

作ったツールの紹介

<https://github.com/noborus>

斉藤 登

PostgreSQLに関連した以下のツールを紹介します

- **trdsq**
テキストとデータのツール
- **ov**
ターミナルページャー
- **pgsp**
pg_stat_progress監視ツール
- **jpug-doc-tool**
マニュアル翻訳ツール

全部Goで書かれたCLIツールです。

trdsql

PostgreSQLユーザーにこそ使ってほしいtrdsql

<https://github.com/noborus/trdsql>

trdsq1とは？

CSV,LTSV,JSON等にSQLを実行できるツール

```
trdsq1 -ih "SELECT id, name, price FROM fruits.csv"
```

```
1, apple, 100  
2, orange, 50  
3, melon, 500
```

同様のツールはいくつか存在する `q`, `textql` ...

trdsq1はDBエンジンを変更できる！

trdsq1はPostgreSQLに接続可能

類似ツールの多くは内部でSQLite3を使用している。

trdsq1もSQLite3を（内蔵して）使用しているが、PostgreSQL,MySQLに変更可能。

`-driver` オプションと接続先を表す `-dsn` オプションで変更可能。

```
trdsq1 -driver postgres -dsn "host='/var/run/postgresql/'" "SELECT * FROM fruits.csv"
```

（設定ファイルにも書けます）。

PostgreSQLのSQL構文、SQL関数のでストレスフリー

PostgreSQLのインポート、エクスポート

trdsq1はファイルにSQLを実行するだけでなく、もっと使いみちがある。

ファイルのJOINだけでなく、ファイルとテーブルのJOINも可能。

テーブルだけに対しても実行可能で、SELECT以外のSQLを実行しても問題ありません。

そのため、インポート、エクスポートもできます。

エクスポート

既存のテーブルにSQLを実行。

```
trdsq1 -driver postgres -dsn "host='/var/run/postgresql/'" -omd "SELECT * FROM actor"
```

actor_id	first_name	last_name	last_update
1	Penelope	Guinness	2013-05-26T14:47:57Z
2	Nick	Wahlberg	2013-05-26T14:47:57Z
3	Ed	Chase	2013-05-26T14:47:57Z
4	Jennifer	Davis	2013-05-26T14:47:57Z
5	Johnny	Lollobrigida	2013-05-26T14:47:57Z

インポート

trdsq1の動作として、元々ファイルを指定した場合にテンポラリテーブルを作成して、インポートしている。

そのため、テンポラリテーブルではなく実テーブルを指定する機能を開発しようとするが、考慮することが爆発的に増える。

- テーブルがすでに存在している場合...
- 列名を変えたい...
- 型を定義したい...

機能開発は中止。

SQLで実行すればよいことに気づいた。

```
trdsq1 "CREATE TABLE fruits AS SELECT id::int,name,price::int FROM fruits.csv"
```

CREATE TABLE テーブル AS の代わりに INSERT INTO テーブル SELECT を使用すれば既存のテーブルにもインポート可能。

ON CONFLICT DO NOTHING や ON CONFLICT DO UPDATE を使用すれば、専用のインポートツールよりも柔軟なインポートが可能。

JSON対応

trdsqはJSONL(NDJSON) やトップが配列になっているJSONが対象だった。
ただ、そこから外れるけど、リスト形式なJSONはよくある。

```
{
  "userList": [
    {
      "userID": 1,
      "nickname": "taro",
    },
    {
      "userID": 2,
      "nickname": "hanoko",
    },
    {
      "userID": 3,
      "nickname": "momoko",
    }
  ]
}
```

そのままだとusrListという1列1行のテーブルになる。

```
trdsq1 -omd "SELECT * FROM sample.json"
```

	usrList	

	[{"nickname": "taro", "userID": 1}, {"nickname": "hanoko", "userID": 2}, {"nickname": "momoko", "userID": 3}]	

前まではSQLのJSON関数でなんとかするか、jqで前処理をしてパイプで渡せば良いと考えていた。

```
jq ".usrList" sample.json | trdsq1 -omd -ijson "SELECT * FROM -"
```

jq構文

`gojq` というGo製のjqクローンが開発されていた。

packageとして使えるので、取り込んでしまった方が便利。

ファイル名 `::` の後にjq構文を書く と解釈してからSQLを実行できるようにした。

```
trdsq1 -omd "SELECT * FROM sample.json::.userList"
```

userID	nickname
1	taro
2	hanoko
3	momoko

⇒ 次のツール

**PostgreSQLの開発当初からあった問題が
2021年に進展したのをご存知でしょうか？**

それは...

lessにヘッダーオプションが入った！

(まだベータリリース版)

```
less --header 2
```

Pager

lessはPagerと呼ばれるジャンルのアプリケーションで、画面に収まらない出力を1画面ずつ表示。

Pagerはいくつかある。more,less,most...

psqlやmysqlのREPLは自分でパイプに渡せないなので内部で使用する。環境変数PAGERで設定されたコマンドを使用する。

Pagerに渡った後はPagerの操作になるので、psqlを使っていると思っている半分はPagerを操作している。

psqlにとってPagerは大事

less

<http://greenwoodsoftware.com/less/>

Pagerのデファクトスタンダードといえる存在がless

いつの間にかGitHubにも置かれていてissue対応してます！

<https://github.com/gwsw/less>

1983年からの開発で2021年にヘッダーオプションが入った。

それまでは最初に列名を出力してもスクロールすると消えていた。

psqlの出力が画面に収まらないうと '\x'で縦に表示しろというのが定番でした。

ヘッダーオプションを使用することで、常に列名が表示できるようになった。

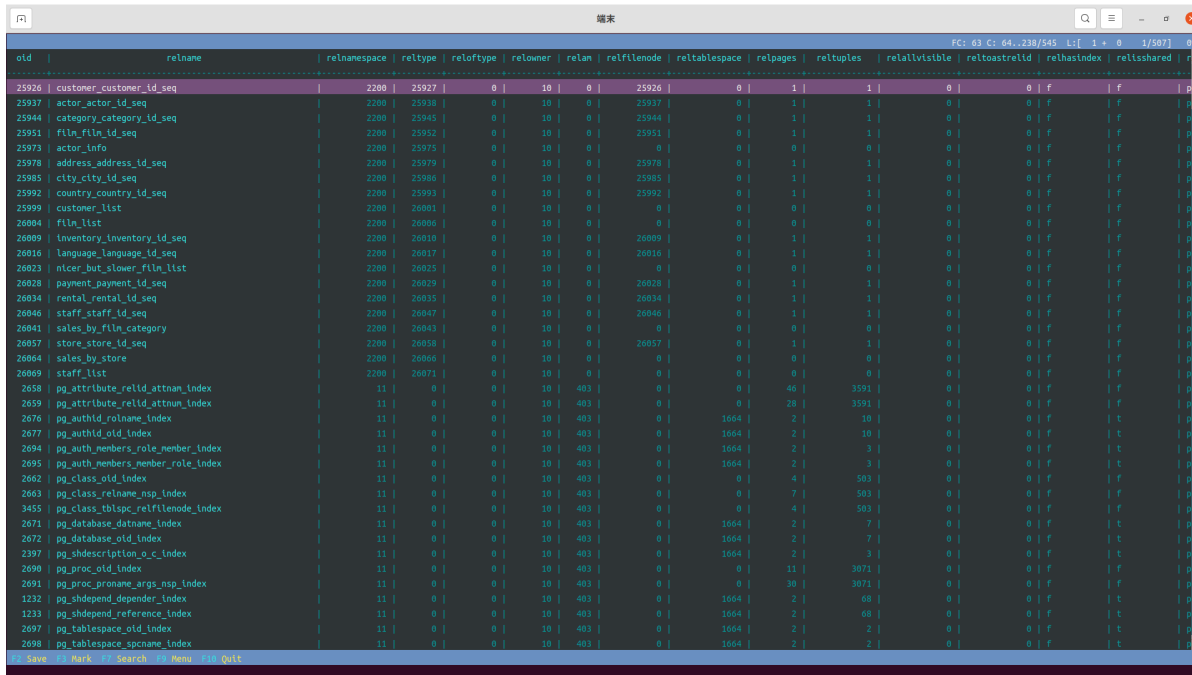
oid	relname	relnamespace	r
2674	pg_depend_reference_index	11	>
3597	pg_seclabel_object_index	11	>
548	pg_foreign_data_wrapper_name_index	11	>
1259	pg_class	11	>
68159	pg_toast_68156	99	>
68161	pg_toast_68156_index	99	>
76735	pg_toast_76732	99	>
68156	s	2200	>
76372	pg_toast_76369	99	>
76737	pg_toast_76732_index	99	>
76374	pg_toast_76369_index	99	>
76369	fruits	2200	>
76732	worldcitiespop	2200	>
76705	pg_toast_76702	99	>
76707	pg_toast_76702_index	99	>
76702	test	2200	>
76717	pg_toast_76714	99	>
76719	pg_toast_76714_index	99	>
76714	inbox	2200	>
(503 rows)			
(END)			

(ヘッダーオプションを使用すると折り返さず横スクロール表示になる)。

pspg

pspg はテーブル表示に特化したページャー。

pspgを使用することで、列名を常に表示、列の固定を利用して閲覧できる。
テーブル表示なので、画面端でも折り返さないで横スクロールして表示する。



oid	relname	relnamespace	reltype	relowner	relrelname	reltablespace	relpages	reltuples	relallvisible	reltoastrelid	relhasindex	relisshared	relkind
25926	customer_customer_id_seq	2200	25927	0	10	0	25926	0	1	1	0	0	f
25937	actor_actor_id_seq	2200	25938	0	10	0	25937	0	1	1	0	0	f
25944	category_category_id_seq	2200	25945	0	10	0	25944	0	1	1	0	0	f
25951	film_film_id_seq	2200	25952	0	10	0	25951	0	1	1	0	0	f
25972	actor_info	2200	25973	0	10	0	0	0	0	0	0	0	f
25978	address_address_id_seq	2200	25979	0	10	0	25978	0	1	1	0	0	f
25985	city_city_id_seq	2200	25986	0	10	0	25985	0	1	1	0	0	f
25992	country_country_id_seq	2200	25993	0	10	0	25992	0	1	1	0	0	f
25999	customer_list	2200	26001	0	10	0	0	0	0	0	0	0	f
26004	film_list	2200	26006	0	10	0	0	0	0	0	0	0	f
26009	inventory_inventory_id_seq	2200	26010	0	10	0	26009	0	1	1	0	0	f
26016	language_language_id_seq	2200	26017	0	10	0	26016	0	1	1	0	0	f
26023	nicer_but_slower_film_list	2200	26025	0	10	0	0	0	0	0	0	0	f
26028	payment_payment_id_seq	2200	26029	0	10	0	26028	0	1	1	0	0	f
26034	rental_rental_id_seq	2200	26035	0	10	0	26034	0	1	1	0	0	f
26040	staff_staff_id_seq	2200	26047	0	10	0	26040	0	1	1	0	0	f
26041	sales_by_film_category	2200	26043	0	10	0	0	0	0	0	0	0	f
26057	store_store_id_seq	2200	26058	0	10	0	26057	0	1	1	0	0	f
26064	sales_by_store	2200	26066	0	10	0	0	0	0	0	0	0	f
26069	staff_list	2200	26071	0	10	0	0	0	0	0	0	0	f
2658	pg_attribute_relid_attnam_index	11	0	0	10	403	0	46	3591	0	0	0	f
2659	pg_attribute_relid_attnum_index	11	0	0	10	403	0	28	3591	0	0	0	f
2670	pg_authidrolname_index	11	0	0	10	403	0	1664	2	10	0	0	f
2677	pg_authid_oid_index	11	0	0	10	403	0	1664	2	10	0	0	f
2694	pg_auth_members_role_member_index	11	0	0	10	403	0	1664	2	3	0	0	f
2695	pg_auth_members_member_role_index	11	0	0	10	403	0	1664	2	3	0	0	f
2662	pg_class_oid_index	11	0	0	10	403	0	4	503	0	0	0	f
2663	pg_class_relname_nsp_index	11	0	0	10	403	0	7	503	0	0	0	f
3455	pg_class_tblspc_relfilenode_index	11	0	0	10	403	0	4	503	0	0	0	f
2671	pg_database_datname_index	11	0	0	10	403	0	1664	2	7	0	0	f
2672	pg_database_oid_index	11	0	0	10	403	0	1664	2	7	0	0	f
2397	pg_shdescription_o_c_index	11	0	0	10	403	0	1664	2	3	0	0	f
2690	pg_proc_oid_index	11	0	0	10	403	0	0	11	3071	0	0	f
2691	pg_proc_proname_args_nsp_index	11	0	0	10	403	0	0	30	3071	0	0	f
1232	pg_shdepend_depender_index	11	0	0	10	403	0	1664	2	68	0	0	f
1233	pg_shdepend_reference_index	11	0	0	10	403	0	1664	2	68	0	0	f
2697	pg_tablespace_oid_index	11	0	0	10	403	0	1664	2	2	0	0	f
2698	pg_tablespace_spcname_index	11	0	0	10	403	0	1664	2	2	0	0	f

pspg はpsqlを想定して作られているので、psqlを使用するならpspgの方が便利。

OV

<https://github.com/noborus/ov>

私が新しく作った汎用ページャー。

1つの表示方法だけでなく、いろんな表示を動的に切り替えられる。

ヘッダー表示しても折返しができるようにした。

画面幅に収まらなくても横スクロールしないで表示できる。

そのために以下の機能を追加。

- 行背景の交互表示
- 列のハイライト

ovは機能盛りだくさん

- 検索 - インクリメンタルサーチ、正規表現のインクリメンタルサーチ
- フォローモード (tail -f相当)、複数ファイルのフォローモード
- 圧縮ファイル対応 (gzip,bzip2,zstd,lz4,xz)
- execモード (標準出力と標準エラー出力を分けて表示)

開発予定

PSQL_WATCH_PAGER

開発中のPostgreSQL 15ではPSQL_WATCH_PAGERという環境変数が追加される予定。

この変数をセットすれば `\watch` の出力をPAGERに出力できる。

pspgが対応（pspgの作者が入れた）

--streamオプションを使う。

`\watch` の出力はスクロールして流れていくので複数行出力しての変化は見づらい。

先頭からの位置が変わらないと変化に気づきやすい。

Unixコマンドで言えばpsコマンドを定期的に行うか、topコマンドを使用するかの違い。

実際に `\watch` を実行した結果がPAGERに渡されると以下のような形式で出力される。

```
Thu Mar 17 15:53:27 2022 (every 1s)  <----- タイトル
                                     <!----- 空行

  a | b
  ---+---
  1 | 2
(1 row)

Thu Mar 17 15:53:28 2022 (every 1s)  <!----- 空行（次の行は1秒後に出力）
                                     <----- タイトル
                                     <!----- 空行

  a | b
  ---+---
  1 | 2
(1 row)

                                     <!----- 空行
```

- 1回目の空行でタイトルが終わって結果がはじまりを表す
- 2回目の空行で結果の終わりを表す

この結果に依存して表示モードを作るのは辛い...

空行が途中で入ったり、空行を取りこぼしたりするとズレて表示が崩れやすい。

結果の区切り文字を`^L(form feed)`追加することを提案して、CommitFestにも登録した。

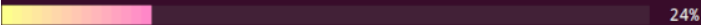
pspg作者のPavel Stehuleさんだけ賛成してくれたけど、他の反応はイマイチ...
ovで`\watch`が使いやすく表示できるのかは不透明。

⇒ 次のツール

pgsp

<https://github.com/noborus/pgsp>

```
$ pgsp
Using config file: /home/noborus/.pgsp.yaml
quit: q, ctrl+c, esc
pg_stat_progress_basebackup
pid          | 401044
phase        | streaming database files
backup_total  | 10976660480
backup_streamed | 2644375552
tablespaces_total | 1
tablespaces_streamed | 0
```



pg_stat_progress_* という、処理中状況を表すViewを表示するだけのシンプルなツール

対象View

バージョンによって増えていきますが。以下のViewを対象にしています。

- pg_stat_progress_analyze
- pg_stat_progress_basebackup
- pg_stat_progress_cluster
- pg_stat_progress_copy
- pg_stat_progress_create_index
- pg_stat_progress_vacuum

```
SELECT * FROM pg_stat_progress_analyze;
```

実行すれば、その時点での状況を教えてくれる。

```
pid | datid | datname | relid | phase | sample_blks_total |
sample_blks_scanned | ext_stats_total | ext_stats_computed | child_tables_total |
child_tables_done | current_child_table_relid
-----+-----+-----+-----+-----+-----+-----+-----+-----+
-+-----+-----+-----+-----+-----+-----+-----+-----+
-+-----+-----+-----+-----+-----+-----+-----+-----+
30481 | 16386 | noborus | 25855 | acquiring sample rows | 30000
| 21320 | 0 | 0 | 0
| 0 |
(1 row)
```

(結果はViewによって異なる。)

問題点

処理中にはレコードが追加されて、処理が終わるとレコードが消える。

psql の \watch を使用すれば監視できるが、
前述の通り \watch はスクロールして流れていくので変化は見づらい。

前述のPSQL_WATCH_PAGERによって改善するかもしれない。

そもそも、わかりやすすくない。

pgsp

プログレスのViewなんだからプログレスバーを表示する監視ツールを探す
が無かった⇒そこで作ったのが pgsp

特徴

- 1つのviewだけでなく、複数のview（デフォルトは全部）に対して定期的に問い合わせる。
- 処理中はわかる範囲でプログレスバーを表示。
- レコードが消えても指定した秒数間は表示し続ける。
- ターミナルの表示域（幅、高さ）によって、表示方法を変更。
- オプションで、監視間隔、終了してから表示し続ける秒数等に対応。

今後

GoのTUIフレームワークBubble Teaを使用

<https://github.com/charmbracelet/bubbletea>

プログレスバー以外にも機能豊富な
ので、統合監視ツールも作れるか
も。

Bubble Tea



release v0.20.0  reference  build passing


The fun, functional and stateful way to build terminal apps. A Go framework based on [The Elm Architecture](#). Bubble Tea is well-suited for simple and complex terminal applications, either inline, full-window, or a mix of both.

```
thunderclap:~ christian $ ./demo

Carrot planting?

Cool, we'll need libgarden and vegeutils...

Downloaded. Exiting in 3 seconds...
```



```
100%
```

Bubble Tea is in use in production and includes a number of features and performance optimizations we've added along the way. Among those is a standard framerate-based renderer, a renderer for high-performance scrollable regions which works alongside the main renderer, and mouse support.

⇒ 次のツール

jpug-doc-tool

PostgreSQL マニュアル翻訳のためのツール

<https://github.com/noborus/jpug-doc-tool/>

自己紹介

PostgreSQLマニュアルの日本語翻訳プロジェクトに参加している齊藤です。

PostgreSQLマニュアルの日本語翻訳の管理がGitHubに移行してからビルドツールの修正等以下のことを担当しています。

- UTF-8への変更
- ビルド環境の修正
- CIの整備
- CSS、HTMLのヘッダーフッター
- PDF作成
- <https://pgsql-jp.github.io/> の管理

翻訳もやってますが、英語は得意じゃないです。

PostgreSQL日本語マニュアル

PostgreSQL日本語マニュアル翻訳プロジェクトはjpug-docという名前で管理されています。

<https://github.com/pgsql-jp/jpug-doc>

詳しくはQiitaの[PostgreSQL日本語マニュアルについて](#)を参照してください。

現在、拡張子はsgmlですが、すべてXML処理系で処理されています。

jpug-doc-tool誕生のきっかけ

PostgreSQL 13のマニュアルには「翻訳が終わらない危機」があった。
中身は変わらないが表記法が変わって変更量が爆発した。

一番影響が大きかったfunc.sgml。

バージョンアップ	変更行数
11.0⇒12.0	2,168
12.0⇒13.0	36,575

変更 (+,-) の行数が実際の行数を上回った。

バージョン	行数
12.0	22,673
13.0	21,153

実際の変更例

内容は変わらないが、タグが変わって、
13のマニュアルに対して12の翻訳をマージできず

```
<entry>round(</entry>
<entry>numeric</entry>
<!--
    <entry>Rounds to nearest integer</entry>
-->
    <entry>最も近い整数への丸め</entry>
```

を以下のようにする必要がある。

```
<entry>
  <para>round</para>
  <para>numeric</para>
  <para>
<!--
    Rounds to nearest integer
-->
    最も近い整数への丸め
  </para>
</entry>
```

英語と日本語のペアのリストを抽出して、
新しいバージョンで日本語訳を挿入する方法にした。

```
en:Rounds to nearest integer  
ja:最も近い整数への丸め
```

実装は正規表現バリバリ。
XML処理系で置き換えようとするインデントが元に戻せなかった...
数パターンに対応したので、他の人に使えるように体裁を整えたのが

```
jpug-doc-tool
```

jpug-doc-toolの機能

チェック機能を追加

英語と日本語のペアのリストにより以下のチェックが可能になった。

- 未翻訳の英語だけのパラグラフがないか
- 英語、日本語訳のタグが同じか
- 日本語に含まれている英単語が英語にもあるか
- 英語と日本語を指定して両方含まれているか `--en merge --ja マージ`
- 数値は同じ数値が含まれているか

13の翻訳完了後も数十カ所発見（最近http->https等の修正が多くあった）。

日本語訳挿入の強化

完全一致した場合のみ日本語訳を適用するだけでなく、類似した文に注意書きを入れて挿入するオプションを追加。

`a SQL` ⇒ `an SQL` のような修正では日本語訳の修正は必要ない。

レーベンシュタイン距離によって同一ファイル内の文から類似しているか比較
(package <https://github.com/agnivade/levenshtein> を使用しているだけ)。

さらにAPIを利用した機械翻訳も！

みんなの自動翻訳@TexTra

<https://mt-auto-minhon-mlt.ucri.jgn-x.jp/>

利用規約にオープンソースライセンスの翻訳に使用
できることが明言されている。

いわゆるAI翻訳で精度も日々向上している。

APIも公開している。

GoからAPIを利用できるパッケージを作成。

<https://github.com/noborus/go-textra>

jpug-doc-toolにも組み込んだ。



アカウント情報を設定ファイルに書いたら、以下を実行すると...

```
jpug-doc-tool replace --mt brin.sgml
```

未翻訳の箇所に日本語訳を挿入する。

```
API...[Some of the built-in operator ] Done
API...[bloom operator classes accept ] Done
API...[Defines the estimated number o] Done
API...[Defines the desired false posi] Done
API...[minmax-multi operator classes ] Done
API...[Defines the maximum number of ] Done
API...[Returns whether all the ScanKe] Done
API...[To write an operator class for] Done
API...[Support procedure numbers 1-10] Done
API...[The minmax-multi operator clas] Done
replace: brin.sgml
```

「みんなの自動翻訳」を利用した日本語訳挿入

```
diff --git a/doc/src/sgml/brin.sgml b/doc/src/sgml/brin.sgml
index 7b90452dd8..0c60b2bc79 100644
--- a/doc/src/sgml/brin.sgml
+++ b/doc/src/sgml/brin.sgml
@@ -778,14 +778,24 @@ LOG:  request for BRIN range summarization for index "brin_wi_idx" page 128 was
    <title>Operator Class Parameters</title>

    <para>
+<!--
        Some of the built-in operator classes allow specifying parameters affecting
        behavior of the operator class.  Each operator class has its own set of
        allowed parameters.  Only the <literal>bloom</literal> and <literal>minmax-multi</literal>
        operator classes allow specifying parameters:
+-->
+<!-- 《機械翻訳》 -->
+一部の組み込み演算子クラスでは、演算子クラスの動作に影響するパラメータを指定できます。
+各演算子クラスには、使用可能なパラメータの独自のセットがあります。
+パラメータを指定できるのは、<literal>bloom</literal>演算子クラスと<literal>minmax-multi</literal>演算子クラスのみです。
    </para>
```

最後に人がチェックして修正し、《機械翻訳》コメントを消せばOK。

「みんなの自動翻訳」をコマンドで利用

コマンドラインから翻訳ツールとしても使用できる。

```
jpug-doc-tool mt "This is a pen."
```

generalNT_en_jaがデフォルトの翻訳エンジンだが、カスタマイズした翻訳エンジンを追加できる。

```
c-1640_en_ja: これはペンです。  
generalNT_en_ja: これはペンです。
```

自動翻訳の利用は、まだほぼされていない。

jpug-doc-toolを利用している人が自分以外いない...

紹介したツール

(全部MITライセンスです)

trdsql

<https://github.com/noborus/trdsql>

ov

<https://github.com/noborus/ov>

pgsp

<https://github.com/noborus/pgsp>

jpug-doc-tool

<https://github.com/noborus/jpug-doc-tool>

みんなの自動翻訳APIクライアント

Goパッケージ

<https://github.com/noborus/go-textra>

おしまい