アシスト
「お客様の最高」のために

LPI-JAPAN
Linux Professional Institute Japan

OSS-DB Exam Silver 技術解説無料セミナー

2015/08/30

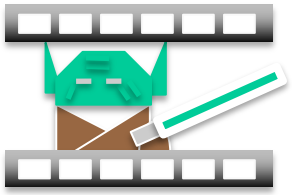
株式会社 アシスト
データベース技術本部
喜田 紘介

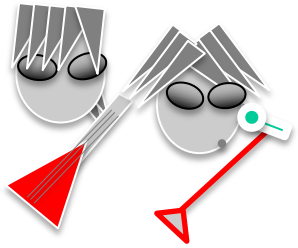
© LPI-Japan 2015. All rights reserved.

講師プロフィール

■プロフィール

- 名前　喜田 紘介 (きだ こうすけ)
- 所属　株式会社 アシスト データベース技術本部
- 趣味



■PostgreSQL関連の活動

- Oracle DBの構築、教育、記事執筆、トラブル対応などのフィールド支援を経て2011年よりPostgreSQLの専任技術として活動。
- 現在は、新規構築するシステムのRDBMS選択支援や、商用DBからの移行難易度の評価を行う移行アセスメント支援を主に担当。

2011年　OSS-DB Silver 取得(12月)
2012年　OSS-DB Gold 取得(12月)
2013年　SQL逆引き大全 363の極意 執筆
2014年　日本PostgreSQLユーザ会 広報・企画担当就任(6月)

© LPI-Japan 2015. All rights reserved.

オープンソースデータベース（OSS-DB）に関する技術と知識を認定するIT技術者認定

OSS-DB /Silver

データベースシステムの設計・開発・導入・運用ができる技術者

OSS-DB /Gold

大規模データベースシステムの
改善・運用管理・コンサルティングができる技術者

OSS-DB技術者認定資格の必要性

商用/OSSを問わず様々なRDBMSの知識を持ち、データベースの構築、運用ができる、または顧客に最適なデータベースを提案できる技術者が求められている

© LPI-Japan 2015. All rights reserved.

OSS-DB /Silver

データベースシステムの設計・開発・導入・運用ができる技術者

- RDBMSやPostgreSQLの構造の理解
  - RDBMSに求められる機能とその実装
    （高性能・同時実行・耐障害性などを満たすデータベースとしての実装）
- メンテナンスコマンドの理解
  - オプションレベルで、何ができるか知っている
  - コマンドから結果を予測できる
- SQL開発

OSS-DB /Gold

大規模データベースシステムの改善・運用管理・コンサルティングができる技術者

- PostgreSQLの詳細な構造の理解（たとえば、データの格納方式）
- メンテナンスや障害対応の必要性の判断と適切な実施
- 広い視野でチューニングができる

© LPI-Japan 2015. All rights reserved.

Silver試験範囲

アシスト 「お客様の最高」のために　OSS-DB

■一般知識

- RDBMSに関する一般知識（リレーショナルモデル・SQL・正規化　など）
- PostgreSQLライセンスやOSSコミュニティについて

■運用管理

- インストール方法（ソースコード・initdb・PGDATA・template　など）
- 設定ファイルと設定方法（postgresql.conf・pg_hba.conf）
- バックアップ・リカバリ
- 運用管理（PostgreSQLの標準ツール、DBオブジェクトのメンテナンス）

■SQL開発

- 基本的なSQL文やデータベースオブジェクト
- 組み込み関数
- トランザクション

OSS-DB出題範囲（http://www.oss-db.jp/outline/examarea.shtml）では
コマンド例などの詳細を確認可能

© LPI-Japan 2015. All rights reserved.

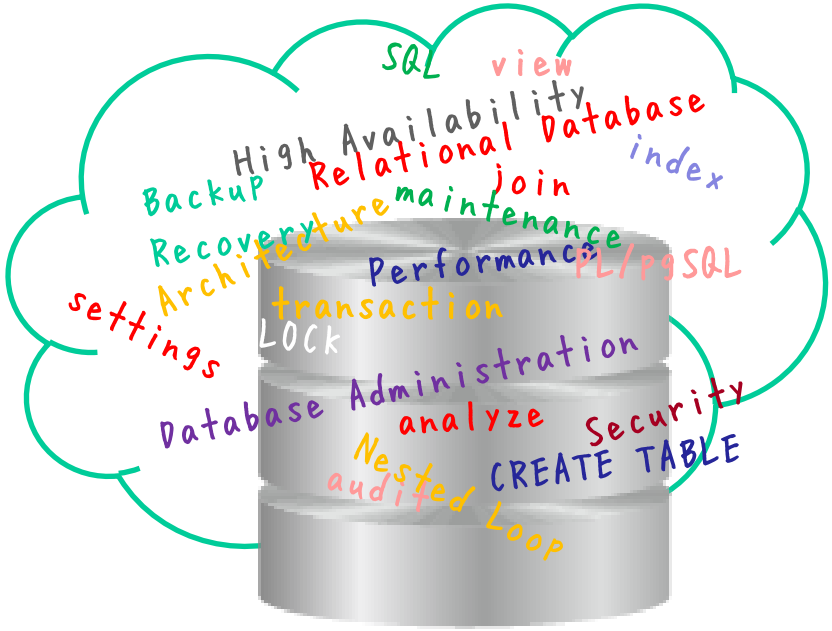
本講義の内容

■データベースの基本を解説

- データベース技術者としての入門レベルであり、PostgreSQLを扱う上で必須知識であるOSS-DB Silver試験に向けた学習のきっかけに
- データベース初級者が、PostgreSQLを使用したデータベース学習を進められることを目標とする

■取り扱う内容

- データベースに求められること
- RDBMSの構造
- SQL開発
- DBA（データベース管理者）のタスク



© LPI-Japan 2015. All rights reserved.

アジェンダ

- **データベースに求められること**

  データベースに求められる「高性能」「同時実行性」「耐障害性」などの基本を整理し、これらを実現するRDBMSの重要なキーワードを解説

- **RDBMSの構造**

  前章で挙げたデータベースとしての基本が、PostgreSQLではどのような仕組みで実装されているかを解説

- **SQL開発**

  RDBMSの共通言語である「SQL」の基本を解説

- **DBA（データベース管理者）のタスク**

  RDBMSの構造から定期的なメンテナンスの必要性を解説し、管理者が実施する具体的なタスクやその実施方法を解説

© LPI-Japan 2015. All rights reserved.

データベースに求められること

■高性能

- 格納された大量のデータから必要なものを高速に検索する

■同時実行

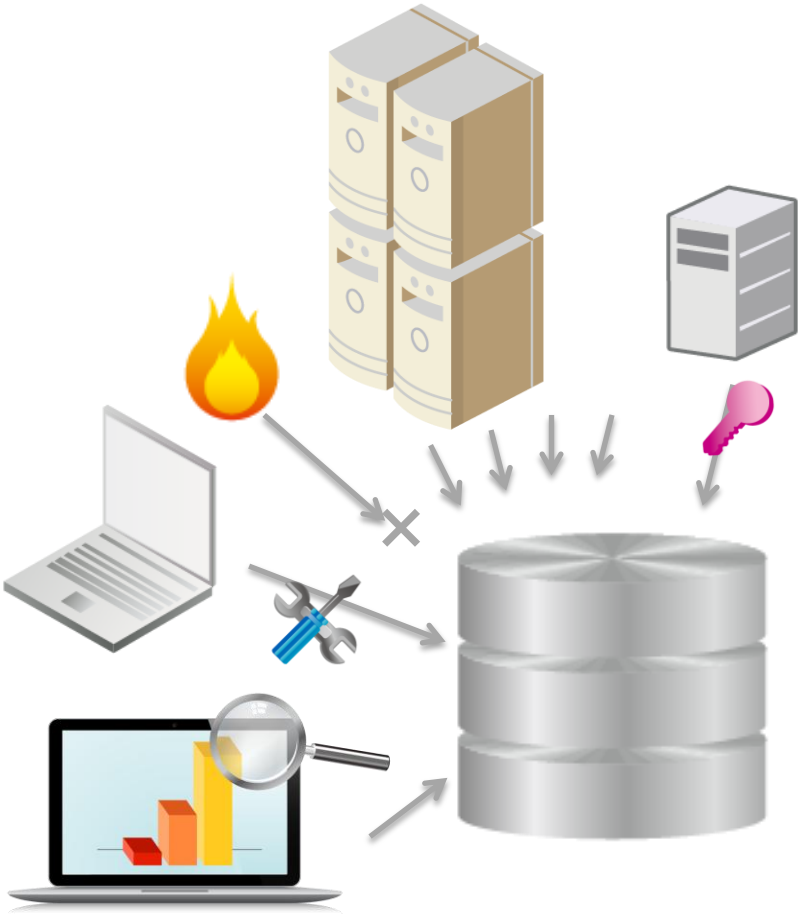
- 同時に多数のユーザがデータを参照し、任意のタイミングでデータを変更する

■耐障害性

- データを確実に保護し、万が一の障害時に復旧を可能とする

■その他

- データへのアクセス方法（SQL）
- 各種チューニング／メンテナンス
- 性能調査／障害調査のためのログ出力・診断機能や、アクセス制御／暗号化／監査などのセキュリティ機能、可用性・負荷分散を可能にするレプリケーションなど

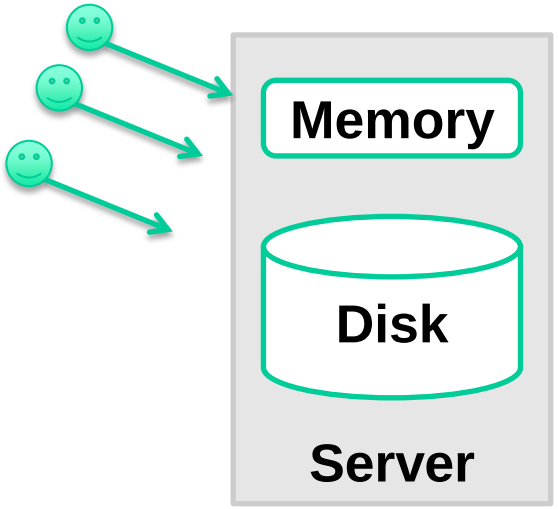


© LPI-Japan 2015. All rights reserved.

■キーワード

- ディスクI/Oの削減
- 共有メモリ
- ログ先行書き込み と チェックポイント
- チューニング

■考え方のポイント



- 性能の観点では、メモリのみで処理を続けることが理想だが、信頼性のためにデータは永続化したい
- 変更履歴をシーケンシャルI/Oで保存する事で、性能影響を抑えて永続化

シーケンシャル I/Oの負荷 << ランダム I/Oの負荷

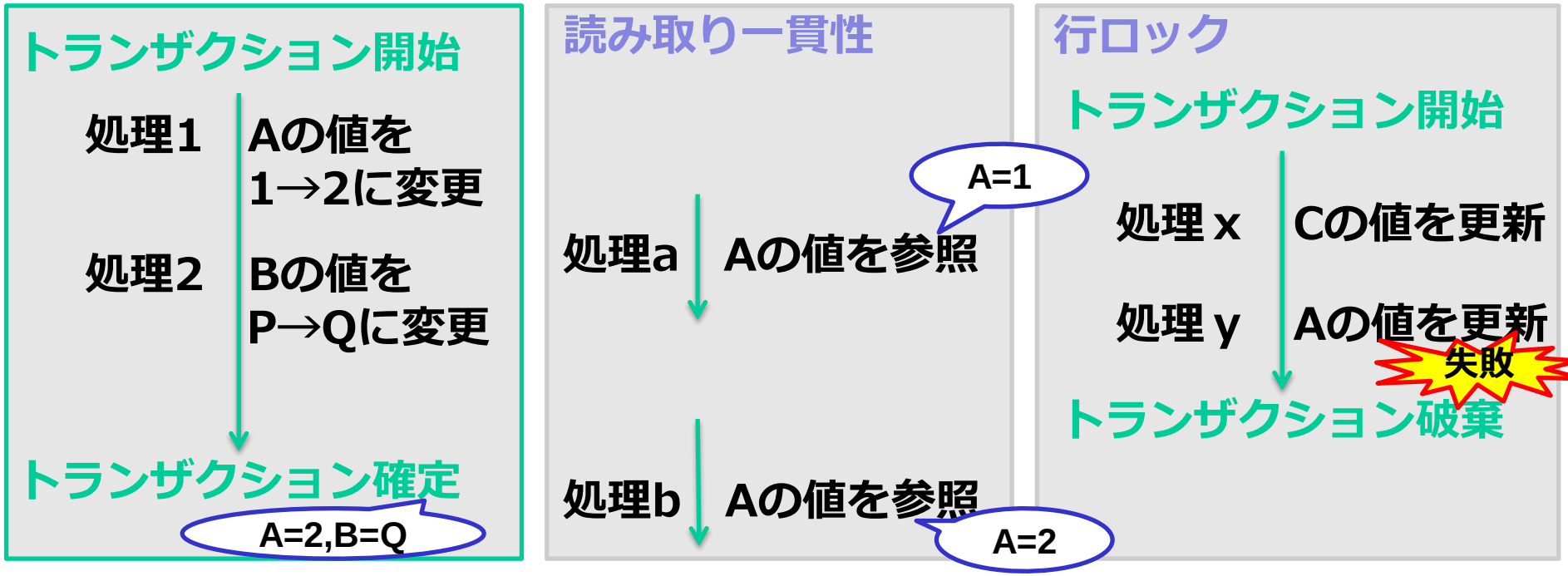
© LPI-Japan 2015. All rights reserved.

同時実行の実現

■キーワード

- トランザクション
- ロック
- 読み取り一貫性

■トランザクションとは



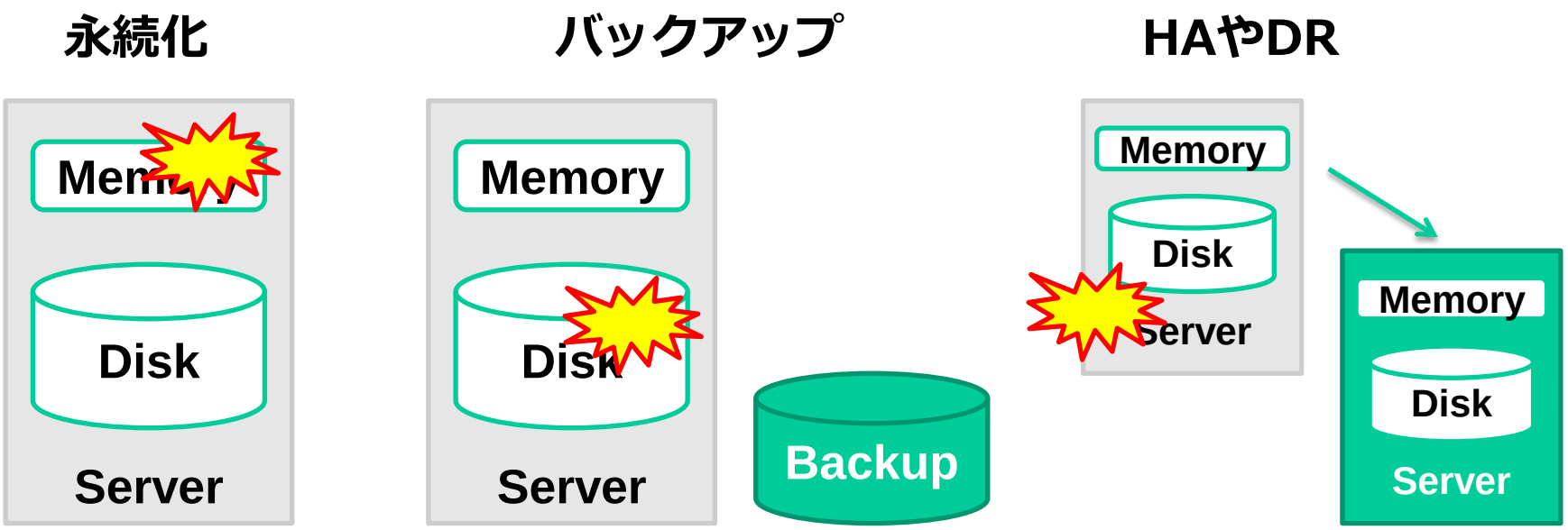
© LPI-Japan 2015. All rights reserved.

耐障害性の実現

■キーワード

- ログ先行書き込み と チェックポイント
- 障害の種類
- バックアップ・リカバリ

■障害からデータを保護する方法



© LPI-Japan 2015. All rights reserved.

アジェンダ

- **データベースに求められること**

  データベースに求められる「同時実行性」「高性能」「耐障害性」などの基本を整理し、これらをRDBMSで実現するための重要なキーワードを解説

- **RDBMSの構造**

  前章で挙げたデータベースとしての基本が、PostgreSQLではどのような仕組みで実装されているかを解説

- **SQL開発**

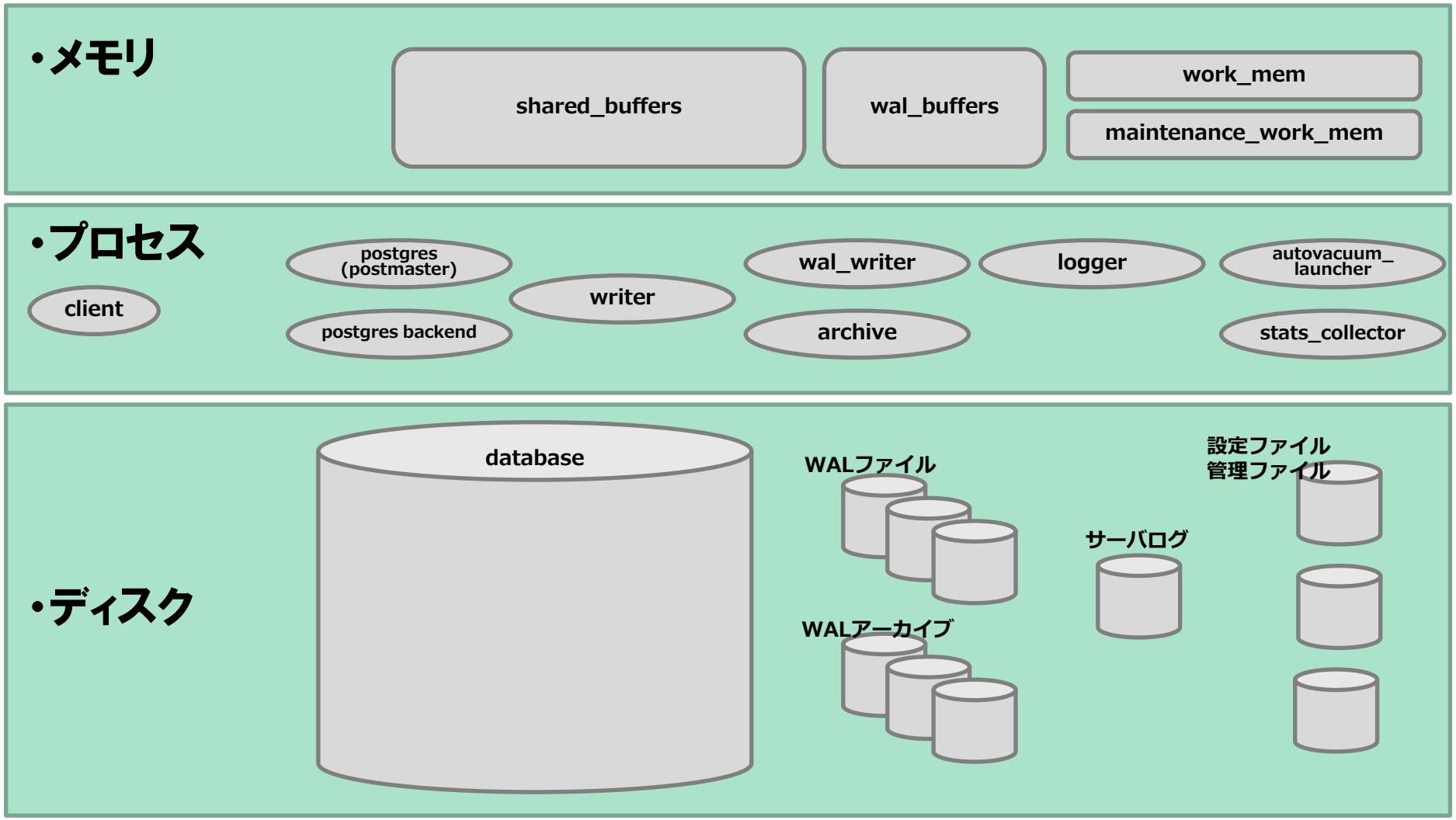
  RDBMSの共通言語である「SQL」の基本を解説

- **DBA（データベース管理者）のタスク**

  RDBMSの構造から定期的なメンテナンスの必要性を解説し、管理者が実施する具体的なタスクやその実施方法を解説

© LPI-Japan 2015. All rights reserved.

■メモリ、プロセス、ディスク領域からなるDB構造を正しく把握



© LPI-Japan 2015. All rights reserved.

■データベースクラスタ

- 1つ以上のデータベースと、管理情報・設定ファイルがあつまったもの
  - PostgreSQLは、データベースクラスタ単位で起動・停止を行う
  - 実体は構築時に指定するPostgreSQL関連の最上位のディレクトリ
    （ディレクトリを指す場合は、データディレクトリと記載される）
  - 環境変数$PGDATAにデータディレクトリのパスを設定しておく

■データベースクラスタの構成要素

内容	指定方法	ディレクトリ(ファイル)名
データディレクトリ	initdb -D	$PGDATA
WALファイル出力先	initdb -X	$PGDATA/pg_xlog
ユーザデータ格納先	TABLESPACE機能	$PGDATA/base
ログファイル出力先	パラメータ	$PGDATA/pg_log
アーカイブ退避先	パラメータ	$PGDATA/<指定した出力先>
状態管理・設定 ファイル群	--	postgresql.conf pg_hba.conf その他

© LPI-Japan 2015. All rights reserved.

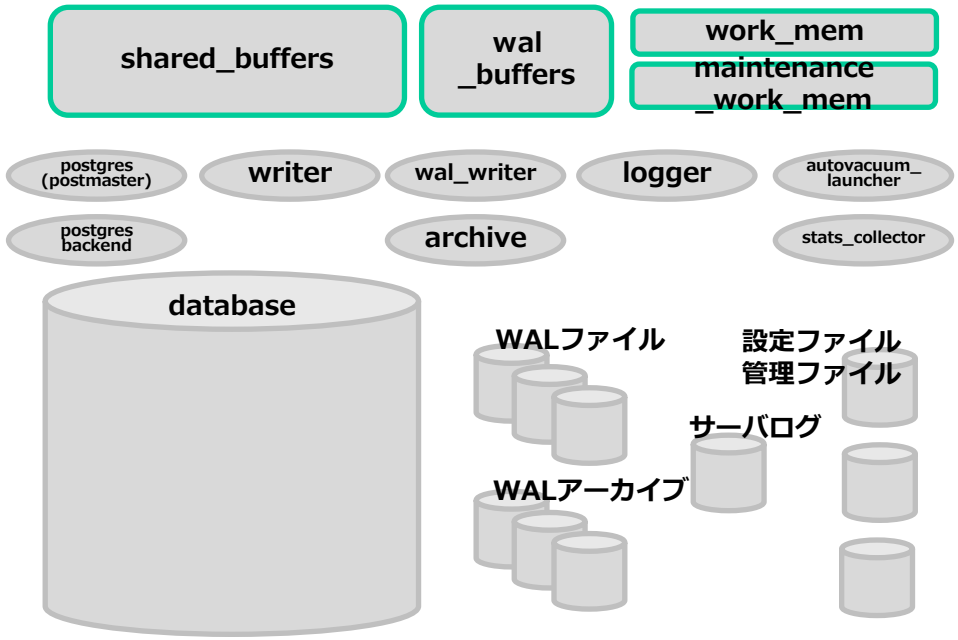
共有メモリ

- 共有バッファ
  - ディスクから読み取ったデータをキャッシュして、以降のユーザ要求に高速に応答
- WALバッファ
  - ログ先行書き込み (Write Ahead Logging)
  - 耐障害性とパフォーマンスを両立するための仕組み
  - WALファイルへのI/Oをシーケンシャルにするために、メモリ上に変更をキャッシュ

セッションメモリ

セッション毎に確保される領域

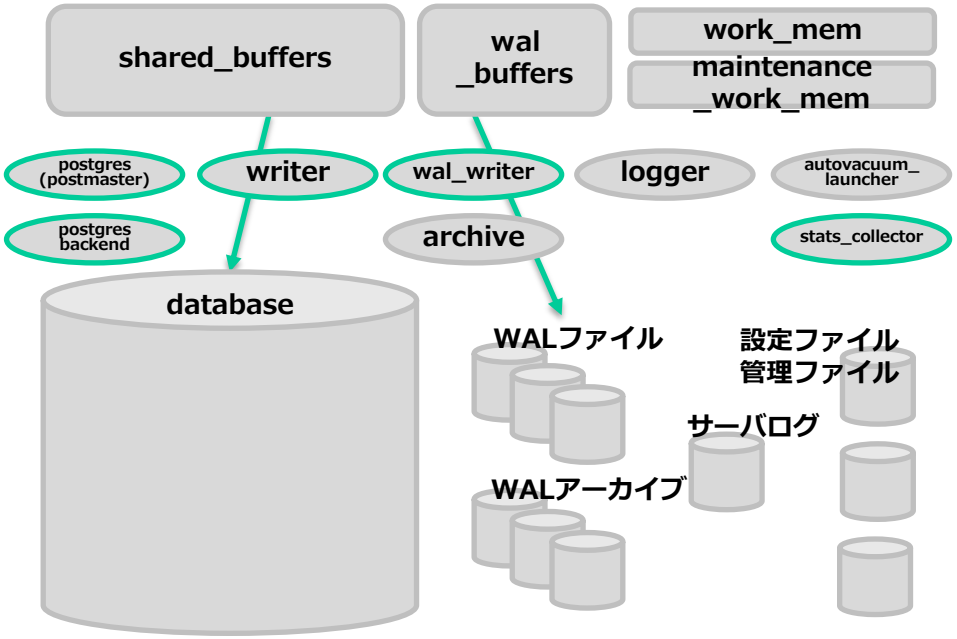
- ワークメモリ
  - ソートやハッシュの一時領域
- メンテナンスワークメモリ
  - メンテナンス操作



© LPI-Japan 2015. All rights reserved.

必須プロセス

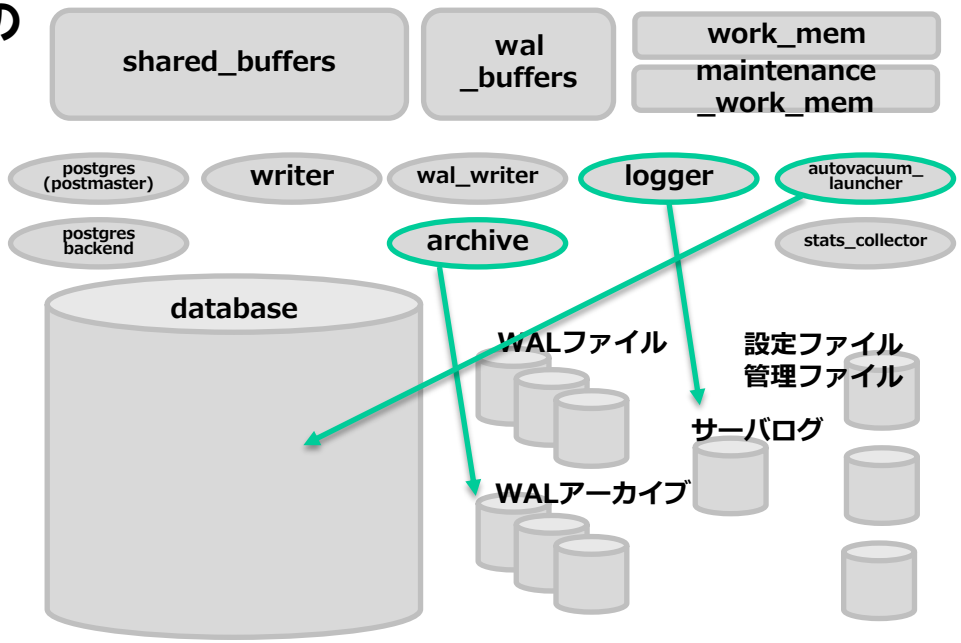
- postgres (postmaster) 、postgres backend
  - クライアントからの接続を待ち受ける、すべてのプロセスの親プロセス (postgres)
  - postgresプロセスによって起動され、クライアントからの処理を担当 (backend)
- writer
  - 共有バッファのデータをディスクに書き込むプロセス
  - チェックポイントやダーティバッファの書き込み
- wal writer
  - データの変更履歴をWALファイルに書き込む
- stats collector
  - 実行時統計情報を収集する



© LPI-Japan 2015. All rights reserved.

パラメータ設定により起動するプロセス

- logger
  - PostgreSQLサーバ実行時のログを記録するプロセス
  - パラメータ設定により有効化し、何をどこに保存するか指定できる
- archive
  - チェックポイント以前の不要なWALをPITRのために別のディスクに退避
- autovacuum launcher/worker
  - 自動VACUUMの閾値を超過したもの(表・列)に対してVACUUMを実行



© LPI-Japan 2015. All rights reserved.

【参考】PostgreSQLインストール

PostgreSQLをソースコードから任意のユーザでインストールする例

- ソースコードの展開

```
# useradd silver -u 2303 -g 1000 -d /home/silver -s /bin/bash
# su - silver
$ mkdir media
/* postgresql-9.4.4.tar.gzを転送しておく */
$ cd media
$ tar zxvf postgresql-9.4.4.tar.gz
$ ls
postgresql-9.4.4  postgresql-9.4.4.tar.gz
$ cd postgresql-9.4.4
$ ls
COPYRIGHT       HISTORY  Makefile  aclocal.m4  configure     contrib  src
GNUmakefile.in  INSTALL  README    config      configure.in  doc
```

- インストール
  - コマンド詳細は https://www.postgresql.jp/document/9.4/html/install-procedure.html
  - オプションの意味など、ここで例示しているコマンドは理解しておくこと

```
$ ./configure --prefix /home/silver/pg_home --with-libxml --with-openssl
$ make world
$ make install-world
PostgreSQL, contrib, and documentation installation complete.
$ ls $HOME/pg_home
bin  include  lib  share
```

© LPI-Japan 2015. All rights reserved.

【参考】データベースクラスタの初期化

インストール後、initdbでデータベースクラスタを初期化

- ディレクトリ作成/環境変数の設定

```
$ mkdir silver_data
$ cd $HOME
$ vi .bash_profile
-----
export PGHOME=/home/silver/pg_home
export PGDATA=/home/silver/silver_data
export PGDATABASE=silver
export PGPORT=2303
export PATH=$PGHOME/bin:.:$PATH
-----
$ source .bash_profile
$ env | grep PGDATA
PGDATA=/home/silver/silver_data
```

- データベースクラスタの初期化

```
$ initdb -E utf8 --no-locale -D $HOME/silver_data
$ vi $PGDATA/postgresql.conf
------
port=2303
------
```

© LPI-Japan 2015. All rights reserved.

■インストール後の確認

- ディレクトリ構造

```
$ ls -ltr $PGDATA
drwx------. 2 silver postgres  4096  8月 18 09:21 2015 pg_snapshots
drwx------. 2 silver postgres  4096  8月 18 09:21 2015 pg_serial
drwx------. 2 silver postgres  4096  8月 18 09:21 2015 pg_dynshmem
drwx------. 2 silver postgres  4096  8月 18 09:21 2015 pg_twophase
drwx------. 2 silver postgres  4096  8月 18 09:21 2015 pg_replslot
drwx------. 4 silver postgres  4096  8月 18 09:21 2015 pg_multixact
drwx------. 2 silver postgres  4096  8月 18 09:21 2015 pg_tblspc
drwx------. 2 silver postgres  4096  8月 18 09:21 2015 pg_stat
drwx------. 4 silver postgres  4096  8月 18 09:21 2015 pg_logical
-rw-------. 1 silver postgres     4  8月 18 09:21 2015 PG_VERSION
drwx------. 2 silver postgres  4096  8月 18 09:21 2015 pg_subtrans
drwx------. 2 silver postgres  4096  8月 18 09:23 2015 pg_notify
drwx------. 2 silver postgres  4096  8月 18 09:39 2015 pg_stat_tmp
-rw-------. 1 silver postgres    34  8月 18 09:23 2015 postmaster.opts
drwx------. 2 silver postgres  4096  8月 18 09:21 2015 pg_clog
-rw-------. 1 silver postgres    88  8月 18 09:21 2015 postgresql.auto.conf
-rw-------. 1 silver postgres  1636  8月 18 09:21 2015 pg_ident.conf
-rw-------. 1 silver postgres  4462  8月 18 09:21 2015 pg_hba.conf
-rw-------. 1 silver postgres 21268  8月 18 09:22 2015 postgresql.conf
drwx------. 3 silver postgres  4096  8月 18 09:21 2015 pg_xlog
drwx------. 6 silver postgres  4096  8月 18 09:24 2015 base
drwx------. 2 silver postgres  4096  8月 18 09:39 2015 global
-rw-------. 1 silver postgres    82  8月 18 09:23 2015 postmaster.pid
```

© LPI-Japan 2015. All rights reserved.

【参考】インスタンスの起動と接続

PostgreSQLインスタンス

- postgresql プロセス、メモリ領域からなるデータベースインスタンスが稼働
- データベースクラスタ単位でインスタンスが稼働
  - 変更履歴 (WAL) やログファイル、各種設定値はデータベース間で共有

pg_ctlコマンド

- postgresql.confからパラメータを反映しインスタンスを起動

```
$ ps -ef | grep postgres /* PostgreSQL関連のプロセスを表示 */
silver   61468     1  0 09:23 pts/0    00:00:00 /home/silver/pg_home/bin/postgres
silver   61470 61468  0 09:23 ?        00:00:00 postgres: checkpointer process
silver   61471 61468  0 09:23 ?        00:00:00 postgres: writer process
silver   61472 61468  0 09:23 ?        00:00:00 postgres: wal writer process
silver   61473 61468  0 09:23 ?        00:00:00 postgres: autovacuum launcher process
silver   61474 61468  0 09:23 ?        00:00:00 postgres: stats collector process
```

pg_ctlコマンド例

操作	コマンド例	意味
起動	pg_ctl start -w -t 60	起動完了まで60秒待機し成功したらプロンプトを戻す
停止	pg_ctl stop -m fast	fastモードを指定してシャットダウン
稼働状態の確認	pg_ctl status	サーバの稼働状態を確認
設定再読み込み	pg_ctl reload	reloadで読み込み可能なパラメータを反映

© LPI-Japan 2015. All rights reserved.

■データベースへの接続

- 初期のデータベース名はpostgres
- 初期のユーザ名は”PostgreSQLをインストールしたOSユーザと同一”

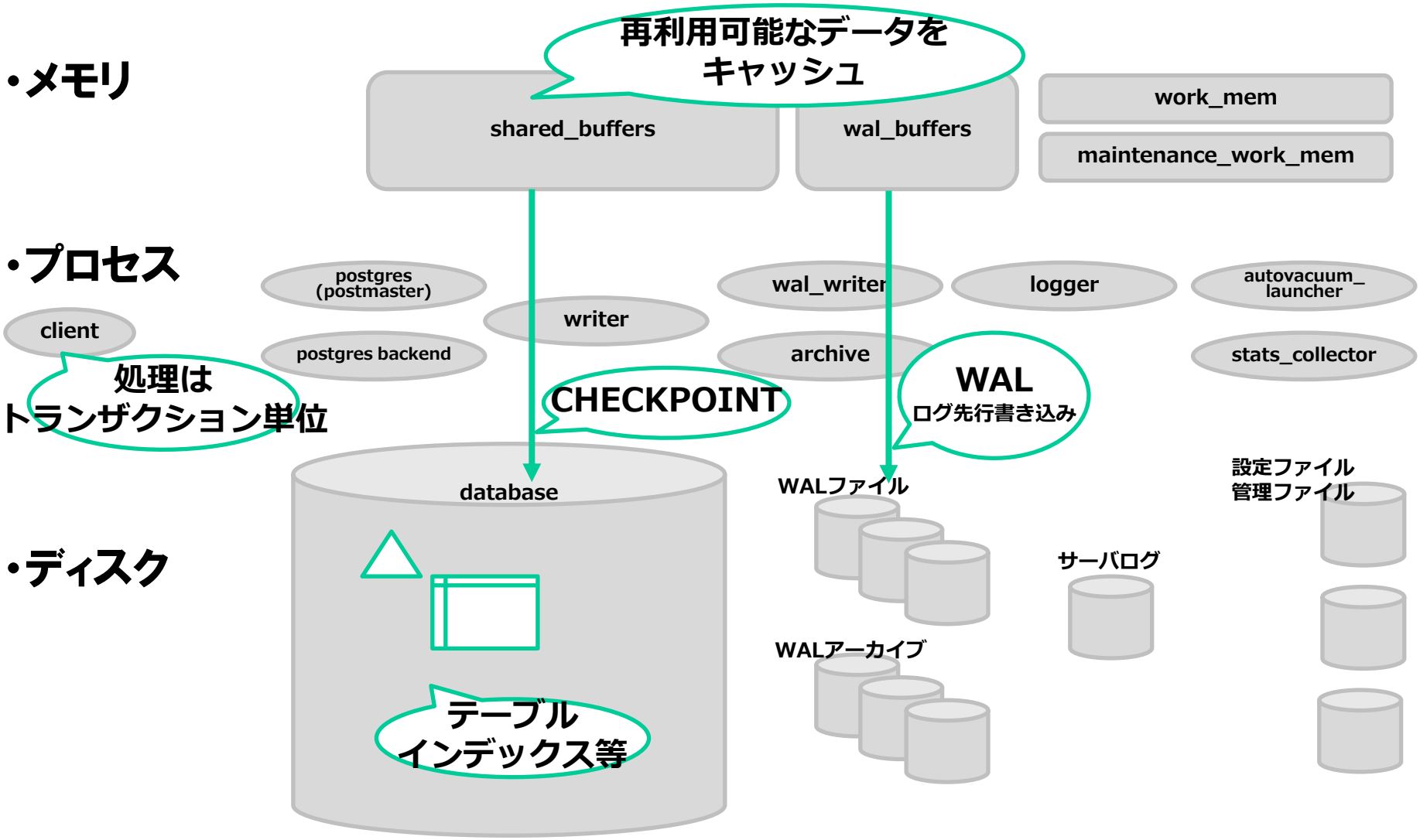
```
$ pg_ctl start   /* データベースクラスタを起動 */
$ psql -h localhost -p 2303 -U silver -d postgres   /* データベースを指定して接続 */
postgres=#
```

- ユーザデータベースの作成と接続

```
$ psql -U silver postgres
postgres=# CREATE DATABASE silver OWNER silver;
postgres=# ¥l
                            List of databases
   Name    | Owner  | Encoding | Collate | Ctype | Access privileges
-----------+--------+----------+---------+-------+-------------------
 postgres  | silver | UTF8     | C       | C     |
 silver    | silver | UTF8     | C       | C     |
 template0 | silver | UTF8     | C       | C     | =c/silver        +
           |        |          |         |       | silver=CTc/silver
 template1 | silver | UTF8     | C       | C     | =c/silver        +
           |        |          |         |       | silver=CTc/silver
postgres=# ¥q
$ psql   /* 環境変数PGPORT、PGUSER、PGDATABASEなどを設定しておくことで補完可能 */
silver=#
```

© LPI-Japan 2015. All rights reserved.

■DB構造をもとに、それぞれがどのように動作するか理解する



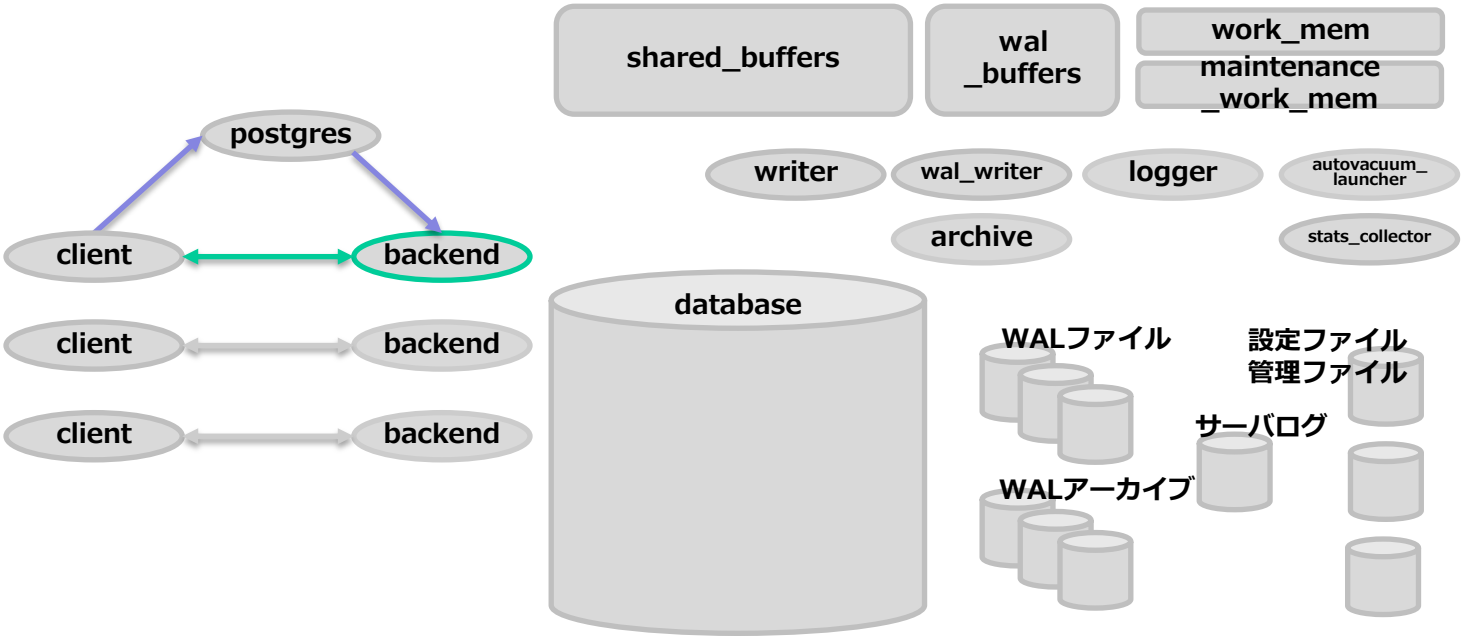
© LPI-Japan 2015. All rights reserved.

■セッション開始：postgresプロセスが認証を担当

①クライアントから認証要求

②postgresプロセスによる認証が完了すると、postgres　backendプロセスが起動し、クライアントとセッションを確立

※セッション毎にbackendプロセスが起動され、クライアントと1対1対応する

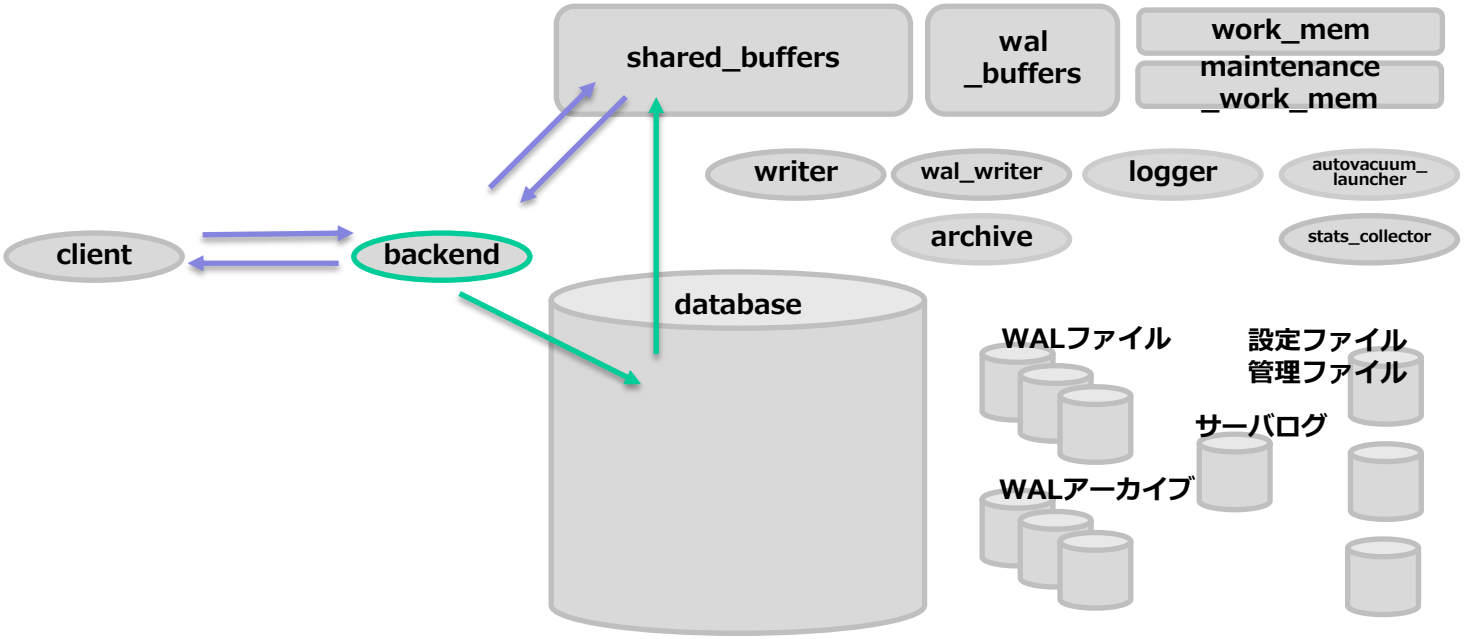


© LPI-Japan 2015. All rights reserved.

■参照：共有バッファを利用

①クライアントからクエリ発行
②backendプロセスが必要なデータを共有バッファから探す
③バッファ上に無い場合は、ディスクの該当ブロックをバッファに載せる
④backendプロセスがクライアントに結果を返却

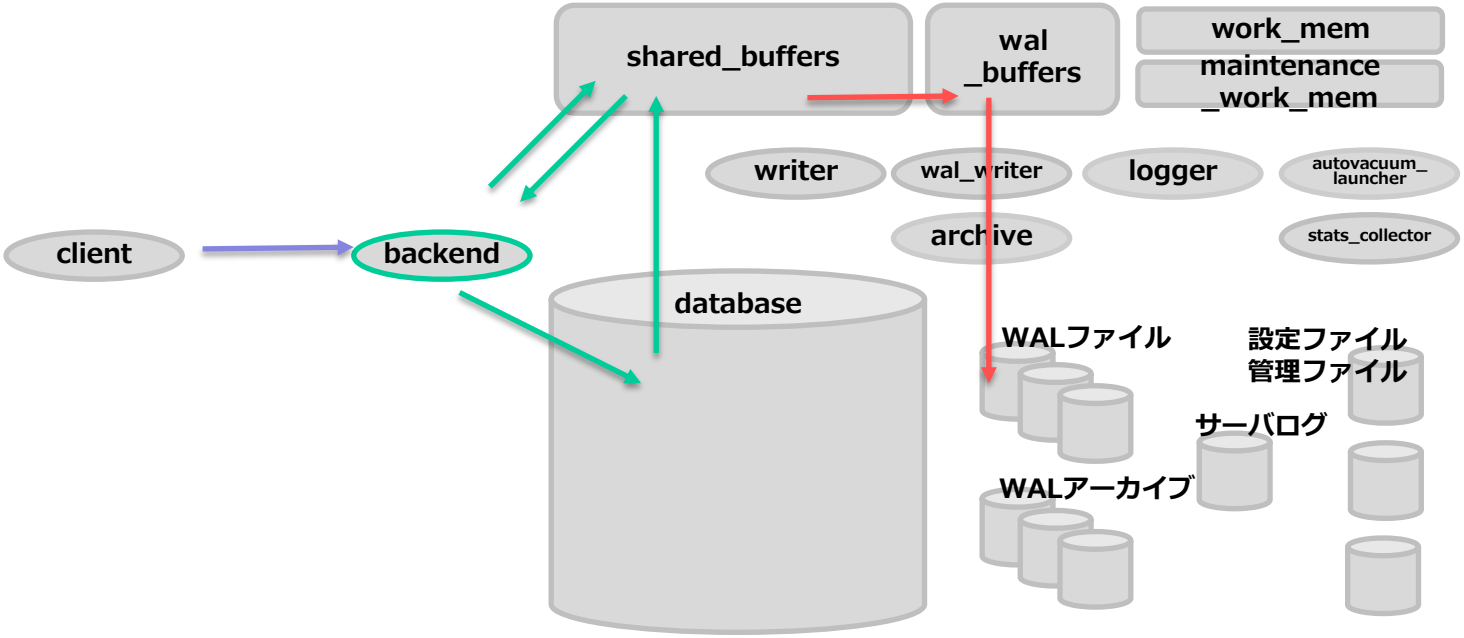
※アクセスしたデータはバッファ上にキャッシュされ、以降は高速に結果を返す仕組み



© LPI-Japan 2015. All rights reserved.

更新：共有バッファ上の更新＋WALによる変更履歴の永続化

①クライアントからクエリ発行（UPDATE、INSERT、DELETE）
②backendプロセスが必要なデータを共有バッファから探す
③バッファ上に無い場合は、ディスクの該当ブロックをバッファに載せる
④変更内容をWALバッファ上のWALレコードとして作成
⑤共有バッファ上のデータを更新
⑥クライアントが変更を確定（COMMIT）すると、WALレコードをWALファイルに永続化し、
WAL書き込みが成功したらクライアントに成功を返す



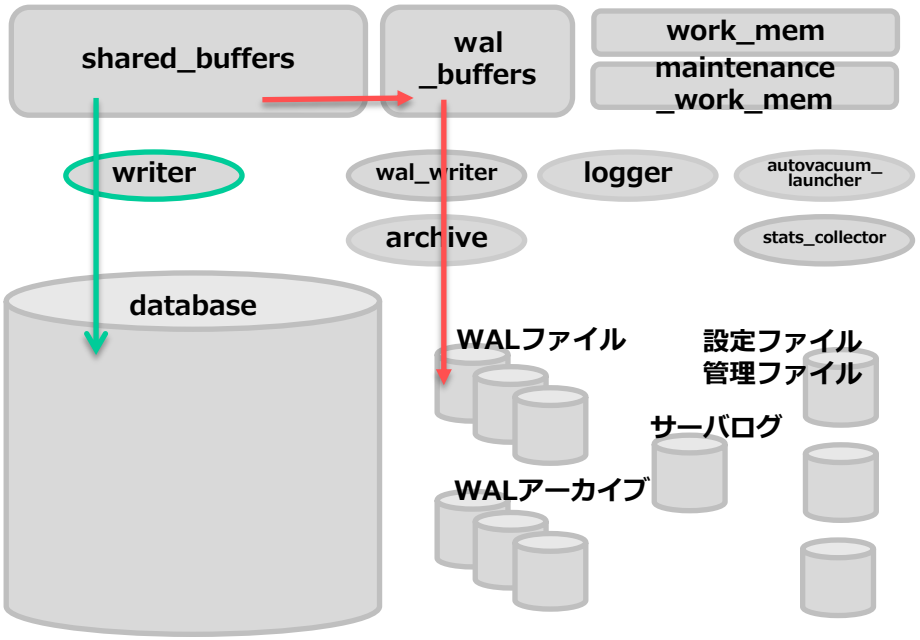
© LPI-Japan 2015. All rights reserved.

共有バッファ上のデータをデータファイルに書き込む

- データファイルへの書き込みは個々のSQL実行とは非同期に行う
- チェックポイントとWAL
  - チェックポイント以降のWALファイルは非常に重要
    - WALは個々のSQLによる変更履歴を追跡可能なように都度ディスクに記録
    - チェックポイントはある時点のバッファの内容をすべてディスクに反映

writerの動作	特徴
ダーティバッファの書き込み	更新とは非同期に、システムの負荷を極力抑えて実行される。どこまで書くか保証しない。
チェックポイント	ある瞬間のバッファの内容が確実にディスクに反映されたことを保証するタイミング。大量のI/Oが発生する。

wal writerの動作	特徴
WALファイルへの書き込み	COMMIT時、確実にディスクに書く。一定時間経過やWALバッファが不足する際にも書き込み。

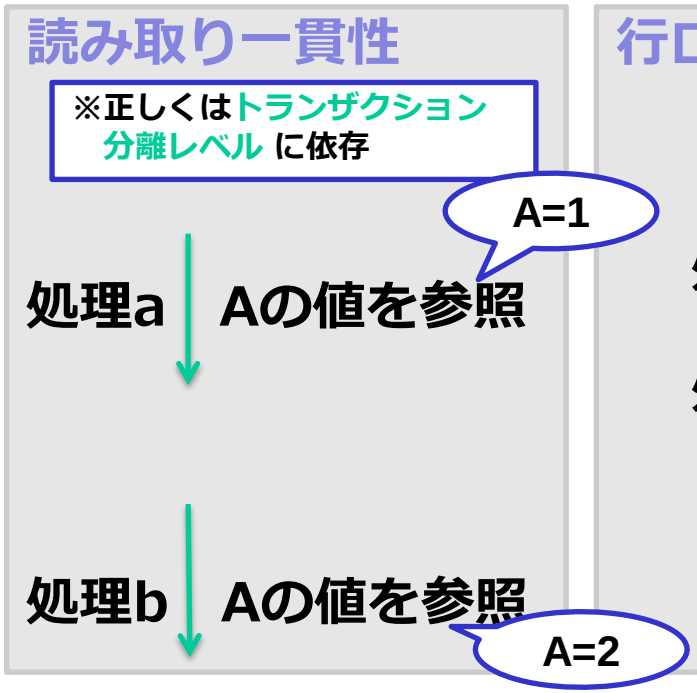


© LPI-Japan 2015. All rights reserved.

トランザクション

■トランザクション＝一連の操作をひとまとまりとして扱う処理単位
- BEGINコマンドでトランザクション開始/ENDまたはCOMMITコマンドで終了
- ABORTまたはROLLBACKコマンドで破棄

■トランザクション中の排他制御
- 変更中のデータは確定まで他のセッションから見れない（読み取り一貫性）
- 同一の行を他のセッションから更新されない（行ロック）



© LPI-Japan 2015. All rights reserved.

MVCC (Multi Version Concurrentry Control)

- 「追記型」と言われるアーキテクチャ
- 同じ行を表す複数の行バージョンが同時に存在する
  - 「Aの値」を参照しているが、実際には同じ行を表す「A」と「A」が同時に存在する
  - Aが更新処理により追記された行
    - 変更がコミットされた場合、他のセッションからもAが見える
    - 変更がロールバックされた場合、Aは無かったものとし、元のAが見える

行バージョン管理用の領域 ↓

id	value
A	1
B	1
C	1
D	1
E	1
A	2

```
BEGIN;
UPDATE test SET vaule=2
WHERE id=A;
END;
SELECT value WHERE
id=A;
2
```

```
ROLLBACK;
SELECT value WHERE
id=A;
1
```

テーブルロックと行ロック

- PostgreSQLでは、SQLコマンド毎に適した強度でテーブルロックを獲得
  - テーブルAをUPDATE中は、同じテーブルAにINSERTできる
  - 同時にテーブルAに対するDROPや、VACUUM FULLはできない
- DML文では、テーブルロックとは別に行ロックを獲得
  - 例えば、テーブルAのid=1をUPDATEするトランザクション実行中、別トランザクションでid＝1をDELETEできない

	現在のロックモード							
要求するロックモード	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE								■
ROW SHARE							■	■
ROW EXCLUSIVE					■	■	■	■
SHARE UPDATE EXCLUSIVE				■	■	■	■	■
SHARE			■	■		■	■	■
SHARE ROW EXCLUSIVE			■	■	■	■	■	■
EXCLUSIVE		■	■	■	■	■	■	■
ACCESS EXCLUSIVE	■	■	■	■	■	■	■	■

※各SQLが獲得するロックレベルは、https://www.postgresql.jp/document/9.4/html/explicit-locking.html　を参照

© LPI-Japan 2015. All rights reserved.

LOCK TABLEでテーブルロックを確保

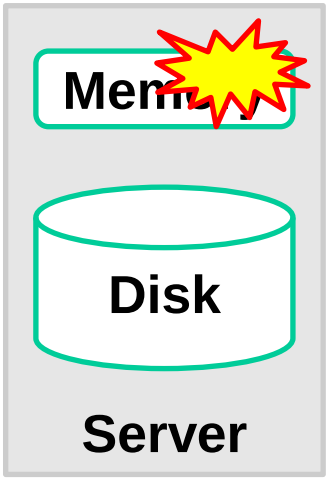
```
silver=# ¥h lock
------------------------------------------------------------------------------
LOCK [ TABLE ] [ ONLY ] name [ * ] [, ...] [ IN lockmode MODE ] [ NOWAIT ]
where lockmode is one of:
    ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE
    | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE
------------------------------------------------------------------------------
silver=# begin;
BEGIN
silver=# LOCK TABLE dog;
LOCK TABLE
          ---------------------------->

silver=# ROLLBACK; ---------------------->



silver=# BEGIN;
BEGIN
silver=# LOCK TABLE dog IN share MODE;
LOCK TABLE

                   ---------------------->



silver=# ROLLBACK;
```

```
/* 同時に別セッションで実行 */
silver=# begin;
BEGIN
silver=# SELECT * FROM dog;
/* ロック解放まで待機 */

 id |  name  |     kind     | owner_cd
----+--------+--------------+----------
  1 | Poppy  | Westy        |        1


silver=# begin;
BEGIN
silver=# SELECT * FROM dog;
/* ロック競合しないので即座に結果が得られる */
 id |  name  |     kind     | owner_cd
----+--------+--------------+----------
  1 | Poppy  | Westy        |        1
```

© LPI-Japan 2015. All rights reserved.

■障害の種類と復旧方法を整理

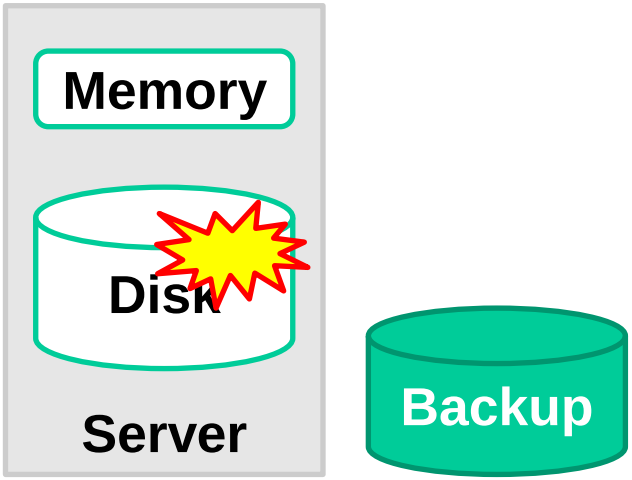
インスタンス障害



発生ケース
- 停電
- 強制停止
- プロセス障害　など

対処
H/W、データが共に正常であり、データベースの再起動で復旧

メディア障害

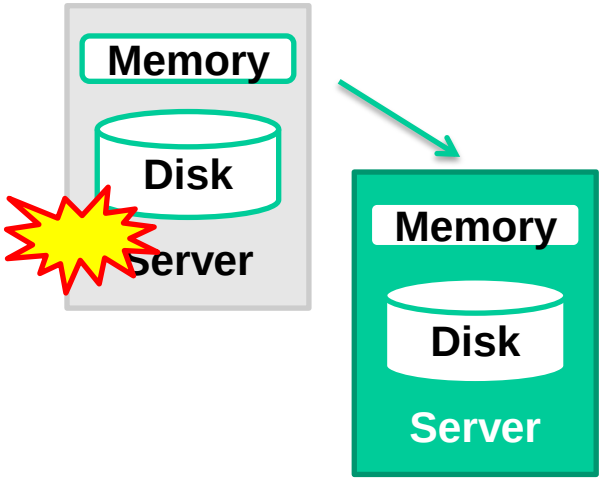


発生ケース
- HDDの故障
- データファイル削除

など

対処
必要なデータを喪失しているため、バックアップからの復旧が必要

サーバ障害



発生ケース
- メモリの故障
- 大規模災害　など

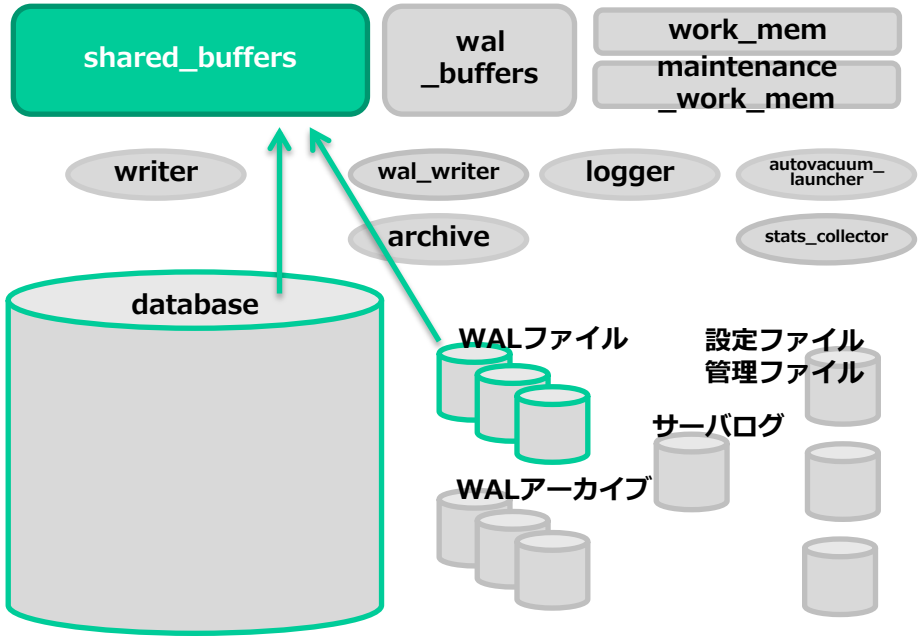
対処
代替機にデータを戻し再起動(HAやBCPも同様の考え方とする)

© LPI-Japan 2015. All rights reserved.

障害復旧（チェックポイントとWAL）

■ディスクに書きこまれた（永続化された）データ
- チェックポイント時点の状態が確実に反映されている「データファイル」
- SQLによる変更を刻一刻と記録している「WALファイル」

■インスタンス障害、電源障害など
- チェックポイント以降のデータファイル＋WALが残っているため、管理者は特別な操作をすることなくデータベースの再起動で復旧が可能



© LPI-Japan 2015. All rights reserved.

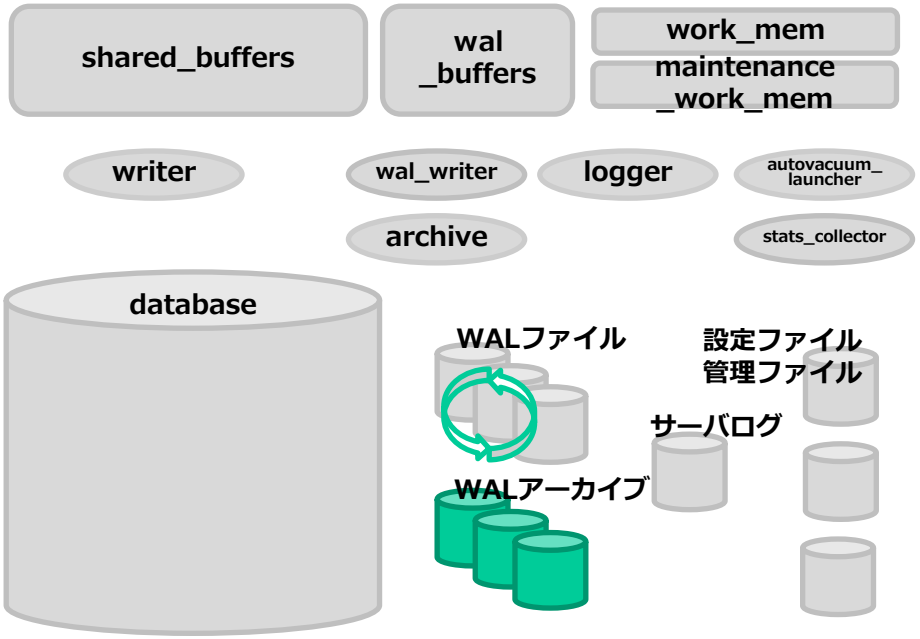
障害復旧（WALの循環とアーカイブ）

■WALファイルの領域は循環利用
- 一定期間変更履歴をもつためには、その分だけディスク領域が必要
- チェックポイント後のWALファイルは不要とみなされ、自動削除される

■アーカイブ運用
- WALファイルを削除前にWALアーカイブとして保存
- 過去のデータファイル＋WALアーカイブ＋（最新の）WALファイル

↓

ある時点のバックアップを活用

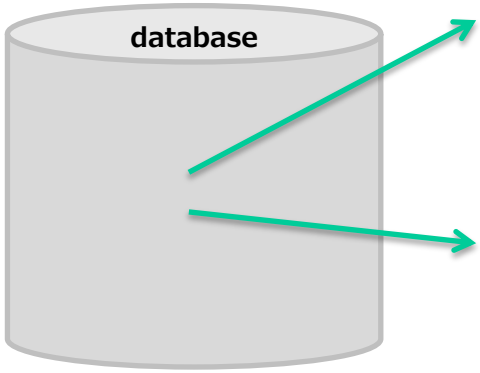


© LPI-Japan 2015. All rights reserved.

■データベースのバックアップ

バックアップ種類		特徴
物理バックアップ	オフラインバックアップ	データベースを停止し、データファイルをコピーする。 データの更新は無いため、そのままオープンして利用可能。
	オンラインバックアップ	データベース稼働中、**バックアップモードに変更することで**常に更新されるデータファイルのコピーを可能とする。 コピー中に行われた変更分を追跡するために**WALアーカイブ・WALファイルとセットで利用。**
論理バックアップ	オンラインで論理的なデータを保存	専用の論理バックアップツールpg_dumpや、COPY文またはSELECT文で取得可能な論理データのバックアップ。 **≒ある時点でのSELECT結果をファイルに保存**

※データは常時更新される可能性があり、稼働中の単純なコピーは不可

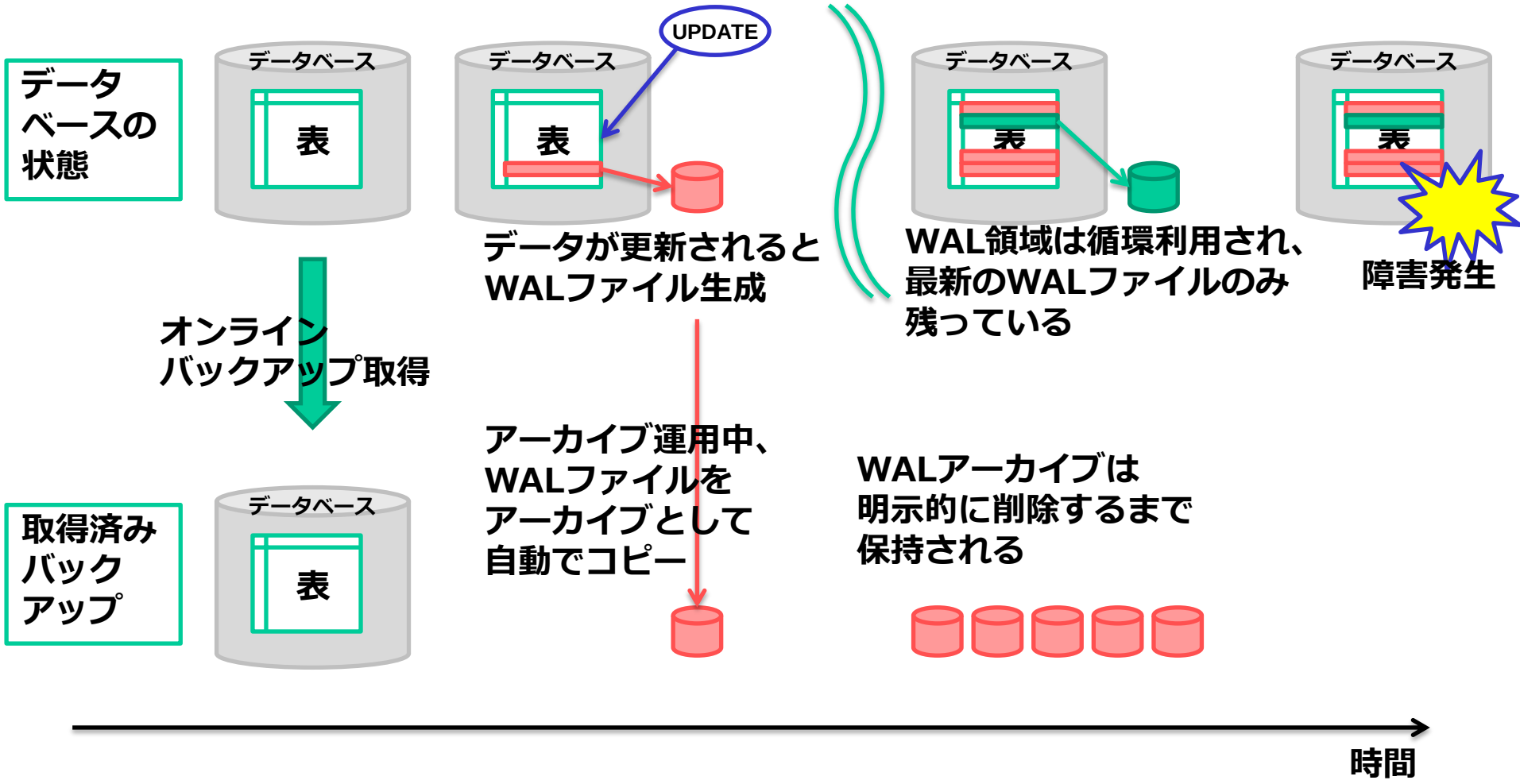


データベースの停止が可能
- **データファイルのコピー可能**

データベース稼働中
- **論理バックアップ**
  ある時点のSELECTした結果を保存
- **物理バックアップ**
  アーカイブ運用・バックアップモード必須

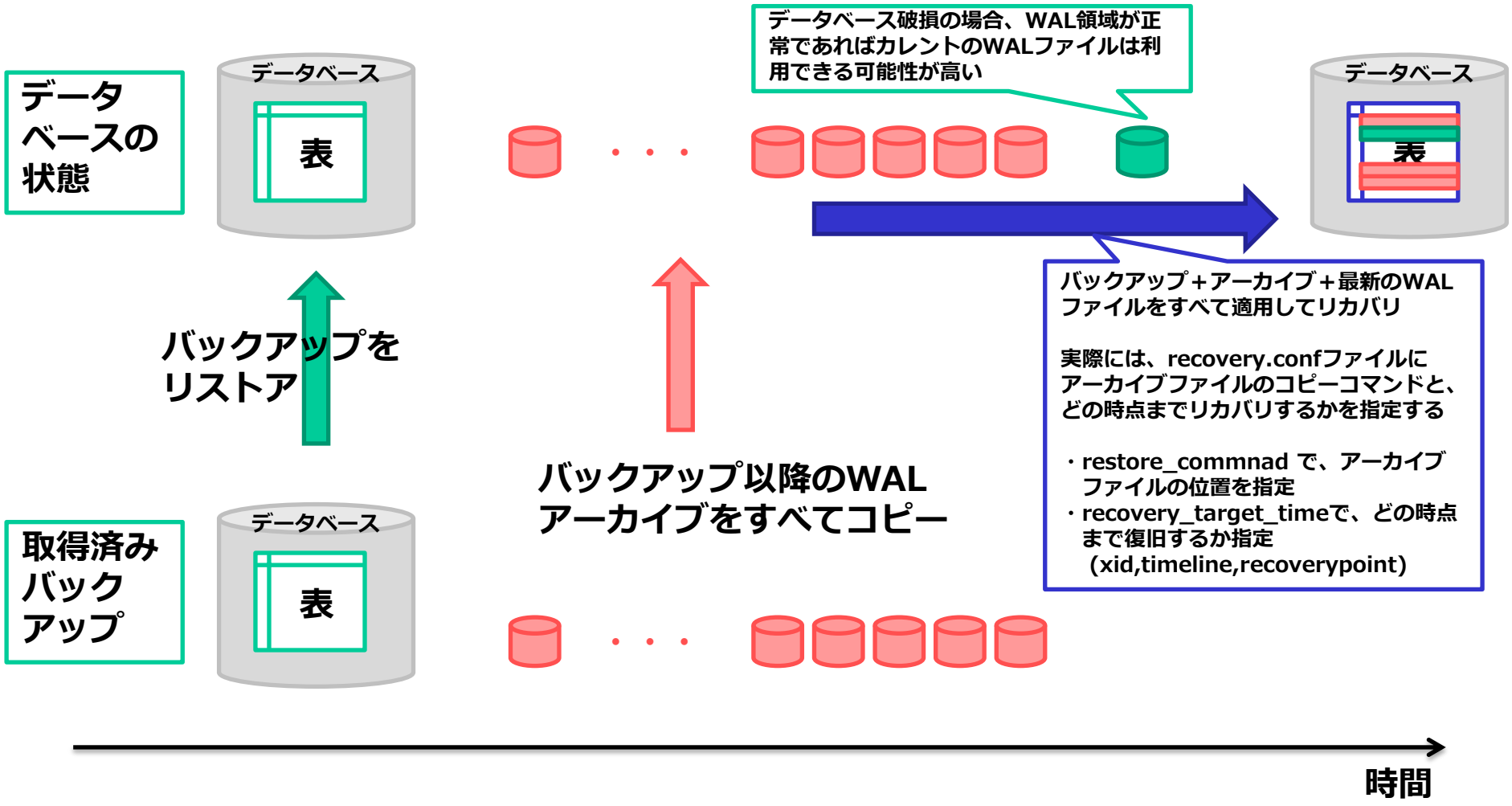
© LPI-Japan 2015. All rights reserved.

■オンライン・バックアップからリカバリ



© LPI-Japan 2015. All rights reserved.

■オンライン・バックアップからリカバリ



© LPI-Japan 2015. All rights reserved.

【参考】バックアップ・リカバリの詳細

アシスト 「お客様の最高」のために

OSS-DB

物理バックアップの参考情報

- アーカイブ運用の設定
  - https://www.postgresql.jp/document/9.4/html/continuous-archiving.html#BACKUP-ARCHIVING-WAL
- ベースバックアップの取得
  - https://www.postgresql.jp/document/9.4/html/continuous-archiving.html#BACKUP-BASE-BACKUP
- リカバリ
  - https://www.postgresql.jp/document/9.4/html/continuous-archiving.html#BACKUP-PITR-RECOVERY
- 押さえておくべきキーワード
  - PITR (Point In Time Recovery) 、timeline、recovery.conf

論理バックアップの参考情報

- pg_dump/pg_dumpall/pg_restore全ての使い方をマスターしておくこと
- pg_dump
  - 論理バックアップを独自形式で取得 (テキスト形式も選択可能)
  - 独自形式のバックアップは、pg_restoreを用いてリストア
- pg_dumpall
  - 論理バックアップをテキスト形式で取得
  - psqlを用いてリストア

© LPI-Japan 2015. All rights reserved.

アジェンダ

- **データベースに求められること**

  データベースに求められる「同時実行性」「高性能」「耐障害性」などの基本を整理し、これらをRDBMSで実現するための重要なキーワードを解説

- **RDBMSの構造**

  前章で挙げたデータベースとしての基本が、PostgreSQLではどのような仕組みで実装されているかを解説

- **SQL開発**

  RDBMSの共通言語である「SQL」の基本を解説

- **DBA（データベース管理者）のタスク**

  RDBMSの構造から定期的なメンテナンスの必要性を解説し、管理者が実施する具体的なタスクやその実施方法を解説

© LPI-Japan 2015. All rights reserved.

表名や列名を指定し、データへのアクセスを可能とする

- すべてのRDBMS共通の言語
- データの物理的な構造を意識せずアクセスが可能
  - テーブル名、列名を指定し、条件に合致する行を抽出
  - データの物理構造やアクセス方法はRDBMSに任せる
- テーブルの作成や稼働状態の確認などもSQLで実施

SQLの分類

分類	コマンド例	特徴
問合せ Query	SELECT	表名、列名を指定して、条件に合致する行データを取得する。**関連する項目を条件に複数の表を結合**できる。
データ操作 DML	INSERT、UPDATE、DELETE	表を指定して新規行データの挿入、既存行の更新・削除を行う。問合せ同様、条件に合致する行に対する操作であり、複数行をまとめて操作できる。
データ制御 DCL	BEGIN、END、ABORT GRANT、REVOKE	トランザクションを明示的に制御するほか、データへのアクセス制御の管理なども行う。
データ定義 DDL	CREATE、DROP、ALTER	表や索引などのオブジェクトを作成、管理する。

© LPI-Japan 2015. All rights reserved.

表名や列名を指定し、データへのアクセスを行う

- 列名（選択リスト）、表名（FROM句）の指定は必須
- 条件の指定（WHERE句）により該当する行を選択

dog表から飼い主の住所を検索

```
SELECT name,owner,address
FROM   dog
WHERE  name = 'Poppy';

name   |  owner |  address
-------+--------+------------
Poppy  | kida   | 千葉県○○市
```

dog表

id	name	kind	owner	address
001	Poppy	Westy	kida	千葉県○○市
002	Mitten	mix	kida	千葉県○○市
003	Pearl	mix	k.kida	東京都△△区
004	Luke	Dachshund	y.kida	神奈川県××市
005	Robbin	Schnauzer	morioka	千葉県○○市
006	Andy	Schnauzer	morioka	千葉県○○市
007	Ace	Jack Russell	sakamoto	東京都△△区

© LPI-Japan 2015. All rights reserved.

■複数の表を結合して、一つの表のように扱う

- 列名（選択リスト）、結合する全ての表名（FROM句、JOIN句）を指定
- 結合条件（JOIN 表名 ON 条件）を指定

■dog表とonwer表を結合

id	name	kind	owner_cd	o_name	o_address
001	Poppy	Westy	001	kida	千葉県○○市
002	Mitten	mix	001	kida	千葉県○○市
003	Pearl	mix	002	k.kida	東京都△△区
004	Luke	Dachshund	003	y.kida	神奈川県××市
005	Robbin	Schnauzer	004	morioka	千葉県○○市
006	Andy	Schnauzer	004	morioka	千葉県○○市
007	Ace	Jack Russell	005	sakamoto	東京都△△区

■結合条件　JOIN ONを指定

```
SELECT d.name,o.o_name,o.o_address
FROM   dog d JOIN owner o
ON     d.owner_cd = o.owner_cd
WHERE  name = 'Poppy';

name   | o_name | o_address
-------+--------+-----------
Poppy  | kida   | 千葉県○○市
```

■dog表

id	name	kind	owner_cd
001	Poppy	Westy	001
002	Mitten	mix	001
003	Pearl	mix	002
004	Luke	Dachshund	003
005	Robbin	Schnauzer	004
006	Andy	Schnauzer	004
007	Ace	Jack Russell	005

■owner表

owner_cd	o_name	o_address
001	kida	千葉県○○市
002	k.kida	東京都△△区
003	y.kida	神奈川県××市
004	morioka	千葉県○○市
005	sakamoto	東京都△△区

© LPI-Japan 2015. All rights reserved.

結合の種類

- （従来の）結合
  - JOIN句を用いず、WHERE条件で結合
- クロス結合
  - とりうる全ての組み合わせを指す
- 外部結合
  - 片方の表にしかデータが無い場合
  - 例） dog表owner_cd列に「里親募集中」を表すコード「000」が入っている
- 自然結合
  - 結合する表に同じ列名が1つある場合に結合条件を自動で補完
  - ただし、データが持つ意味は考慮されず、同名の列が複数ある場合は使用不可
  - 例） 両者のowner_cd列は同名であり、共通の意味を持つデータを格納

クエリ例

```
-- 従来の結合
SELECT d.name,o.o_name,o.o_address
FROM   dog d,owner o
WHERE  d.owner_cd = o.owner_cd
AND    name = 'Poppy';

name   | o_name | o_address
-------+--------+------------
Poppy  | kida   | 千葉県○○市

-- クロス結合
SELECT d.name,o.o_name,o.o_address
FROM   dog d,owner o
--WHERE  d.owner_cd = o.owner_cd
WHERE  name = 'Poppy';

-- 外部結合
SELECT d.name,o.o_name,o.o_address
FROM   dog d LEFT OUTER JOIN owner o
ON     d.owner_cd = o.owner_cd;

-- 自然結合
SELECT d.name,o.o_name,o.o_address
FROM   dog d NATURAL JOIN owner o
WHERE  name = 'Poppy';
```

© LPI-Japan 2015. All rights reserved.

検索結果の上位〇件を表示

- LIMITで指定した以降のデータの取得を中断し、高速に結果を返す
- OFFSET以降、LIMITまで
- 通常はORDER BY (ソート) と組み合わせ、指定した順位の上位を検索

dog表(idで降順にソート)

id	name	kind	owner_cd
007	Ace	Jack Russell	005
006	Andy	Schnauzer	004
005	Robbin	Schnauzer	004
004	Luke	Dachshund	003
003	Pearl	mix	002
002	Mitten	mix	001
001	Poppy	Westy	001

```
SELECT * FROM dog
ORDER BY id desc
LIMIT 3 OFFSET 2;

 id |  name  |   kind    | owner_cd
----+--------+-----------+----------
  5 | Robbin | Schnauzer |        4
  4 | Luke   | Duchshund |        3
  3 | Pearl  | mix       |        2
(3 rows)
```

© LPI-Japan 2015. All rights reserved.

WHERE句の条件に別の問合せ結果を用いる

dog表

id	name	kind	owner_cd
001	Poppy	Westy	001
002	Mitten	mix	001
003	Pearl	mix	002
004	Luke	Dachshund	003
005	Robbin	Schnauzer	004
006	Andy	Schnauzer	004
007	Ace	Jack Russell	005

```
SELECT * FROM dog
WHERE owner_cd = (
  SELECT owner_cd FROM owner
  WHERE  o_name = 'k.kida');
```

```
 id | name  | kind | owner_cd
----+-------+------+----------
  3 | Pearl | mix  |        2
(1 row)
```

owner表

owner_cd	o_name	o_address
001	kida	千葉県○○市
002	k.kida	東京都△△区
003	y.kida	神奈川県××市
004	morioka	千葉県○○市
005	sakamoto	東京都△△区

© LPI-Japan 2015. All rights reserved.

■サブクエリの結果が1行とは限らない

- LIKEによる曖昧検索

```
SELECT * FROM owner
WHERE  o_name LIKE '%kida%';
 owner_cd | o_name |  o_address
----------+--------+-------------
        1 | kida   | 千葉県○○市
        2 | k.kida | 東京都△△区
        3 | y.kida | 神奈川県××市
```

- WHERE句の演算子を「＝」でなく、「IN」に変更

```
SELECT * FROM dog
WHERE owner_cd =  (SELECT owner_cd FROM owner WHERE o_name LIKE '%kida%');
ERROR:  more than one row returned by a subquery used as an expression

SELECT * FROM dog
WHERE owner_cd IN (SELECT owner_cd FROM owner WHERE o_name LIKE '%kida%');

 id |  name  |   kind    | owner_cd
----+--------+-----------+----------
  1 | Poppy  | Westy     |        1
  2 | Mitten | mix       |        1
  3 | Pearl  | mix       |        2
  4 | Luke   | Duchshund |        3
(4 rows)
```

© LPI-Japan 2015. All rights reserved.

FROM句に副問合せ結果を用いる

- 結合、ソート、集計済みの結果に対する条件指定をしたい場合
- PostgreSQLでは、インライン・ビューの別名が必須

dog表とonwer表を結合

id	name	kind	owner_cd	o_name	o_addre
001	Poppy	Westy	001	kida	千葉県○
002	Mitten	mix	001	kida	千葉県○
003	Pearl	mix	002	k.kida	東京都△
004	Luke	Dachshund	003	y.kida	神奈川県
005	Robbin	Schnauzer	004	morioka	千葉県○
006	Andy	Schnauzer	004	morioka	千葉県○
007	Ace	Jack Russell	005	sakamoto	東京都△△区

```
SELECT * FROM (
  SELECT * FROM dog NATURAL JOIN owner)
  AS dog_with_owner
WHERE o_name = 'k.kida';


 id | name  | kind | owner_cd
----+-------+------+----------
  3 | Pearl | mix  |        2
(1 row)
```

dog表

id	name	kind	owner_cd
001	Poppy	Westy	001
002	Mitten	mix	001
003	Pearl	mix	002
004	Luke	Dachshund	003
005	Robbin	Schnauzer	004
006	Andy	Schnauzer	004
007	Ace	Jack Russell	005

owner表

owner_cd	o_name	o_address
001	kida	千葉県○○市
002	k.kida	東京都△△区
003	y.kida	神奈川県××市
004	morioka	千葉県○○市
005	sakamoto	東京都△△区

© LPI-Japan 2015. All rights reserved.

■雑種（kind列が「mix」）を飼っているownerを調べる

- 参照したい結果を想像する

犬の名前	犬種	オーナー名

という形で取り出せれば良い

- 犬種とオーナー名は別のテーブルにあるので、結合

  結合のキー列として、共通の意味を持つ項目を考える

- WHERE句に（kind列が「mix」）条件を書く

```
SELECT
FROM
ON
WHERE
```

■dog表

id	name	kind	owner_cd
001	Poppy	Westy	001
002	Mitten	mix	001
003	Pearl	mix	002
004	Luke	Dachshund	003
005	Robbin	Schnauzer	004
006	Andy	Schnauzer	004
007	Ace	Jack Russell	005

■owner表

owner_cd	o_name	o_address
001	kida	千葉県○○市
002	k.kida	東京都△△区
003	y.kida	神奈川県××市
004	morioka	千葉県○○市
005	sakamoto	東京都△△区

© LPI-Japan 2015. All rights reserved.

■雑種（kind列が「mix」）を飼っているownerを調べる

- 参照したい結果を想像する

犬の名前	犬種	オーナー名

という形で取り出せれば良い

→ SELECT 犬の名前,犬種,オーナー名 FROM ・・・

- 犬種とオーナー名は別のテーブルにあるので、結合

→ FROM dog JOIN owner

結合のキー列として、共通の意味を持つ項目を考える

→ ON dog.owner_cd = owner.owner_cd

- WHERE句に（kind列が「mix」）条件を書く

→ WHERE kind= 'mix'

```
SELECT name,kind,o_name
FROM dog JOIN owner
ON   dog.owner_cd = owner.owner_cd
WHERE kind='mix';

 name   | kind | o_name
--------+------+--------
 Mitten | mix  | kida
 Pearl  | mix  | k.kida
(2 rows)
```

■dog表

id	name	kind	owner_cd
001	Poppy	Westy	001
002	Mitten	mix	001
003	Pearl	mix	002
004	Luke	Dachshund	003
005	Robbin	Schnauzer	004
006	Andy	Schnauzer	004
007	Ace	Jack Russell	005

■owner表

owner_cd	o_name	o_address
001	kida	千葉県○○市
002	k.kida	東京都△△区
003	y.kida	神奈川県××市
004	morioka	千葉県○○市
005	sakamoto	東京都△△区

© LPI-Japan 2015. All rights reserved.

■表名や列名を指定し、データへの操作を行う

- 列名（選択リスト）、表名（FROM句）、条件（WHERE）句を指定
  - UPDATE・DELETEは、WEHRE条件が無い場合は列に対する操作
  - INSERTは、列名の指定が無い場合は列の順に挿入する値のリストを記述

■dog表に対する操作

```
-- データのINSERT
INSERT INTO dog
VALUES (008,'Candy','mix','kida','千葉県○○市');

-- データのUPDATE
UPDATE dog SET onwer='a.kida' WHERE id=003;

-- データのDELETE
DELETE FROM dog WHERE id=004;
```

■dog表

id	name	kind	owner	address
001	Poppy	Westy	kida	千葉県○○市
002	Mitten	mix	kida	千葉県○○市
003	Pearl	mix	k.kida	東京都△△区
004	Luke	Dachshund	y.kida	神奈川県××市
005	Robbin	Schnauzer	morioka	千葉県○○市
006	Andy	Schnauzer	morioka	千葉県○○市
007	Ace	Jack Russell	sakamoto	東京都△△区
008	Candy	mix	kida	千葉県○○市
003	Pearl	mix	a.kida	東京都△△区

© LPI-Japan 2015. All rights reserved.

ロック

- 同じ行に対する更新を防ぐ仕組み
- データ操作の対象行はロックされ、別トランザクションによる操作は待機する

デッドロック

- 2つのトランザクションがロックを取り合う状態
  - 片方がエラーになりトランザクション失敗
  - 他方はロック待ちが終わり成功
- デッドロックが発生しないよう、アプリケーション側で考慮

dog表

id	name	kind	owner
001	Poppy	Westy	kida
002	Mitten	mix	kida
003	Pearl	mix	k.kida
004	Luke	Dachshund	y.kida
005	Robbin	Schnauzer	morioka
006	Andy	Schnauzer	morioka
007	Ace	Jack Russell	sakamoto

```
-- トランザクションA
BEGIN;
-- データのUPDATE
UPDATE dog SET onwer='a.kida'
WHERE id=003;

-- データのUPDATE
DELETE FROM dog WHERE id=004;
--トランザクションBの確定を待機
```

```
-- トランザクションB
BEGIN;
-- データのUPDATE
UPDATE dog SET onwer='k.kida'
WHERE id=004;

-- データのUPDATE
DELETE FROM dog WHERE id=003;
--トランザクションAの確定を待機
--デッドロックを検知しエラー
```

© LPI-Japan 2015. All rights reserved.

■表名、列名とデータ型を定義する

- CREATE TABLE文
- テーブル定義からCREATE TABLE文を作成

  または

  テーブル定義からデータのサンプルを想像

■dog表のCREATE TABLE文

```
CREATE TABLE dog
( id       integer
 ,name     text
 ,kind     text
 ,owner_cd integer
);
```

■dog表のテーブル定義

```
dog
--------------------------------
id       integer
name     text
kind     text
owner_cd integer
```

■dog表の定義からデータを想像する

id	name	kind	owner_cd
001	Poppy	Westy	001
002	Mitten	mix	001
003	Luke	Dachshund	002
999	xxxxxx	xxx	100

データまで想像するとわかること

・犬一頭につき1行のデータ
・飼い主は重複する可能性がある
　ただしdog表とowner表の件数は同等規模
　(常識的に、例えば1000対1となるような超多頭飼いは無い)
・NULL値の可能性を予測
　データの意味を考え、idやnameがNULLの可能性は低い

© LPI-Japan 2015. All rights reserved.

■列に格納されるデータに対する制約条件を定義する

- CREATE TABLE 時に指定
  ALTER TABLE ALTER COLUMN などで指定

- PRIMARY KEY制約
- UNIQUE KEY制約
- NOT NULL制約
- CHECK制約
- FOREIGN KEY制約 (参照整合性制約)

■dog表のテーブル定義(イメージ)

```
dog
----------------------------------------
id        integer  PRIMARY KEY
name      text     NOT NULL
kind      text
owner_cd  integer  FOREIGN KEY(owner)
```

■dog表のテーブル定義(確認例)

```
postgres=# ¥d dog
      テーブル "public.dog"
    列    |   型      |  修飾語
----------+---------+----------
 id       | integer | not null
 name     | text    | not null
 kind     | text    |
 owner_cd | integer |
インデックス:
    "dog_pkey" PRIMARY KEY, btree (id)
外部キー制約:
    "dog_owner_cd_fkey" FOREIGN KEY
(owner_cd) REFERENCES owner(owner_cd)
```

© LPI-Japan 2015. All rights reserved.

■dog表とowner表の作成、データの投入

```
DROP TABLE dog;
DROP TABLE owner CASCADE;
CREATE TABLE owner (owner_cd  integer primary key
                   ,o_name    text
                   ,o_address text);
¥d owner
CREATE TABLE dog ( id       integer primary key
                  ,name     text    not null
                  ,kind     text
                  ,owner_cd integer references owner(owner_cd) );
¥d dog

insert into owner values (001,'kida','千葉県○○市');
insert into owner values (002,'k.kida','東京都△△区');
insert into owner values (003,'y.kida','神奈川県××市');
insert into owner values (004,'morioka','千葉県○○市');
insert into owner values (005,'sakamoto','東京都△△区');

insert into dog values (001,'Poppy','Westy',001);
insert into dog values (002,'Mitten','mix',001);
insert into dog values (003,'Pearl','mix',002);
insert into dog values (004,'Luke','Duchshund',003);
insert into dog values (005,'Robbin','Schnauzer',004);
insert into dog values (006,'Andy','Schnauzer',004);
insert into dog values (007,'Ace','Jack Russell',005);

SELECT * FROM dog d NATURAL JOIN owner o;
```

© LPI-Japan 2015. All rights reserved.

■データベースに求められること

データベースに求められる「同時実行性」「高性能」「耐障害性」などの基本を整理し、これらをRDBMSで実現するための重要なキーワードを解説

■RDBMSの構造

前章で挙げたデータベースとしての基本が、PostgreSQLではどのような仕組みで実装されているかを解説

■SQL開発

RDBMSの共通言語である「SQL」の基本を解説

■DBA（データベース管理者）のタスク

RDBMSの構造から定期的なメンテナンスの必要性を解説し、管理者が実施する具体的なタスクやその実施方法を解説

© LPI-Japan 2015. All rights reserved.

DB管理者（データベースアドミニストレータ、DBA）の担当業務

分類	タスク	備考
サーバ構築 初期設定	サーバサイジング OS設定、インストール パラメータ設定 セキュリティ設定　など	構築時の初期設定は代表的なパラメータの変更など、ある程度決まった設定で対応可能(**Silver**) 上級では、システム要件から必要なサーバスペックを見積もり、OS設定等を含めた対応が求められる(**Gold**)
監視	死活監視 領域監視 エラー監視 パフォーマンス監視　など	サーバログ出力設定を行い、基礎的なメッセージを理解し対処を行う。また、正常稼働中のステータス確認やプロセスの状態を知っている。(**Silver**) 各種監視を行い障害を未然に防止する(**Gold**)
メンテナンス	オブジェクトのメンテナンス ユーザのメンテナンス 起動・停止	オブジェクト作成や基本のメンテナンス (**Silver**) 監視情報からメンテナンスの必要性を判断・対処し、障害を未然に防止する(**Gold**)
チューニング	ボトルネックの把握 データベースチューニング SQLチューニング	初期設定時に基本的なチューニングを実施(**Silver**) 監視情報からボトルネックを判断し、適切なチューニングを行う(**Gold**)
障害復旧	バックアップの取得 HA、BCP対策 リストア・リカバリ	標準的なバックアップの手法を理解し、対応可能な障害の種類を整理する(**Silver**) レプリケーション・HA、BCPや環境固有の対策(クラウド機能によるHAなど)を含めた計画を立て、可用性を高く保つ(**Gold**)

© LPI-Japan 2015. All rights reserved.

標準付属ツール

- メンテナンスコマンド
  - <インストール先>/bin配下
  - オプションの意味まで理解しておく
  - --helpで使い方が確認可能
- その他

運用管理の実施

- 初期設定
  - 初期化パラメータ
  - ユーザ作成・管理
- 監視
  - プロセス監視
  - サーバーログ監視
- オブジェクトのメンテナンス
  - テーブルのメンテナンス

代表的なメンテナンスコマンド

分類	タスク
initdb	データベースクラスタの初期化
pg_ctl pg_isready	データベースの起動・停止 データベース稼働状態の確認 パラメータのリロード
psql	データベースに接続、SQL発行
createdb dropdb	データベースの作成・削除
vacuumdb	データベースまたはテーブルを指定したVACUUMの実施
pg_dump pg_restore pg_dumpall	論理バックアップの取得 論理バックアップを使用したリストア

© LPI-Japan 2015. All rights reserved.

■パラメータ設定

- 初期化パラメータpostgresql.confを変更して、サーバ再起動
- 同じくpostgresql.confを変更して、pg_ctl reload
- セッション単位で動的に変更可能　　などパラメータによって方法が異なる

```
/* pg_settingsビューから現在の設定を参照 */
postgres=# ¥x
postgres=# SELECT name,setting,unit,context FROM pg_settings;

/* pg_settingsビューからパラメータの分類を確認 */
SELECT distinct context FROM pg_settings;
Internal    ・・・変更不可(構築時設定確認用)
postmaster  ・・・サーバ起動時
Sighup      ・・・設定ファイルの再読み込み
Backend     ・・・セッション確立時に決定
Superuser   ・・・スーパユーザ権限で動的変更可能
User        ・・・一般ユーザで動的変更可能
```

■代表的なパラメータ

接続関連	port 、listen_addresses、max_connections
メモリ関連	shared_buffers、work_mem、maintenance_work_mem
チェックポイント関連	checkpoint_segments、checkpoint_timeout
ログ出力関連	logging_collector、log_line_prefix

© LPI-Japan 2015. All rights reserved.

■データベースユーザの作成

- 初期ユーザ（一般にpostgresユーザと表記される）はスーパーユーザ
- ログイン属性を持つユーザを作成
  - （標準ツール）createuserコマンド
  - （SQL）CREATE ROLE文

■アクセス制御

- pg_hba.confファイルに記載し、pg_ctl reload
- どのデータベース/どのユーザへの接続を、どの接続元から許可（拒否）

```
/* pg_hba.confにアクセス制御リストを記述  */
cd $PGDATA
vi pg_hba.conf
-------------------------------------------------------------------------
# TYPE  DATABASE        USER            ADDRESS                 METHOD

local   all             all                                     trust
host    silver          kkida         192.168.10.xx/32          md5
host    gold            all           192.168.10.xx/24          reject
-------------------------------------------------------------------------
/* 設定を読み込み */
pg_ctl reload
```

© LPI-Japan 2015. All rights reserved.

サーバログ監視

サーバログ出力設定

- 初期化パラメータpostgresql.conf の logging_collector = on
- log_line_prefixに時刻やSQL Stateを記録するよう指定 (推奨)

サーバログの何を監視するか

- エラーラベルの監視　log_min_messagesのエラーラベル
  - INFO、NOTICE、WARNING、ERROR、LOG、FATAL、PANIC
  - 重要度の高いものは以下

エラーレベル	内容
PANIC	サーバが停止している
FATAL	セッションが切断されている(他のセッションは正常)
ERROR	該当の処理が失敗し、セッションは残っている

© LPI-Japan 2015. All rights reserved.

死活監視

サーバの死活監視はプロセス監視またはクライアント接続で確認する

- OSコマンド(ps -ef など)で監視
  - postgresプロセスのPIDを確認
  - $PGDATA/postmaster.pidファイルに記録されたPIDと一致
  - 他のプロセスは、postgresプロセスが自動的に再起動する
- SQLによる死活監視
  - 数分間隔で SELECT 1; などの単純なSQLを実行
- 専用コマンドによる死活監視
  - pg_isreadyコマンド(9.3～)
  - 管理コマンドとしてインストールされ、死活監視に利用

正常時	接続不可の場合
$ pg_isready /tmp:5432 – accepting connections $ echo $? 0	$ pg_isready -h localhost -p 5433 localhost:5433 - rejecting connections $ echo $? 1 : 起動中などで接続を拒否 2 : 無応答 3 : pg_isreadyの実行に失敗

© LPI-Japan 2015. All rights reserved.

■テーブルの肥大化を抑制する

- VACUUMの必要性
  - PostgreSQLは追記型であり、1行が何度も更新されるとテーブルは肥大化
  - 不要な行を記録しておき、次の挿入や更新時に再利用する
- VACUUMの動作イメージ
  - Visibrity Mapから不要行を検索
  - 使用可能領域としてFree Space Mapに記録

■dog表の不要領域を追跡

id	name	kind	owner	address	
001	Poppy	Westy	kida	千葉県○○市	
002	Mitten	mix	kida	千葉県○○市	
003	Pearl	mix	k.kida	東京都△△区	←更新済みの行
004	Luke	Dachshund	y.kida	神奈川県××市	←削除済みの行
005	Robbin	Schnauzer	morioka	千葉県○○市	
006	Andy	Schnauzer	morioka	千葉県○○市	
007	Ace	Jack Russell	sakamoto	東京都△△区	
008	Candy	mix	kida	千葉県○○市	
003	Pearl	mix	a.kida	東京都△△区	

VM

Visiblity Map
・不要行を追跡可能

FSM

Free Space Map
・不要行の位置を記録
・更新時、FSMから空き領域を再利用

© LPI-Japan 2015. All rights reserved.

VACUUMの種類

- VACUUMの種類と内容

種類	内容
(通常の)VACUUM	不要行をFree Space Mapに登録し、再利用可能にする
VACUUM FULL	表の再作成を行い、不要行を詰めて物理ファイルの縮小を行う ※一時的に表サイズの2倍の領域を使用するため、ディスク不足時の領域確保には使えない
VACUUM FREEZE	トランザクションID周回問題への対処

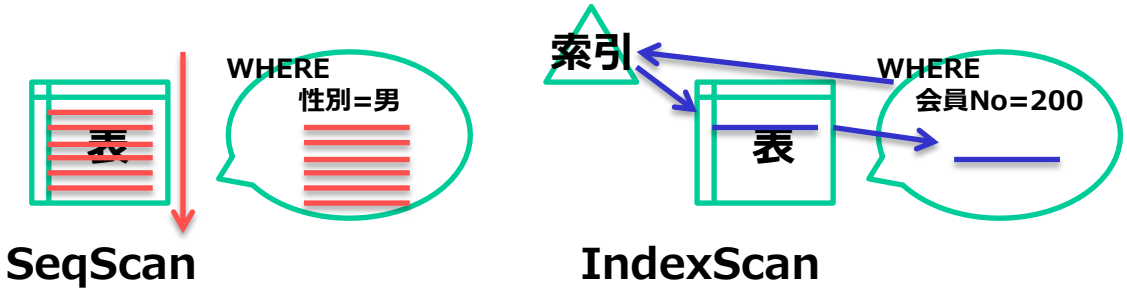
※VACUUMが適切に実行されることで、表は一定サイズ以上には肥大しない

- 自動VACUUM
  - デフォルトでは自動VACUUMが有効
  - テーブルに対する更新量を検査し、一定量の更新があるとVACUUMを行う
  - 同じ検査の仕組みで自動ANALYZEも実行されている

© LPI-Japan 2015. All rights reserved.

ANALYZEによる列統計の収集

- ANALYZEの必要性
  - 必要なデータを高速に検索する仕組みとして「実行計画」がある
  - 例）テーブル全体をディスクから読みこむ　SeqScan
  - 索引を使って必要な行だけ読み込む　IndexScan　　・・・どっちが高速？
  - PostgreSQLが実行計画の候補を複数作成し、最適なものを実行する



  - ANALYZEで、対象列にどのようなデータがどんな分布で格納されているか見積もっておくと、最適（な可能性が高い）実行計画を作る事ができる。
- ANALYZEの実行
  - VACUUM時に併せて実行
  - ANALYZEコマンドで実行

© LPI-Japan 2015. All rights reserved.

テーブルの再編成

- VACUUM FULLまたはCLUSTERコマンド
  - 大量更新や、長期間の運転などで、通常のVACUUMではファイルの肥大化が避けられないケースがある
    - SeqScanで読み取るデータ量の肥大化
    - バックアップ取得の長時間化
  - テーブルの不要領域を取り除き、物理ファイルサイズを縮小
  - CLUSTERコマンドでは、同時に索引を指定することで索引の並び順にソート

- テーブル再編成の影響
  - 以下の影響があるため、24時間稼働するシステムでは難易度が高い
    - テーブル全体のロック(Access Exclusiveモード)を確保し、参照をブロック
    - 論理バックアップ→リストア　に近い動作であり、一時的にディスク容量を使用

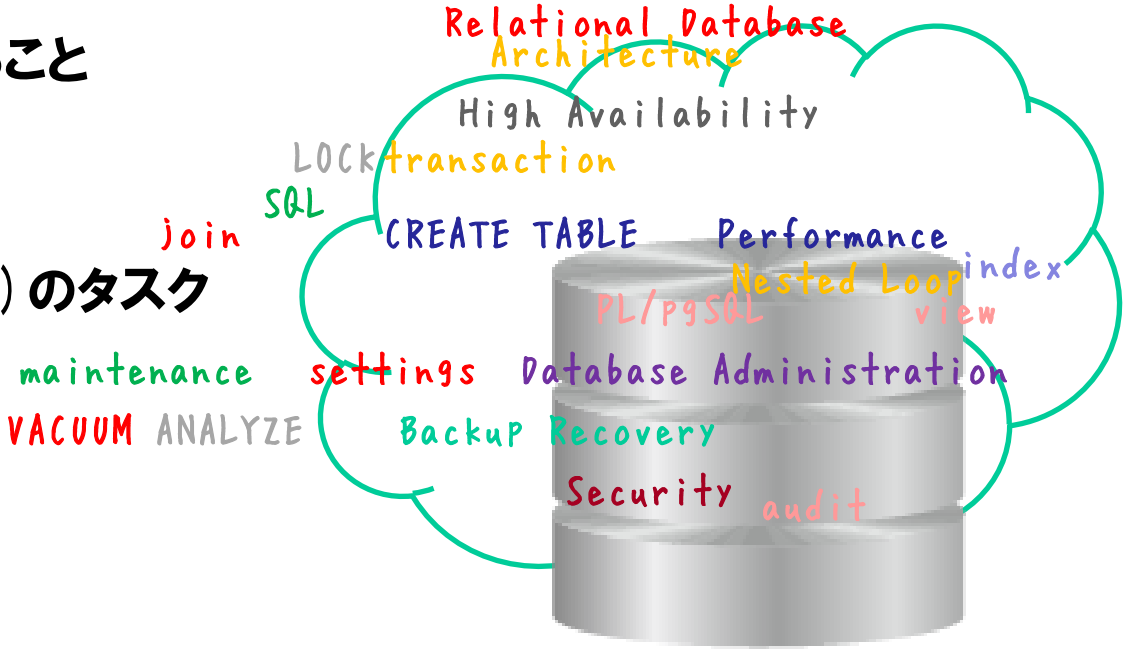
© LPI-Japan 2015. All rights reserved.

■データベースの基本を解説

- データベース技術者としての入門レベルであり、PostgreSQLを扱う上で必須知識であるOSS-DB Silver試験に向けた学習のきっかけに
- データベース初級者が、PostgreSQLを使用したデータベース学習を進められることを目標とする

■取り扱う内容

- データベースに求められること
- RDBMSの構造
- SQL開発
- DBA（データベース管理者）のタスク



© LPI-Japan 2015. All rights reserved.

参考文献

■講演資料

- OSS-DB Exam Silver技術解説無料セミナー　2015/05/16
  株式会社メトロシステムズ　佐藤千佳　氏

■Webサイト

- PostgreSQLマニュアル
  https://www.postgresql.jp/document/9.4/html/index.html

■書籍

- 徹底攻略　OSS-DB Silver 問題集 [OSDBS-01] 対応
  インプレスジャパン刊　ISBN978-4844331933
- SQL逆引き大全363の極意
  秀和システム刊　ISBN978-4798038520
- これならわかる Oracle 超入門教室 第2版 (DB Magazine SELECTION)
  翔泳社刊　ISBN978-4798114262

© LPI-Japan 2015. All rights reserved.

終わりに

OSS-DBの普及

- 現代の契約社会を支えるデータベース技術では、これまで商用製品が圧倒的なシェアを有していたが、近年の製品品質の向上や、国内での情報整備、サービス提供企業の存在から、急速にOSS化が進んでいる
- 商用/OSSを問わず様々なRDBMSの知識を持ち、データベースの構築、運用ができる、または顧客に最適なデータベースを提案できる技術者が求められている

OSS-DB資格の重要性

- 製品選択の観点から、体系的な知識を持った技術者の存在は重要であり、ベンダ資格がないPostgreSQLにとっては普及の起爆剤となる
- 前述の通り、製品自体が普及してきていることから、資格取得による個人のキャリアアップと、さらなる製品の普及促進の両面から非常に重要

© LPI-Japan 2015. All rights reserved.

ご清聴ありがとうございました。

■お問い合わせ■
株式会社アシスト
喜田 紘介
Mail：kkida@ashisuto.co.jp

© LPI-Japan 2015. All rights reserved.