士通のPostgreSQLデータベース「FUJITSU Software Enterprise Postgres」の主にデータベース移行アセスメント業務に従事。
- 現在は、プロモーションや、PostgreSQL開発者向けに「PostgreSQL技術インデックス」の執筆を通して、PostgreSQLのポテンシャル・魅力をお伝えしている。



PostgreSQLインサイド https://www.fujitsu.com/jp/products/software/resources/feature-stories/postgres/
Enterprise Postgres https://www.fujitsu.com/jp/products/software/middleware/database/enterprisepostgres/

講師の紹介 (2/2)

■富士通では、導入検討から開発、運用まで
レベルに合わせたトレーニングをご用意

#OSS-DB

フェーズ	eラーニング		講習会	
導入検討・設計	富士通のデータベース概説	初級	PostgreSQL 導入トレーニング	初級
	PostgreSQL技術者のための 富士通のデータベース適用	中級		
開発・構築	実習で学ぶ PostgreSQLによるSQL開発	初級	PostgreSQLクラスタシステム 構築トレーニング	上級
	富士通版PostgreSQLによる データベース構築と運用のポイント	初級		
運用・保守	実習で学ぶ PostgreSQL DB構築と運用操作	上級	PostgreSQL 運用管理トレーニング	中級

フェーズと講座の関係は、一例です。上記ラインナップは、データベーストレーニングの一部です。
詳細は、「 https://www.fujitsu.com/jp/software/resources/training/ 」ご覧ください。

OSS-DBとは

#OSS-DB

オープンソースのデータベースソフトウェア「PostgreSQL」を扱うことができる技術力の認定です。様々な分野でPostgreSQLの利用拡大が進む中でOSS-DBの認定を持つことは、自分のキャリアのアピールにもつながります。

- ✓OSS-DB Goldは設計やコンサルティングができる技術力の証明

  PostgreSQLについての深い知識を持ち、データベースの設計や開発のほか、パフォーマンスチューニングやトラブルシューティングまで行えることが証明できます

- ✓OSS-DB Silverは導入や運用ができる技術力の証明

  PostgreSQLについての基本的な知識を持ち、データベースの運用管理が行えるエンジニアとしての証明ができます

- ✓対象のバージョンはPostgreSQL 11

今回のテーマ

■PostgreSQLのしくみ(アーキテクチャ)を知り、動作をイメージする
- 運用時に知っておいて欲しいしくみについて説明する

#OSS-DB

■PostgreSQLを最適に動作するよう設定する
- PostgreSQLは、様々な運用規模・形態に対応できるよう、
  多くの設定パラメータで調整できる作りになっている
- Silverの出題範囲をやや超える部分もあるが、
  しくみと動作をイメージすることが、PostgreSQLの理解を深め、
  応用が利くようになる

■その他
- 実施例については、以下の環境で確認した
  - Red Hat Enterprise Linux 7.7
  - PostgreSQL 11.12

目次

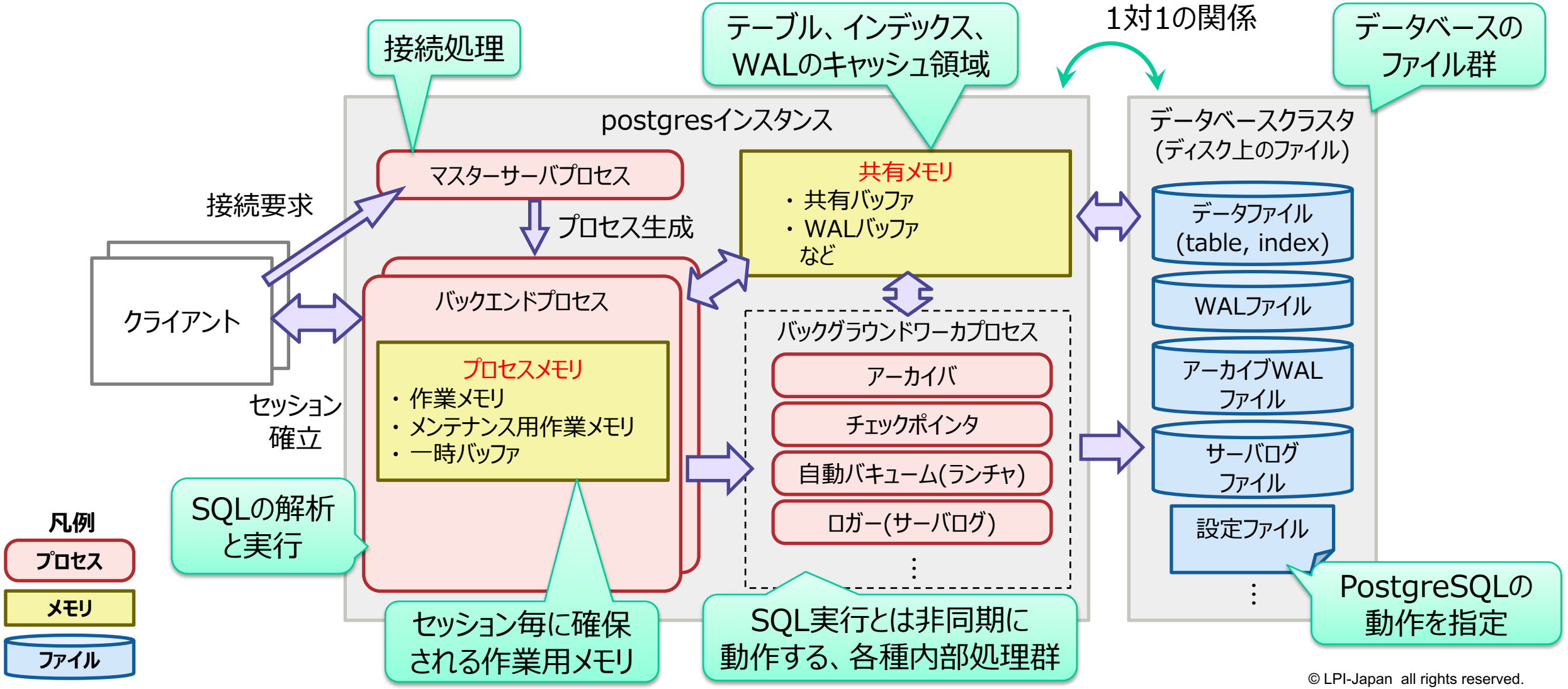
- PostgreSQLの基本構成（概要）
- PostgreSQLを利用するまでの４ステップ
  - インストール
  - クラスタ作成
  - PostgreSQLサーバ起動
  - 接続と利用
- 設定ファイル(postgresql.conf)について
- 運用時に意識すべきPostgreSQLのしくみとパラメータ設定
  - 接続と認証
  - データアクセス
  - VACUUM(バキューム)とANALYZE(アナライズ)
  - ロガー

#OSS-DB

PostgreSQLの基本構成

PostgreSQLを構成する要素（概要）※Silver関連の範囲

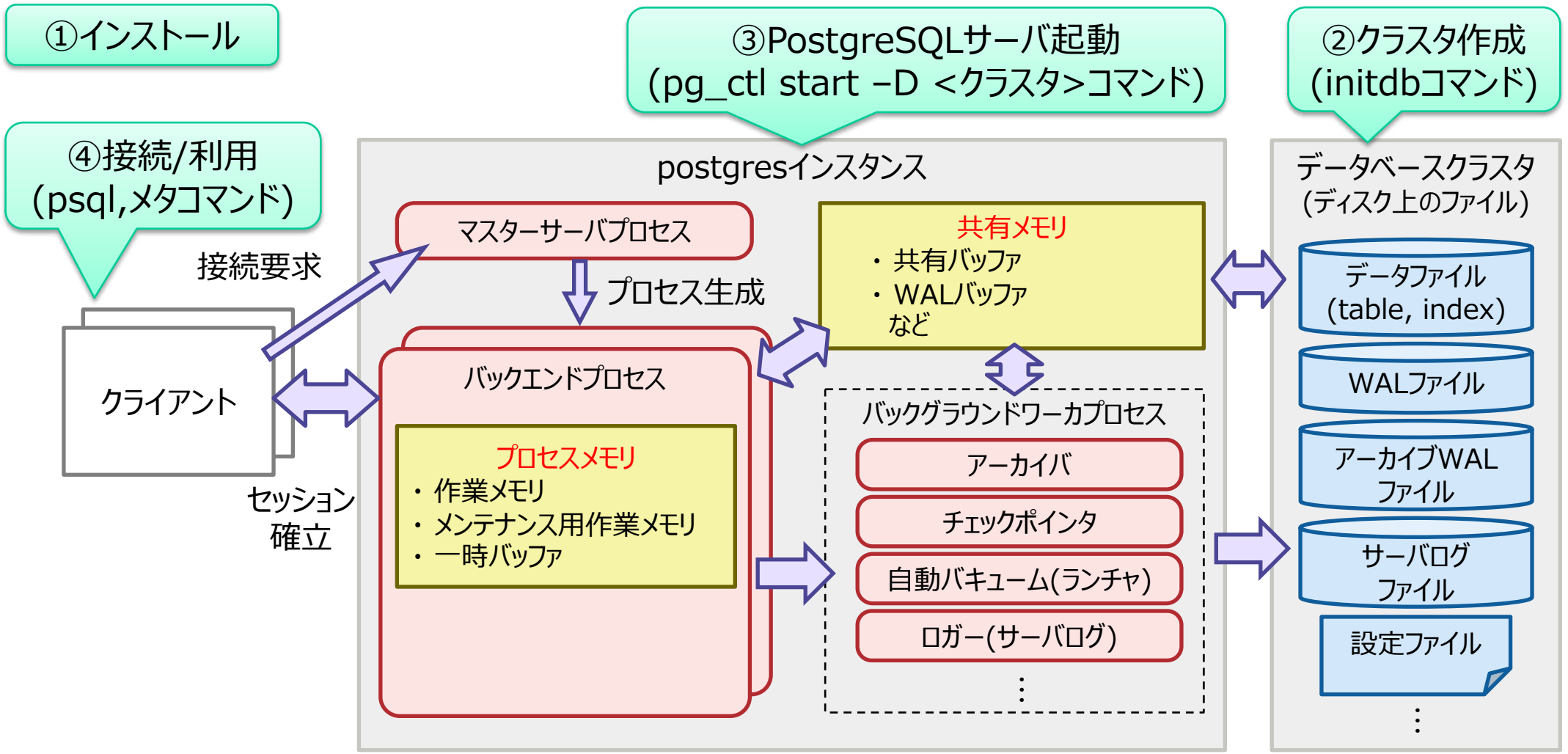
#OSS-DB



PostgreSQLを利用するまでの４ステップ

■インストールから利用までの簡単な流れについて説明

#OSS-DB



①インストール (Red Hat Enterprise Linuxの場合)

■ソースコードからのインストールの例(すべてデフォルト)

#OSS-DB

- **PostgreSQL管理用のユーザを作成しておく**

```
# useradd postgres
# passwd postgres
# su - postgres
```

事前にOSの必要パッケージもインストールしておく！

- **ソースコードをコミュニティのサイトからダウンロードし展開する**

```
$ tar xvzf postgresql-11.12.tar.gz
```

- **展開したフォルダに移動してコンパイルする**

※デフォルトのインストール先：/usr/local/pgsql
(configureのオプションにて、別フォルダも指定できる)
※makeコマンドの成功メッセージ：
All of PostgreSQL is successfully made. Ready to install.

```
$ cd postgresql-11.12
$ ./configure
$ make
$ make check
```

- **OSの管理者ユーザでインストール**

※make installの成功メッセージ：
PostgreSQL installation complete.

```
$ sudo make install
```

■設定

- **データベースクラスタ用のフォルダを準備**

```
# mkdir -p /usr/local/pgsql/data
# chown postgres:postgres /usr/local/pgsql/data
```

- **環境変数を設定する**

```
# vi ~/.bash_profile
# source ~/.bash_profile
```

```
export PATH=/usr/local/pgsql/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/pgsql/lib:
$LD_LIBRARY_PATH
export PGDATA=/usr/local/pgsql/data
```

実行ファイル、ライブラリ、データ格納先のパスを指定

②クラスタ作成

■initdbコマンドを実行

環境変数PGDATAで定義済みなら、省略可能

#OSS-DB

- **initdb --no-locale --encoding=UTF8 -D /usr/local/pgsql/data**

日本語を扱う場合、文字処理の性能面から、ロケールはno-locale(無効)を推奨

日本語を扱う場合、エンコーディング(文字セット)はUTF8かEUC_JPかEUC_JIS_2004を指定

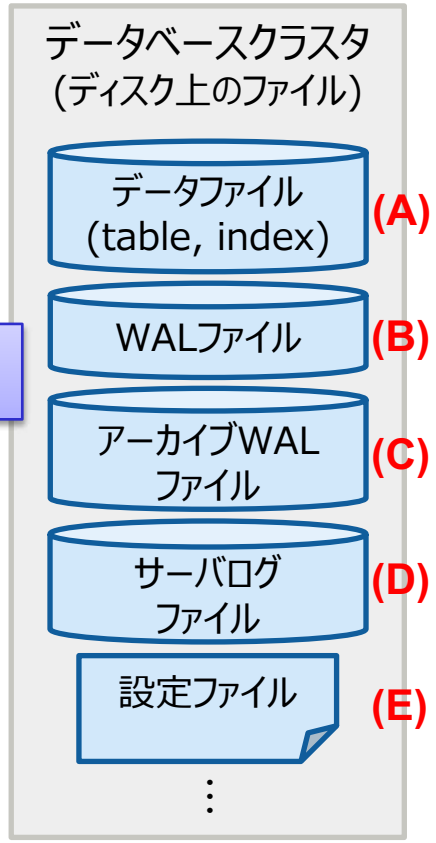
■データベースクラスタのフォルダ構成 (青色はフォルダ)

```
$ ls /usr/local/pgsql/data/
PG_VERSION     pg_dynshmem     pg_multixact  pg_snapshots  pg_tblspc     postgresql.auto.conf
base (A)       pg_hba.conf (E) pg_notify     pg_stat       pg_twophase   postgresql.conf (E)
global         pg_ident.conf   pg_replslot   pg_stat_tmp   pg_wal (B)    postmaster.opts
pg_commit_ts   pg_logical      pg_serial     pg_subtrans   pg_xact
```

デフォルトで以下の3つのデータベースが作成される

```
base (A)
  1   ←template1データベース用フォルダ
    (実データ)
  13286   ←template0データベース用フォルダ
    (実データ)
  13287   ←postgresデータベース用フォルダ
    (実データ)
```

(C)、(D)
PostgreSQL管理者ユーザが用意し、設定ファイルでパスを指定する。



③PostgreSQLサーバ起動

PostgreSQLサーバの起動方法

環境変数PGDATAで定義済みなら、省略可能

#OSS-DB

- pg_ctl start -D /usr/local/pgsql/data

postgresインスタンス内の
各種プロセスが存在する

```
[root@VM037311 ~]# ps aux | grep ^postgres
postgres 11963  0.0  0.7 272088 13168 pts/0    S    10:44   0:00 /usr/local/pgsql/bin/postgres
postgres 11965  0.0  0.1 272208  2908 ?        Ss   10:44   0:00 postgres: checkpointer
postgres 11966  0.0  0.1 272088  2136 ?        Ss   10:44   0:00 postgres: background writer
postgres 11967  0.0  0.2 272088  5040 ?        Ss   10:44   0:00 postgres: walwriter
postgres 11968  0.0  0.1 272500  2072 ?        Ss   10:44   0:00 postgres: autovacuum launcher
postgres 11969  0.0  0.0 127020  1072 ?        Ss   10:44   0:00 postgres: stats collector
postgres 11970  0.0  0.0 272500  1596 ?        Ss   10:44   0:00 postgres: logical replication launcher
```

pg_ctlのその他の主な機能

- stop　　指定されたデータディレクトリで稼働しているサーバを停止する
  - mオプション（smart:全クライアントの接続終了を待つ / fast:トランザクションをすべてロールバックする / immediate:緊急停止）
- restart　　stopとstartを実行する（設定ファイルの設定値が全て反映される）
- reload　　設定ファイルの再読み込み（パラメータの一部は反映のために再起動が必要）
- status　　サーバが稼働しているかどうかを確認する

※postgresql.confのコメントに
”(change requires restart)”が入っている項目

④接続と利用 (1/2)

psqlコマンド（PostgreSQLの対話的ターミナル）

#OSS-DB

- $ psql [オプション] [データベース名(ユーザ名)]
  - 主なオプション　-h: ホスト、-p: ポート、-U:ユーザ名
    -l: データベース一覧表示、-c: 指定コマンドの実行、
    -f: 指定ファイル内のコマンドの実行、-d: データベース名
- 実行例(スーパーユーザで実行した場合)

```
$ psql postgres
psql (11.12)
Type "help" for help.

postgres=# SQL文;
postgres=# メタコマンド
```

postgresはデータベース名

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

デフォルトのデータベース(postgres)、および、
テンプレートデータベース(template0, template1)が作成されている
template0: データベースの初期内容を保持（書き込み不可）
template1: データベースのひな形（書き込み可能）

- メタコマンド
  - ¥q(psqlを終了する)、¥d(テーブル、ビュー、シーケンスの一覧を表示する)　など

④接続と利用 (2/2)

■テーブル作成・アクセス

#OSS-DB

```
$ psql postgres
psql (11.12)
Type "help" for help.

postgres=# create table tbl (id int, name text);
CREATE TABLE
postgres=# insert into tbl values(1, 'suzuki');
INSERT 0 1
postgres=# select * from tbl;
 id |  name
----+--------
  1 | suzuki
(1 row)

postgres=# ¥q
```

設定ファイル(postgresql.conf)について

#OSS-DB

用途は？

- PostgreSQLの動作を決める設定ファイル

postgresql.confの記述方法

- 書式「パラメータ名 = 設定値」（=は空白でもよい）
- 1行に1つずつ
- 空白行は無視され、#から行末まではコメント
- パラメータ名は、大文字/小文字を区別しない
- 設定値の型：論理値(Boolean)、整数(integer)、
  浮動小数点(floating point)、
  文字列(string)、列挙型(enum)
- 論理値：on, off, true, false, yes, no, 1, 0　(t,f,y,nも使える)
- 文字列：複数指定や空白を含める場合は、'(シングルクォート)で囲む
  文字列内のシングルクォートは"(ダブルクォート)で囲む
- メモリの単位：B(バイト), kB(キロバイト), MB(メガバイト), GB(ギガバイト), TB(テラバイト)
- 時間の単位：ms(ミリ秒), s(秒), min(分), h(時間), d(日)

postgresql.confの設定値の例

```
#listen_addresses = 'localhost'
#port = 5432
max_connections = 100

shared_buffers = 128MB
#logging_collector = off
#log_rotation_age = 1d

#search_path = '"$user", public'
```

大文字/小文字の区別なし

単位については、大文字/小文字の区別あり

設定ファイル(postgresql.conf)に関する例題

postgresql.conf の記述方法について、正しいものを2つ選びなさい。

#OSS-DB

A) 論理型のパラメータで真を表す値として、on、true、t、yes、y などが使用できる。

B) 数値型のパラメータの値は、読みやすくするため 12,345 のように3桁ごとにカンマを入れて記述する。

C) 時間の単位にはms（ミリ秒）、s（秒）、m（分）、h（時間）、d（日）があり、大文字と小文字は区別されない。

D) メモリの単位は B、kB、MB、GB、TB があるが、キロバイトの k だけが小文字、他はすべて大文字で記述する。

E) 時間やメモリのパラメータ値を記述するとき、単位を省略して数値だけ記述すると、それぞれデフォルトとしてミリ秒、バイトが使用される。

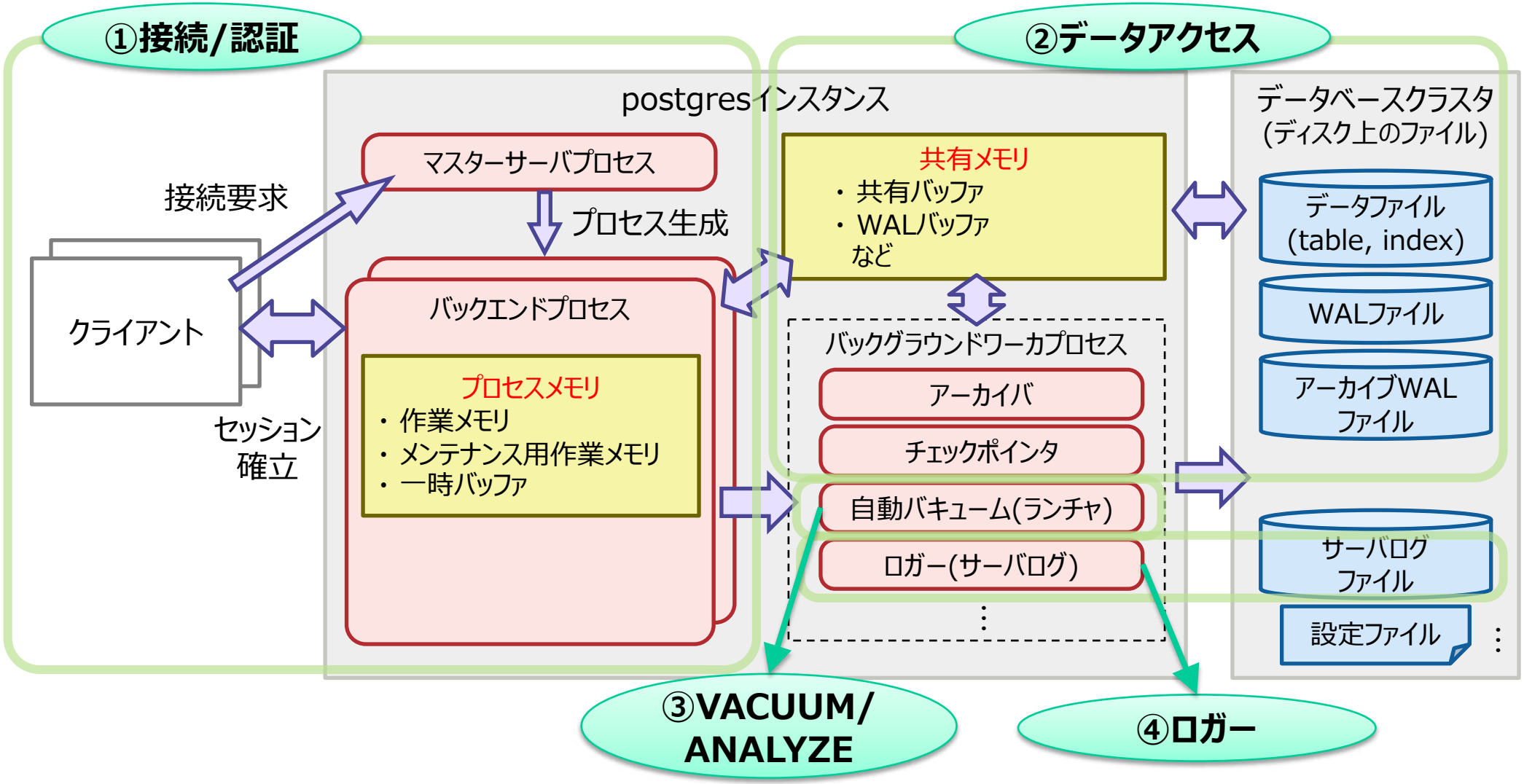
※この例題は実際のOSS-DB技術者認定試験とは異なります

- **OSS-DB公式サイトの「Silverの例題解説」より引用**
  https://oss-db.jp/sample/silver_management_05/98_191113

運用時に意識すべきPostgreSQLのしくみ

■PostgreSQLの主な機構を①～④に分けて説明

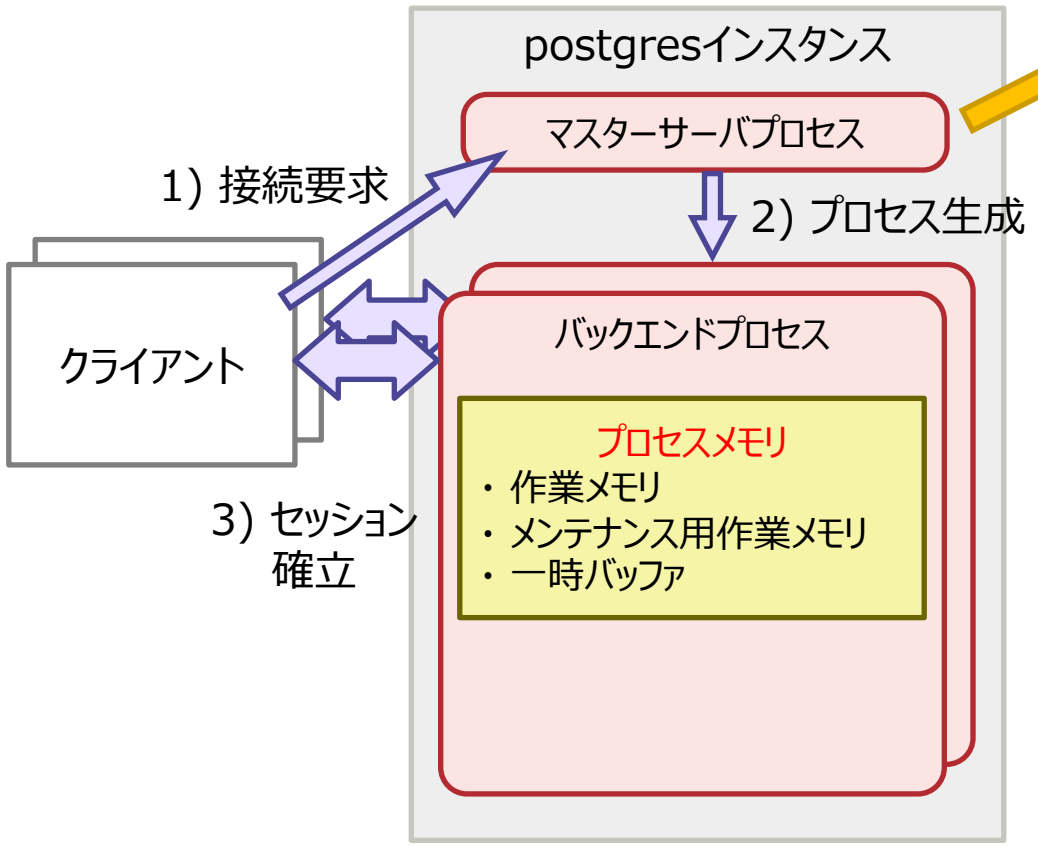
#OSS-DB



①接続と認証のしくみ

■接続要求で、認証/バックエンドプロセス起動/セッション確立が行われる　#OSS-DB

■クライアントの認証方法は pg_hba.conf に記述する



クライアント認証用のファイル(pg_hba.conf)

認証

```
# TYPE   DATABASE   USER   ADDRESS        METHOD
# IPv4 local connections:
host     all        all    127.0.0.1/32   md5
# IPv6 local connections:
host     all        all    ::1/128        md5
 . . .
```

「どこから(ADDRESS)誰(USER)がどのデータベース(DATABASE)にどんな接続方式(TYPE)でどのような認証方式(METHOD)でアクセスするか」を1行で指定する。認証時は、先頭行から順にチェックされる。

意味

クライアントのIPアドレス127.0.0.1/32(*1)のすべてのユーザが、(レプリケーション以外の)すべてのデータベースにTCP/IP接続でmd5暗号化によるパスワード認証(*2)での接続を許可する。

(*1) ローカルループバックアドレス。自分自身のIPアドレス。localhostも同じ意味。
(*2) クライアントからのパスワードが暗号化されて送信される。

参考）テスト用に無条件でTCP/IP接続を許可させる設定

```
host    all    all    0.0.0.0/0    trust
```

①設定パラメータ（postgresql.conf）

■接続用

#OSS-DB

パラメータ	デフォルト値	設定方法	注意
listen_addresses	localhost	クライアントからの接続を受け付けるPostgreSQLサーバ側のIPアドレスかホスト名を指定する。	(‘*’を指定すると、すべてのIPアドレスから接続できるようになる)
port	5432	接続時のTCPポートを指定する。	同一サーバでPostgreSQLを複数インスタンスを起動する場合には、それぞれ異なるportを指定する。
max_connections	100	最大受付数(*)を指定する。 (*)インスタンスへの同時接続数 + superuser_reserved_connections(デフォルト値:3)の値	max_connections数分のプロセスメモリが消費される。プロセスメモリを76MBで見積もっても、×100セッションで最大7.6GBのメモリが必要になる。
search_path	'"$user", public'	SQLでスキーマを省略した場合の、スキーマ検索順を指定する。	(スキーマを扱う場合は考慮が必要)
default_transaction_isolation	'read committed'	トランザクションのデフォルトの分離レベルを指定する。	(トランザクション分離レベルを意識する場合には考慮が必要)
client_encoding	sql_ascii	クライアントエンコーディングを指定する。	文字変換できないとエラーになる。
tcp_keepalives_xxx	0(OSのデフォルト値を使用)	クライアントから応答がない場合に自動切断できるよう指定する。	クライアントとの通信が不通になると、長時間リソースが回収されなくなる。

①設定パラメータ（postgresql.conf）

■プロセスメモリ用

#OSS-DB

パラメータ	デフォルト値	設定方法	注意
work_mem (作業メモリ)	4MB	SQLの処理において、並び替えやハッシュテーブル操作用に使用するメモリ容量を指定する。	複雑なSQLの場合、並行して処理が実行されるため、設定値×並列数分のメモリを消費する場合がある。
meaintenance_work_mem (メンテナンス用作業メモリ)	64MB	VACUUM、CREATE INDEX、およびALTER TABLE ADD FOREIGN KEYの様な保守操作で使用されるメモリの最大容量を指定する。	大きい値を設定することで、VACUUM処理などの性能が良くなることがあるが、autovacuum_max_workersの数分消費される。
temp_buffers (一時バッファ)	8MB	一時テーブルにアクセスする時にのみ使用されるバッファの最大量を指定する。	ー

接続／認証に関する例題

#OSS-DB

■ **PostgreSQLサーバはIPアドレス 192.168.1.10 のサーバで動作しており、pg_hba.conf の内容が以下である。**

```
local all all ident
host all all 127.0.0.1/32 ident
host foo bar 192.168.1.0/24 md5
host all bar 192.168.1.0/24 reject
host foo all 192.168.1.0/24 trust
```

IPアドレスが 192.168.1.11 のクライアントからこのデータベースに接続するときの記述について、選択肢の中から正しいものをすべて選びなさい。

A) ユーザ foo がデータベース bar にアクセスするときはパスワードを入力する必要がある。

B) ユーザ bar がデータベース foo にアクセスするときはパスワードを入力する必要がある。

C) ユーザ foo はデータベース foo にはアクセスできない。

D) ユーザ bar はデータベース bar にはアクセスできない。

E) ユーザ hoge はデータベース foo にパスワードなしでアクセスできる。

※この例題は実際のOSS-DB技術者認定試験とは異なります

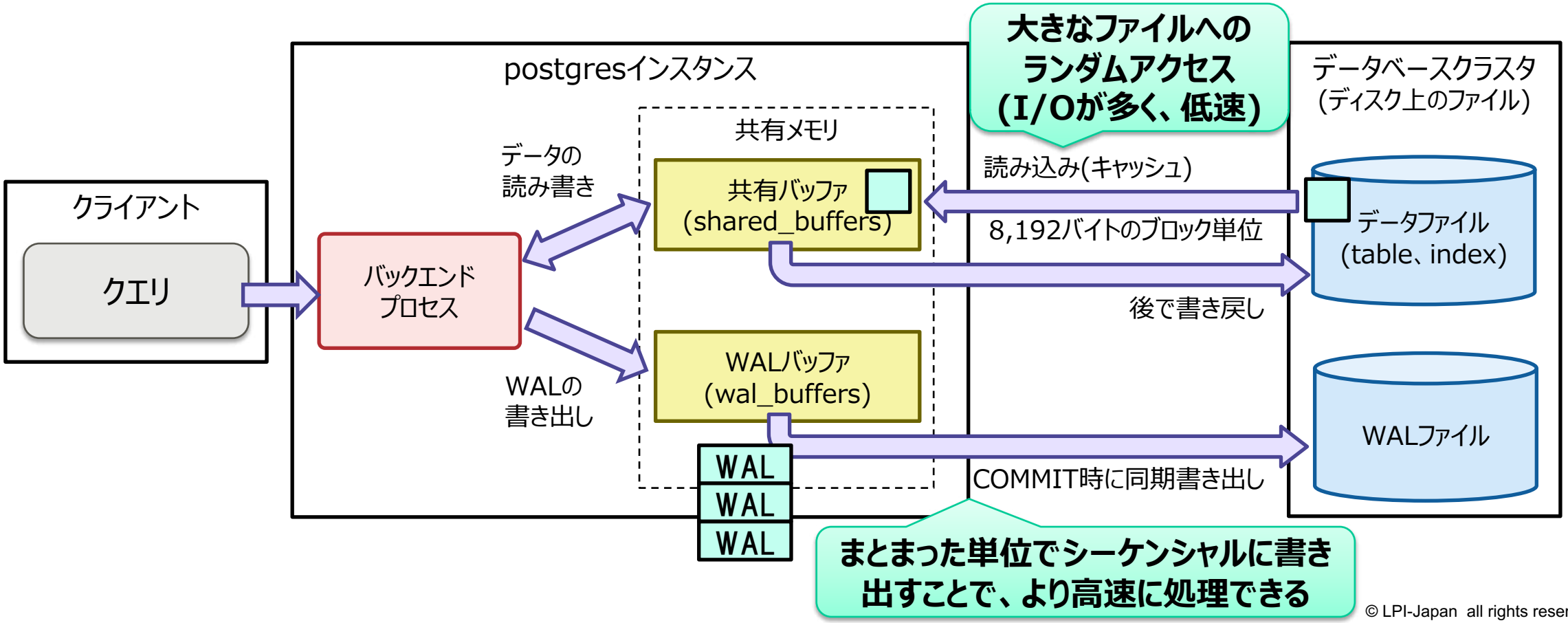
・**OSS-DB公式サイトの「Silverの例題解説」より引用**
https://oss-db.jp/sample/silver_management_01/14_120418

②データアクセスのしくみ（書き込み保証）

- 「高速なデータ更新」と「データ保証」を両立する
  - データ更新は共有メモリ上にキャッシュして高速に行う
  - ファイルの更新記録（トランザクションログ、WAL）をメモリに貯める
  - トランザクションがCOMMITしたら、まとまった単位でシーケンシャルにディスクに書き出す

#OSS-DB

※WAL:ログ先行書き込み(Write Ahead Logging)



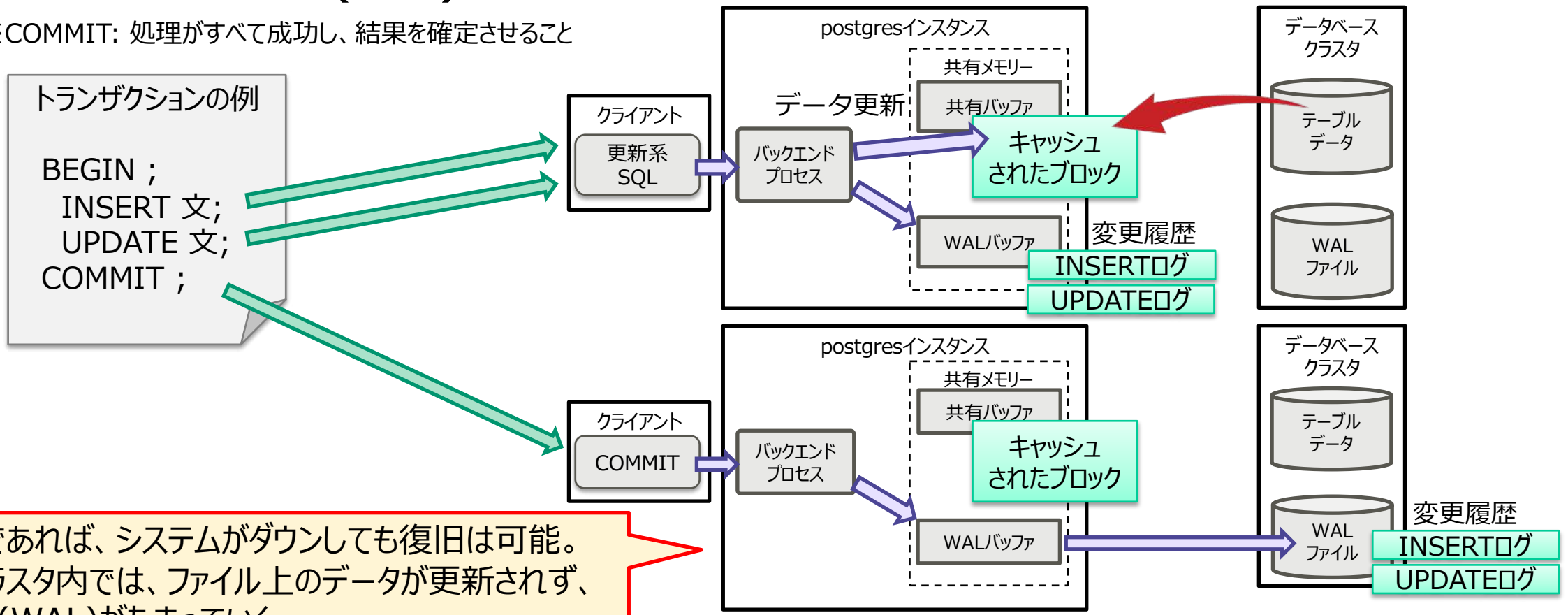
②データアクセス（書き込み保証）(1/3)

■SQLが実行されたときのデータアクセスの流れ

#OSS-DB

1. 共有バッファに更新すべきデータが無いと、ファイルからブロック単位でキャッシュする
2. 共有バッファ上でデータを更新し、変更履歴(WAL)をWALバッファに書き込む
3. COMMITで変更履歴(WAL)をWALファイルに書き込む

※COMMIT: 処理がすべて成功し、結果を確定させること



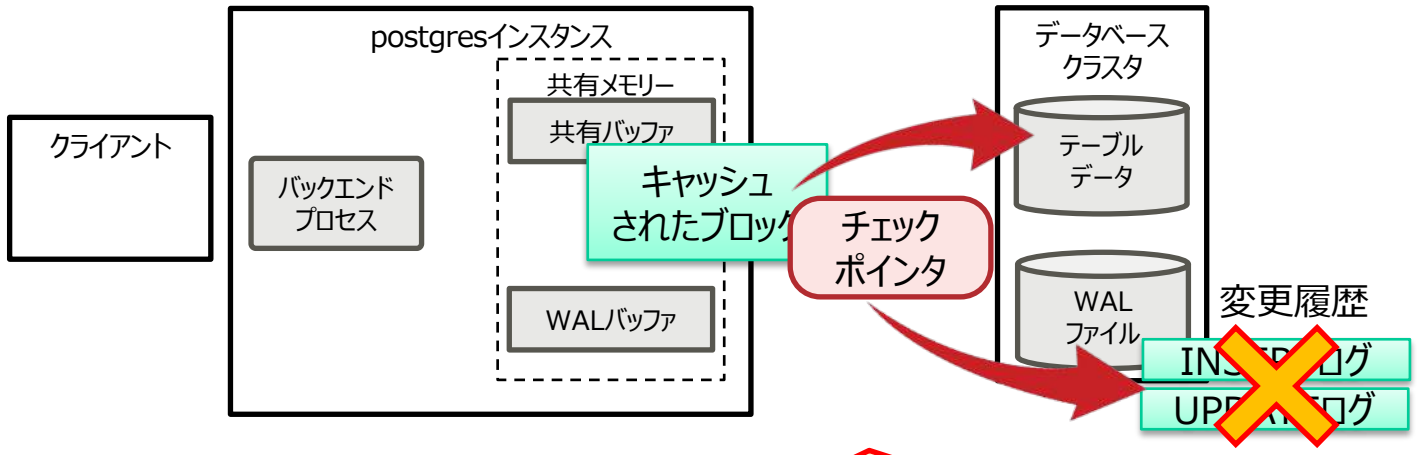
この状態であれば、システムがダウンしても復旧は可能。
しかし、クラスタ内では、ファイル上のデータが更新されず、
変更履歴(WAL)がたまっていく。

②データアクセス（書き込み保証）(2/3)

■チェックポイント処理（「チェックポインタ」プロセス）

#OSS-DB

- SQL実行とは非同期に動作する
- 定期的に共有バッファ内の更新されたブロックをファイルに書き出す
- 不要になった変更履歴(WAL)はリサイクルあるいは削除される

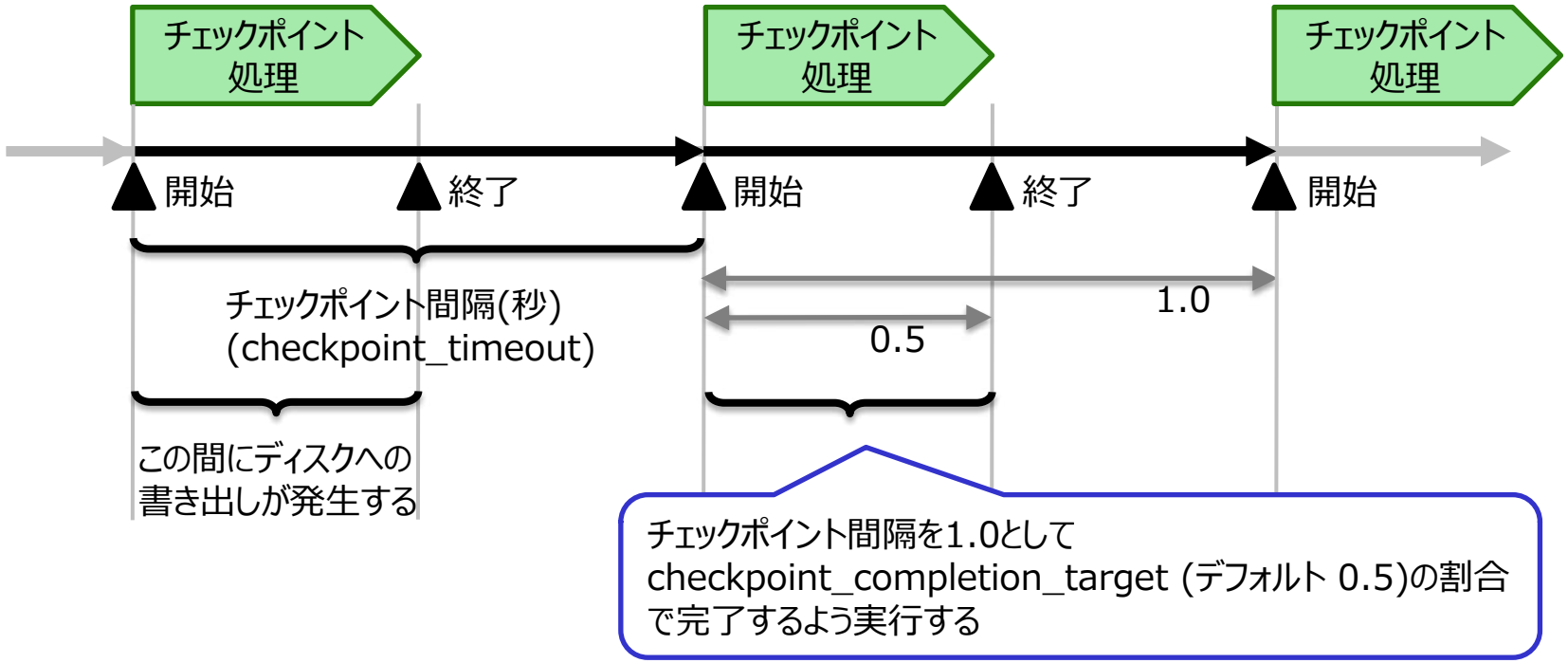


これで、テーブルデータが更新され、WALファイルがあふれる心配がない。
しかし、データファイルの更新には多くのI/Oが発生するため、
チェックポイント処理はシステムに負荷が掛かる。

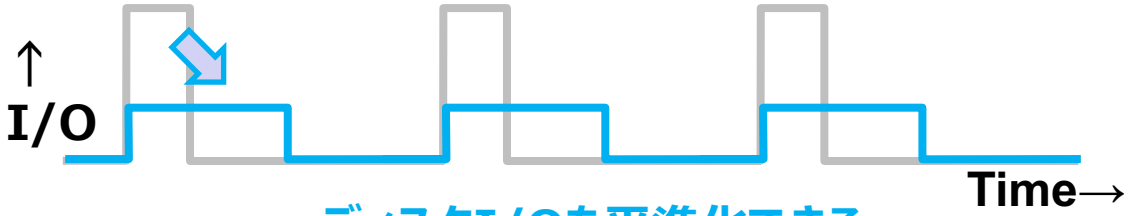
②データアクセス（書き込み保証）(3/3)

■オンライン処理に影響を与えないように、そ～っと実行する

#OSS-DB



注意）WALの量がmax_wal_sizeで設定された量に達すると、チェックポイント間隔で
指定された時間を待たずに次のチェックポイント処理は開始される



ディスクI/Oを平準化できる

②設定パラメータ（postgresql.conf）

■共有メモリ

#OSS-DB

パラメータ	デフォルト値	設定方法	注意
shared_buffers (共有バッファ)	128MB	データベースサーバが共有バッファのために使用するメモリ量として、搭載メモリーの約25～30%程度を指定する。	サイズを大きくし過ぎるとOS側で利用できるキャッシュメモリが減り、サーバ全体の性能が落ちる。
wal_buffers (WALバッファ)	shared_buffersの1/32	COMMIT前にWALバッファがあふれないようなサイズを指定する。そのためには、実際の業務のピーク時に発生するディスクI/Oの量を実測しながら調整する。（Linuxの場合はiostatコマンドなどで確認）	

②設定パラメータ（postgresql.conf）

WAL関連

#OSS-DB

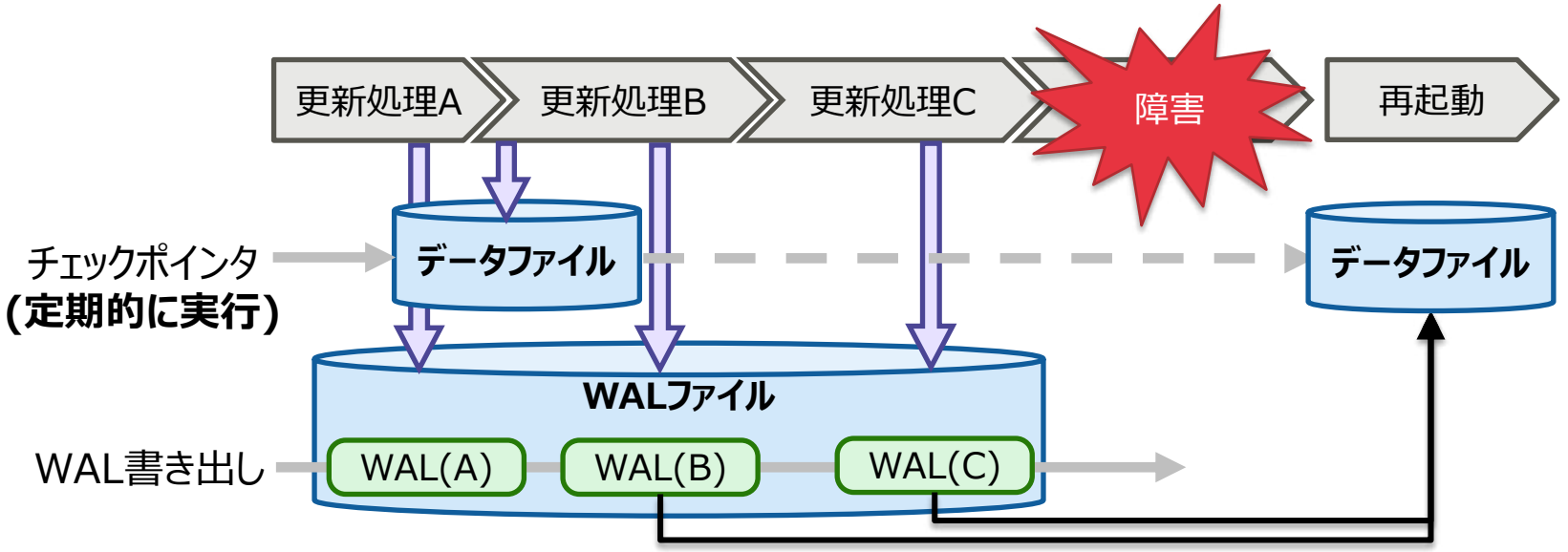
パラメータ	デフォルト値	設定方法	注意
wal_level	replica	WALに書かれる情報量を選択する。 (minimal < replica < logical) replica:WALアーカイブやレプリケーション利用時 logical:WALをロジカルデコーディングをする場合	設定値によってWALの情報量が変化するため、バックアップ/リカバリ時のデータ量に影響する。
max_wal_size	1GB	チェックポイント処理中に発生するWALの最大サイズを指定する。WALのサイズがおおよそmax_wal_size / (1+checkpoint_completion_target) に達するとチェックポイントが実施される。 ログメッセージに「LOG: checkpoints are occurring too frequently ・・・」が出力された場合にはサイズを増やす。	サイズが不足すると、頻繁にチェックポイント処理が動作する。また、サイズを大きくすると、クラッシュリカバリに必要な時間が長くなる。
checkpoint_timeout	5分	チェックポイント間隔を指定する。 (30分程度が推奨)	
checkpoint_completion_target	0.5	通常のオンライン業務の負荷が高い場合は、大きくても0.9まで増やすことで、チェックポイントによる負荷を減らすことができる。	最大値1.0を指定すると、ある時点でチェックポイントが完了しなくなる場合がある。

補足）クラッシュリカバリ

WALを利用してリカバリする

#OSS-DB

- トランザクション更新処理のWALは随時、WALファイルに書き出される
- 更新処理Aの後にチェックポイント処理が行われた場合、更新処理Aの結果はデータファイルに反映済み
- 障害発生後のリカバリは、PostgreSQLを再起動することで、未適用のWAL(B)(C)を検出し適用される

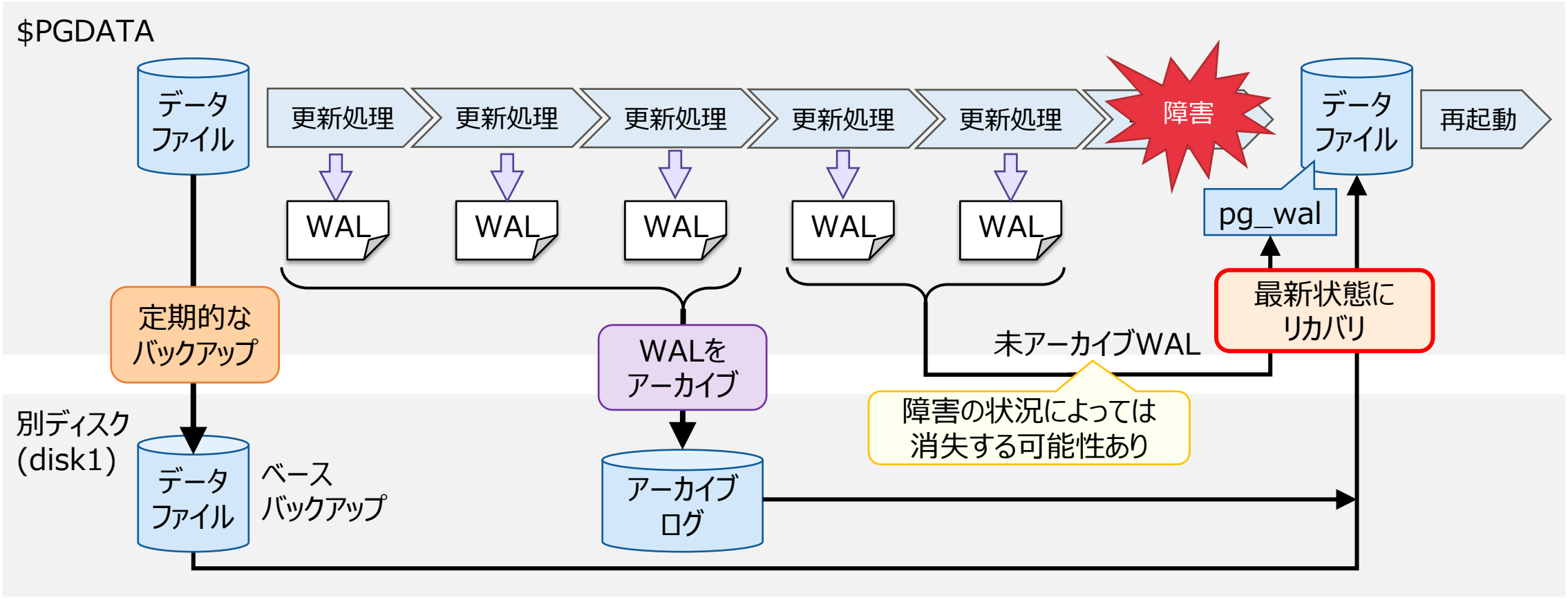


WALの応用）WALを利用したバックアップ「PITR」(1/3)

■オンラインバックアップとPITR(Point In Time Recovery)のしくみ

#OSS-DB

- 運用時：定期的にベースバックアップとアーカイブログを取得しておく
- 障害発生後：ベースバックアップ、アーカイブログ、WALから最新の状態まで復旧する



WALの応用）WALを利用したバックアップ「PITR」(2/3)

#OSS-DB

■オンラインバックアップ（運用中）

- **WALアーカイブを取得するための設定(postgresql.conf)**

```
wal_level = replica
archive_mode = on
archive_command = 'cp "%p" "/disk1/archivedir/%f"'
```

WALアーカイブ用のフォルダを準備しておく

- **定期的なベースバックアップの実行**

PostgreSQLは停止しなくてよいスーパーユーザで実施する

1. **pg_start_backup('ラベル名')関数の実行**
2. **データベースクラスタをバックアップ**

```
# cd /usr/local/pgsql
# tar czvfp /disk1/backup/backup.tar.gz data
```

OSのコマンドでよい

3. **pg_stop_backup()関数を実行**

1.～3.は、pg_basebackupに置き換えることも可能
（1.と3.の処理は、pg_basebackup内部に含まれているため実施不要）

```
$ pg_basebackup -D /disk1/backup -X s -P -h ホスト名 -U ユーザ名
```

WALの応用）WALを利用したバックアップ「PITR」(3/3)

PITRによるリカバリ（障害発生後の復旧）

1. **PostgreSQLが停止していることを確認**

```
$ pg_ctl status
```

2. **データベースクラスタから未アーカイブのWALを退避**

```
# cp -rp /usr/local/pgsql/data/pg_wal /disk1
# rm -rf /usr/local/pgsql/data
```

クラスタは削除しておく

3. **ベースバックアップのリストア**

```
# cd /usr/local/pgsql
# tar xzvfp /disk1/backup/backup.tar.gz
```

OSのコマンドでよい

4. **未アーカイブのWALファイルを復元**

```
# rm -rf /usr/local/pgsql/data/pg_wal
# cp -rp /disk1/pg_wal /usr/local/pgsql/data
```

5. **recovery.confの作成**

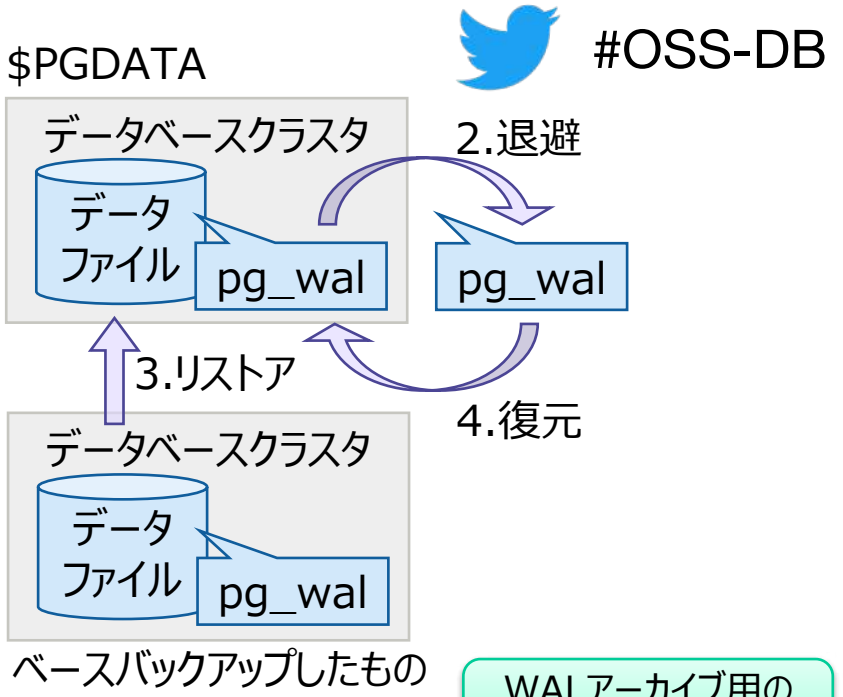
```
$ vi /usr/local/pgsql/data/recovery.conf
```

```
# 以下の行を追加
restore_command = 'cp "/disk1/archivedir/%f" "%p"'
# リカバリポイントも指定できる
#recovery_target_time = '2021/06/22 00:00:00'
```

WALアーカイブ用のフォルダ

6. **PostgreSQLを起動し、recovery.confの拡張子がdoneになったことを確認**

```
$ pg_ctl start
$ ls /usr/local/pgsql/data/recovery.*
/usr/local/pgsql/recovery.done
```



WALの応用）設定パラメータ（postgresql.conf）

PITR関連(WALアーカイブの設定)

#OSS-DB

パラメータ	デフォルト値	設定方法	注意
wal_level	replica	WALアーカイブを利用する場合は、replicaかlogicalを指定する。	
archive_mode	off	onかalwaysに設定することでWALがアーカイブされるようにする。	
archive_command	''	アーカイブスクリプトを設定する。 例：(%p:アーカイブ先フォルダ、%f:ファイル名) 'cp "%p" "/disk1/archivedir/%f"'	アーカイブ先は、システムの信頼性を高めるために、別ディスクを指定することが望ましい

PITR（バックアップ方式）に関する例題

■**ポイントインタイムリカバリのためのバックアップ手順として、間違っているものを1つ選びなさい。**

#OSS-DB

A) データベースに管理者ユーザで接続し、pg_start_backup を実行する。

B) データベースを停止する。

C) データベースクラスタのディレクトリ全体を、tar などのOSコマンドを利用してコピーする。

D) データベースに管理者ユーザで接続し、pg_stop_backup を実行する。

E) 上記の一連の処理の代わりに、OSのコマンドラインから pg_basebackup コマンドを実行する。

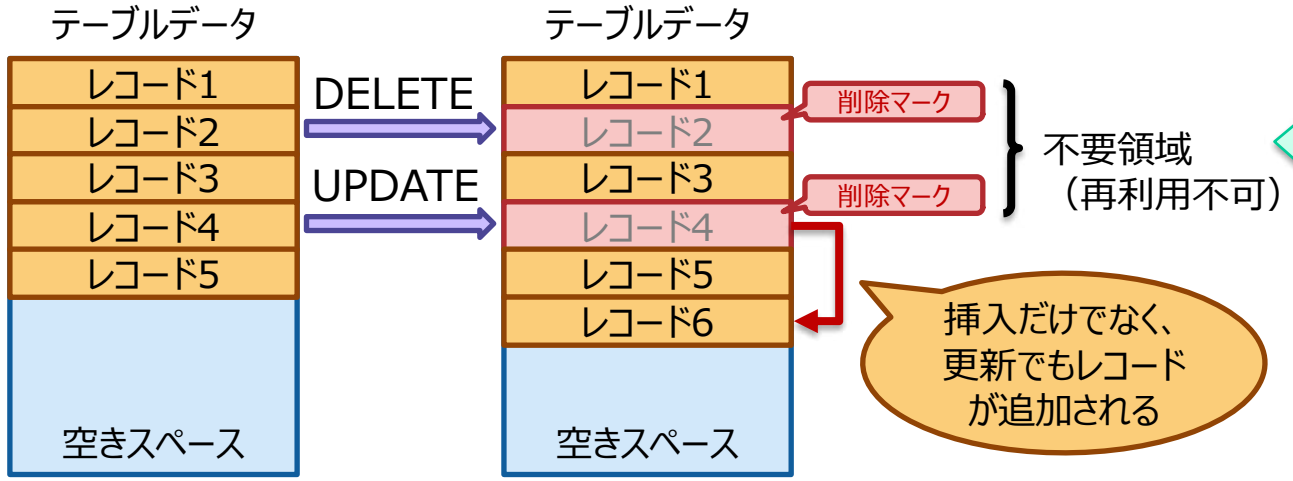
※この例題は実際のOSS-DB技術者認定試験とは異なります

・**OSS-DB公式サイトの「Silverの例題解説」より引用**
https://oss-db.jp/sample/silver_management_06/102_200312

③VACUUMとANALYZEのしくみ (1/4)

■PostgreSQLは追記型アーキテクチャを採用

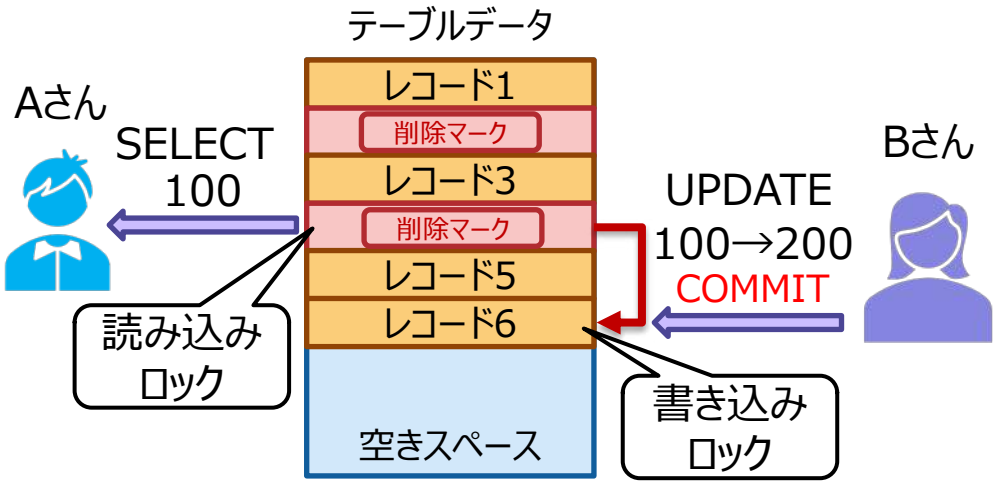
#OSS-DB



「不要領域」は、データファイルのサイズを増大させ、ディスク領域の圧迫とI/O増加による性能低下につながる。
この「不要領域」を回収する手段がVACUUM。

■トランザクションの同時実行制御方式（MVCC）を実装

※MVCC:多版型同時実行制御 (Multi Version Concurrency Controlの略)



Aさんが「読み込みロック」を獲得してレコード4を参照中、Bさんは「書き込みロック」を獲得して、同じデータを更新することが可能になる。
ロック競合の観点でメリットになる。

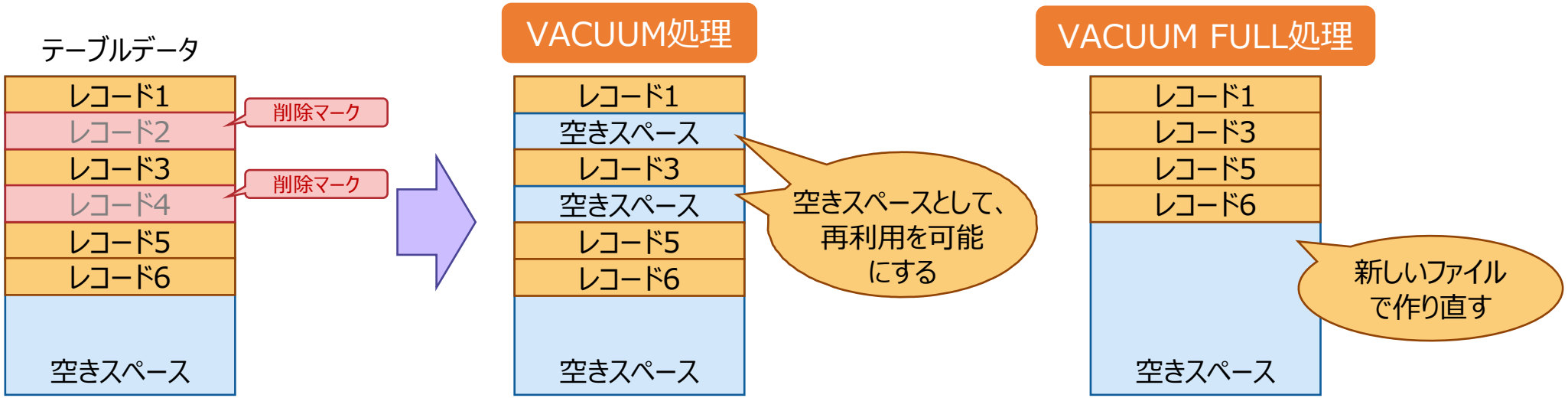
③VACUUMとANALYZEのしくみ (2/4)

■VACUUMで「不要領域」を再利用可能にする（2種類の方法）

#OSS-DB

- VACUUM テーブル名;
  - 同時に実行される殆どのSQLとは、テーブルへのロックが競合しない。
  - 運用中に実行できる。
- VACUUM FULL テーブル名;
  - 不要領域を物理的にディスクから削除。
  - 排他的ロックが必要なため、保守作業中に実行する必要がある。

※テーブル名を省略するとすべてのテーブルが対象。
※VACUUMは、指定したテーブルに指定されているインデックスも処理対象にする。



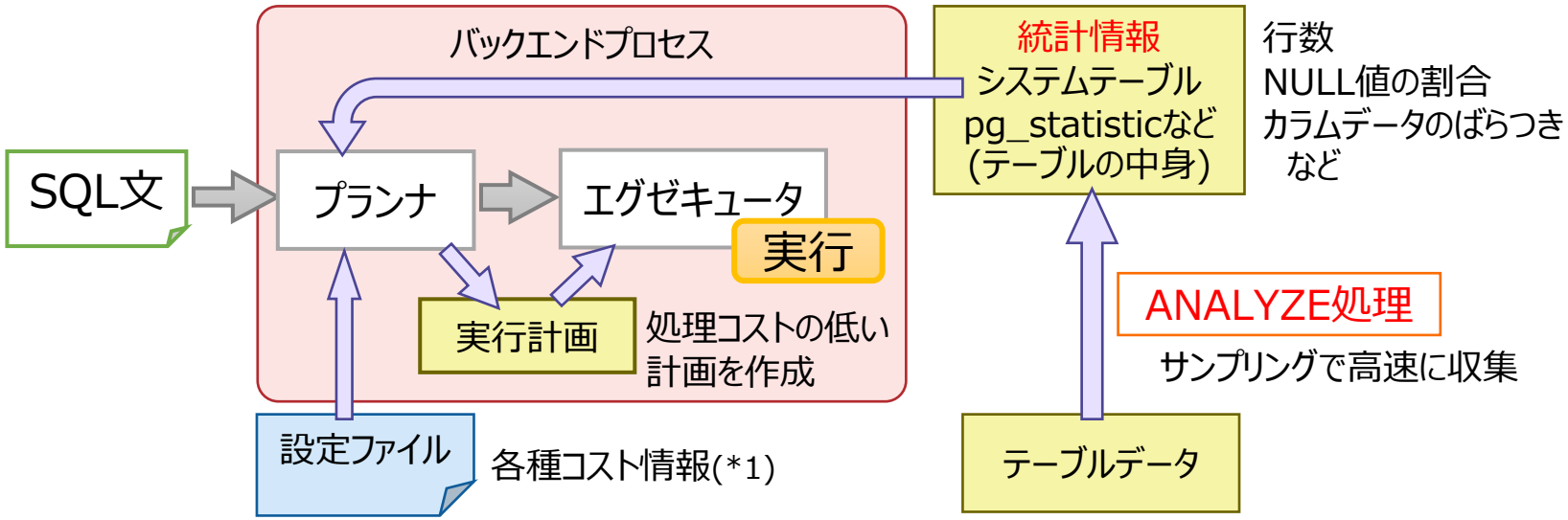
③VACUUMとANALYZEのしくみ (3/4)

■ANALYZEで『統計情報』を更新する

#OSS-DB

- PostgreSQLは、クライアントからSQLを受け取ると、テーブルにどのようなデータが格納されれているかを示す『統計情報』に基づいて、最も効率の良い処理方式と処理順序を組み立てて実行する。この統計情報がテーブルの実態と合っていないとSQLが効率的に実行できない。
- 『統計情報』を更新するには「ANALYZE テーブル名;」を実行する。

※テーブル名を省略するとすべてのテーブルが対象。



(*1)各種コスト情報
seq_page_cost = 1.0
random_page_cost = 4.0
cpu_tuple_cost = 0.01
cpu_index_tuple_cost = 0.005
cpu_operator_cost = 0.0025
・・・

③VACUUMとANALYZEのしくみ (4/4)

#OSS-DB

VACUUMとANALYZEをまとめて実行する方法

- **VACUUM ANALYZE テーブル名;** ※テーブル名を省略するとすべてのテーブルが対象。
  ※VACUUMを実行すると、ほぼ全テーブルをスキャンするため、ANALYZEも一緒に実行した方が効率がよい。

autovacuum(自動バキューム)について

- **VACUUMを定期的に自動で実行する機能（自動メンテナンス機能）。**
- **デフォルトで有効に設定されている。**
  - **設定パラメータautovacuum = on(デフォルト)**
- **テーブル毎に、更新/削除された行数が、指定された”しきい値”を超えた場合に実行される。**
- **VACUUMを実行後、ANALYZEも一緒に実行される。**
- **実際の運用において、autovacuumが定期的に実行されていれば、不要領域や実行計画の問題は発生しにくい。**

大量データの挿入／削除の直後や、高頻度のデータ更新時は、
統計情報の更新が追いつかないことがあるため、
手動でのANALYZEを併用することも検討が必要。

③設定パラメータ（postgresql.conf）

■VACUUMとANALYZE

#OSS-DB

パラメータ	デフォルト値	設定方法	注意
autovacuum	有効(on)	onを指定すると、autovacuumが実行されるようになる。	track_countsパラメータも有効(on)になっている必要がある。
autovacuum_naptime	1分	autovacuumの実行チェックが行われる間隔を変更する場合に指定する。	テーブルの統計情報の変化に応じて設定値を検討する。
autovacuum_vacuum_threshold	50レコード	以下の計算値（しきい値）を超えたテーブルでVACUUMが実行される。 ［ autovacuum_vacuum_threshold ＋ autovacuum_vacuum_scale_factor × レコード数 ］	更新が少ないテーブルやレコード数が非常に大きいテーブルは、なかなかVACUUMの対象にならない。（→不要領域の回収が遅れる）
autovacuum_vacuum_scale_factor	0.2		
autovacuum_analyze_threshold	50レコード	以下の計算値（しきい値）を超えたテーブルでANALYZEが実行される。 ［ autovacuum_analyze_threshold ＋ autovacuum_analyze_scale_factor × レコード数］	更新が少ないテーブルやレコード数が非常に大きいテーブルは、は、なかなかANALYZEの対象にならない。（→統計情報の更新頻度が低くなる）
autovacuum_analyze_scale_factor	0.1		

VACUUMとANALYZEに関する例題

■**バキューム（VACUUM）と自動バキュームの違いの説明として適切なものを3つ選びなさい。**

#OSS-DB

A) VACUUMはオプションで指定しなければ不要領域の回収のみを実行するのに対し、自動バキュームは不要領域の回収と解析（ANALYZE）の両方を実行する。

B) VACUUMは対象とするテーブルのリストを指定する必要があるが、自動バキュームは挿入・更新・削除されたデータが一定量以上のテーブルが自動的に対象となる。

C) VACUUMは一時テーブル（temporary table）を対象とできるが、自動バキュームでは一時テーブルは対象外である。

D) VACUUMはオプションを指定することでファイルサイズを縮小することができるが、自動バキュームには同等の機能がない。

E) VACUUMはOSの機能などを使って定期的に自動実行するようにできる。自動バキュームはPostgreSQLサーバの稼働中で、トランザクションの実行量が少ないときに自動的に実行されるので、OSの機能の設定は不要である。

※この例題は実際のOSS-DB技術者認定試験とは異なります

・**OSS-DB公式サイトの「Silverの例題解説」より引用**
https://oss-db.jp/sample/silver_management_06/109_201029

VACUUMとANALYZEに関する例題

■**VACUUM と VACUUM FULLの違いの説明として適切なものを2つ選びなさい。**

#OSS-DB

A) VACUUM FULLはデータベース内の全テーブルが対象になるが、VACUUMは指定した特定のテーブルのみを対象とする。

B) VACUUM FULLはテーブルのANALYZEも実行するが、VACUUMはANALYZEを実行しない。

C) VACUUM FULLは処理対象のテーブルの排他的ロックを取得するため、通常のテーブルの読み書きの処理と競合が発生する。VACUUMは排他的ロックを取得しない。

D) VACUUM FULLを実行すると通常はテーブルを構成するファイルのサイズが小さくなるが、VACUUMを実行しても通常はテーブルを構成するファイルのサイズは変わらない。

E) OSのコマンドラインから vacuumdb コマンドを発行することで VACUUM を実行できるが、VACUUM FULL はコマンドラインからは実行できない。

※この例題は実際のOSS-DB技術者認定試験とは異なります

・**OSS-DB公式サイトの「Silverの例題解説」より引用**
https://oss-db.jp/sample/silver_management_06/108_200902

VACUUMとANALYZEに関する例題

■**ANALYZEについて述べたものとして、適切なものを2つ選びなさい。**

#OSS-DB

A) テーブルがアクセスされる頻度、更新される頻度といった統計情報を取得する。

B) デフォルトではテーブル全体を解析するため、巨大なテーブルのANALYZEには時間がかかる。

C) 実行時のパラメータにより、データベース全体、データベース内の特定のテーブルのみ、特定のテーブルの特定の列のみ、などANALYZEの対象を制御できる。

D) 実行時のパラメータにより、統計情報の取得の目標値を変更できるので、これにより、統計情報の正確さやANALYZEに要する時間を制御できる。

E) 自動バキュームの実行時に自動的に実行される。

※この例題は実際のOSS-DB技術者認定試験とは異なります

・**OSS-DB公式サイトの「Silverの例題解説」より引用**
https://oss-db.jp/sample/silver_management_04/66_160808

④ロガーのしくみ

#OSS-DB

ロガーとは？

- サーバログをstderr、syslog、csvlog等へ送る機構。

※ stderr: 標準エラー出力、csvlog: CSV形式での標準エラー出力

設定方針

- デフォルト設定のままだと問題の検出や調査に必要な情報がログ出力されないため、ログファイルに出力する必要がある
- ログ行の先頭に各種プレフィックスに付加すると、障害時の調査等に役立つ
- ログファイルはローテーションさせる

postgresql.confへの設定例

① サーバログをsyslogとログファイルへ出力

② ログファイルは$PGDATA/pg_log/postgres-Mon.logのような名前で、毎日のログを1週間でローテーションする

③ プレフィックスに、タイムスタンプ、ユーザ名、データベース名、プロセスID、ログ行番号を出力

④ 5秒以上かかったSQL文は記録

⑤ ワーニング以上のエラーはログファイルへ出力し、クライアント側にも伝える

```
① log_destination = 'syslog, stderr'
② logging_collector = on
② log_directory = 'pg_log'
② log_filename = 'postgres-%a.log'
② log_truncate_on_rotation = on
② log_rotation_age = 1440
③ log_line_prefix = '[%t] %u %d %p [%l] '
④ log_min_duration_statement = 5000
⑤ log_min_messages = warning
⑤ log_min_error_statement = warning
⑤ client_min_messages = log
```

④設定パラメータ（postgresql.conf）

■サーバログの出力設定(サーバログ系)

#OSS-DB

パラメータ	デフォルト値	設定方法	注意
log_destination	'stderr'	サーバログの出力先を選択する。'stderr'、'syslog'、 'csvlog'などから、カンマ区切りで複数指定が可能。	csvlogの場合、logging_collectorがonであること。
logging_collector	off	onにすると、標準エラー出力へ送られたサーバログをファイルにリダイレクト(書き出す)。併せて、log_directory、log_filenameも設定する。	
log_directory	'log'	ログファイルを格納するフォルダを、絶対パス、あるいは、$PGDATAからの相対パスで指定する。	logging_collectorがon(サーバログをファイルへ出力)のときに指定する。
log_filename	'postgresql-%Y-%m-%d_%H%M%S.log'	ログファイル名を指定する。(%Y→年、%m→月、%d→日、%H→時、%M→分、%S→秒に変換)	
client_min_messages	notice	どのメッセージ階層をクライアントに送るかを指定する。有効な値は、debug5、 debug4、debug3、debug2、 debug1、log、notice、 warning、error、fatal、panic。	設定値は、それぞれ後方の階層も含む。(例えば、fatalを指定するとpanicも含む) logの優先度が双方で異なる。
log_min_messages	warning	どのメッセージ階層をサーバログに書き込むかを指定する。有効な値は、debug5、debug4、 debug3、debug2、debug1、 info、notice、warning、error、log、fatal、panic。	

④設定パラメータ（postgresql.conf）

■サーバログの出力設定(サーバログ系　のつづき)

#OSS-DB

パラメータ	デフォルト値	設定方法	注意
log_min_error_statement	error	エラーの原因となったSQL文をサーバログに記録する。 有効な値は、debug5、debug4、 debug3、debug2、debug1、 info、notice、warning、 error、log、fatal、panic。	設定値は、それぞれ後方の階層も含む。(例えば、fatalを指定するとpanicも含む)
log_line_prefix	'%m [%p] '	ログ行の先頭に出力するprintfの書式文字列を指定する。 (%m:ミリ秒付きタイムスタンプ, %p:プロセスID)	障害解析、性能解析に必要な情報が出力されるようにしておく。
log_truncate_on_rotation	off	ログをローテーションする場合には、onを指定する。併せて、log_filename、log_rotation_ageも設定する。 例)毎日のログを1週間(7日)分保持する場合 log_truncate_on_rotation = on log_filename = 'postgres-%a.log' log_rotation_age = 1440	logging_collectorが有効になっている必要がある。
log_rotation_age	1d	ログファイルの最大寿命を指定します。指定した時間(分単位)を経過すると、新しいログファイルが生成されます。	logging_collectorが有効になっている必要がある。

④設定パラメータ（postgresql.conf）

■サーバログの出力設定(性能監視系)

#OSS-DB

パラメータ	デフォルト値	設定方法	注意
log_statement	none	どのSQL文をログに記録するかを指定する。指定できる値は以下のとおり。 none(off) / ddl(CREATE/ALTER/DROP) / mod(ddl+INSERT/UPDATE/DELETE/TRUNCATE/COPY FROM) / all(全てのメッセージ)	より広い範囲のログを記録すると、ログの量が大きくなりシステムに負荷がかかる。
log_min_duration_statement	-1(記録無効)	指定したミリ秒数以上かかったSQL文について、それぞれの文の実行に要した時間をログに記録する。 0を指定すると、すべてのSQL文が対象になる。	

ロガーに関する例題

■**postgresql.conf で設定するパラメータについて、正しい説明を2つ選びなさい。**

#OSS-DB

A) log_destination には、ログ出力先となるファイル名を指定する。

B) logging_collector を on にすることで、標準エラー出力に送られたログメッセージをログファイルにリダイレクトできる。

C) log_connections を on にすることで、クライアントからサーバへの接続試行がログに出力される。

D) log_statement を on にすることで、SQL 文をログに記録できる。

E) log_line_prefix を on にすることで、ログ情報の出力時刻、ユーザ名、プロセスIDなど様々な付加情報がログファイルに出力される。

※この例題は実際のOSS-DB技術者認定試験とは異なります

・**OSS-DB公式サイトの「Silverの例題解説」より引用**
https://oss-db.jp/sample/silver_management_01/19_120815709

まとめ

以下の内容について述べた。

#OSS-DB

■PostgreSQLの基本構成
- PostgreSQLを利用するまでの４ステップ
  インストール ～ クラスタ作成 ～ PostgreSQLサーバ起動 ～ 接続と利用

■運用時に意識すべきPostgreSQLのしくみ＋設定パラメータの設定方法
- 接続と認証
- データアクセス
- VACUUM(バキューム)とANALYZE(アナライズ)
- ロガー