アシスト
「お客様の最高」のために

LPI-JAPAN
Linux Professional Institute Japan

OSS-DB Exam Gold 技術解説無料セミナー

2017/03/19

株式会社 アシスト データベース技術本部
NPO法人 日本PostgreSQLユーザ会
喜田 紘介

講師プロフィール

アシスト 「お客様の最高」のために　OSS-DB

■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
  NPO法人 日本PostgreSQLユーザ会

EDB Postgres のプリセールス、
技術支援、新機能検証などを担当

- 最近のこと

  2012年に取得したOSS-DB Goldの認定期限が今年切れます。
  一緒にOSS-DB合格を目指して勉強しましょう！
  また、この5年間、多くの方との関わりの中でここまでやってきたんだと実感します。今年は改めてPostgreSQL、DB技術の普及など頑張りたいと思っています！

初心者向け情報発信　地方需要の活性化　エコシステム拡大

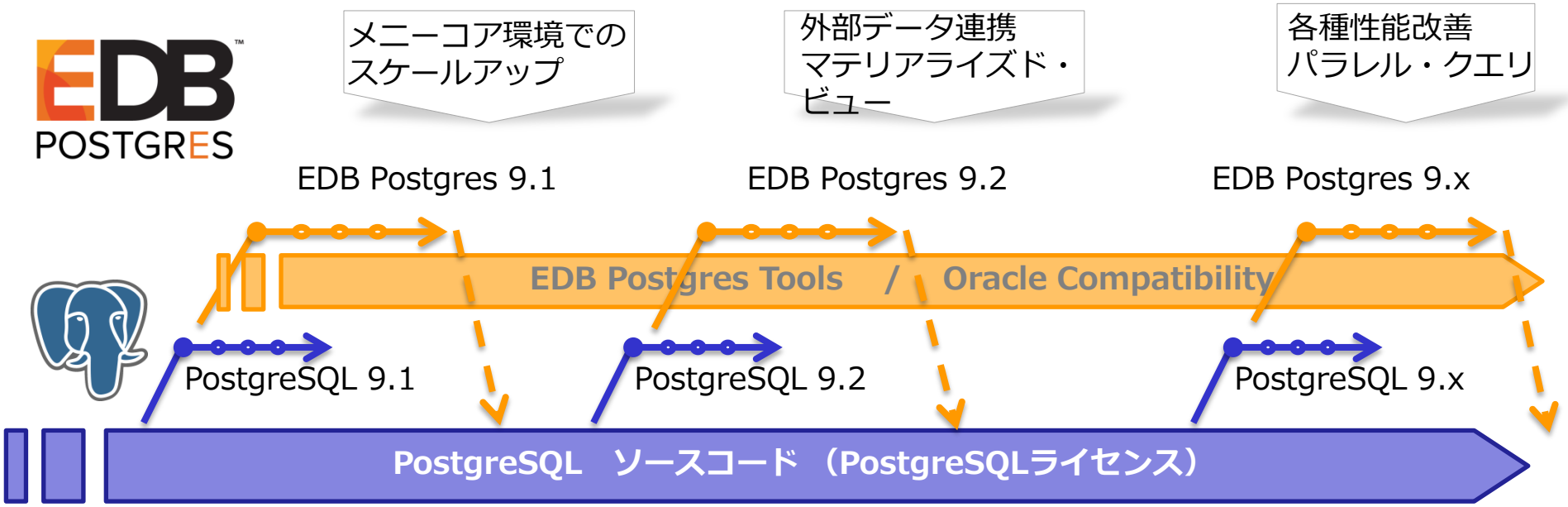
ご参加、応援などよろしくお願いします。

■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
  NPO法人 日本PostgreSQLユーザ会

EDB Postgres のプリセールス、
技術支援、新機能検証などを担当

■EDB Postgresについて



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、データベースの設計、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 /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
  - データの物理的な格納方式
  - 適切なメンテナンスの実施
  - レプリケーションなど大規模環境で有用な機能の詳細を理解
- メンテナンスや障害対応の必要性の判断と適切な実施
  - システムカタログからデータベースの状態を把握
  - 適切なメンテナンスの実施
- 広い視野でチューニングができる
  - OS側の対処を含めた対策の立案
  - システムの特性から事前に影響を予測

Gold試験範囲

■運用管理
- データベース構造や管理コマンド全般について詳細まで理解し、健全なデータベースを構築できる

  ※レプリケーションで構築した大規模環境を含む

■性能監視
- アクセス統計情報、プランナ統計情報、各種ビュー、実行計画について理解し、その他の性能監視手法も活用できる

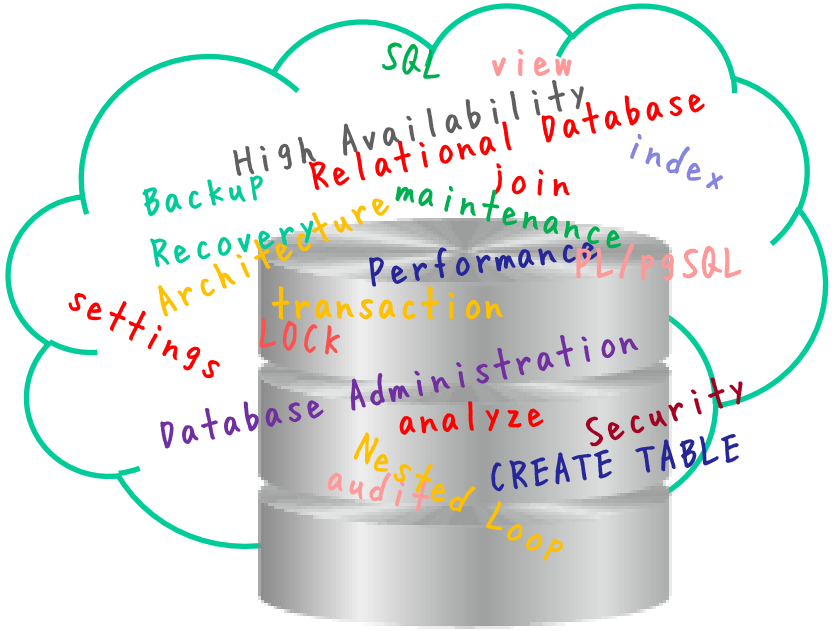
■チューニング
- 性能関連パラメータの理解や、性能監視の結果を正しく判断し、チューニングを行う

■障害対応
- 障害パターンとそれぞれに対する状態確認や復旧方法を理解

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

本講義の内容

- 広範囲におよぶ出題範囲から日々のDBA業務に当てはめて理解
  - 大規模データベースを適切に運用管理するプロフェッショナルであるデータベース管理者（DBA）の業務にあてはめ出題範囲を整理
  - 具体的なコマンドではなく、後からマニュアルを読みこなせるような詳細な実装や、動作のイメージを持つ



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**)

データベース管理者(DBA)の業務

■DBA＝データベース運用管理のプロフェッショナル

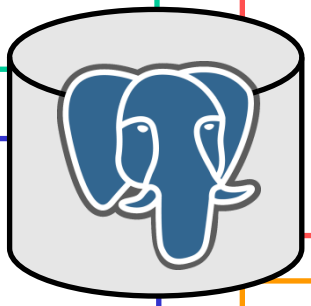
アプリ開発者（Developer）

・データベースの利用者
・プログラム（SQL）を書く
・パフォーマンスの改善

データベース管理者（DBA）

Database Administrator

・データベースの運用管理を担当
・DBの安定稼働を使命とする
・プログラムを書かない
・運用管理に必要なSQLは書く（後述）
・OSレベルの情報も見る
・必要に応じてメンテナンス操作を実施
・オペレーター向け手順書の整備



セキュリティ管理者

・データベースには
アクセスできない
・DBAや開発者による不正な
操作がないことを確認する
・監査情報に対する権限

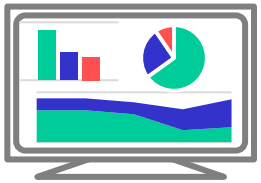
オペレーター

・手順書に従って各種対応を行う
・データベースへの限られた操作

データベース管理者(DBA)の業務

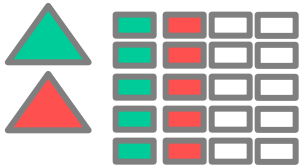
■DBA＝データベース運用管理のプロフェッショナル

稼動状態の監視



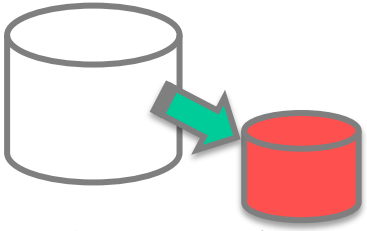
- ✓ 死活監視
- ✓ ログ監視
- ✓ OSレベルの監視
- ✓ パフォーマンス監視

メンテナンス操作



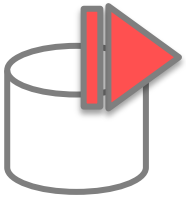
- ✓ 表や索引のメンテナンス
- ✓ 格納領域の管理
- ✓ システム統計情報の管理
- ✓ オプティマイザ統計の管理

バックアップ管理

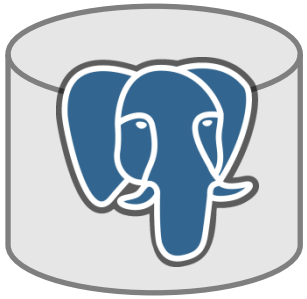


- ✓ バックアップの取得
- ✓ バックアップ世代管理
- ✓ リストア・リカバリ計画
- ✓ リカバリ手順の作成

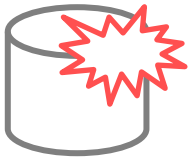
オペレーション



- ✓ 起動・停止
- ✓ パラメータ検討・設定
- ✓ DBサーバーの構築



障害復旧



- ✓ 各種障害への対応

広範囲な試験範囲をDB管理者の業務に当てはめて理解

- サーバ構築
  - OSの設定～PostgreSQLのインストール
- 初期設定
  - サーバサイジング
  - パラメータ設定
  - セキュリティ

Silver試験で学習したレベルを
おさらいしつつ、
Goldで問われる内容を学習

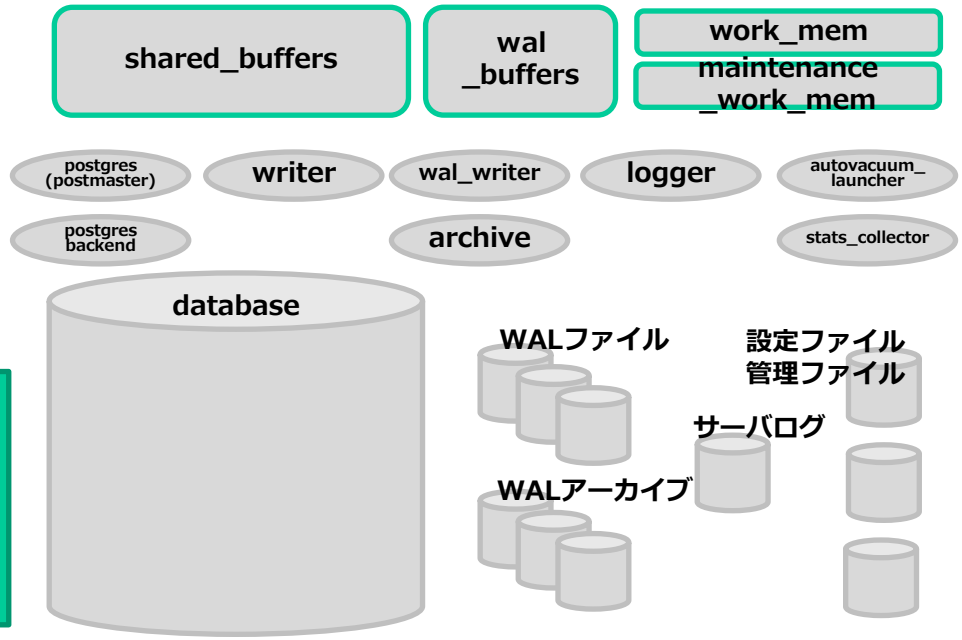
- 監視
- メンテナンス
- チューニング
- 障害復旧
- レプリケーション

申し訳ありません。
本日は時間の関係で
さっと紹介する程度です。

領域ごとに用途を確認

- 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
  - 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
- work_mem
- maintenance_work_mem
  - セッション毎に確保される領域
  - ソートやハッシュの一時領域
  - メンテナンス操作

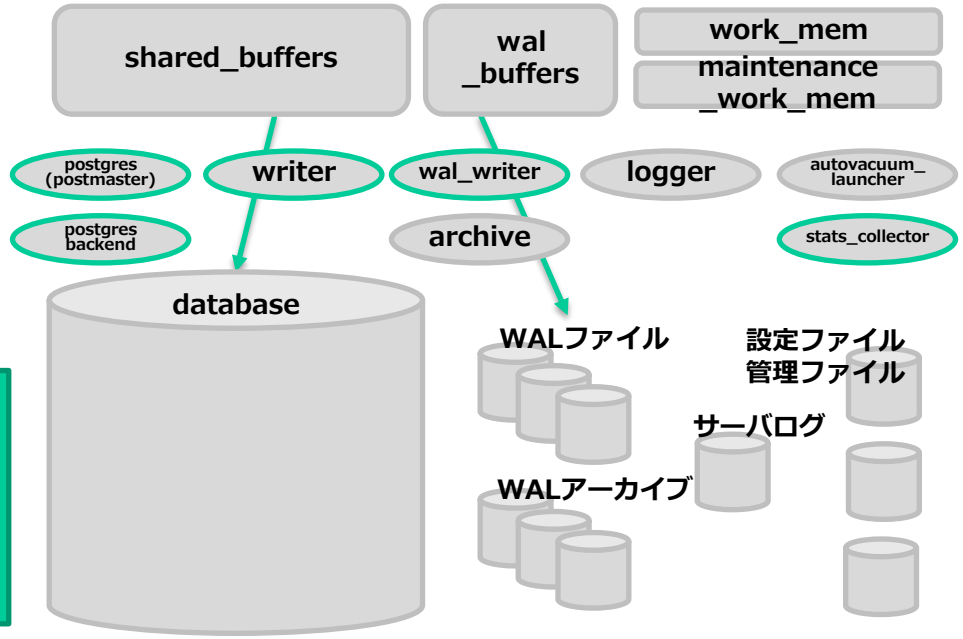
メモリ関連では、設定パラメータと役割、不足したときにどういう情報からそれを検知するか、チューニング方法を整理してを整理しておく。



必須プロセス

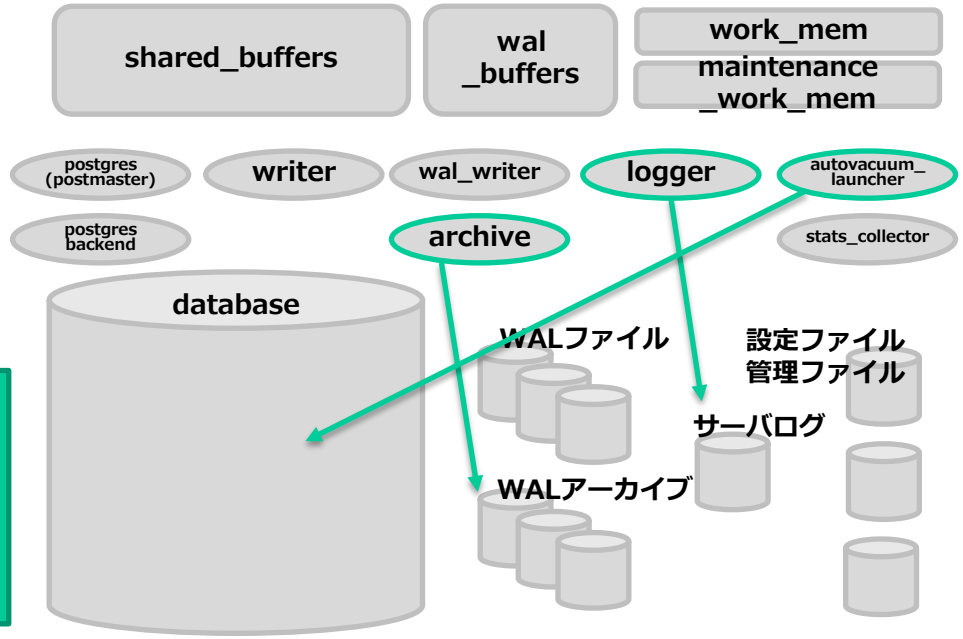
- postgres(postmaster)、postgres backend
  - クライアントからの接続を待ち受ける、すべてのプロセスの親プロセス
  - バックエンドがクライアントからの処理を担当
- writer
  - 共有バッファのデータをディスクに書き込むプロセス
  - チェックポイントやダーティバッファの書き込み
- wal writer
  - データの変更履歴を WALファイルに書き込む
- stats collector
  - 実行時統計情報を収集する

プロセス関連では、各プロセスが担う役割を詳細に理解し、動作を調整できることが期待される



パラメータ設定により起動するプロセス

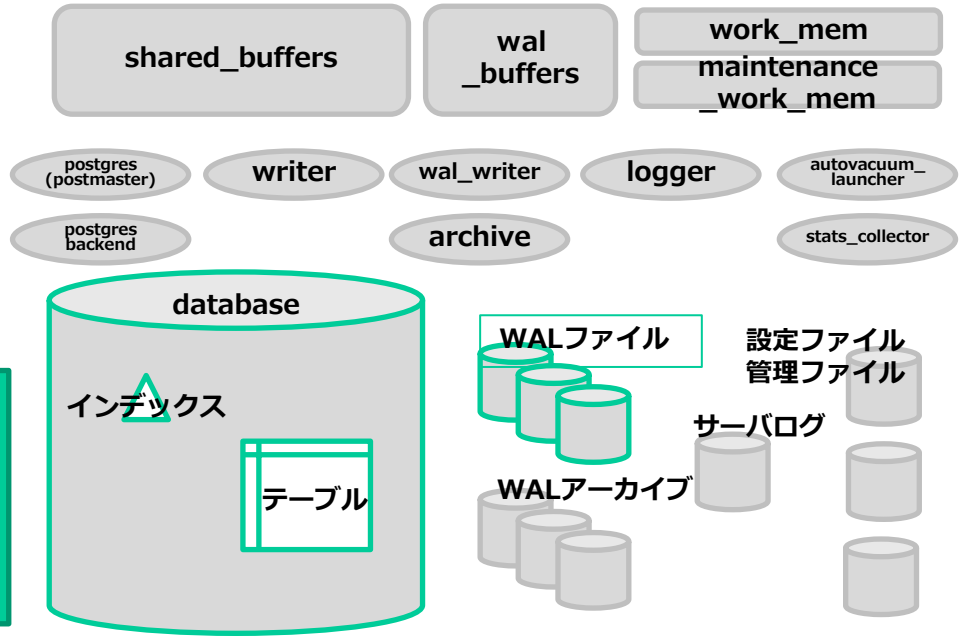
- logger
  - PostgreSQLサーバ実行時のログを記録するプロセス
- archive
  - チェックポイント以前の不要なWALをPITRのために別のディスクに退避
- autovacuum launcher/worker
  - 自動VACUUMの閾値を超過したもの(表・列)に対してVACUUMを実行



これらのプロセスはGoldの範囲では有効が前提であり、非常によく問われる詳細設定まで理解している必要がある

データベースクラスタの物理構造

- データディレクトリ
  - PostgreSQLサーバのルートディレクトリ
- 配下のディレクトリ・ファイル
  - base　　：データベース毎にディレクトリが分かれ、データを格納
  - global　：システムカタログなどの管理用オブジェクトのデータ
  - pg_xlog：WALセグメントファイル
  - その他
    - 各種設定ファイル
    - 障害対応で必要なもの



構築、運用管理、性能、障害対応全てに関わるため、詳細に理解しておく
各ファイルに対し考慮点が多数あるため
まずはここから解説

ディスク領域設計に対する考慮点

アシスト 「お客様の最高」のために

OSS-DB

- 安定性や性能を考慮した構築ポイント
  - ①各ファイルの配置先はどう決めるか
  - ②テーブルデータを格納する物理ファイル
  - ③インデックスデータを格納する物理ファイル
  - ④WALファイルに関する構築ポイント

①物理ファイル配置先

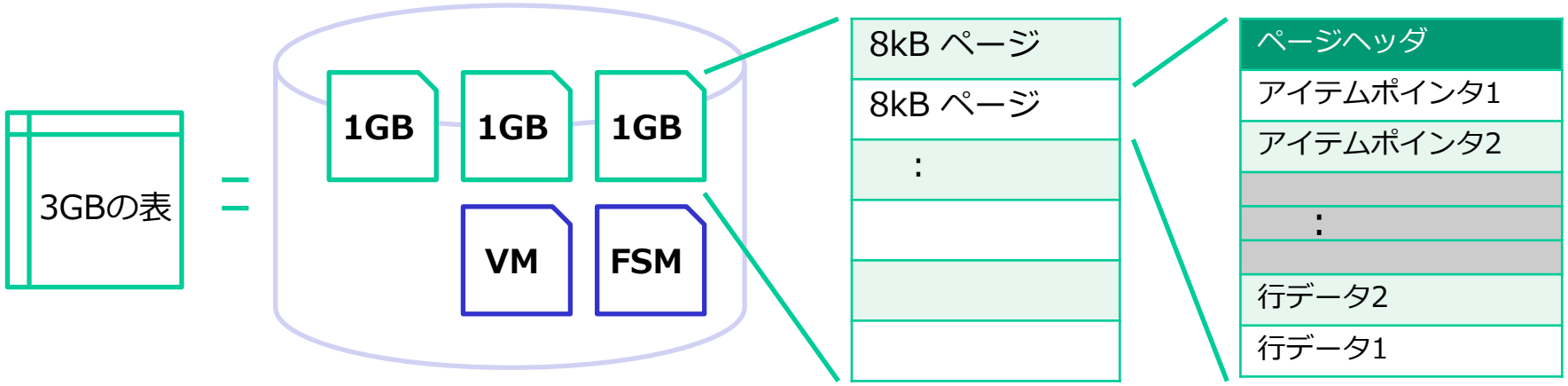
- I/O分散や、故障リスク分散の方法を理解しておく
  - DBを作る際に、ユーザが位置を指定できるもの

内容	指定方法	デフォルト位置
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	--

  - 考慮時のポイント
    - データとWALは、同時に破損すると最新状態への復旧が困難になるため、WALファイル出力先を指定し、別のディスクとする
    - WALファイルは二重化できないためRAIDによる保護を検討
    - テーブル、インデックスはTABLESAPCE機能を使用して配置先を分散しI/Oの効率化を図る

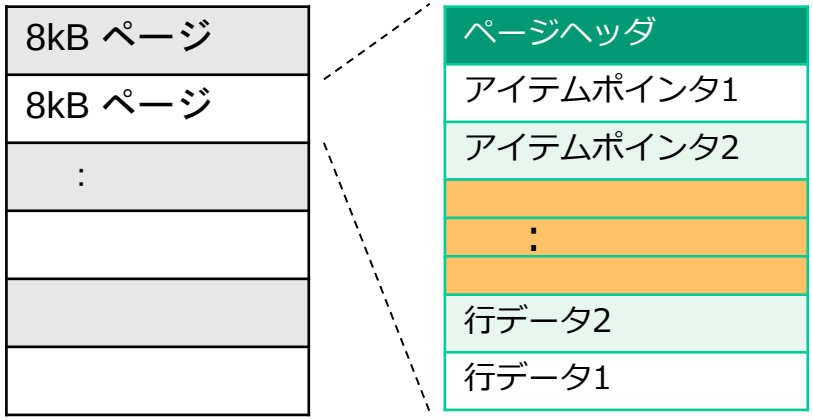
■物理ファイルの構造は詳細に問われる

- ファイル(ディレクトリ)構造や、ページ構造は必ず理解しておくこと
  - –1つのオブジェクト　＝ 1GB毎のセグメントファイル ＋ VM、FSM
  - –セグメントファイルは8KB単位のページで構成される



テーブルサイズの見積もり

- テーブルのページヘッダは固定で**24 byte**
- 1行追加されるごとに アイテムポインタ（行ヘッダ）**28 byte**
- 行データが消費するサイズ




```
postgres=# ¥d dog
--------  ---------
dogno      int
regdate    timestamp
dogname    text   ・・・平均12byte
```

int	timestamp	text
4 byte	8 byte	(4 + avg) byte
4 byte	8byte	(4 + avg) byte

1）**1 行のサイズ** = [行ヘッダ**28 byte**] + [行データ**4+8+ (4+平均文字サイズ) byte**]

2）**1ページ中のデータ領域** =(**8192 byte** – **24 byte**) × FILLFACTOR
(割合に変換)

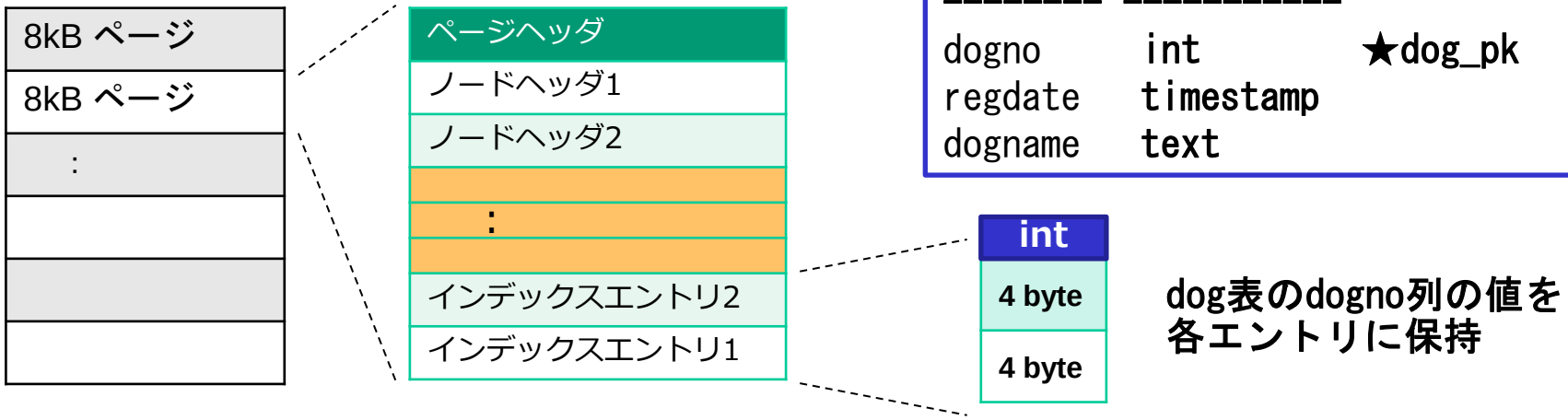
3）**ページに格納できる行数**＝**データ領域** ÷ **1行のサイズ**

4）テーブル行数÷**ページに格納できる行数**＝**必要なページ数**

5）**必要なページ数 × 8192 byte = テーブルサイズ**

インデックスサイズの見積もり

- インデックスページの固定領域は **40 byte**
- 1エントリ追加されるごとに ノードヘッダ **12 byte**
- 1エントリが消費するサイズ



1）**1 ノードのサイズ** = [ノードヘッダ**12 byte**] + [int型のエントリ **4 byte**]

2）**1ページ中のデータ領域** =(**8192 byte** – **40 byte**) × FILLFACTOR
(割合に変換)

3）**ページに格納できるノード数**=**データ領域** ÷ **1ノードのサイズ**

4）インデックスエントリ数（テーブル行数）÷**ページに格納できるノード数**=**必要なページ数**

5）**必要なページ数 × 8192 byte = インデックスサイズ**

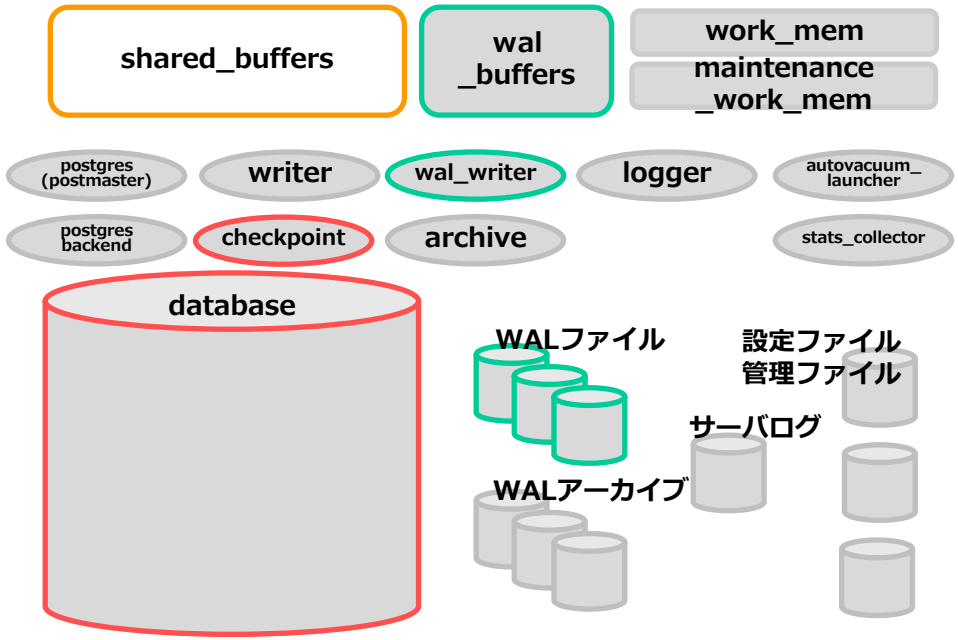
■平常時の動作

- 1つ16MB のWALセグメントファイルを循環利用
  - –WALファイルはチェックポイントが済めば不要
  - –不要なWALは自動で消去される　→max_wal_sizeの設定　(9.5～)
    PostgreSQLでは2世代前のチェックポイント以降のWALが残る
- ただし、リカバリ時はベースバックアップ以降のWALが必要となる

チェックポイント
更新が一定量行われたことを契機に
共有バッファの内容をディスクに書く

WAL
更新の度にディスクに書くと遅いので
メモリ上で更新を完結し、変更履歴のみ
ディスクに永続化

ある時点の チェックポイントが 完了した データファイル ＋ チェックポイント 以降の WALファイル ＝ 最新の データベース



④WALファイル

■平常時と異なる動作

- WALファイルの領域が枯渇するとDB停止に至る
- WALファイルが蓄積されるケース
  - アーカイブモード
    - アーカイブモードはWAL消失を防ぐための設定
    - アーカイブ先の空き領域が無い場合、WAL消失しないため本来循環されるWALファイルを残しておく
  - レプリケーション
    - スタンバイに転送予定だがまだ転送が完了していないWAL
    - レプリケーション・スロット(～9.4)

構築時の設定–メモリ

shared_buffersの初期設定

- ディスクから読み取ったデータをキャッシュして高速に応答
- チューニングポイント　**shared_buffers**パラメータ
  - –初期設定値は、物理メモリの 25%-40% 程度とする
  - –頻繁にアクセスされるデータがキャッシュできるようにサイズを調整

- キャッシュヒット率の確認
  - –キャッシュヒット率の算出
    - – pg_stat_databaseに記録されるアクセスしたブロック数
      - メモリ上でヒット（**blks_hit**）とディスクI/O（**blks_read**）したブロック数
      - **blks_hit** + **blks_read**　= 必要とした総ブロック数
  - –キャッシュヒット率

$$= \frac{\text{blks\_hit}}{\text{blks\_hit} + \text{blks\_read}} \times 100(\%)$$

構築時の設定–メモリ

アシスト 「お客様の最高」のために　OSS-DB

WALバッファの初期設定

- WALファイルのI/Oをシーケンシャルにするため、メモリ上に変更をキャッシュ
- チューニングポイント
  - –WALフラッシュ回数を減らすよう**wal_buffers**のサイズを調整
  - –バージョン9.x以降では自動調整されるが、自動調整の上限は16MB
  - –実際にはWALセグメントサイズの倍(32MB)程度とするのが望ましい
- WALフラッシュのタイミング
  - –トランザクションがコミットされたとき
  - –wal_writer_dilay時間が経過したとき　デフォルト3秒
  - –WALバッファが一杯になったとき

構築時の設定–プロセス

アシスト 「お客様の最高」のために

OSS-DB

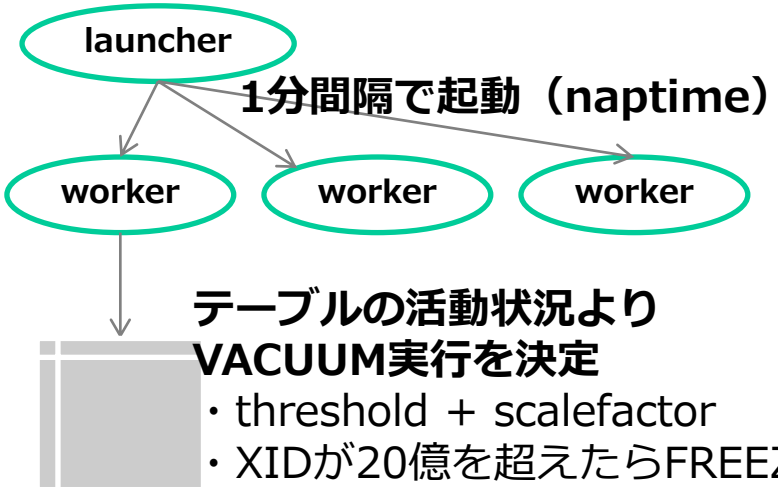
loggerプロセス

- デフォルトは無効だが、有効化しておくべき
  - –logging_collector　= on
  - –log_destination = `ログ出力先ディレクトリ(デフォルトでOK)`
  - –ログ循環設定
  - –ログ出力設定

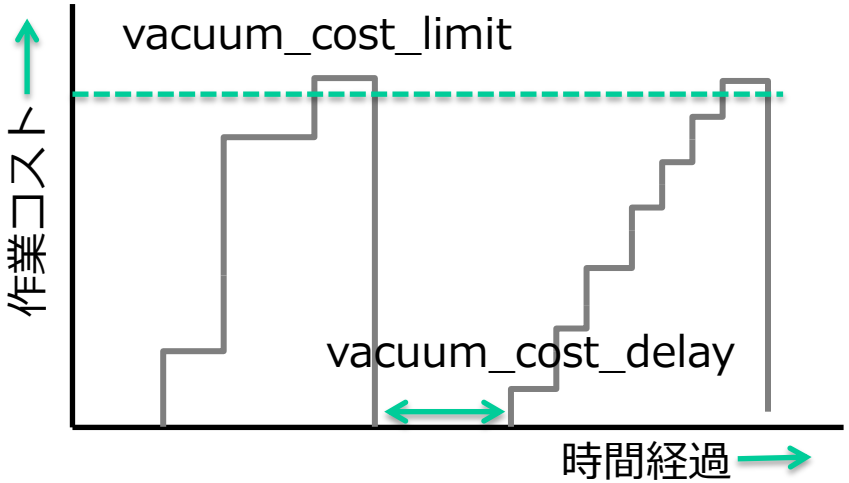
autovacuum_launcherプロセス

- 自動VACUUMを活用
  - –autovaccuum = on
- VACUUMの負荷はシステム全体の性能に多少の影響を与える
  - –自動VACUUMのタイミングを理解しておく
  - –autovaccuum_vacuum_threshold
    autovacuum_vacuum_scalefactor
  - –性能影響を抑えるために、ゆるやかにVACUUMさせる(遅延VACUUM)

自動VACUUMの実行間隔



遅延VACUUM



構築時の設定-プロセス

アーカイブ運用

- 障害発生直前に確定されたデータまで復旧したい場合に必須の設定
  - archive_mode
  - archive_command ：WALファイルをコピーするコマンド(CPやSCP)
  - wal_level：最新版ではreplica/logical、以前はarchive/hot_standby

レプリケーション構成

- wal sender およびwal recieverプロセスが起動
  - マスター側で、max_wal_sendersで指定した数のスタンバイを持てる
- スタンバイ側では、ホットスタンバイモードを指定し recovery.confにマスターへの接続情報を記載
  - hot_standby = on
  - recovery.confの設定

セキュリティ関連設定

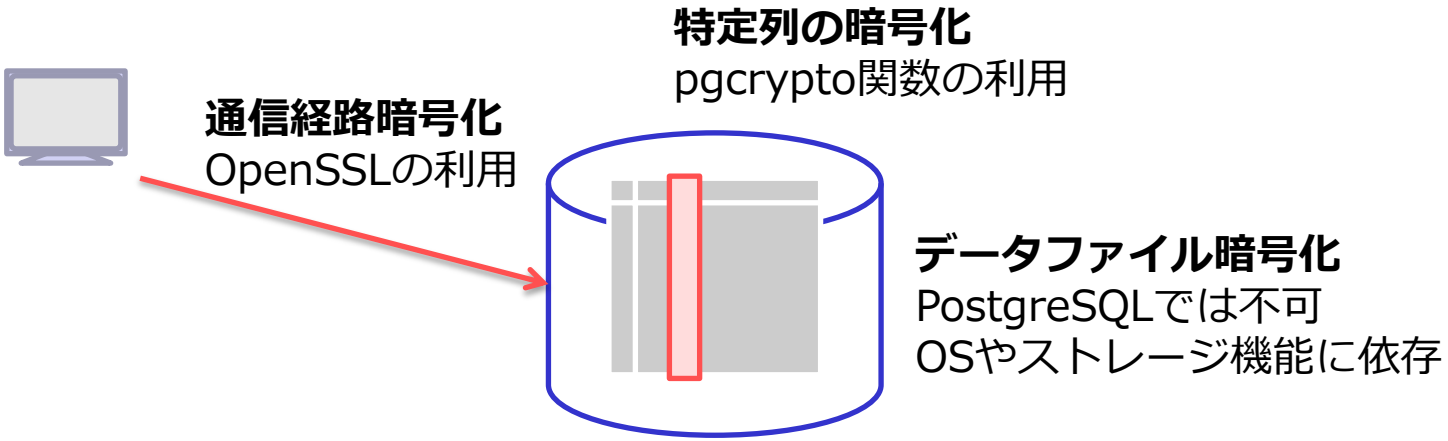
アシスト 「お客様の最高」のために　OSS-DB

- PostgreSQLでできることの整理
  - ①データ暗号化に関して
  - ②通信経路暗号化
  - ③監査情報の取得

- 試験ではPostgreSQLでできることのみでなく、
  通常対策しておくべき課題、OSや他のソフトウェア機能での
  実装を検討する内容も問われる

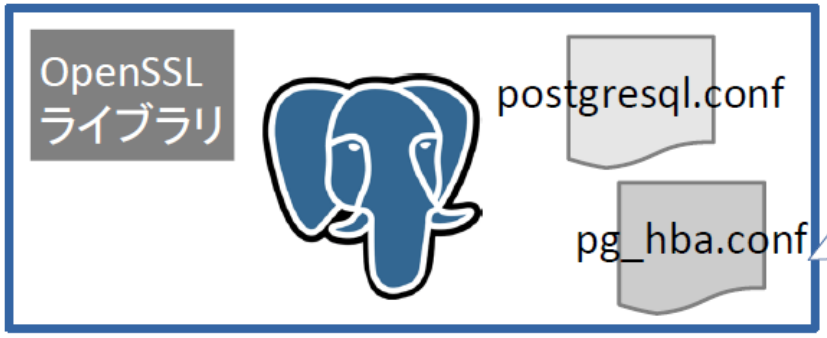
①データ暗号化

- 暗号化で検討される一般的な考慮点
  - **格納時**の暗号化：個人情報を含む列のみを暗号化するなどAP側で実装
  - **通信経路**暗号化：実行されるSQL文の盗聴を防ぐ(次項で解説)
  - **ファイル**暗号化：バックアップの持ち出しなどから保護
- PostgreSQLでの対応
  - PostgreSQLでは、**データ格納時にpgcrypto関数**を使用して列単位の暗号化を行うことができる
  - **通信経路暗号化**はOpen SSLライブラリをロードして使用できる
  - **データファイル暗号化はできず**、暗号化機能を持ったストレージ等を検討



②通信経路暗号化

- クライアント-サーバ間のSSL通信設定を理解しておく
  - PostgreSQLのインストール時
    - configure実行時に--with openssl オプションを追加
    - RPM版によるインストールでは本設定も有効化されている
  - データベースクラスタの設定
    - ssl = on
    - SSL用サーバ証明書の設定や、秘密鍵を含む設定ファイルを用意し指定
  - 認証設定
    - クライアント毎にSSL通信を強制するかどうかを指定可能
    - 認証設定ファイル pg_hba.conf ファイルにtype = hostsslを指定



pg_hba.confの書式に従いアクセス制御

type	database	user	host	method
hostssl	db1	usr1	xx.xx.xx.xx	md5
hostnossl	db2	usr2	xx.xx.xx.xx	md5

③監査情報の取得

アシスト 「お客様の最高」のために　OSS-DB

監査で検討される一般的な考慮点

- 実行された処理を記録
  - 実行されたSQL文の記録
  - 失敗したSQL文の記録　：アドホックなSQLを実行する管理者からの処理
  - 特定のテーブルに対する処理の記録　：個人情報を含むテーブルのみ監視
- ログイン/ログアウトを記録
  - DBAアカウントによる操作の記録　：申請に基づく操作が行われているか
- 専用の監査ログファイルの作成
  - DBA/セキュリティ担当者の職務分掌

PostgreSQLでの対応

- log_statementの指定によるSQL文を記録
  - ユーザ単位で指定可能
  - テーブル単位やSELECTのみは不可
- log_(dis)connectionの指定による
  ログイン/ログアウトの記録

log_statementの設定

none	出力無し(デフォルト)
ddl	実行されたDDL文のみ記録
mod	実行されたDDL文とDML文を記録 ※COPY FROや、PREPARE、EXPLAIN ANALYZEの該当するSQL文を含む
all	実行された全SQLを記録

広範囲な試験範囲をDB管理者の業務に当てはめて理解

- サーバ構築
- 初期設定
- 監視
  - 状態監視
  - 死活監視
  - パフォーマンス監視
- メンテナンス
  - オブジェクトのメンテナンス
  - ユーザーのメンテナンス
- チューニング
- 障害復旧
- レプリケーション

カタログを確認して、
そこから何が読みとれるか
理解する

運用管理

- 監視
  - 容量監視
  - プロセス監視
  - サーバーログ監視
  - パフォーマンス監視
- オブジェクトのメンテナンス
  - VACUUM(自動VACUUMを含む)
  - ANALYZE
  - HOT と FILLFACTOR
  - REINDEX
- ユーザ(セッション)の管理
  - 長時間実行しているSQL、ロック
  - ユーザ設定

容量監視

- ディスク容量、データベース、オブジェクト等のサイズを監視
  - ディスク容量はOSコマンド(df 、du –s など)で監視
  - 特に以下の領域に注意
    - –データファイル
    - –WALファイル
    - –アーカイブ領域(ローカルディスクに配置している場合)
  - データベースやオブジェクトサイズは関数で確認
    - –pg_database_size(‘database’)
    - –pg_relation_size(‘object’)
    - –pg_total_relation_size(‘table’)

死活監視

- サーバの死活監視はプロセス監視またはクエリ実行で確認する
  - 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

パフォーマンス監視

- OSリソースやシステムカタログの監視、遅いSQLを監視
  - OSリソースの監視
    - sar、top、vmstat、mpstat、iostatなど
    - 平常時と比較し、CPU使用率やメモリ使用率が高くないか
  - システムカタログの監視

カタログ	用途
pg_stat_database	キャッシュヒット率の確認
pg_stat_bgwriter	チェックポイント間隔の適正値把握
pg_stat_all_tables	キャッシュヒット率やHOT更新、IndexScan割合 VACUUM設定の適正値把握
pg_statio_all_tables	同上(ブロック数で表示)
pg_stat_activity	ユーザセッション毎に実行されているSQLの情報を格納
pg_locks	pg_stat_activityとjoinし、他セッションをブロックしているSQLを特定

  - スロークエリの検出
    - ログ監視設定（log_min_duration_statement）で解説

サーバログ監視

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

- 閾値超過したSQLの監視

カタログ	用途
log_min_error_statement	指定のエラーラベルに対してSQLを記録する
log_min_duration_statement	実行に長時間要したSQLの特定
log_lock_waits / deadlock_timeout	ロック獲得に要した時間がtimeoutを超過

- その他指定イベントのログ出力
  - log_checkpoints
  - log_(dis)connections

オブジェクトのメンテナンス

テーブルの肥大化を抑制する

- 適切なVACUUMの実行
  - VACUUMの種類と内容

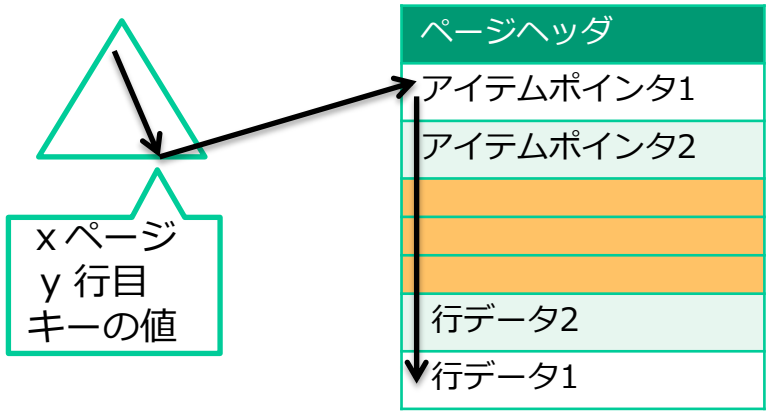
種類	内容
(通常の)VACUUM	不要行をFree Space Mapに登録し、再利用可能にする
VACUUM FULL	表の再作成を行い、不要行を詰めて物理ファイルの縮小を行う ※一時的に表サイズの2倍の領域を使用するため、ディスク不足時の領域確保には使えない
VACUUM FREEZE	トランザクションID周回問題への対処

  - 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
  - VACUUMが適切に実行されることで、一定サイズ以上には肥大しない

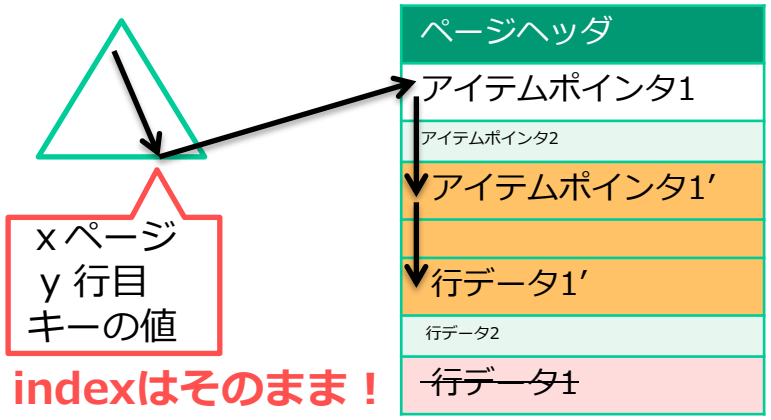
HOTによる更新とFILFACTOR

- HOTは行データが更新されてもインデックス更新をしない仕組み
- かつ、変更前データの領域はVACUUM不要で再利用可能となる

インデックスを使った
データへのアクセス



HOTによる更新

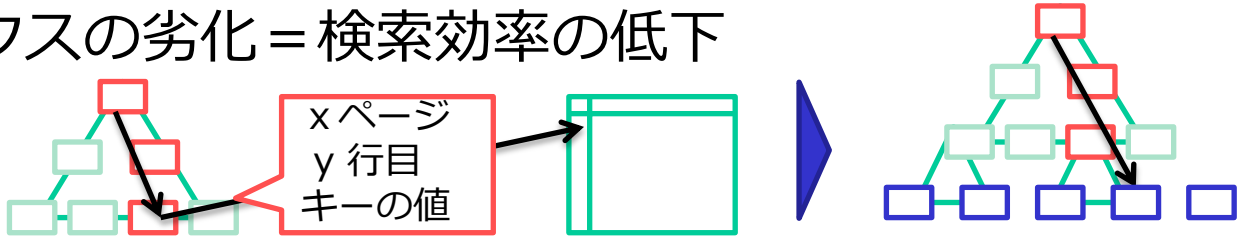


indexはそのまま！

変更前の領域は再利用可に

- HOT更新が有効になる条件
  - 同一ページ内に更新後のデータが格納できる場合
  - インデックスに格納されたキーの値自体が変更されない
  - 元の領域が再利用可能になるためには、ロングトランザクションが古いデータを見ていないこと

■インデックスの劣化＝検索効率の低下



- インデックスは、表の更新に合わせて自動で書き換えられるが
  - –ソートされたデータ構造を保つためブロック内に空きが無いと劣化が進む
  - –インデックスを指定したVACUUMはできない
    （テーブルのVACUUM時にページ単位で空きがあれば回収してくれる）

■メンテナンス手段

- 以下を比較して、ページ数が非常に大きい場合には再作成を検討
  - –pg_class の reltuples ： インデックスエントリの行数
    relpages ： インデックスのページ数
- 再作成の手段・・・ロック強度と所要時間のトレードオフ
  - –REINDEX
  - –DROP INDEX/CREATE INDEX
  - –CREATE INDEX CONCURRENTRY

ユーザー(セッション)の管理

ユーザーの処理が他に影響を与えるケース

- pg_stat_activity：**トランザクション開始時刻や実行SQL**
  - 長時間実行されているSQLは強制終了対象
    - 多くのユーザーをロック待ちさせている可能性
    - HOTやVACUUMの妨げ（参照されている行は回収不可）
- pg_locks：あるセッションが**どの表のロックを確保/待機しているか**
  - pg_stat_activityとJOINすることでどのSQLが悪いか特定
  - 各構文が必要とする表ロックの強度は覚えておくこと
- pg_stat_statements：**実行されたSQLを記録、分析**
  - 所要時間が長い/実行回数が多いSQLはチューニング効果大
  - contribツール
    - rpmやソースコードから有効化の方法を理解しておくこと
    - CREATE EXTENSION、shared_preload_libraries

アジェンダ

■広範囲な試験範囲をDB管理者の業務に当てはめて理解

- サーバ構築
  - –OSの設定～PostgreSQLのインストール
- 初期設定
  - –サーバサイジング
  - –パラメータ設定
  - –セキュリティ
- 監視
- メンテナンス
- チューニング
- 障害復旧
- レプリケーション

申し訳ありません。
本日は時間の関係で
さっと紹介する程度です。

チューニングの領域へ

DBチューニングとSQLチューニング

- DBチューニング
  - 構築段階からある程度の設定が可能で、大まかな設定でも効果がでる
  - パラメータチューニングが主
  - 評価指標としてシステム全体でどの程度の処理が可能かを表すTPSなど
- SQLチューニング
  - 問題のあるSQLを特定、実行計画を調整することで大きな効果を期待
  - 索引を使用しているかどうか、パラメータ設定による実行計画の強制
  - 評価指標として、対象SQLのレスポンスタイムなど
- 双方が与える影響
  - 高負荷な1つのSQLが改善することでシステム全体が最適化され、TPSも向上する可能性もある
  - SQLチューニングのために索引を作成したことで、他の処理に悪影響を及ぼす可能性もある

ベンチマークの取得

- TPSを計測するモデル　pgbench　・・・contribツール

チューニングの観点

- 各プロセスの詳細動作をチューニング
  - 自動VACUUMの遅延設定、ダーティバッファ書き出し
  - チェックポイント
  - これらの設定根拠にpg_stats_xxx ビューのどの列を見れば良いか
    どのようなログ出力を見れば良いか
- マイナーなパラメータも出題傾向あり
  - fsync
  - full_page_writes
  - synchronous_commit
  - commit_delay
  - wal_sync_method

いずれもWAL関連の設定

WALを吐かない設定=信頼性を犠牲にして
大幅な性能向上

各パラメータがどのようなリスクと効果を
期待できるか理解しておく必要がある

SQLチューニングの要素

- チューニング対象の決定
  - pg_stat_activityやpg_stat_statements
  - エラーログから実行時間の長いSQLを特定
- 実行計画
  - EXPLAINとEXPLAIN ANALYZE
  - 読み解き方の基本を理解（costの理解、actual rowsやtime、loop）
- 行推定
  - 統計情報≒プランナ統計とは
    - 実行計画作成時に利用。列に含まれるデータをサンプリングしたもの
    - アクセス統計（stats collectorで取得）とは別モノ
  - pg_statistics表と、pg_statsビュー

障害の種類

■障害の種類を整理しておく

- 電源障害、プロセス障害
  - –メモリ上のデータが消失するが、起動時にリカバリされる
- ファイル破損
  - –正常なバックアップがあれば、復旧は可能
  - –限定的な障害の場合、少ない影響で復旧させることを検討

→エラーログから障害箇所を特定

PANIC:could not create file pg_xlog・・・

故障個所	対処方針
WALの破損(pg_xlog)	pg_resetxlogの検討
データの破損 (baseやテーブルスペース)	zero_damaged_pages 使い方と、その後の高級対処策
システムデータの破損(global)	シングルユーザモードで接続、REINDEX
インデックス (baseやテーブルスペース)	インデックス再作成

- データ(テーブルやインデックス)、WALの物理ディスク障害
  - 適切なバックアップがあればリストア・リカバリできる
  - 障害の種類を把握して影響を最小限に対処するには？

インデックスデータの場合

- データの破損ではないため、まずはインデックス再作成を検討する
- システムテーブルのインデックスの場合は注意が必要
- インデックス再作成は、システムに与える影響を考慮して検討

WALファイル

- 起動時に読み込まれるため、WALが破損していると起動できない
- pg_resetxlogを使用してWALのリセットを行う
- または、正常なトランザクション位置
  (アーカイブ化されたもの)を指定してPITRを行う

$ pg_resetxlog −f −x [ ] $PGDATA

pg_clog配下のXIDを確認
次のXIDを指定してWALをリセット
例） pg_clog配下が**0011**なら、1を加えて
末尾に0を5つ付けた **0012000000**

容量不足時

■データ、WAL領域、それぞれで発生する問題を整理

- 容量不足により各ファイルへの書き込みが失敗するとどうなるか

データファイル

–データ破壊を引き起こす可能性がある

–ディスク容量不足に対して、以下の対処は誤り

– SELECT処理のみのシステムであり、そのまま運用する

– VACUUM (FULL)を実行する

– ＄PGDATA/base配下を手動で×××

WALファイル

–領域が不足すると、データベースクラスタが停止する

PANIC: could not create file “ pg_xlog/xxxxx ”

- 対処の案

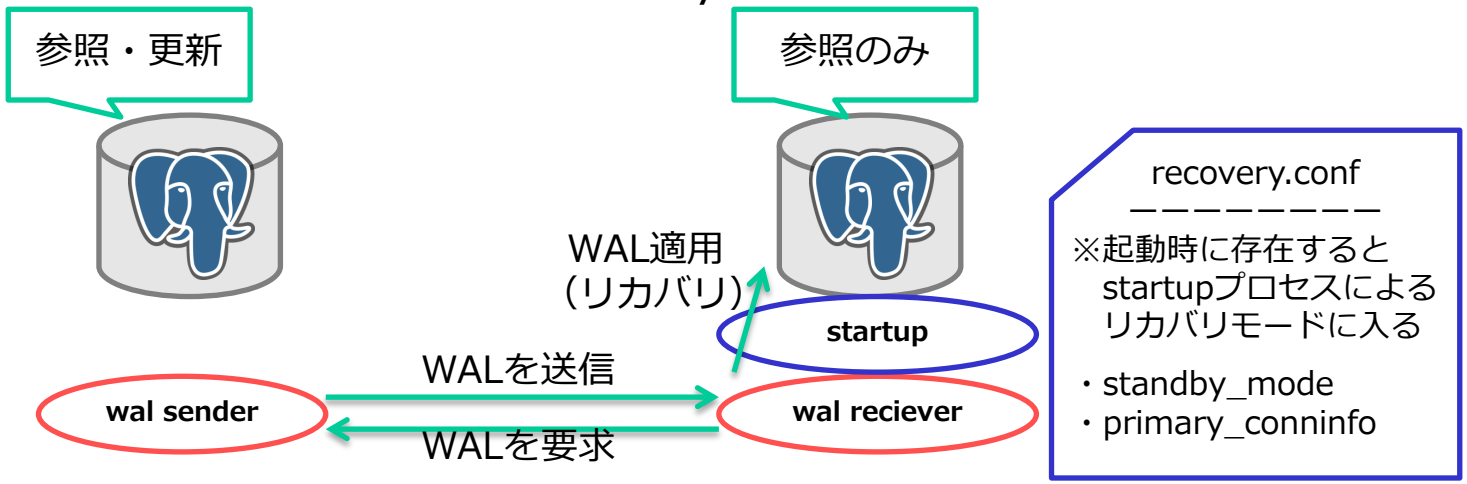
–LinuxのLVMなどで領域追加

–テーブルスペース機能で別領域に移動（データファイル）

–シンボリックリンクで別領域に移動（WALファイル）

レプリケーション構成について

- バックアップリカバリの応用
  - ある時点のベースバックアップにWALを順次適用する仕組み
  - ホットスタンバイモードで参照可能
  - ストリーミングレプリケーション（wal sender、wal reciever）
  - スタンバイ側の設定（recovery.conf）



- プライマリ障害時の操作
  - スタンバイ側で pg_ctl promoteすると新プライマリに昇格
  - このときタイムラインが1繰り上がる

レプリケーションの多様な構成

■データ同期モード

- 更新がどこまで伝搬したらマスターで成功とみなすか

	パラメータ	
	synchronous_commit	**synchronous_standby_names**
完全同期	remote_apply	同期対象スタンバイのリスト
準同期	on または remote_write	
非同期	指定なし	指定なし

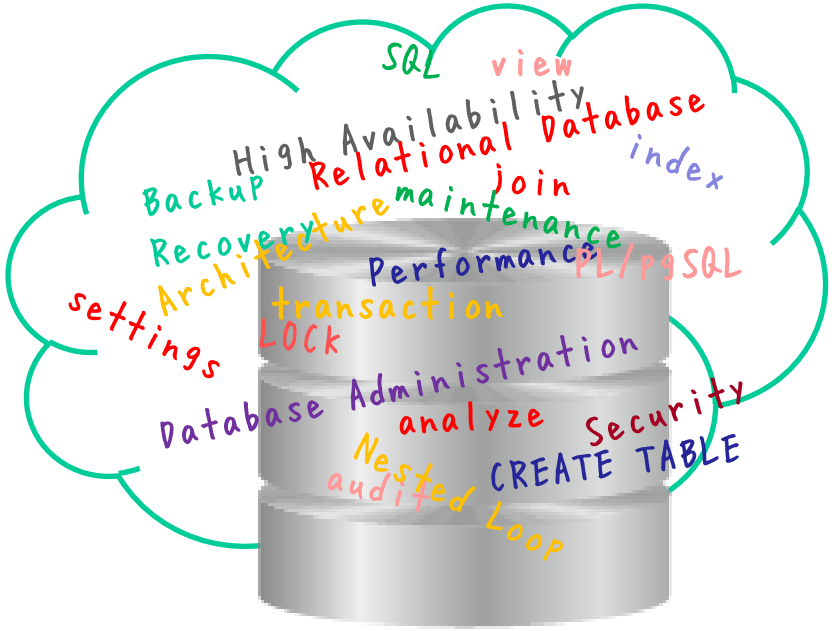
■マルチスタンバイ

- 上記の「同期対象スタンバイのリスト」は通常は先頭の1台が同期、その他のスタンバイは非同スタンバイ期
- 以下の書式で複数台(**N台**)の同期設定が可能（9.6～）

synchronous_standby_names = **N(**slave1,slave2,slave3**)**

※N=2のとき、先頭2台のslave1、slave2が同期対象になる

- 広範囲におよぶ出題範囲をDBA業務に当てはめて理解
  - 大規模データベースを適切に運用管理するプロフェッショナルであるデータベース管理者（DBA）の業務にあてはめ出題範囲を整理
  - 具体的なコマンドでは無く、後からマニュアルを読みこなせるような詳細な実装や、動作のイメージを持つ



参考文献

Webサイト

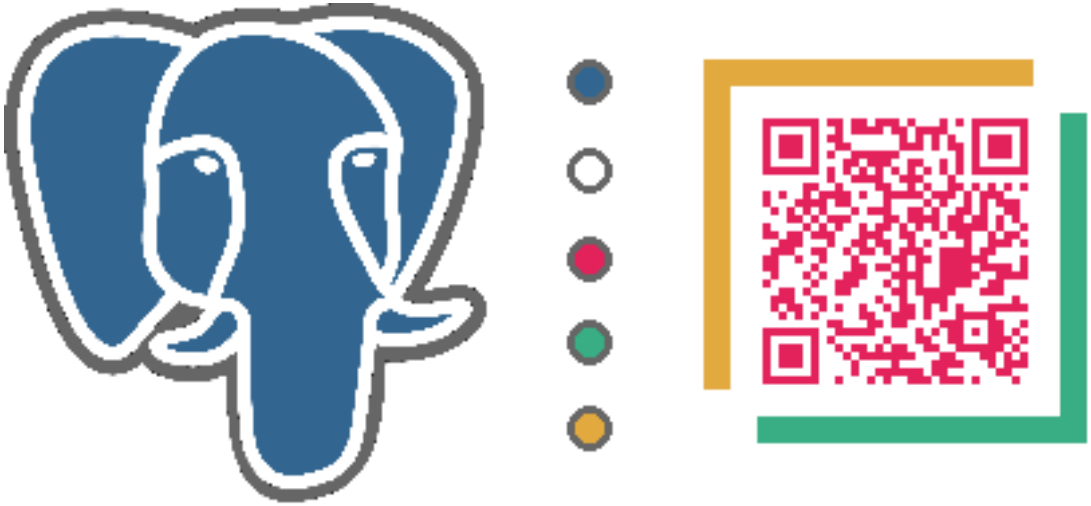
- PostgreSQLマニュアル
  https://www.postgresql.jp/document/9.4/html/index.html

書籍

- LPI-Japan OSS-DB Gold認定教材
  PostgreSQL高度技術者育成テキスト
  NTTソフトウェア株式会社
- 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

- 日本PostgreSQLユーザ会（ https://www.postgresql.jp/ ）
  - 全国各地で勉強会の開催、講師派遣など
  - ドキュメント翻訳
  - pgAdmin翻訳　　など

- PostgreSQLのslack部屋（ http://tinyurl.com/pgsql-slackin ）



質問、相談、雑談、お知らせなど、PostgreSQLやデータベースにまつわる話題なら何でもOK

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

- 体系的な知識を持った技術者の存在は採用する製品を選定する観点で重要視され、ベンダ資格がないPostgreSQLにとっては普及の起爆剤となる。
- データベースに限らずOSSを使うことが一般化してきていることから、本資格取得による個人のキャリアアップの面と、さらなる製品の普及促進の面から非常に重要。

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