psql 、私の好きなツールです

Open Source Conference Kyoto 2022 (2022-07-30)

自己紹介

- ぬこ@横浜 , @nuko_yokohama
- にゃーん
- 趣味でポスグレをやってる者だ
- 今日は JPUG の人として来ました
- psql 、私の好きな言葉です

目次

40 分では
psql 全機能の説明は
無理なので、
ピックアップして
紹介します

psql とは何か？

psql とは何か？

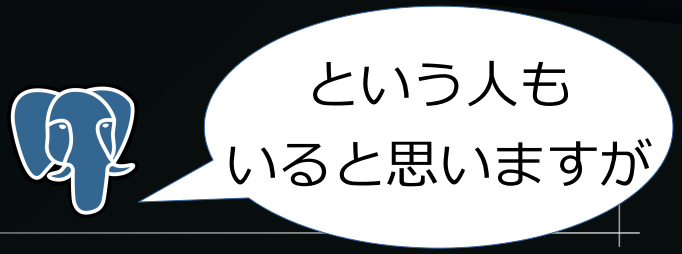
- PostgreSQL をコマンドラインから扱うユーティリティ
  - psql - PostgreSQL interactive terminal （PostgreSQL Document での説明文）
  - ログインして SQL コマンドを実行する。
  - バッチ処理用のファイルを入力して処理も可能

```
$ psql -p 10014 postgres
psql (14.3)
Type "help" for help.

postgres=# SELECT 'Welcome to psql world!';
        ?column?
------------------------
 Welcome to psql world!
(1 row)

postgres=# ¥q
$
```

実は公式文書では
1300 行を超える
ボリュームの
ユーティリティです

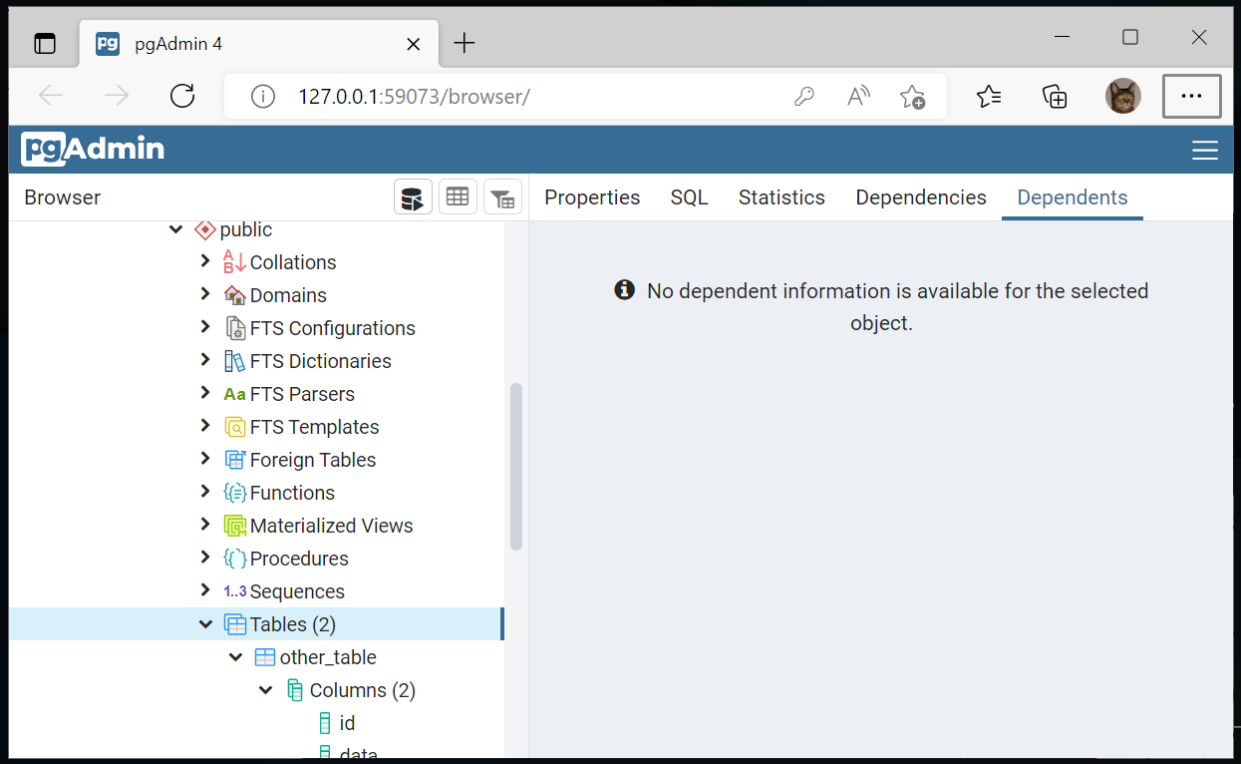
いまどき psql ？

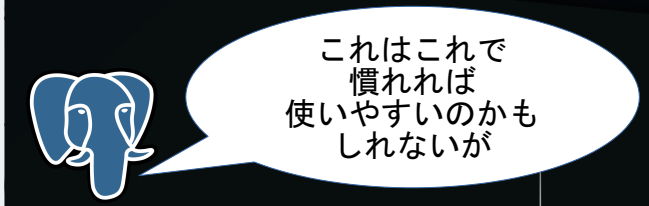
- いまどきコマンドラインインタフェース？
- 地味・・・
- 黒い (白い) 画面こわい
- pgAdmin4 みたいな GUI クライアントあるじゃん？



いまどき psql ？

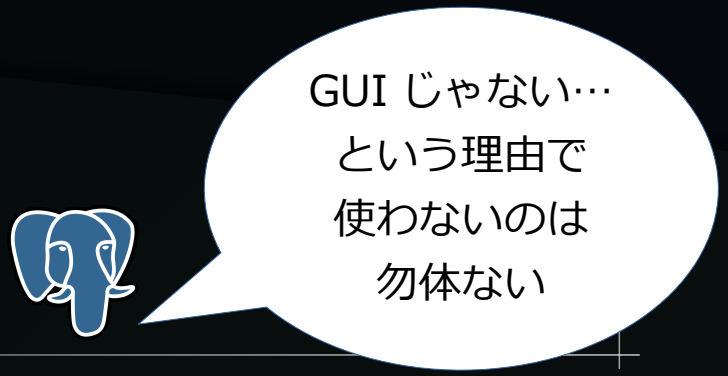
- 参考： pgAdmin4 の画面





psql の推しポイント

- PostgreSQL 標準機能として提供されている
- 軽い！
- システムカタログ参照機能
- タブ補完機能
- psql 独自機能
- シェルスクリプトとの相性も良い
- PostgreSQL 本体とともに進化



psql はいいぞ

- psql は地味だけど使い慣れると便利
- psql はいいぞ、が今日のテーマです。

```
$ psql testdb -c "SELECT * FROM pg_banner('psql is good')"
                                        data
-----------------------------------------------------------------------------------
 ######   #####   #####  #         ###  #####     #####  ####### ####### ######
 #     # #     # #     # #          #  #    #    #     # #     # #     # #     #
 #     # #       #     # #          #  #         #       #     # #     # #     #
 ######   #####  #     # #          #   #####    #  #### #     # #     # #     #
 #           # #   # # #          #       #    #     # #     # #     # #     #
 #       #     # #    #  #          #  #    #    #     # #     # #     # #     #
 #        #####   #### # #######    ###  #####     #####  ####### ####### ######
(7 rows)

$
```

注： psql 独自機能ではなく、
https://github.com/nuko-yokohama/pg_scripts/blob/master/pg_banner.sql を使用しています。

導入編

psql のインストール

- yum インストール
  - インストール完了時点で使えるようになる使用可能。
- RPM インストール
  - postgresqlXX-XX.X-XPGDG.*.rpm インストール完了時点で使用可能。
- Windows 版インストーラ
  - インストール完了時点で使用可能。
  - Stack Builder は入れなくても OK
- ソースからビルドインストール
  - make && make install した時点で使用可能。
  - contrib の追加インストールは不要

psql は
PostgreSQL の
コア機能として
組み込まれている

基本編

psql の基本

- psql への接続
- SQL コマンドの実行
- バッチ処理
- 定義情報の確認
- タブ補完（ readline 機能）

psql への接続

- psql への接続方法
  - オプションによる指定
  - 接続文字列
- psql の接続オプション

```
psql is the PostgreSQL interactive terminal.

Usage:
  psql [OPTION]... [DBNAME [USERNAME]]
 (略)
Connection options:
  -h, --host=HOSTNAME      database server host or socket directory (default: "local socket")
  -p, --port=PORT          database server port (default: "10014")
  -U, --username=USERNAME  database user name (default: "postgres")
  -w, --no-password        never prompt for password
  -W, --password           force password prompt (should happen automatically)
```

psql への接続（オプションによる指定）

- オプションによる指定方法

項目	ショートオプション	ロングオプション	備考
ホスト名	-h < ホスト名リスト >	--host=< ホスト名リスト >	複数指定可 （後のスライド参照）
ポート番号	-p < ポート番号 >	--port=< ポート番号 >	
データベース名	（なし）	< データベース名 >	オプションではなく引数の最後に指定する。
ユーザ名	-U < ユーザ名 >	--username=< ユーザ名 >	
パスワード	-W < パスワード >	--password=< パスワード >	パスワード指定をしない場合には、 -w/--no-password オプションを指定する。

psql への接続（オプションによる指定）

- ホスト名の指定方法
  - ホスト名： TCP/IP 接続
  - ディレクトリ名： Unix ドメインソケット接続
  - この指定がない場合は、 Unix ドメインソケット接続
- ホスト名はカンマ区切りリストで複数指定可能
  - リスト先頭から接続を順に試行
  - リスト指定可能なのはホスト名のみ。

psql への接続（接続文字列）

- データベース名の代わりに、接続文字列を指定する。
  - 接続文字列の詳細→PostgreSQL 13文書 接続文字列
  - 接続文字列で詳細な接続時のオプションを指定可能
- 接続文字列による接続の例

```
$ psql "port=10014 dbname=test user=postgres target_session_attrs=read-write"
```

```
$ psql "postgresql://postgres@:10014/testdb"
```

psql への接続（接続の確認）

- 接続方式は、 ¥conninfo コマンドで確認

```
$ psql -U postgres testdb
psql (14.3)
Type "help" for help.

testdb=# ¥conninfo
You are connected to database "testdb" as user "postgres" via socket in "/tmp" at port "10014".
testdb=#
```

```
$ psql -h localhost -U postgres testdb
psql (14.3)
Type "help" for help.

testdb=# ¥conninfo
You are connected to database "testdb" as user "postgres" on host "localhost" (address "127.0.0.1") at port "10014".
testdb=#
```

SQL コマンドの実行

- コマンド実行の基本
- プロンプト七変化
- 出力結果の書式
- コマンド実行結果のファイル出力

コマンド実行の基本（入力）

- プロンプトが出ている状態で任意の SQL コマンドを入力
- セミコロン（ ; ）が SQL コマンドの終端
  - psql コマンド（ ¥d 等）はセミコロン不要なので注意
- SQL コマンド終端まで入力したら、リターンキーで実行

```
testdb=# SELECT * FROM foo WHERE id = 1;
 id | data
----+------
  1 | ABC
(1 row)
```

コマンド実行の基本（入力）

- SQL コマンドの途中で改行しても OK
  - その場合、プロンプトが変化する

```
testdb=# SELECT *
testdb-# FROM foo
testdb-# WHERE id = 1
testdb-# ;
 id | data
----+------
  1 | ABC
(1 row)
```

プロンプト七変化

- psql のプロンプトは大別すると 3 種類ある。
- 各種類のプロンプトの表示書式は psql 変数で設定可能

psql 変数	意味	psql 変数のデフォルト値
PROMPT1	SQL コマンド入力待ちの状態 ログイン直後はこの状態になる	’%/%R%x%#’
PROMPT2	SQL コマンド入力途中で改行された状態	’%/%R%x%#’
PROMPT3	COPY FROM STDIN 実行中 値の入力待ち状態	’>>’

プロンプト七変化（ % 文字の意味）

- デフォルト値として設定されている % 文字の意味

% 文字	意味
%/	接続中のデータベース名
%R （PROMPT1）	入力中の状態。詳細は後のスライド参照
%R （PROMPT2）	入力中の状態。詳細は後のスライド参照
%x	トランザクションの状態。詳細は後のスライド参照
%#	ユーザの属性の状態。 # ならスーパーユーザ、そうでなければ >

プロンプト七変化（ %R 文字の意味）

- PROMPT1 での %R 文字の意味

文字	意味
=	通常の状態
@	条件ブロックで使用されない箇所
^	シングル行モード
!	データベースとの接続が切れた状態

プロンプト七変化（ %R 文字の意味）

- PROMPT2 での %R 文字の意味

文字	意味
-	継続行
*	/* ... */ コメントの途中
'	単一引用符で引用中
"	二重引用符で引用中
$	ドル引用符で引用注
(	小括弧が閉じていない

プロンプト七変化（ %x 文字の意味）

- %x プロンプトの表示の意味

表示文字	意味
（空白）	トランザクションブロック外
*	トランザクションブロック内
!	失敗したトランザクション内
?	トランザクション状態不明（未接続状態など）

プロンプト七変化（変化例）

- プロンプトの変化例

```
testdb=# SET ROLE user_a;  -- 一般ロールにスイッチ
SET
testdb=> BEGIN;            -- トランザクション開始
BEGIN
testdb=*> SELECT 'hoge'    -- SQL コマンドの継続
testdb-*> ,/* 単一引用符が完了していない */ 'huga
testdb'*> ',/* ドル引用符が完了していない */ $quote$
testdb$*> hige$quote$      -- SQL コマンドの継続
testdb-*> ;                -- これで SQL コマンドは完了 ただしトランザクション内。
 ?column? | ?column? | ?column?
----------+----------+----------
 hoge     | huga    +|         +
          |          | hige
(1 row)

testdb=*> SELECT error;     -- トランザクション全体をエラーにする
ERROR:  column "error" does not exist
LINE 1: SELECT error;
               ^
testdb=!> ROLLBACK;         -- トランザクションを終了する
ROLLBACK
testdb=> RESET ROLE;        -- 特権ロールに戻る
RESET
testdb=#
```

プロンプト七変化（カスタマイズ）

- プロンプト自体は psql 変数（ PROMPT1 ～ PROMPT3 ）を変更してカスタマイズ可能。
- プロンプトに設定可能な特殊文字の一覧
  - 特殊文字は 10 種類以上

    https://www.postgresql.jp/document/13/html/app-psql.html#APP-PSQL-PROMPTING
- 通常はカスタマイズ不要。

出力結果の書式

- 出力形式
- aligned/unaligned
- tuples-only
- フィールドセパレータ

出力結果の書式

- psql の出力形式は行 / 列 ×TEXT/HTML の 4 パターン

種別	コマンド	種別	意味
行 / 列	¥x	通常形式	ヘッダ行には列名、 データ行に全列の情報が出力される
		拡張テーブル形式	1 行に列名とその列のデータが出力される
TEXT/ HTML	¥html	TEXT/HTML	TEXT 形式で出力される
		TEXT/HTML	HTML の TABLE タグソースが出力される

出力結果の書式（通常形式）

- 通常形式

```
testdb=# SELECT * FROM foo;
 id | data
----+------
  1 | ABC
  2 |
  3 |
(3 rows)
```

```
postgres=# SELECT * FROM pg_database WHERE datname = 'testdb';
  oid  | datname | datdba | encoding | datcollate | datctype | datistemplate | datallowconn |
datconnlimit | datlastsysoid |
datfrozenxid | datminmxid | dattablespace | datacl
-------+---------+--------+----------+------------+----------+---------------+--------------
+--------------+---------------+-
-------------+------------+---------------+--------
 17682 | testdb  |     10 |        6 | C          | C        | f             | t            |          -
1 |         13948 |
         726 |          1 |          1663 |
(1 row)
```

列数が多かったり
値が長いと見づらい

出力結果の書式（拡張テーブル形式）

- 拡張テーブル形式

```
testdb=# ¥x
Expanded display is on.
testdb=# SELECT * FROM pg_database WHERE datname = 'testdb';
-[ RECORD 1 ]-+-------
oid           | 17682
datname       | testdb
datdba        | 10
encoding      | 6
datcollate    | C
datctype      | C
datistemplate | f
datallowconn  | t
datconnlimit  | -1
datlastsysoid | 13948
datfrozenxid  | 726
datminmxid    | 1
dattablespace | 1663
datacl        |

testdb=#
```

列数が多かったり
値が長いときに便利

出力結果の書式（HTML 形式）

- HTML テーブル形式＆通常形式

<table> タグを
使った HTML 表の
ソースを出力

```
testdb=# ¥html
Output format is html.
testdb=# SELECT * FROM pg_database WHERE datname = 'testdb';
<table border="1">
  <tr>
    <th align="center">oid</th>
    <th align="center">datname</th>
    <th align="center">datdba</th>
    <th align="center">encoding</th>
    <th align="center">datcollate</th>
 (略)
    <th align="center">datlastsysoid</th>
    <th align="center">datfrozenxid</th>
    <th align="center">datminmxid</th>
    <th align="center">dattablespace</th>
    <th align="center">datacl</th>
  <tr>
 (略)
</table>
<p>(1 row)<br />
</p>
testdb=#
```

oid	datname	datdba	encoding	datcollate	datctype	datistemplate	datallowconn	datco
17682	testdb	10	6	C	C	f	t	

(1 row)

出力結果の書式（HTML 形式）

- HTML テーブル形式＆拡張テーブル形式

```
testdb=# ¥x
Expanded display is on.
testdb=# ¥html
Output format is html.
testdb=# SELECT * FROM pg_database WHERE datname = 'testdb';
<table border="1">

  <tr><td colspan="2" align="center">Record 1</td></tr>
  <tr valign="top">
    <th>oid</th>
    <td align="right">17682</td>
  </tr>
 (略)
  <tr valign="top">
    <th>datacl</th>
    <td align="left">  </td>
  </tr>
</table>

testdb=#
```

Record 1	
oid	17682
datname	testdb
datdba	10
encoding	6
datcollate	C
datctype	C
datistemplate	f
datallowconn	t
datconnlimit	-1
datlastsysoid	13948
datfrozenxid	726
datminmxid	1
dattablespace	1663
datacl	

拡張テーブル形式の
HTML 表の
ソースを出力

aligned/unaligned

- aligned
  - その列の一番長い表示長に合わせて全行揃える。
  - 人間が見やすい形式。
  - psql のデフォルトの出力形式は aligned 。
- unaligned
  - 長さを合わせずにそのまま表示する。
  - 出力結果を機械的に処理するのに向いた形式。
- ¥a コマンド： aligned⇔unaligned

aligned/unaligned

- aligned

```
testdb=# TABLE foo;
   id    |  data
---------+---------
       1 | abcdefg
   10002 | xyz
 1000003 |
(3 rows)
```

- unaligned

```
testdb=# ¥a
Output format is unaligned.
testdb=# TABLE foo;
id|data
1|abcdefg
10002|xyz
1000003|
(3 rows)
testdb=#
```

unaligned は
レコードセパレータと
値の間に何も入らない

tuples-only

- デフォルト
  - 列ヘッダ行、データ行、行数フッタ行
- tuples-only モード
  - データ行のみ
  - 【注】拡張テーブル形式では無効。
- ¥t コマンド：デフォルト⇔ tuples-only モード

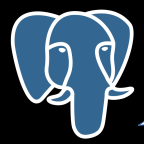
tuples-only

- デフォルト

```
testdb=# TABLE foo;
   id    |  data
---------+---------
       1 | abcdefg
   10002 | xyz
 1000003 |
(3 rows)

testdb=#
```

- tuples-only モード

```
testdb=# ¥t
Tuples only is on.
testdb=# TABLE foo;
       1 | abcdefg
   10002 | xyz
 1000003 |

testdb=#
```



データだけ
出力したいときに有用

フィールドセパレータ

- フィールドセパレータのデフォルトは縦棒（ | ）

```
testdb=# ¥a
Output format is unaligned.
testdb=# TABLE foo ;
id|data|ts
1|ABC|2022-06-09 12:00:00
2|XYZ|2022-06-17 06:17:42.35611
(2 rows)
```

- psql 変数 fieldsep で任意のフィールドセパレータに変更可能

フィールドセパレータ

- psql 変数 fieldsep の設定によるフィールドセパレータの変更例
  - 【注】 aligned モードの場合には、この指定は無視される

```
Output format is unaligned.
testdb=# ¥pset fieldsep
Field separator is "|".
testdb=# ¥pset fieldsep ,
Field separator is ",".
testdb=# TABLE foo ;
id,data,ts
1,ABC,2022-06-09 12:00:00
2,XYZ,2022-06-17 06:17:42.35611
(2 rows)
testdb=#
```

バッチ処理

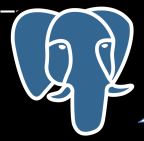
- 実行コマンド指定オプション
- SQL ファイル指定
- psql 実行結果の判定
- 結果のファイル出力

実行コマンド指定オプション

- psql のパラメータオプション -c < コマンド文字列 >
  - コマンドを実行して psql を終了する。
  - SQL コマンドではなく psql コマンド（ ¥d など）も実行可能。

```
$ psql testdb -c "TABLE foo"
 id | data |            ts
----+------+----------------------------
  1 | ABC  | 2022-06-09 12:00:00
  2 | XYZ  | 2022-06-17 06:17:42.35611
(2 rows)

$
```

```
$ psql testdb -c "¥d foo"
                          Table "public.foo"
 Column |            Type             | Collation | Nullable | Default
--------+-----------------------------+-----------+----------+--------
 id     | integer                     |           |          |
 data   | text                        |           |          |
 ts     | timestamp without time zone |           |          |

$
```



シェルからコマンドだけ
実行したい場合に使う

実行コマンド指定オプション

- -c オプションにはセミコロン（;）区切りで複数の SQL コマンドを渡すことも可能
- 複数の SQL コマンドを渡した場合
  - 途中のコマンドも実行される。
  - 出力されるのは、最後の SQL コマンドの結果のみ。

```
$ psql testdb -c "TABLE foo;INSERT INTO foo VALUES (3,'にゃーん');TABLE foo;"
 id |   data
----+----------
  1 | ABC
  2 | XYZ
  3 | にゃーん
(3 rows)

$
```

SQL ファイル指定

- 複数の SQL コマンドや、 psql コマンドをファイルに保存し、そのファイルを psql に与えてバッチ的な処理も可能。
- -f < ファイル名 > で指定
- 明示的に BEGIN, COMMIT を入れない場合は、自動コミットモードになる。（デフォルト時）
- -f < ファイル名 > 指定は複数指定可能。
  - コマンドラインの記述順に処理される。
- -e, -a オプションにより出力される内容が異なる。

SQL ファイル指定

- SQL ファイルの内容

```
$ cat test-1.sql
-- SQL ファイルのテスト
TRUNCATE foo;
--  テーブル定義の確認
¥d foo
--  挿入後に参照
INSERT INTO foo VALUES (1, 'ABC', '2022-06-30 12:30:00'), (2, 'XYZ', NULL);
TABLE foo;
$
```

SQL ファイル指定

- SQL ファイルによる実行例

```
$ psql -f test-1.sql testdb
TRUNCATE TABLE
                          Table "public.foo"
 Column |            Type             | Collation | Nullable | Default
--------+-----------------------------+-----------+----------+---------
 id     | integer                     |           |          |
 data   | text                        |           |          |
 ts     | timestamp without time zone |           |          |

INSERT 0 2
 id | data |         ts
----+------+---------------------
  1 | ABC  | 2022-06-30 12:30:00
  2 | XYZ  |
(2 rows)

$
```

デフォルトでは
実行結果のみが
出力される

SQL ファイル指定

- SQL ファイルによる実行例（ -e オプションつき）

```
$ psql -e -f test-1.sql testdb
TRUNCATE foo;
TRUNCATE TABLE
                                Table "public.foo"
 Column |            Type             | Collation | Nullable | Default
--------+-----------------------------+-----------+----------+---------
 id     | integer                     |           |          |
 data   | text                        |           |          |
 ts     | timestamp without time zone |           |          |

INSERT INTO foo VALUES (1, 'ABC', '2022-06-30 12:30:00'), (2, 'XYZ', NULL);
INSERT 0 2
TABLE foo;
 id | data |         ts
----+------+---------------------
  1 | ABC  | 2022-06-30 12:30:00
  2 | XYZ  |
(2 rows)

$
```

SQL コマンドも
出力される

SQL ファイル指定

- SQL ファイルによる実行例（ -a オプションつき）

```
$ psql -a -f test-1.sql testdb
-- SQL ファイルのテスト
TRUNCATE foo;
TRUNCATE TABLE
--  テーブル定義の確認
¥d foo
                              Table "public.foo"
 Column |            Type             | Collation | Nullable | Default
--------+-----------------------------+-----------+----------+---------
 id     | integer                     |           |          |
 data   | text                        |           |          |
 ts     | timestamp without time zone |           |          |

--  挿入後に参照
INSERT INTO foo VALUES (1, 'ABC', '2022-06-30 12:30:00'), (2, 'XYZ', NULL);
INSERT 0 2
TABLE foo;
 id | data |         ts
----+------+---------------------
  1 | ABC  | 2022-06-30 12:30:00
  2 | XYZ  |
(2 rows)

$
```

SQL コマンド、
psql コマンド、
コメント文も
出力される

SQL ファイル指定（複数ファイル指定）

- SQL ファイルの内容

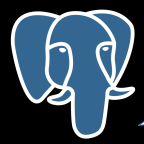
```
$ cat test-1.sql
-- SQL ファイルのテスト
TRUNCATE foo;
--  テーブル定義の確認
¥d foo
--  挿入後に参照
INSERT INTO foo VALUES (1, 'ABC', '2022-06-30 12:30:00'), (2, 'XYZ', NULL);
TABLE foo;
$
```

```
$ cat test-2.sql
-- SQL ファイルのテスト
--  更新・削除後に参照
UPDATE foo SET data = 'abc', ts = now() WHERE id = 1;
DELETE FROM foo WHERE id = 2;
TABLE foo;
$
```

SQL ファイル指定（複数ファイル指定）

- 複数の SQL ファイルによる実行例

```
$ psql -f test-1.sql -f test-2.sql testdb
TRUNCATE TABLE
                                Table "public.foo"
 Column |            Type             | Collation | Nullable | Default
--------+-----------------------------+-----------+----------+---------
 id     | integer                     |           |          |
 data   | text                        |           |          |
 ts     | timestamp without time zone |           |          |

INSERT 0 2
 id | data |         ts
----+------+---------------------
  1 | ABC  | 2022-06-30 12:30:00
  2 | XYZ  |
(2 rows)

UPDATE 1
DELETE 1
 id | data |             ts
----+------+----------------------------
  1 | abc  | 2022-06-18 19:08:57.020594
(1 row)

$
```



test-1.sql の後に
test-2.sql が実行される

psql の実行結果判定

- psql コマンドの終了ステータス

値	意味
0	正常終了
1	FATAL エラー ・メモリ不足 ・指定ファイルなし
2	実行中にサーバ接続断
3	スクリプト内でエラー発生（ ON_ERROR_STOP 変数設定時）

psql の実行結果判定

- スクリプトファイル内容

```
$ cat return-status.sql
¥set ON_ERROR_STOP

SELECT * FROM my_schema.baz;
$
```

- 正常終了時

```
$ psql testdb -e -f return-status.sql
SELECT * FROM my_schema.baz;
 id | data
----+------
(0 rows)

$ echo $?
0
$
```

psql の実行結果判定

- 異常終了時（指定ファイルなし）

```
$ psql testdb -e -f not_found.sql
not_found.sql: No such file or directory
$ echo $?
1
$
```

- 異常終了時（スクリプトファイル内エラー）

```
$ psql testdb -e -f return-status.sql
SELECT * FROM my_schema.baz;
psql:return-status.sql:3: ERROR:  relation "my_schema.baz" does not exist
LINE 1: SELECT * FROM my_schema.baz;
                      ^

$ echo $?
3
$
```

結果のファイル出力

- psql の処理結果をファイル出力可能。
  - ¥o < ファイル名 >
    - ¥o のみ指定すると標準出力に出力される。
  - < ファイル名 >
    - 絶対パス
    - psql 起動ディレクトリからの相対パス
- ファイル出力対象
  - 標準出力に出力されるものが対象
  - エラーメッセージは出力されない。

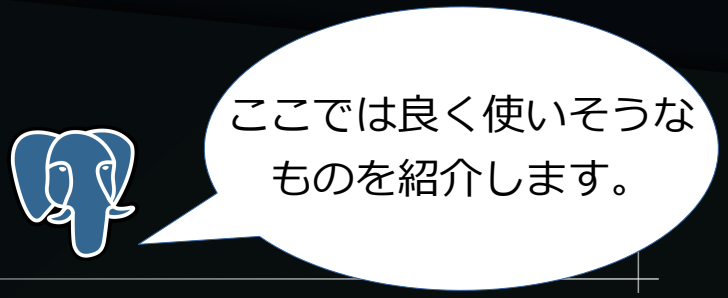
結果のファイル出力（実行例）

- ¥o < ファイル名 >
- 以降、 SQL コマンド結果がファイルに書き込まれる。

```
testdb=# ¥x
Expanded display is on.
testdb=# ¥o /tmp/psql-output.txt
testdb=# TABLE foo;
testdb=# ¥q
$ cat /tmp/psql-output.txt
-[ RECORD 1 ]--------------------
id   | 1
data | abc
ts   | 2022-06-18 19:08:57.020594

$
```

定義情報の確認

- 各種定義情報の確認用 psql コマンドが豊富。
  - 一覧は次スライド参照
- データベース一覧
- スキーマ一覧
- テーブル一覧
- テーブル情報の詳細
- 定義済み設定一覧



定義情報の確認（ PostgreSQL 14 ）

コマンド	概要	コマンド	概要
¥d[S+]	リレーション全般	¥dL[S+]	手続き言語
¥da[S]	集約関数	¥dn[S+]	スキーマ（名前空間）
¥dA[+],¥dAc[+],¥dAf[+], ¥dAo[+],¥dAp[+]	アクセスメソッド関連	¥do[S+]	演算子
¥db[+]	テーブル空間	¥dO[S+]	照合順序
¥dc[S+]	文字エンコード変換	¥dp	アクセス権限
¥dC[+]	型変換（キャスト）	¥dP[itn+]	パーティション
¥dd[S]	制約、オペレータ系、ルール、トリガー	¥drds	ユーザまたはデータベースの設定（ALTER 文による設定）
¥dD[S+]	ドメイン	¥dRp[+]	パブリッシャ
¥ddp	デフォルトアクセス権限	¥dRs[+]	サブスクライバ
¥dE[S+],¥di[S+],¥dm[S+], ¥ds[S+],¥dt[S+],¥dv[S+]	リレーションの種別毎の一覧	¥dT[S+]	データ型
¥des[+],¥det[+],¥deu[+],¥dew[+]	外部テーブル関連	¥du[S+]	データベースロール（ユーザ）
¥df[anptwS+]	関数	¥dx[+]	拡張機能
¥dF[+],¥dFd[+],¥dFd[+], ¥dFp[+],¥dFt[+]	全文検索関連	¥dX	拡張統計情報
¥dg[S+]	データベースロール（¥du と同じ）	¥dy[+]	イベントトリガ
¥dl	ラージオブジェクト	¥l	データベース

データベース一覧

- ¥l コマンド

```
testdb=# ¥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
 testdb    | postgres | UTF8     | C       | C     |
(4 rows)

testdb=#
```

testdb 以外の
データベースは
初期構築ずみ

スキーマ一覧

- ¥dn/¥dn+ コマンド

```
testdb=# ¥dn
   List of schemas
   Name     |  Owner
-----------+----------
 my_schema | postgres
 public    | postgres
(2 rows)

testdb=# ¥dn+
                              List of schemas
   Name     |  Owner   |  Access privileges   |      Description
-----------+----------+----------------------+------------------------
 my_schema | postgres | postgres=UC/postgres+|
           |          | user_a=U/postgres    |
 public    | postgres | postgres=UC/postgres+| standard public schema
           |          | =UC/postgres         |
(2 rows)

testdb=#
```

public スキーマは
データベース作成時に
デフォルト作成される

テーブル一覧

- ¥dt/¥dt+ コマンド（スキーマ指定）

```
testdb=# ¥dt my_schema.*
          List of relations
  Schema   | Name | Type  |  Owner
-----------+------+-------+----------
 my_schema | bar  | table | postgres
 my_schema | baz  | table | postgres
 my_schema | foo  | table | postgres
(3 rows)

testdb=# ¥dt+ my_schema.*
                                   List of relations
  Schema   | Name | Type  |  Owner   | Persistence | Access method |    Size    | Description
-----------+------+-------+----------+-------------+---------------+------------+-------------
 my_schema | bar  | table | postgres | permanent   | heap          | 0 bytes    |
 my_schema | baz  | table | postgres | unlogged    | heap          | 16 kB      |
 my_schema | foo  | table | postgres | permanent   | heap          | 8192 bytes |
(3 rows)

testdb=#
```

¥dt コマンドは
テーブルのみを
対象とする

テーブル詳細

- ¥d < テーブル名 >/¥d+ < テーブル名 > コマンド

```
testdb=# ¥d my_schema.foo
                Table "my_schema.foo"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 data   | text    |           |          |
Indexes:
    "foo_pkey" PRIMARY KEY, btree (id)

testdb=# ¥d+ my_schema.foo
                                        Table "my_schema.foo"
 Column |  Type   | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
--------+---------+-----------+----------+---------+----------+-------------+--------------+-------------
 id     | integer |           | not null |         | plain    |             |              |
 data   | text    |           |          |         | extended |             |              |
Indexes:
    "foo_pkey" PRIMARY KEY, btree (id)
Access method: heap

testdb=#
```

¥d+ では
詳細な列属性や、
表のアクセスメソッドが
表示される

定義済み設定一覧

- ¥drds コマンド

```
testdb=# ALTER ROLE postgres SET log_statement = 'all';
ALTER ROLE
testdb=# ALTER DATABASE testdb SET max_parallel_workers_per_gather = 0;
ALTER DATABASE
testdb=# SHOW log_statement;
 log_statement
---------------
 none
(1 row)

testdb=# SHOW max_parallel_workers_per_gather;
 max_parallel_workers_per_gather
---------------------------------
 2
(1 row)

testdb=# ¥drds
                    List of settings
   Role   | Database |             Settings
----------+----------+-------------------------------------
 postgres |          | log_statement=all
          | testdb   | max_parallel_workers_per_gather=0
(2 rows)

testdb=#
```

ALTER...SET で設定した
ロールやデータベースの
設定のリストを表示

ちょっと便利な機能編

覚えておくと便利な Tips

- タブ補完
- SQL の実行時間出力
- CSV 出力
- null の表示
- psql の設定ファイル

タブ補完

- psql はこのタブ補完が強力無比！
  - Powered by readline
- 補完対象
  - SQL コマンド / サブコマンド
  - オブジェクト名
  - OS のパス名

注意！
Windows 版
PostgreSQL の psql では
この機能は使えない！

タブ補完（実行例）

- SQL コマンド補完

```
testdb=# DROP
```

- 候補 SQL キーワード出力

```
testdb=# DROP
ACCESS METHOD        EXTENSION            OPERATOR             SCHEMA               TRANSFORM
AGGREGATE            FOREIGN DATA WRAPPER OWNED                SEQUENCE             TRIGGER
CAST                 FOREIGN TABLE        POLICY               SERVER               TYPE
COLLATION            FUNCTION             PROCEDURE            STATISTICS           USER
CONVERSION           GROUP                PUBLICATION          SUBSCRIPTION         USER MAPPING FOR
DATABASE             INDEX                ROLE                 TABLE                VIEW
DOMAIN               LANGUAGE             ROUTINE              TABLESPACE
EVENT TRIGGER        MATERIALIZED VIEW    RULE                 TEXT SEARCH
testdb=# DROP
```

DROP 可能な
オブジェクトって
こんなにあるのか

タブ補完（実行例）

- 前方一致補完の例

```
testdb=# DROP T
```

```
testdb=# DROP T
TABLE        TABLESPACE   TEXT SEARCH  TRANSFORM    TRIGGER      TYPE
testdb=# DROP T
```

- オブジェクト名補完の例

```
testdb=# DROP TABLE
baz                  my_schema.           pg_toast.
information_schema.  pg_catalog.          public.
testdb=# DROP TABLE
```

タブ補完（実行例）

- サブコマンド補完

```
testdb=# ALTER TABLE my_schema.baz
ADD                     DISABLE                 NO                      SET
ALTER                   DROP                    OWNER TO                VALIDATE CONSTRAINT
ATTACH PARTITION        ENABLE                  RENAME
CLUSTER ON              FORCE ROW LEVEL SECURITY REPLICA IDENTITY
DETACH PARTITION        INHERIT                 RESET
testdb=# ALTER TABLE my_schema.baz ALTER
COLUMN      CONSTRAINT  data        id
testdb=# ALTER TABLE my_schema.baz ALTER COLUMN
data  id
testdb=# ALTER TABLE my_schema.baz ALTER COLUMN data
ADD     DROP    RESET   RESTART  SET     TYPE
testdb=# ALTER TABLE my_schema.baz ALTER COLUMN data TYPE varchar(10);
ALTER TABLE
testdb=#
```

便利すぎて
SQL コマンド構文を
覚えなくなるという
弊害もあるw

タブ補完（実行例）

- OS 上のパス名補完

```
testdb=# ¥copy my_schema.baz FROM '/
.autorelabel  data/         home/         local/        opt/          run/          sys/          var/
bin/          dev/          lib/          media/        proc/         sbin/         tmp/
boot/         etc/          lib64/        mnt/          root/         srv/          usr/
testdb=# ¥copy my_schema.baz FROM '/tmp/
```

```
testdb=# ¥copy my_schema.baz FROM '/tmp/
baz.txt
 （略）
.XIM-unix/
testdb=# ¥copy my_schema.baz FROM '/tmp/b
```

```
testdb=# ¥copy my_schema.baz FROM '/tmp/baz.txt'
```

PostgreSQL 12 までは
タブ補完で引用符が
消えるバグがあったが
PostgreSQL 13 で解消

SQL の実行時間出力

- ¥timing コマンド
  - on ⇔ off
  - SQL コマンドの実行後に実行時間を出力 (ms 単位）。
  - EXPLAIN ANALYZE ： SELECT/DML
  - ¥timing ：全 SQL コマンド
    - psql コマンド（ ¥d 等）は対象外
- psql 上での実行時間≠サーバ内の実行時間
  - EXPLAIN ANALYZE はサーバ内の実行時間
  - SQL がエラーになった場合にも時間を表示する。

SQL の実行時間

- 実行例

```
testdb=# ¥timing
Timing is on.
testdb=# ¥timing
Timing is off.
testdb=# ¥timing
Timing is on.
testdb=# SELECT COUNT(*) FROM my_schema.foo;
  count
---------
 1000000
(1 row)

Time: 56.586 ms
testdb=# SELECT * FROM my_schema.foo;
   id    |                data
---------+----------------------------------
       1 | 487bb6ad9aa4324afee542c661ec9501
 (略)
Time: 257.563 ms
testdb=#
```

¥timing は psql へのデータ転送時間も含むため COUNT(*) より * のほうが時間がかかる

CSV 出力

- ¥copy で CSV ファイルに出力
- ¥copy (*query*) TO '*filename*' WITH (FORMAT CSV)
- 列値内のエスケープ等は ¥copy にお任せ

CSV 出力

- 実行例

```
testdb=# TABLE hoge ;
 id | data1 | data2 |         ts
----+-------+-------+---------------------
  1 | ABC   | XYZ   | 2022-06-21 12:00:00
  2 | a,b,c | X"Y"Z | 2022-06-22 00:00:00
  3 |       | X'Y'Z | 2022-06-23 12:00:00
(3 rows)

testdb=# ¥copy (SELECT * FROM hoge) TO '/tmp/foo.csv' WITH ( FORMAT CSV )
COPY 3
testdb=# ¥q
[ec2-user@ip-10-0-1-10 ~]$ cat /tmp/foo.csv
1,ABC,XYZ,2022-06-21 12:00:00
2,"a,b,c","X""Y""Z",2022-06-22 00:00:00
3,"",X'Y'Z,2022-06-23 12:00:00
[ec2-user@ip-10-0-1-10 ~]$
```

値内に、カンマや
二重引用符がある場合、
¥copy 機能が
引用＆エスケープする

null の表示

- デフォルトの設定だと null は空文字と同じように表示される。
- null の場合に表示する文字列を設定可能
  - ¥pset null *any_text*
  - 例： (null) を設定する。

```
testdb=# SELECT * FROM foo;
 id | data
----+------
  1 | ABC
  2 |
  3 |
(3 rows)

testdb=# ¥pset null (null)
Null display is "(null)".
testdb=# SELECT * FROM foo;
 id |  data
----+--------
  1 | ABC
  2 | (null)
  3 |
(3 rows)

testdb=#
```

psql の設定ファイル

- よく使う設定は、設定ファイルに書いておくと便利。
- 設定ファイルの場所
  - pg_config --sysconfdir ディレクトリ配下の psqlrc
  - 環境変数 PGSYSCONFDIR の指定
  - $HOME/psqlrc （ Windows の場合： %APPDATA%¥postgresql¥psqlrc.conf ）
  - psql 変数 psqlrc の設定
- 設定ファイル内容を読み込ませずに psql を起動も可能
  - -X or --no-psqlrc オプション

psql の設定ファイル

- 自分の環境の例
  - null 表示の設定
  - プロンプト（PROMPT1）設定

```
$ cat $HOME/.psqlrc
-- 自分のpsql設定
¥pset null (null)
¥set PROMPT1 '%[%033[1;34;40m%]%n@%/%R%[%033[0m%]%# '
$ psql testdb
Null display is "(null)".
psql (14.3)
Type "help" for help.

postgres@testdb=#
```

psql の面白機能編

psql の面白機能

- 直前コマンドの繰り返し実行
- シェルの実行
- クロス集計
- コマンドの自動生成＆実行
- psql 変数
- 条件分岐

直前コマンドの繰り返し実行

- 直前に実行した SQL コマンドを一定間隔で繰り返し実行
- ¥watch 実行間隔
  - 実行間隔は秒単位・小数指定可能
- pg_stat_progress_* を psql 上で監視するときに有用

直前コマンドの繰り返し実行

- pg_stat_progress_copy の監視例

```
postgres@bench# SELECT relid, command, type, bytes_processed, tuples_processed FROM pg_stat_progress_copy ;
 relid | command | type | bytes_processed | tuples_processed
-------+---------+------+-----------------+------------------
(0 rows)

postgres@bench# ¥watch 0.5
        Sun 26 Jun 2022 09:07:42 AM JST (every 0.5s)

 relid | command | type | bytes_processed | tuples_processed
-------+---------+------+-----------------+------------------
 17907 | COPY TO | FILE |       131372888 |          1360000
(1 row)

        Sun 26 Jun 2022 09:07:42 AM JST (every 0.5s)

 relid | command | type | bytes_processed | tuples_processed
-------+---------+------+-----------------+------------------
 17907 | COPY TO | FILE |       227205087 |          2337920
(1 row)
```

別ターミナルで
COPY ... TO ... を
実行している

シェルの実行

- psql 上でシェルコマンドを実行する
- ¥! [command]
  - commad がないときはサブシェルに制御が移る。サブシェル終了（ exit 実行等）後は psql に戻る。
  - command が指定されたときは、そのコマンドを実行する。
- サブシェル、コマンドの実行ステータスをとることはできない。
- サブシェルや実行されたコマンドはトランザクション管理外になる。

シェルの実行（簡単な実行例）

- 引数なしで ¥! コマンド実行→ date コマンドを実行

```
postgres@bench# ¥!
[ec2-user@ip-10-0-1-10 ~]$ date
Sun Jun 26 09:27:22 JST 2022
[ec2-user@ip-10-0-1-10 ~]$ exit
exit
postgres@bench#
```

- 引数に date コマンドをつけて ¥! コマンドを実行

```
postgres@bench# ¥! date
Sun Jun 26 09:29:11 JST 2022
postgres@bench#
```

シェルの実行（トランザクション管理外）

- トランザクション内で実行された !¥ コマンドはロールバックされない

```
postgres@testdb# TABLE foo;
 id | data
----+------
  1 | ABC
  2 | XYZ
(2 rows)

postgres@testdb# BEGIN;
BEGIN
postgres@testdb# UPDATE foo SET data = 'abc' WHERE id = 1;
UPDATE 1
postgres@testdb# ¥! psql testdb -c "UPDATE foo SET data = 'xyz' WHERE id = 2"
UPDATE 1
postgres@testdb# ROLLBACK;
ROLLBACK
postgres@testdb# TABLE foo;
 id | data
----+------
  1 | ABC
  2 | xyz
(2 rows)

postgres@testdb#
```

¥! の実行内容は
トランザクション管理外！

クロス集計

- 検索結果内２列を縦 / 横に指定して、クロス表形式で出力
  - 検索結果には３列以上必要
- ¥crosstabview [colV [colH [colD [sortcolH]]]]
  - colV ：縦方向に展開する列
  - colH ：横方向に展開する列
  - colD ：セル内に表示する列
  - sortcolH ：水平方向のヘッダをソートする列
- ２列の GROUP BY ＋集約関数結果の表示に有用
- ¥crosstabview 直前のクエリの終端にはセミコロンは必須ではない

クロス集計

- 2012 年～、横浜市内で食べたラーメンの区別の数

```
postgres@ramendb# SELECT s.area, r.year, count(*)
FROM shops s JOIN reviews_year r ON (s.sid = r.sid)
WHERE r.uid = 8999 AND pref = '神奈川県' AND area ~ '横浜市' AND year >= '2012' AND r.category = 'ラーメン'
GROUP BY area, year
ORDER BY year;
       area       | year | count
------------------+------+-------
 横浜市中区        | 2012 |    55
 横浜市保土ケ谷区  | 2012 |     3
 横浜市南区        | 2012 |     5
 (略)
 横浜市金沢区      | 2022 |     4
 横浜市青葉区      | 2022 |     1
 横浜市鶴見区      | 2022 |     3
(168 rows)

postgres@ramendb#
```

クロス集計

- 「2012年～、横浜市内で食べたラーメンの区別の数」のクロス集計

```
postgres@ramendb=# SELECT s.area, r.year, count(*)
FROM shops s JOIN reviews_year r ON (s.sid = r.sid)
WHERE r.uid = 8999 AND pref = '神奈川県' AND area ~ '横浜市' AND year >= '2012' AND r.category = 'ラーメン'
GROUP BY area, year
ORDER BY year
ramendb-# ¥crosstabview area year
```

area	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022
横浜市中区	55	90	143	46	59	81	90	36	69	30	23
横浜市保土ケ谷区	3	4	3	9	7	2	5	4	4	3	3
横浜市南区	5	7	11	10	16	19	26	10	11	8	3
横浜市戸塚区	4	2	2	1	2			1	1	6	2
横浜市旭区	7	1	5	4	6	1	4	2	4	4	2
横浜市栄区	3		1		1		1		1	2	1
横浜市泉区	1	1		1	1				2	6	1
横浜市港北区	7	6	1	6	2	3	7		4	2	6
横浜市港南区	5	1	4	4	1	7	5	5	3	4	3
横浜市瀬谷区	4	3			1	1	1		1	2	1
横浜市磯子区	3	6	1	1	1		11		8	3	2
横浜市神奈川区	21	20	12	7	6	7	10	1	5	8	4
横浜市緑区	5		1	2	1	1	1	1	1	3	1
横浜市西区	24	34	27	33	35	44	37	19	12	13	7
横浜市都筑区	1	3		2		1	1	1	2	4	3
横浜市金沢区	2	1			1				1	7	4
横浜市青葉区	1			1					1	3	1
横浜市鶴見区	7	3	5	7	21	8	4	3	4	6	3

(18 rows)

コマンドの自動生成＆実行

- 直前の SELECT 文結果を各行ごとに SQL として実行する。
- ¥gexec
- COPY 文や DDL を自動生成＆実行するときに有用
- ¥gexec　直前の SQL コマンドはセミコロン不要

コマンドの自動生成＆実行

- 5 個のハッシュパーティションを自動生成＆実行

```
postgres@testdb# SELECT 'CREATE TABLE my_part.child_' || generate_series(0, 4) || ' PARTITION OF my_part.parent
FOR VALUES WITH ( MODULUS 5, REMAINDER ' || generate_series(0, 4) || ');'
postgres@testdb# ¥gexec
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
postgres@testdb# ¥d+ my_schema.par

postgres@testdb# ¥d+ my_part.parent
                                    Partitioned table "my_part.parent"
  Column  |  Type   | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
----------+---------+-----------+----------+---------+----------+-------------+--------------+-------------
 hash_key | integer |           |          |         | plain    |             |              |
 data     | text    |           |          |         | extended |             |              |
Partition key: HASH (hash_key)
Partitions: my_part.child_0 FOR VALUES WITH (modulus 5, remainder 0),
            my_part.child_1 FOR VALUES WITH (modulus 5, remainder 1),
            my_part.child_2 FOR VALUES WITH (modulus 5, remainder 2),
            my_part.child_3 FOR VALUES WITH (modulus 5, remainder 3),
            my_part.child_4 FOR VALUES WITH (modulus 5, remainder 4)

postgres@testdb#
```

コマンドの自動生成＆実行

- 5 個のハッシュパーティションを自動生成＆実行（種明かし）

```
postgres@testdb# SELECT 'CREATE TABLE my_part.child_' || generate_series(0, 4) || ' PARTITION OF my_part.parent
FOR VALUES WITH ( MODULUS 5, REMAINDER ' || generate_series(0, 4) || ');';
                                                ?column?
-------------------------------------------------------------------------------------------------------
 CREATE TABLE my_part.child_0 PARTITION OF my_part.parent FOR VALUES WITH ( MODULUS 5, REMAINDER 0);
 CREATE TABLE my_part.child_1 PARTITION OF my_part.parent FOR VALUES WITH ( MODULUS 5, REMAINDER 1);
 CREATE TABLE my_part.child_2 PARTITION OF my_part.parent FOR VALUES WITH ( MODULUS 5, REMAINDER 2);
 CREATE TABLE my_part.child_3 PARTITION OF my_part.parent FOR VALUES WITH ( MODULUS 5, REMAINDER 3);
 CREATE TABLE my_part.child_4 PARTITION OF my_part.parent FOR VALUES WITH ( MODULUS 5, REMAINDER 4);
(5 rows)

postgres@testdb#
```

- このクエリ実行直後に ¥gexec を実行すると、上記の CREATE TABLE 文のテキストを順次実行する。

psql 変数

- psql 変数
  - psql 内で任意の変数を定義・参照できる。
- 設定時
  - ¥set [変数名 値]
  - 引数なしの ¥set は設定済みの変数名と設定値のリストを表示
- 参照時
  - : 変数名
  - 単一引用符つきの値で参照したい場合は特殊な記法になる

psql 変数（設定と参照例）

- psql 変数の設定と参照

```
postgres@testdb# ¥set foo hoge
postgres@testdb# ¥echo :foo
hoge
postgres@testdb#
postgres@testdb#
postgres@testdb# ¥set bar ’にゃーん’
postgres@testdb# ¥echo :bar
にゃーん
postgres@testdb# ¥echo :’bar’
’にゃーん’
postgres@testdb# ¥set baz ″ぱおーん″
postgres@testdb# ¥echo :baz
″ぱおーん″
postgres@testdb#
```

- 単一引用符つきで展開したい場合には、 :’変数名’ のような記法になる。

psql 変数（SQL 結果を設定）

- ¥gset コマンドで直前の SELECT 文の結果を、列名と同じ変数名に格納できる。

```
postgres@testdb# SELECT generate_series(1,1), 'にゃーん' AS voice;
 generate_series |  voice
-----------------+----------
               1 | にゃーん
(1 row)

postgres@testdb# ¥gset
postgres@testdb# ¥echo :voice
にゃーん
postgres@testdb#
```

- ¥gset は１行のみ返却する検索結果だけに対応する。２行以上はエラーになる。

```
postgres@testdb# SELECT generate_series(1,2), 'にゃーん' AS voice;
 generate_series |  voice
-----------------+----------
               1 | にゃーん
               2 | にゃーん
(2 rows)

postgres@testdb# ¥gset
more than one row returned for ¥gset
postgres@testdb#
```

条件分岐

- psql スクリプト内に条件分岐を書ける。
  - ¥if, ¥elif, ¥else, ¥endif
- 条件分岐はネスト可能

```
¥if expression
  ¥if expression
  -- 処理1-1
  ¥elsif expression
  -- 処理1-2
  ¥else
  -- 処理1-3
  ¥endif
¥else
-- 処理2
¥endif
```

条件分岐（簡単な例）

- 条件分岐用の psql 変数に boolean を設定する SQL を実行
- ¥gset で psql 変数に設定し、¥if で分岐する。

```
$ cat if-sample.sql
-- ¥if サンプル
WITH t AS (SELECT random() AS val)
SELECT (t.val > 0.8) AS very_good, (t.val > 0.4) AS good, val FROM t
¥gset
¥echo :val :very_good :good

¥if :good
  ¥if :very_good
    ¥echo 大吉
  ¥else
    ¥echo 吉
  ¥endif
¥else
  ¥echo 凶
¥endif

$
```

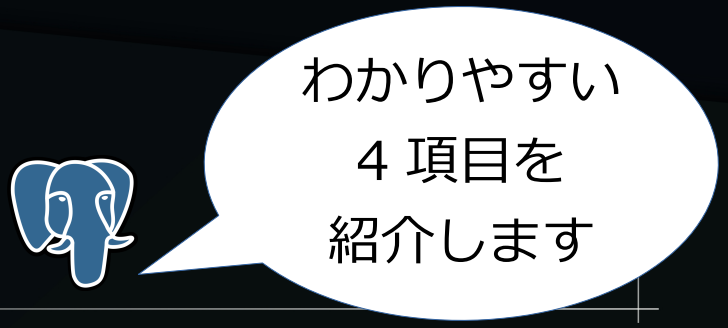
- 実行例

```
$ psql testdb -f if-sample.sql
0.8953974298938512 t t
大吉
$ psql testdb -f if-sample.sql
0.32283707228951997 f f
凶
$ psql testdb -f if-sample.sql
0.7752079435760777 f t
吉
$
```

PostgreSQL 15 追加予定の改善項目

PostgreSQL 15 の psql 改善項目（予定）

- psql の ¥copy コマンドの性能向上
- 環境変数設定
- ¥dconfig コマンド
- 旧バージョンのサーバ接続時に警告



psql の ¥copy コマンドの性能向上

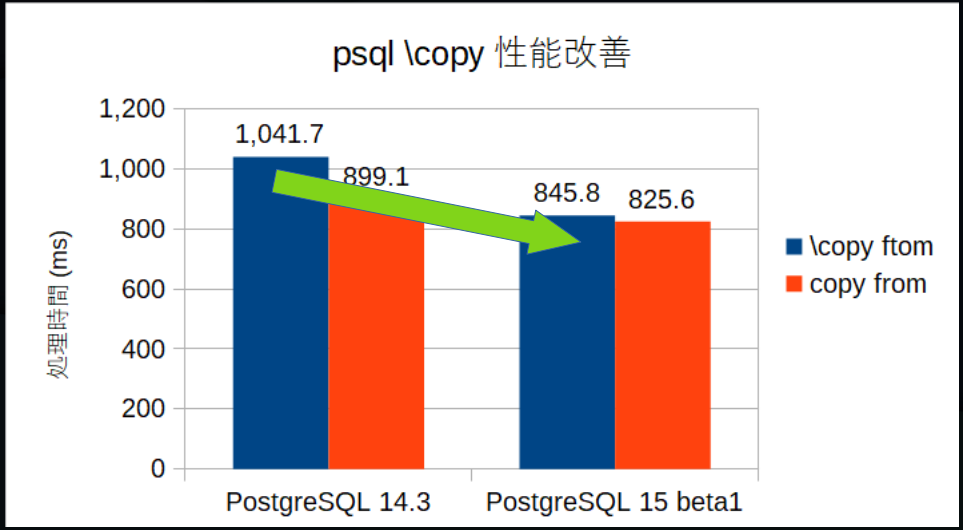
- ¥copy コマンド
  - テキストファイルをテーブルにバルクコピーする
  - サーバコマンド COPY を psql から使うコマンド
  - DBaaS （ AWS Aurora PostgreSQL 互換等）でも使える。
- PostgreSQL 15 では ¥copy がちょっとだけ高速になった

psql の ¥copy コマンドの性能向上

- 測定用テーブル定義

```
CREATE UNLOGGED TABLE test (id int, num_data numeric, txt_data text);
```

- このテーブルに 1000000 件を ¥copy/COPY でロードする時間を測定
- 19 %程度の性能向上効果



環境変数設定

- PostgreSQL 15 からは環境変数の内容を psql 変数として設定可能になった。
- ¥getenv <psql 変数 > < 環境変数 >
  - 指定した環境変数の値を指定した psql 変数に格納する。
- 用途
  - 環境変数にテーブル名をセットしてから、 ¥getenv を使ったスクリプトを実行
  - 条件分岐 (¥if 等) で評価する変数を環境変数から取得するようにして、環境変数設定値によって動作変更

環境変数設定

- 設定されている環境変数

```
$ export VOICE="にゃーん"
$ echo $VOICE
にゃーん
$
```

- 環境変数を psql 変数に設定する

```
test=# ¥getenv voice VOICE
test=# SELECT :'voice';
 ?column?
----------
 にゃーん
(1 row)

test=#
```

¥dconfig コマンド

- ¥dconfig はサーバ変数を表示する
- PostgreSQL コマンド SHOW の違い
  - SHOW ：パターンマッチ未対応
  - ¥dconfig ：パターンマッチ対応

¥dconfig コマンド

- 例： max_parallel_workers_per_gather

```
test=# SHOW max*para*gather;
ERROR:  syntax error at or near "*"
LINE 1: SHOW max*para*gather;
```

```
test=# ¥dconfig max*para*gather
    List of configuration parameters
            Parameter            | Value
--------------------------------+-------
 max_parallel_workers_per_gather | 2
(1 row)

test=#
```

パラメータ名を
覚えなくなるという
諸刃の剣

旧バージョンのサーバ接続時に警告

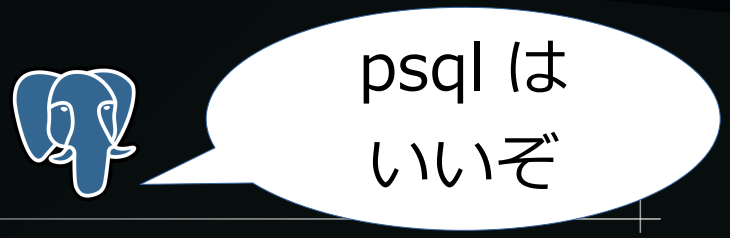
- PostgreSQL 9.1 以前のサーバに接続しようとすると警告メッセージが出力される。
  - ログイン自体は可能

```
$ ~/pgsql/pgsql-15b1/bin/psql -p 10091 postgres
psql (15beta1, server 9.1.24)
WARNING: psql major version 15, server major version 9.1.
         Some psql features might not work.
Type "help" for help.

postgres=# SELECT version();
                                                    version
-----------------------------------------------------------------------------------------------------------------
 PostgreSQL 9.1.24 on x86_64-unknown-linux-gnu, compiled by gcc (GCC) 7.3.1 20180712 (Red Hat 7.3.1-13), 64-bit
(1 row)
```

おわりに

改めて、 psql はいいぞ

- 手軽に使える。
- GUI ツールにはない機能が豊富。
- PostgreSQL の最新機能に追随。



参考情報

参考情報

- PostgreSQL 文書
  - https://www.postgresql.org/docs/
  - https://www.postgresql.jp/document/
- 篠田の虎の巻シリーズ
  - https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/lcc/pdf/PostgreSQL_15_Beta_1_New_Features_ja_20220524-1.pdf
- PostgreSQL 13 ～ PostgreSQL 15 までの psql 機能調査記事
  - https://qiita.com/nuko_yokohama/items/e253dd2619c639558a23
  - https://qiita.com/nuko_yokohama/items/919e657fbdd7794a8f87
  - https://qiita.com/nuko_yokohama/items/db02d5d5f1e0260b161a