OSS-DB

OSS-DB Exam Silver
技術解説無料セミナー
2023/1/21 開催

主題	運用管理
副題	基本的な運用管理作業 【重要度：7】 ・データベースロール（権限、属性）の設計 ・データベースの運用必要なシステムカタログ

本日の講師

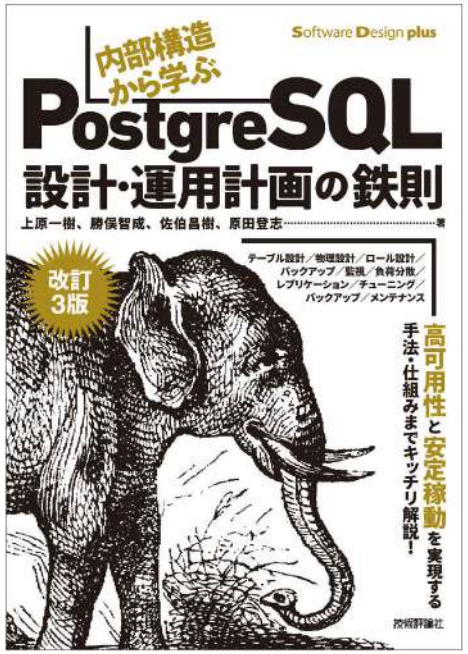
NTT テクノクロス株式会社
上原 一樹

LPI-JAPAN

講師紹介

■上原　一樹

- NTTテクノクロス株式会社　DBチーム所属
- PostgreSQL関連業務に従事
  - テクニカルサポート、コンサルティング
  - データベースの移行支援
  - 研修講師
- 改訂第3版 PostgreSQL 設計・運用計画の鉄則を執筆
  - PostgreSQL 14までの機能をカバー、一部15の情報も解説しています。



2022年11月に発売！

■NTTテクノクロス株式会社（https://www.ntt-tx.co.jp/）

- DBチームでは、PostgreSQLを中心に各種サポートを提供。
  - オンプレのPostgreSQL、パブリッククラウド上のマネージドサービスを対象。
  - PostgreSQLに対応する周辺ツールについても対応（pg_statsinfo, pg_hint_plan等々）
- 技術ブログ（情報畑でつかまえて）では、DBをはじめ、様々な情報を投稿しています。

■OSSクラウド基盤トータルサービス

- https://www.ntt-tx.co.jp/products/osscloud/
- PostgreSQL移行・運用支援サービス、チケットサービスを提供しています。

■OSS-DBとは

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

認定試験の傾向と対策

■公式ドキュメントは使用するバージョンのドキュメントを読むべき！

- バージョンによって、機能の違いがあるため、採用するバージョンとあったものを読む。
- 本セミナー資料でも参考として、バージョン11のPostgreSQL文書のリンクを記載する。

■GUCパラメータやシステムテーブル・ビューは、単純に意味を覚えるのではなく、影響まで理解しなければならない

例 deadlock_timeout

意味: ロック状態になった時にデッドロック検出処理を開始するまでの待機時間

影響: 値を小さくすると、デッドロックの検出は早くなるが、 実際にはデッドロックが発生していないのに検出処理が 動くことが多くなるため、CPUに無駄な負荷がかかる可能性も高くなる

■実機での動作確認は極めて重要

- 自分が予想した通りに動作しなければ、何かしら理解不足があるということ。

セミナーのテーマについて

■本日のテーマ

- データベース管理者として押さえるべき運用管理コマンドとPostgreSQLが提供する運用機能について、Silverの出題範囲をベースに解説します。
  - 過去のセミナで解説済みであり、補足で解説する事項がないものは本日の対象から外しています。
- 本セミナーではSilverの範囲を中心に解説を行いますが、普段の業務で活かせるようにPostgreSQLを利用する上で理解してもらいたい内容は、最新のPostgreSQL 15の情報に加えて紹介させていただきます。

■過去のオンラインセミナーのテーマ（どれも重要なテーマですので、ご活用ください。）

- 「PostgreSQLのバックアップ方法」
  - https://oss-db.jp/__/download/5f0fb75b37471d78510554e4/20200719-silver-01.pdf
- 「VACUUM、ANALYZEの目的と使い方」と「自動バキュームの概念と動作」
  - https://oss-db.jp/__/download/5f3de5a37ea72b5fa4113218/20200905-silver-01.pdf
- 「トランザクションの概念、SQLコマンド、Ver.2.0で追加された項目」
  - https://oss-db.jp/__/download/5f90e6a2c1fa8478bd3cd9a0/20201017-silver-01.pdf
- 「標準ツールの使い方、SQLコマンド（データ操作、データ型、インデックス）」
  - https://oss-db.jp/__/download/6073d1843701c51de41e20ee/20210410-silver-02.pdf
- 「運用管理、設定ファイル」
  - https://ferret-one.akamaized.net/files/60dae99cd38ec80f4ec003f8/20210629-silver.pdf

アジェンダ

運用管理（52%）
基本的な運用管理作業【重要度：7】

- **説明：**
  データベース管理者として実行する基本的な運用管理コマンドに関する知識を問う

- **主要な知識範囲：**
  PostgreSQLの起動・停止方法
  データベースロール / ユーザの概念
  データベースロール / ユーザの追加・削除・変更方法
  VACUUM、ANALYZEの目的と使い方
  自動バキュームの概念と動作
  システム情報関数
  情報スキーマとシステムカタログ
  テーブル単位の権限（GRANT/REVOKE）

- **重要な用語、コマンド、パラメータなど：**
  pg_ctl start / stop
  CREATE/ALTER/DROP ROLE/USER
  VACUUM
  ANALYZE
  vacuumdb
  autovacuum
  current_user
  version
  information_schema
  GRANT
  REVOKE

アジェンダ

運用管理（52%）
基本的な運用管理作業【重要度：7】

- **説明：**
  データベース管理者として実行する基本的な運用管理コマンドに関する知識を問う

- **主要な知識範囲：**
  PostgreSQLの起動・停止方法
  データベースロール / ユーザの概念
  データベースロール / ユーザの追加・削除・変更方法
  VACUUM、ANALYZEの目的と使い方
  自動バキュームの概念と動作
  システム情報関数
  情報スキーマとシステムカタログ
  テーブル単位の権限（GRANT/REVOKE）

- **重要な用語、コマンド、パラメータなど：**
  pg_ctl start / stop
  CREATE/ALTER/DROP ROLE/USER
  VACUUM
  ANALYZE
  vacuumdb
  autovacuum
  current_user
  version
  information_sche
  GRANT
  REVOKE

2020/9/5 開催のセミナー資料で解説されていますが、非常に重要な機能でもあるため、本日は概要の説明と運用の考え方に絞って説明します。

機能の詳細は過去のセミナー資料をご参考ください。

環境情報

■VMを用意する

- Rocky Linux 8

```
[root@ossdb ~]# cat /etc/redhat-release
Rocky Linux release 8.5 (Green Obsidian)
```

※本日は実際にPostgreSQLを動かしながら説明をします。研修では、VirtualBox上にVMを1つ立てた1台構成の環境で実施します。可能な方はぜひ実機で動かしてみてください。

■PostgreSQLのインストール

- リポジトリ登録

```
[root@ossdb ~]# rpm -ivh https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

- PostgreSQL 11のインストール

```
[root@ossdb ~]# yum install -y postgresql11-server
```

参考：PostgreSQLのVersioning Policyについて

Versioning Policy

- 各メジャーバージョンはリリースから5年間サポートされる。
- 現在サポートされているのは、以下の5バージョン。

Version	Current minor	Supported	First Release	Final Release
15	15.1	Yes	October 13, 2022	November 11, 2027
14	14.6	Yes	September 30, 2021	November 12, 2026
13	13.9	Yes	September 24, 2020	November 13, 2025
12	12.13	Yes	October 3, 2019	November 14, 2024
11	11.18	Yes	October 18, 2018	November 9, 2023

参考：https://www.postgresql.org/support/versioning/

- PostgreSQL 10は2022年11月にEOLを迎えています。

■OSユーザの切り替え

- postgresユーザはRPMを用いたインストールでは自動で作成される。

```
[root@ossdb ~]# su - postgres
```

■環境変数の設定

- postgresユーザの .bash_profile は、~/.pgsql_profile を読み込む仕様になっている。

```
[postgres@ossdb ~]$ cat << EOF > ~/.pgsql_profile
PATH=/usr/pgsql-11/bin:$PATH
PGDATA=/var/lib/pgsql/local/11/data
export PATH PGDATA
EOF
```

- 再ログインまたは、sourceコマンドで環境変数を反映する。

```
[postgres@ossdb ~]$ source ~/.pgsql_profile
```

■データベースクラスタを作成

- initdbコマンドでDBクラスタの初期化を行う。

```
[root@ossdb ~]# initdb -D ~/local/11/data --no-locale --encoding=utf8
```

  - ロケールは no-locale を指定。
  - エンコーディングは UTF8 を指定。

運用管理(52%)
基本的な運用管理作業【重要度:7】

- **説明：**
  データベース管理者として実行する基本的な運用管理コマンドに関する知識を問う

- **主要な知識範囲：**
  PostgreSQLの起動・停止方法
  データベースロール / ユーザの概念
  データベースロール / ユーザの追加・削除・変更方法
  VACUUM、ANALYZEの目的と使い方
  自動バキュームの概念と動作
  システム情報関数
  情報スキーマとシステムカタログ
  テーブル単位の権限（GRANT/REVOKE）

- **重要な用語、コマンド、パラメータなど：**
  pg_ctl start / stop
  CREATE/ALTER/DROP ROLE/USER
  VACUUM
  ANALYZE
  vacuumdb
  autovacuum
  current_user
  version
  information_schema
  GRANT
  REVOKE

PostgreSQLの起動・停止方法

PostgreSQLの起動

- pg_ctl -D <DBクラスタのPATH> start

```
[postgres@ossdb ~]$ pg_ctl -D ~/local/11/data start
```

- 起動の確認は pg_isreadyコマンドで確認できる

```
[postgres@ossdb ~]$ pg_isready
/var/run/postgresql:5432 - accepting connections
```

PostgreSQLの停止

- pg_ctl -D <DBクラスタのPATH> stop [-m smart / fast (default) / immediate]

```
[postgres@ossdb ~]$ pg_ctl -D ~/local/11/data stop
```

- シャットダウンモードは3種類提供されている。
  - smart：新規の接続を禁止し、クライアントからの接続が全て閉じられたら、サーバを停止
  - fast：全ての実行中の処理をロールバックし、クライアントとの接続を切断後、サーバを停止
  - immediate：クリーンアップ処理なしで、全てのサーバプロセスを即座に中断（クラッシュ状態）

参考：データベースサーバの起動状態の確認方法

PostgreSQLが起動しているかどうかはプロセスからも確認が可能

```
[postgres@ossdb ~]$ ps -ef | grep postgres
root         2786    2342  0 16:19 pts/0    00:00:00 su - postgres
postgres     2787    2786  0 16:19 pts/0    00:00:00 -bash
postgres     2857       1  0 16:19 pts/0    00:00:00 /usr/pgsql-11/bin/postgres -D /var/lib/pgsql/local/11/data
postgres     2858    2857  0 16:19 ?        00:00:00 postgres: logger
postgres     2860    2857  0 16:19 ?        00:00:00 postgres: checkpointer
postgres     2861    2857  0 16:19 ?        00:00:00 postgres: background writer
postgres     2862    2857  0 16:19 ?        00:00:00 postgres: walwriter
postgres     2863    2857  0 16:19 ?        00:00:00 postgres: autovacuum launcher
postgres     2864    2857  0 16:19 ?        00:00:00 postgres: stats collector
postgres     2865    2857  0 16:19 ?        00:00:00 postgres: logical replication launcher
postgres     2866    2787  0 16:19 pts/0    00:00:00 ps -ef
postgres     2867    2787  0 16:19 pts/0    00:00:00 grep --color=auto postgres
```

- 設定やバージョンによって、起動するプロセスは異なる。
- 上記はPostgreSQL 11をデフォルトの設定で起動した場合の結果である。

参考：データベースサーバのバージョン確認方法

■データベースサーバのバージョン確認

- データベースへの接続

```
[postgres@ossdb ~]$ psql postgres
psql (11.9)
Type "help" for help.

postgres=#
```

- バージョン情報を取得

```
postgres=# SELECT * FROM version();
                                                 version
-------------------------------------------------------------------------------------------------
 PostgreSQL 11.9 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.3.1 20191121 (Red Hat 8.3.1-5), 64-bit
(1 row)
```

システム情報関数

PostgreSQLの制御や監視を行うために用意された機能

- セッション情報関数
  - pg_backend_pid(), current_user, version() , ...
- アクセス権限照会関数
  - has_any_column_privilege(), ...
- システムカタログ情報関数
  - pg_get_indexdef(), ...

括弧が不要な関数も存在する。

参考：https://www.postgresql.jp/document/11/html/functions-info.html

※システム情報関数とは別にシステム管理関数がある。OSS-DB Goldの範囲となるが運用で使用する機会が多いので、どういうものがあるか把握しておくことをおすすめする。

参考：https://www.postgresql.jp/document/11/html/functions-admin.html

ポイント

- システム情報関数としてどういう機能が提供されているか。
- どういった運用で必要になる機能か事前に把握しておく。

システムカタログ、システムビュー、情報スキーマ

■システムカタログとは

- 作成されたオブジェクトの内部的な情報を格納するテーブル。
- システムカタログは、通常のテーブルと変わらない形で扱われる。
  - SQLで操作可能であるが、内部データが破損する可能性があるため、手動での操作は実行してはいけない。

情報の実体はシステムカタログ上にあるが、基本的にユーザが使用するのは、システムビューまたは情報スキーマ（information_schema）である。

■システムビューとは

- システムカタログへの問い合わせを簡単に実施できるように用意されたビュー。
- PostgreSQL固有の機能に関する情報も扱う。

■情報スキーマ（information_schema.*）

- SQL標準で定められたビュー。
- SQL標準で定義された内容を扱うため、PostgreSQL固有の情報は扱わない。
- 標準化されているため、システムカタログやシステムビューと異なり、移植性がある。

システムカタログ、システムビュー、情報スキーマ

■システムビューには具体的にどういった情報が格納されているのか

- pg_locks
  - 現在取得されているロック情報（ロック対象、要求されたロックモード、PID等々）を確認できる。
- pg_settings：
  - SHOW文、SET文としての機能を持つことに加えて、各パラメータの最大や最小、設定値の反映契機等の情報を確認できる。

■利用例：ロックされたテーブルに関する情報取得

```
[postgres@ossdb ~]$ psql postgres -c "SELECT pn.nspname, pc.relname, pl.mode, pl.pid, pl.granted FROM pg_locks pl, pg_class
  pc, pg_namespace pn WHERE pl.relation = pc.oid AND pn.oid = pc.relnamespace AND pc.relkind = 'r' AND pl.pid <>
  pg_backend_pid()"
  nspname   |   relname    |      mode       | pid  | granted
------------+--------------+-----------------+------+---------
 public     | foo          | AccessShareLock | 1639 | t
 pg_catalog | pg_namespace | AccessShareLock | 1639 | t
 pg_catalog | pg_class     | AccessShareLock | 1639 | t
(3 rows)

[postgres@ossdb ~]$ psql postgres -c "SELECT pid, usename, xact_start, state, query FROM pg_stat_activity WHERE pid = 1639"
 pid  | usename  |          xact_start           |        state        |        query
------+----------+-------------------------------+---------------------+---------------------
 1639 | postgres | 2022-12-26 14:45:10.378127+09 | idle in transaction | SELECT * FROM foo ;
(1 row)
```

サンプル問題（システムカタログ）

■OSS-DB Silver サンプル問題/例題解説 から本テーマに関するものを紹介します。

- 2分後に解説を行うので、ぜひ回答を考えてみてください。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

サンプル問題 1.106

情報スキーマについて最も適切な記述を2つ選びなさい。

A) PostgreSQL以外も含め、すべてのリレーショナル・データベースで利用できる。

B) 標準SQLで仕様が定められている。

C) データベース内のオブジェクトに関する様々な情報を保持している。

D) informationという名前のスキーマ内のビューとして定義される。

E) 内容はシステムカタログと同等である。

引用元

- https://oss-db.jp/sample/silver_management_03/53_150511

運用管理(52%)
基本的な運用管理作業【重要度:7】

- **説明：**
  データベース管理者として実行する基本的な運用管理コマンドに関する知識を問う

- **主要な知識範囲：**
  PostgreSQLの起動・停止方法
  データベースロール / ユーザの概念
  データベースロール / ユーザの追加・削除・変更方法
  VACUUM、ANALYZEの目的と使い方
  自動バキュームの概念と動作
  システム情報関数
  情報スキーマとシステムカタログ
  テーブル単位の権限（GRANT/REVOKE）

- **重要な用語、コマンド、パラメータなど：**
  pg_ctl start / stop
  CREATE/ALTER/DROP ROLE/USER
  VACUUM
  ANALYZE
  vacuumdb
  autovacuum
  current_user
  version
  information_schema
  GRANT
  REVOKE

ロールについて

PostgreSQLにおけるロールの概念

- 「ユーザの概念」と「グループの概念」が含まれており、「ユーザ」、「グループ」、もしくは、その両方として扱われている。（Oracleや他のDBMSでは、ロールは権限の集合を指し、ユーザとは異なる概念である。）
- セキュリティ設計の要素である「認証：データベースへの接続の制御」と「認可：データベースオブジェクトへのアクセスの管理」で使用される。

自動で作成されるロールについて

- RPMでインストールした場合、デフォルトで postgres ロールが作成される。
  - postgresロールは強力な superuser属性 が付与されているため、取り扱いには注意が必要。
- 定義済みロール（旧：デフォルトロール）
  - superuser属性が必要となるような操作を部分的に切り出したロール。このロールを付与することで一般ユーザでも、特定の処理を行うことが可能になる。
  - 詳細は次ページを参照。

参考：定義済みロール（旧：デフォルトロール）

■定義済みロールを含むセキュリティ関連の機能は年々強化されている

- superuser属性が必要な操作を部分的に、一般ユーザ許可できるように定義済みロールが提供されている。
  - https://www.postgresql.jp/document/11/html/default-roles.html
  - https://www.postgresql.org/docs/15/predefined-roles.html
- 古いバージョンにおいては、superuser属性にしか許可されない操作が多く、管理者用ユーザで業務を実施するか、管理者以外のロールにsuperuser属性を与えるかしか選択肢がなく、権限分掌が難しいケースがある。

ROLE名	9.6	10	11	12	13	14	15	説明
pg_read_all_data						○	○	全てのスキーマへのUSAGE権限、全てのテーブル、ビュー、シーケンスに対してSELECT権限と同等のアクセスを許可する。BYPASSRLS属性は持たない。
pg_write_all_data						○	○	全てのスキーマへのUSAGE権限、全てのテーブル、ビュー、シーケンスに対してINSERT、UPDATE、DELETE権限と同等のアクセスを許可する。BYPASSRLS属性は持たない。
pg_read_all_settings		○	○	○	○	○	○	全ての設定変数の参照を許可する。
pg_read_all_stats		○	○	○	○	○	○	全てのpg_stat_*ビューの参照を許可し、統計関連の拡張機能を使用を許可する。
pg_stat_scan_tables		○	○	○	○	○	○	長時間シェアロックを取得する可能性があるモニタリング関数の実行を許可する。
pg_monitor		○	○	○	○	○	○	pg_read_all_settings、pg_read_all_stats、pg_stat_scan_tablesを包括して許可する。
pg_database_owner						○	○	暗黙のうちに現在のデータベース所有者のメンバとして構成される。
pg_signal_backend	○	○	○	○	○	○	○	他のバックエンドへの、クエリのキャンセルやセッションの終了を許可する。
pg_read_server_files			○	○	○	○	○	サーバ上でアクセスできる場所からファイルの読み込みを許可する。
pg_write_server_files			○	○	○	○	○	サーバ上でアクセスできる場所へのファイルの書き込みを許可する。
pg_execute_server_program			○	○	○	○	○	データベースを実行するユーザで、サーバ上のプログラムを実行することを許可する。
pg_checkpointer							○	CHECKPOINTコマンドの実行を許可する。

ロールの設計について

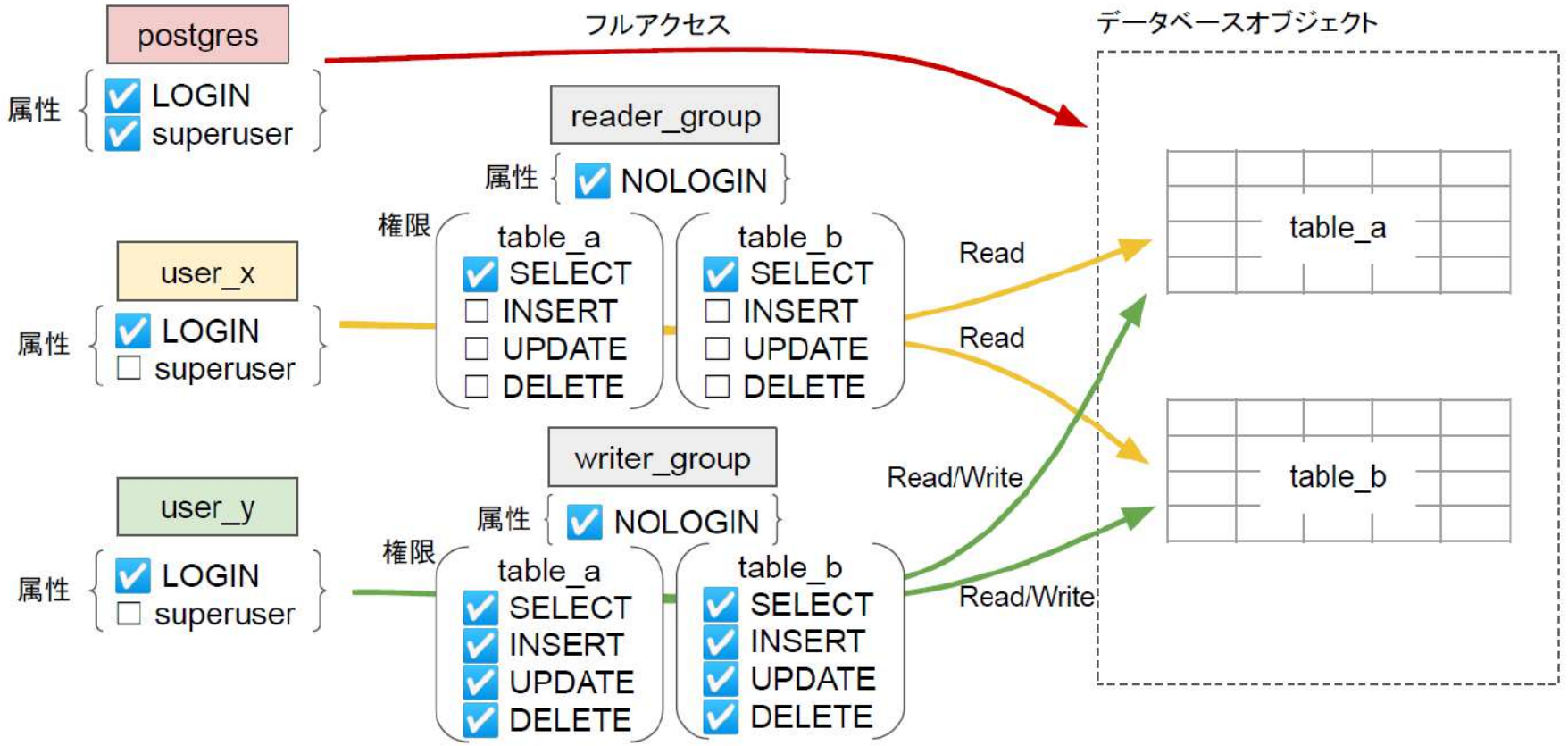
設計のポイント

- superuser属性は強力すぎるため、通常の業務では使用しない。
  - superuser属性を持つロールは制限なく全てのオブジェクトにアクセス可能（OSのroot権限相当）
- 原則として、データベースオブジェクトにアクセスするための最小権限のみを付与すべき。

ロールの作成及び権限付与について

■下記のようなロールを定義する場合のコマンド例を次ページで紹介する。

- 例では、ロールの定義や操作は postgres ロールを使用して実施しているが、CREATEROLE属性をもつロールであれば、ロールの作成や変更は可能。
- テーブルは事前に作成済みという前提での手順となっているので、実際に試す場合は対象となるテーブルを作成してから実施すること。



ロールの作成/削除

CREATE ROLE（データベースロールの作成）

PostgreSQL文書：https://www.postgresql.jp/document/11/html/sql-createrole.html

- 新しいデータベースロールを定義する。
- LOGIN権限を持つロールを作成する CREATE USER文も提供されている。その他にもOSコマンドとして、createuserコマンドが提供されている。

コマンド例：

```
postgres=# CREATE ROLE user_x LOGIN; -- LOGIN権限をもつ user_x ロールを作成する
CREATE ROLE
postgres=# CREATE ROLE reader_group; -- reader_group ロールを作成する
CREATE ROLE
```

```
postgres=# ¥du
                                   List of roles
  Role name   |                         Attributes                         |   Member of
--------------+------------------------------------------------------------+---------------
 postgres     | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 reader_group | Cannot login                                               | {}
 user_x       |                                                            | {}
```

DROP ROLE（データベースロールの削除）

PostgreSQL文書：https://www.postgresql.jp/document/11/html/sql-droprole.html

- 対象のロールを削除する。

ロールの操作

ALTER ROLE（データベースロールの変更）

PostgreSQL文書：https://www.postgresql.jp/document/11/html/sql-alterrole.html

- 対象のロールの変更を行う。属性の変更が必要な場合はALTER ROLEで行う。

コマンド例：

```
postgres=# CREATE ROLE user_y; -- LOGIN権限をもたない user_y ロールを作成する
CREATE ROLE
postgres=# CREATE ROLE writer_group; -- LOGIN権限をもたない writer_group ロールを作成する
CREATE ROLE
postgres=# ALTER ROLE user_y LOGIN; -- user_y ロールにLOGIN属性を持たせる
ALTER ROLE
```

```
postgres=# ¥du
                                   List of roles
  Role name   |                         Attributes                         |   Member of
--------------+----------------------------------------------------------------+---------------
 postgres     | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 reader_group | Cannot login                                                   | {}
 user_x       |                                                                | {}
 user_y       |                                                                | {}
 writer_group | Cannot login                                                   | {}
```

権限の操作

GRANT（権限付与）/REVOKE（権限取り消し）

- 対象のロールのデータベースオブジェクトに対する権限を付与/剥奪する。
- 指定したロール内のメンバ資格を他のロールに付与/剥奪する。

PostgreSQL文書：https://www.postgresql.jp/document/11/html/sql-grant.html

コマンド例：

```
postgres=# GRANT reader_group TO user_x; -- reader_group ロールを user_x ロールに付与する
GRANT ROLE
postgres=# GRANT writer_group TO user_y; -- writer_group ロールを user_y ロールに付与する
GRANT ROLE
```

```
postgres=# ¥du
                                   List of roles
  Role name   |                         Attributes                         |   Member of
--------------+----------------------------------------------------------------+---------------
 postgres     | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 reader_group | Cannot login                                                   | {}
 user_x       |                                                                | {reader_group}
 user_y       |                                                                | {writer_group}
 writer_group | Cannot login                                                   | {}
```

権限の操作

■コマンド例（続き）：

```
postgres=# GRANT SELECT ON ALL TABLES IN SCHEMA public TO reader_group;
GRANT
postgres=# GRANT SELECT, UPDATE, INSERT, DELETE ON ALL TABLES IN SCHEMA public TO writer_group;
GRANT
```

```
postgres=# ¥dp table_a;
                              Access privileges
 Schema |  Name   | Type  |     Access privileges      | Column privileges | Policies
--------+---------+-------+----------------------------+-------------------+----------
 public | table_a | table | postgres=arwdDxt/postgres +|                   |
        |         |       | reader_group=r/postgres   +|                   |
        |         |       | writer_group=arwd/postgres |                   |
(1 row)
```

INSERT, SELECT, UPDATE, DELETE権限を
postgres ロールから付与された　と書かれている。

サンプル問題（ROLE）

■OSS-DB Silver サンプル問題/例題解説 から本テーマに関するものを紹介します。

- 2分後に解説を行うので、ぜひ回答を考えてみてください。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

サンプル問題 1.106

PostgreSQLにおけるCREATE USERとCREATE ROLEの違いの説明として、適切なものを2つ選びなさい。

A) CREATE ROLEは一般ユーザでも実行できるが、CREATE USER の実行はスーパーユーザ権限が必要である。

B) CREATE ROLEで作成したロールを GRANT コマンドによりユーザに付与することができるが、CREATE USERで作成したユーザをGRANTで他のユーザに付与することはできない。

C) CREATE ROLEで作成したロールのLOGIN属性はNOLOGIN、CREATE USERで作成した場合はLOGINがデフォルトになっている。

D) CREATE USERでユーザを作成したら、同じ名前のスキーマが自動的に作成されるが、CREATE ROLEの場合はスキーマが作成されない。

E) OSのコマンドラインからCREATE USERと同等の機能を実行するために createuser というコマンドが提供されているが、createrole というコマンドは提供されていない。

引用元

- https://oss-db.jp/sample/silver_management_06/106_200708

運用管理(52%)
基本的な運用管理作業【重要度:7】

- **説明：**
  データベース管理者として実行する基本的な運用管理コマンドに関する知識を問う

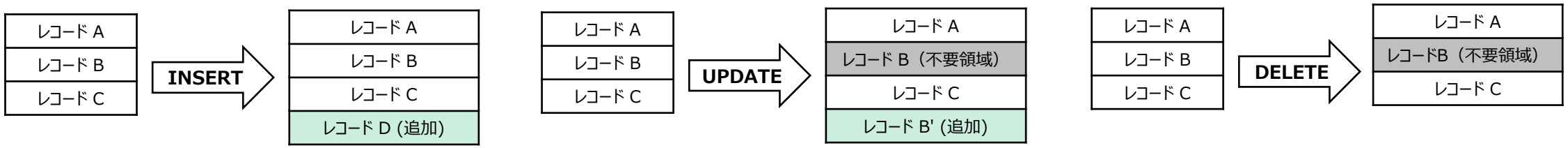
- **主要な知識範囲：**
  PostgreSQLの起動・停止方法
  データベースロール / ユーザの概念
  データベースロール / ユーザの追加・削除・変更方法
  VACUUM、ANALYZEの目的と使い方
  自動バキュームの概念と動作
  システム情報関数
  情報スキーマとシステムカタログ
  テーブル単位の権限（GRANT/REVOKE）

- **重要な用語、コマンド、パラメータなど：**
  pg_ctl start / stop
  CREATE/ALTER/DROP ROLE/USER
  VACUUM
  ANALYZE
  vacuumdb
  autovacuum
  current_user
  version
  information_schema
  GRANT
  REVOKE

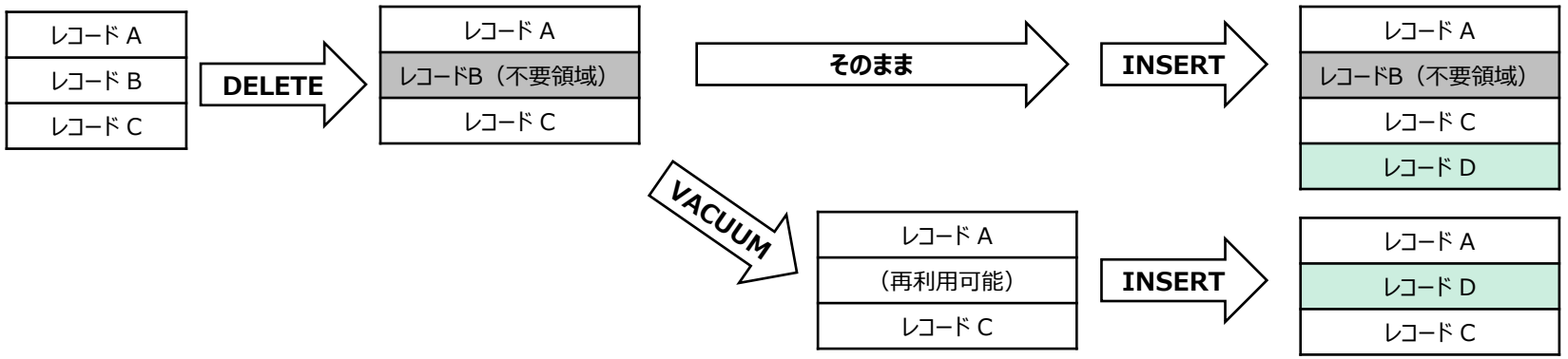
データベースのメンテナンスについて話す前に

■追記型アーキテクチャについて

- PostgreSQLは追記型のアーキテクチャであり、UPDATEやDELETEで削除された古いレコードは不要領域（dead tuple）として残る。
- 追記型アーキテクチャのイメージは以下のとおり。



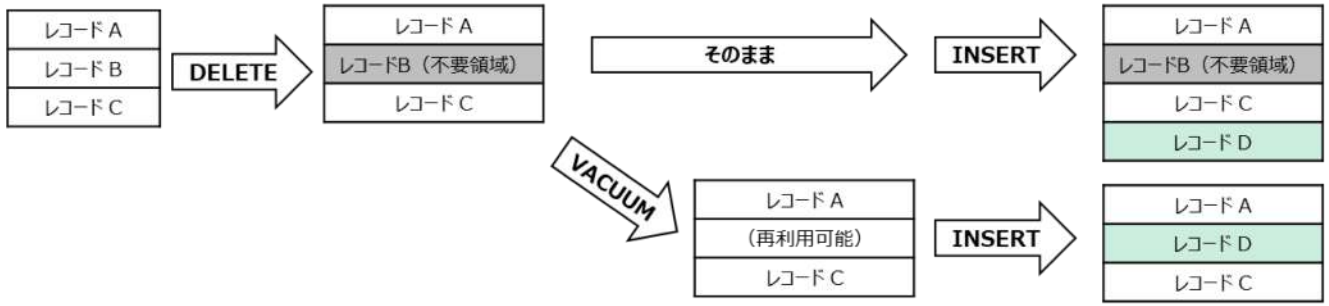
- 不要領域（dead tuple）は、そのままでは再利用できない。
  - DBのメンテナンス処理（VACUUM）によって、不要領域の回収を行う必要がある。
  - VACUUMを適切に実施しないとデータ量が拡大し、テーブル肥大化等の原因となる。



VACUUMによる不要領域の回収

■VACUUM：データベースの不要領域を回収する

- 追記型アーキテクチャを採用しており、削除された領域はVACUUM処理で回収するまで再利用することができない。



- 実行コマンド
  - VACUUM [テーブル名];
    - テーブル名を省略した場合は接続中のDB全体にVACUUMを実行する

```
postgres=# VACUUM pgbench_accounts;
VACUUM
```

  - vacuumdb
    - 接続対象のDB全体にVACUUMを実行する

```
-bash-4.2$ vacuumdb
```

VACUUMによる不要領域の回収

VACUUM：データベースの不要領域を回収する

- 設計のポイント
  - 不要領域を適正なタイミングで実施することで、DBを健全な状態を保つことができる。
    - 過剰なVACUUMの実施は、マシンリソース（CPU、メモリ等）の浪費につながる。
    - VACUUMが不足していると、テーブルの肥大化が発生するので注意する。

- 現在存在する不要領域を確認する方法
  - n_dead_tupとしてシステムカタログから確認することができる。

```
postgres=# postgres=# select n_dead_tup from pg_stat_user_tables where relname = 'pgbench_accounts';
 n_dead_tup
------------
       3850
(1 row)
```

3850レコードが不要領域として存在

FREEZE：タプルを凍結しXIDを再利用する

- PostgreSQLはトランザクション処理をトランザクションID（XID）で管理しており、32bitの数値（約40億）を再利用しながら使用している。
  - XIDの回収ができずに、使い切ってしまうとDBは最悪停止してしまう。
  - XIDはDBクラスタ単位で管理されている。

```
postgres=# SELECT datname,age(datfrozenxid) FROM pg_database ;
  datname  |  age
-----------+-------
 postgres  | 18713
 template1 | 18713
 template0 | 18713
(3 rows)
postgres=# VACUUM FREEZE;
VACUUM
postgres=# SELECT datname,age(datfrozenxid) FROM pg_database ;
  datname  |  age
-----------+-------
 postgres  |     0
 template1 | 18713
 template0 | 18713
(3 rows)
```

age()で、XIDがどれだけ消費されているか確認できる。

VACUUM FREEZEによって、XIDの回収をおこなったことでカウントが「0」に戻った。

FREEZE（続き）

- 設計のポイント
  - FREEZE処理は、VACUUM FREEZEで明示的に実行することもできるが、VACUUM処理を実行した際、裏でFREEZE処理は行われている。
    - VACUUM処理の中でどれくらいをFREEZEするかはチューニング要素となる。
  - VACUUM処理が十分に実行できていれば、FREEZEを厳密に設計する必要はない。

```
postgres=# SELECT datname,age(datfrozenxid) FROM pg_database WHERE datname = 'postgres';
 datname  |    age
----------+------------
 postgres | 1610593462
(1 row)
postgres=# VACUUM;
VACUUM
postgres=# SELECT datname,age(datfrozenxid) FROM pg_database WHERE datname = 'postgres';
 datname  |   age
----------+----------
 postgres | 50000000
(1 row)
```

XIDが消費された状態

テーブルを指定しない場合、DB全体にVACUUMが行われる

5000万XID分を残して、FREEZEされた。

ANALYZEによる統計情報の更新

ANALYZE：データベース内のテーブルの内容に関する統計情報を収集する

- PostgreSQLはSQL問い合わせに対して、統計情報（データ分布等）を利用して、SQLを実行するためのベストな手順（実行計画）を作成するため、適切な処理を行うには実データと一致した統計情報を保つ必要がある。
  - 実行計画は以下のようにEXPLAIN文で確認することができる。

```
postgres=# EXPLAIN SELECT pc.relname, pl.mode FROM pg_locks pl ,pg_class pc
            WHERE pl.relation = pc.oid AND locktype = 'relation';
                                  QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=12.56..34.74 rows=5 width=96)
   Hash Cond: (pc.oid = l.relation)
   ->  Seq Scan on pg_class pc  (cost=0.00..20.18 rows=518 width=68)
   ->  Hash  (cost=12.50..12.50 rows=5 width=36)
         ->  Function Scan on pg_lock_status l  (cost=0.00..12.50 rows=5 width=36)
               Filter: (locktype = 'relation'::text)
(6 rows)
```

JOINの方法、スキャンの方法等、SQLを実行した際にどのように処理が行われるか出力できる

- 実行コマンド
  - ANALYZE [テーブル名];
    - テーブル名を省略した場合は接続しているDB全体に実行する。

```
postgres=# VACUUM pgbench_accounts;
VACUUM
```

ANALYZEによる統計情報の更新

■ANALYZE：データベース内のテーブルの内容に関する統計情報を収集する

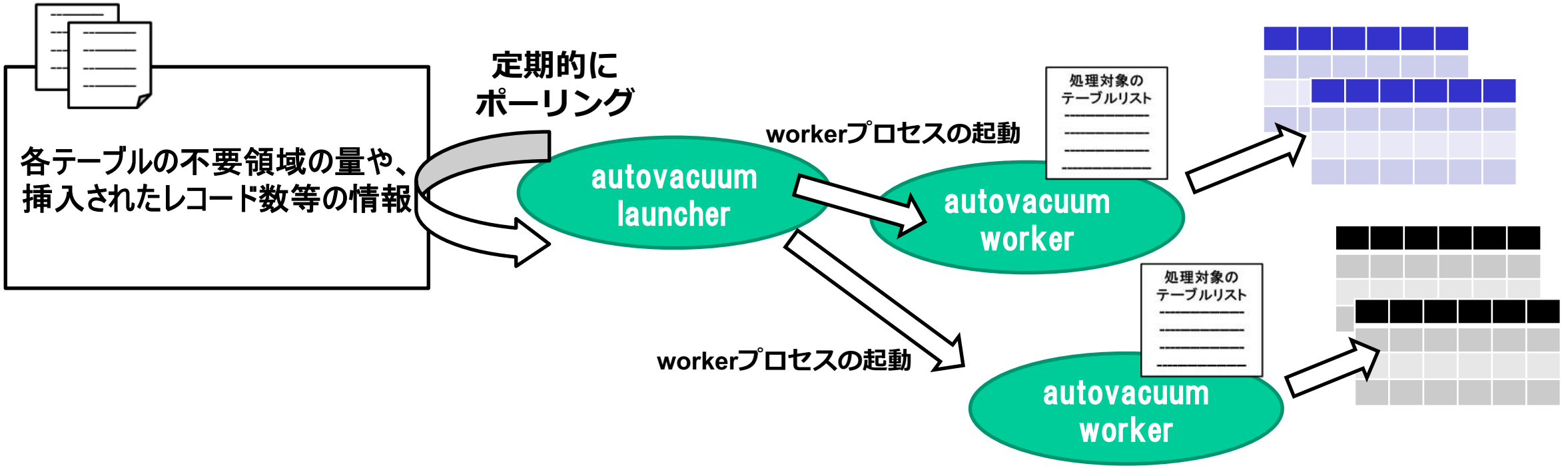
- 設計のポイント
  - 実際の格納された内容と統計情報が一致しない状況が発生する可能性があり、そういった状況においては不適切な実行計画が採用され、性能問題等の原因になる可能性がある。
  - バックアップからリストアした直後は統計情報がクリアされるため、統計情報の取得が必要。
  - データ分布が変わるような大量のデータ操作を行った後には、統計情報の更新が必要。
  - 中間データを格納するようなテーブルの内容を頻繁に入れ替えるようなテーブルについても、統計情報と実データが一致しない可能性があり、適宜統計情報を最新化する必要がある。

autovacuum機能による自動メンテナンスの実行

■メンテナンス処理は、適切なタイミングで実行する必要がある。

- PostgreSQLでは、メンテナンス処理（VACUUM、ANALYZE、FREEZE）を自動で行うautovacuum機能（自動バキューム機能）が備わっている。

■autovacuumの動作イメージ



autovacuum機能による自動メンテナンスの実行

設計のポイント OSS-DB /Gold

- autovacuumによるメンテナンス処理の閾値
  - 各テーブルごとに閾値を決めることができ、INSERT、UPDATE、DELETEされた量が閾値を超えた場合に起動される。
- autovacuumによるVACUUMの処理負荷の制御
  - 不要領域を【全力で回収】するか　or　IO負荷軽減のため、【緩やかに回収】するか
- DBのメンテナンス処理を阻害する主な原因
  - ロングトランザクション（トランザクションを開始した状態が長時間続いている処理）
    - 誰かがアクセスする可能性があるデータはVACUUM処理やFREEZE処理を実施できない。
    - 放置すると最終的にはDBが停止する可能性がある。

サンプル問題（DBのメンテナンス）

■OSS-DB Silver サンプル問題/例題解説 から本テーマに関するものを紹介します。

- 2分後に解説を行うので、ぜひ回答を考えてみてください。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

サンプル問題 1.109

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。

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。

引用元

- https://oss-db.jp/sample/silver_management_06/109_201029

■お問い合わせ■

NTTテクノクロス株式会社　ソフト道場

https://www.ntt-tx.co.jp/products/soft_dojyo/