# Perl & Postgres
# трудный путь, потому что это проще.

Без DBI(x)::Class, поэтому он более эффективен (для меня)

Салем Даниэль

ИТ-Кибернетик, Консультант

Perl-Conf.Ru/25

# Впечатления из моей галактики

- Время - это самое важное.
- Жизнь - это не спринт, это марафон.
- Ознакомьтесь с документацией,
  через 6 месяцев неважное станет важным.
- Документируйте свой опыт. Опирайтесь на свой опыт.
- Сложность убивает.
- Только потому, что это технически возможно,
  мы не должны этого делать.
- Нет никаких “коротких путей”.
- Избегайте сладкого яда
  (средства для быстрого достижения результатов).

Не всегда это возможно, но вы должны знать о последствиях.

# Стиль работы: Что такое работа?

| | | |
|---|---|---|
| ‘‘реальное’’ содержание работы | => “Логика” | |
| ‘‘вспомогательная’’ работа | => “Логистика” | = admin/docu/... |

**Моя цель:** сосредоточиться на логике, оптимизировать логистику

**Но:** мой стиль программирования - “логистика для создания логики”!

=> Оптимизируйте стиль программирования:

... время показывает, что я ненавижу...

1. постоянно меняющийся параметр-список функций
2. постоянно меняющееся количество возвращаемых значений из функций
3. слишком много переменных (что, когда, где,...)
4. слишком много строк кода
5. ...

# Я ... “hash-ref” человек

hash-ref structure

для

- данные
- конфигурация
- управляющий параметр

как

- параметр для sub
- контейнер для возвращаемых значений

И именно за это я люблю Emacs:

Где Emacs, там и shell есть!
shell / eshell (*nixTools)

очень полезно для Win /Mac

```
File  Edit  Options  Buffers  Tools  Complete  In/Out  Signals  Help
$|=1;      # emacs: Alt-x shell & Enter => shell-console
use strict;
use warnings;
use Data::Dumper; $Data::Dumper::Sortkeys = \&my_filter;
sub my_filter {return [(sort keys %{$_[0]})];}

my $h = { a => 3, b => 7};

sub f {
    my $h_ = $_[0];
    $h_->{c} = $h_->{a} - $h_->{b};
    return ($h_->{a} < $h_->{b} ) ? -1 : 1;
}

my $ret = f($h);    # two information channels:
print $ret,"\n";    # - operational information
print Dumper($h);   # - data
U:---  z_perl_konf_02_perl_001.pl   All   L1     (Perl +2)
~/702_perl_konferenzen/202509> perl z_perl_konf_02_perl_001.pl
-1
$VAR1 = {
          'a' => 3,
          'b' => 7,
          'c' => -4
        };
~/702_perl_konferenzen/202509>
U:**-  *shell*        All   L8     (Shell:run +1)
M-x shell
```

# DBI::Class - Simple Database Abstraction (I)

**perldoc:**
It not only provides a simple **database to object mapping layer**, but can be used **to implement several higher order database functions at the application level, rather than at the database.**

.. useful when using a database which doesn't support these (MySQL), or to be portable across multiple databases ...

.. aims to make it simple to introduce **'best practice'** when dealing with data stored in a relational database.

---

*Вы вводите дополнительный уровень сложности.*
*Вы перемещаете работу с данными подальше от "мощных инструментов", известных как базы данных!*
*почему??*

# DBI::Class - Simple Database Abstraction (II)

**background information:**

*базы данных “прозрачны” => Компания хочет скрыть код от проверки.*

*Java (запущена в 1995 году) создает байт-код, но как скрыть логику базы данных?*

**Java знает только объекты!**

*=> Java “нуждается” в “сопоставлении базы данных с объектом”,*
*чтобы скрыть логику базы данных в байт-коде.*

*==> 2001 Hibernate = Object-Relational Mapping*

# DBI::Class - Simple Database Abstraction (II)

**background information:**

*базы данных "прозрачны" => Компания хочет скрыть код от проверки.*

*Java (запущена в 1995 году) создает байт-код, но как скрыть логику базы данных?*

**Java знает только объекты!**

*=> Java "нуждается" в "сопоставлении базы данных с объектом",*
*чтобы скрыть логику базы данных в байт-коде.*

*==> 2001 Hibernate = Object-Relational Mapping*

---

**Но это Perl, а не Java....**

1. *Perl все равно позволяет вам проверять код*
2. *Perl использует хэши и массивы в качестве контейнера данных, и им не нужно создавать объекты.*
3. *Нужна ли нам "best/лучшая практика Java" в Perl?*

# Настройка рабочего места I: установите PostgreSQL

1. > cd; mkdir 001_build system; cd 001_build
2. **загрузите PostgreSQL в каталог 001_build**
3. > tar -xjf postgresql-15.3.tar.bz2; cd postgresql-15.3
4. > ./configure -prefix=$HOME/system/postgresql-15.3/
5. > make world && make install-world
6. **разверните файл .bashrc с помощью**:

   export PATH="$HOME/system/postgresql-15.3/bin:$PATH";

   export LD_LIBRARY_PATH="$HOME/system/postgresql-15.3/lib:$LD_LIBRARY_PATH"

   export LD_LIBRARY_PATH="$HOME/system/postgresql-15.3/lib64:$LD_LIBRARY_PATH"

   export MANPATH="$HOME/system/postgresql-15.3/share/man/:$MANPATH";
7. **документация?**

   file:///$HOME/system/postgresql-15.3/share/doc/html/index.html

# Настройка рабочего места II: Скрипты Bash

1. > cd; mkdir -p work/001_postgres_data; cd work
2. **002_postgres_start.bash**:

   ```
   pg_ctl start -w -D "$HOME/work/001_postgres_data/" -o "-p 5432"
   ```

3. **003_postgres_stop.bash**:

   ```
   pg_ctl stop -w -D "$HOME/work/001_postgres_data/"
   ```

4. **009_postgres_init.bash**:

   ```
   echo "sic-exit"; exit;
   data_dir="$HOME/work/001_postgres_data/"
   rm -r $data_dir
   initdb –no-locale –encoding=utf-8 -D $data_dir
   pg_ctl start -w -D $data_dir -o "-p 5432"
   createuser -s -d -r -E -p 5432 postgres
   createdb -e -p 5432 test_work
   pg_ctl stop -w -D $data_dir
   ```

# Настройка рабочего места III: Инициализировать/запуск базы данных

## Инициализировать базы данных

1. Инициализировать: > bash 009_postgres_init.bash
   (*отключите строку "sic-exit"для запуска*)
2. создайте каталог “sql_file”: > mkdir sql_file

## работа с двумя bash-терминалами

1. первый bash-терминал

   > bash 002_postgres_start.bash

   ... log-output ...

   > bash 003_postgres_stop.bash

2. второй bash-терминал для взаимодействия

   > $HOME/system/postgresql-15.3/bin/psql –help

   > $HOME/system/postgresql-15.3/bin/psql -d test_work -p 5432

   > psql -d test_work -p 5432

# Способ Emacs

**100_postgres_emacs_run.bash**:

bash 002_postgres_start.bash; emacs -r . ; bash 003_postgres_stop.bash

```
bash-5.2$ bash 100_postgres_emacs_run.bash
waiting for ser
PostgreSQL 15.3
2025-08-13 15:3
ort 5432
2025-08-13 15:3
5432"
2025-08-13 15:3
-13 14:40:18 MS
2025-08-13 15:3
ctions
 done
server started
```

work - GNU Emacs at Standard_PC

File Edit Options Buffers Tools Operate Mark Regexp Immediate Subdir Help

```
1. create new file  => Ctrl-x Ctrl-f "001_demo.sql"
```

```
-:---   001_doc_short.org    5%   L3     (Org +2 Ind)
  /home/test/work:
  insgesamt 32K
  drwxr-xr-x  4 test test 4096 13. Aug 11:23 .
  drwxr-xr-x 12 test test 4096 13. Aug 14:15 ..
  drwx------ 19 test test 4096 13. Aug 15:34 001_postgres_data
  -rw-r--r--  1 test test   68 12. Aug 13:13 002_postgres_start.bash
  -rw-r--r--  1 test test   55 12. Aug 12:58 003_postgres_stop.bash
  -rw-r--r--  1 test test  321 12. Aug 13:24 009_postgres_init.bash
  -rw-r--r--  1 test test   95 12. Aug 16:53 100_postgres_emacs_run.bash
  drwxr-xr-x  2 test test 4096 13. Aug 14:36 sql_files

U:%%-   work                 All   L2     (Dired by name)
Find file: ~/work/sql_files/001_demo.sql
```

# Emacs: подключение к базе данных

```
File  Edit  Options  Buffers  Tools  Minibuf  Help
2. connect to postgres => Alt-x sql-postgres => Enter
   User: ..; Database: "tab" => DB-List

-:---                                          +2 Ind)
U:---   001_demo.sql   All   L1    (SQ
User (default test):

Type M-<down> or M-<up> to move point between comp

4 possible completions:
template0                                templa
test                                     test_w
U:%*-d  *Completions*   All   L1    (Completion l
Database (default test): te

U:---   001_demo.sql   All   L1    (SQL[ANSI])
Server:

U:---   001_demo.sql   All   L1    (SQL[ANSI])
M-x sql-postgres
```

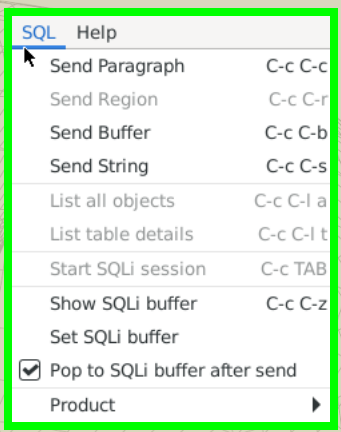
```
U:---   001_demo.sql   All   L1    (SQL[AI
psql (15.3)
Type "help" for help.

test_work=# help
You are using psql, the command-line interf
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to ex
       \q to quit
test_work=#

U:**-   *SQL: Postgres*   All   L11   (SQ
```

type “help” for ...

# Emacs: отправка sql-кода в PostgreSQL

```
File  Edit  Options  Buffers  Tools  SQL  Help
  4. Switch to file-frame;
     "SELECT now();" & save Ctrl-x Ctrl-s
     Send command to sql-console: Ctrl-c Ctrl-c
-:---   001_doc_short.org   45%   L11    (Org +2 Ind)

SELECT now();

U:---   001 demo.sql   All   L2     (SQL[Postgres])
test_work=#
               now
-------------------------------
 2025-08-13 14:36:41.859335+03
(1 row)

test_work=#
U:**-   *SQL: Postgres*   All   L17    (SQLi[Postgres]:run)
Sent string to buffer *SQL: Postgres*
```

```
test_work=# \q

Process SQL: Postgres finished
```




Иногда это необходимо
“Set SQLi buffer” to “*SQL: Postgres*”
Menu or Alt-x “sql-set-sqli-buffer”

обрезка длинных строк
Alt-x “toggle-truncate-lines”

# Что предлагает база данных (PostgreSQL)?

| | | |
|---|---|---|
| Tables (data) | Views | (combining tables/abstract layer) |
| | Index | (performance, sorting) |
| | Trigger | (event driven process = functions) |
| Functions | *plpgsql* | (also possible Perl, R, C, ...) |

**!!Реальные базы данных - это мощные инструменты для управления данными. Используйте их!!**

shortened list of data types

| data type | Description |
|---|---|
| xml / bytea | XML data / binary data (“byte array”) |
| json / jsonb | textual/binary JSON data |
| uuid | universally unique identifier |
| bit [ (n) ] | fixed-length bit string |
| cidr / inet / macaddr | IPv4/6 network/host / MAC address |
| line / lseg | infinite line / line segment on a plane |
| box / circle | rectangular box / circle on a plane |

# работа с “jsonb”: jsonb (database) => hash (perl) : SELECT

```
CREATE TABLE user_feedback(
 username     varchar,
 feedback     jsonb -- DEFAULT '{}'::jsonb
);

INSERT INTO user_feedback (username, feedback) VALUES 'Otto','{}'::jsonb;

SELECT username, feedback FROM user_feedback WHERE username = 'Otto';
```

```
$h_->{sql} = qq|SELECT feedback FROM user_feedback WHERE username = 'Otto';|;
dbh_execute(\&do_dbh_func,$h_);
my $data   = $h_->{data}->[0];
utf8::encode($data->{feedback});
$h_->{feedback_hash} = decode_json $data->{feedback};
```

```
sub dbh_execute {
    my ($dbh_execute_func,$func_par) = @_;
    $dbh_execute_func->($dbh,$func_par);
    $dbh->commit;
}
```

```
sub do_dbh_func{
    my $sth = $_[0]->prepare($_[1]->{sql}); $sth->execute();
    my @row;
    while ( my $h_refrow = $sth->fetchrow_hashref )
    { push @row, $h_refrow; }
    $_[1]->{data} = [@row];
}
```

*“jsonb как естественное расширение хэша в PostgreSQL!”*

# работа с “jsonb”: hash (perl) => jsonb (database) : INSERT/UPDATE

*Давайте распространим “стиль хэш-программирования” на базу данных!*

```
-- $jsonb = encode_json {"username":"Otto","feedback":{"2":1}};
-- $h_->{sql} = qq|SELECT * FROM user_feedback_insert(E'$jsonb'::jsonb);|;
-- dbh_execute(\&do_dbh_func,$h_);
```

```
DROP FUNCTION IF EXISTS    user_feedback_insert(jsonb);
CREATE OR REPLACE FUNCTION user_feedback_insert(IN json_ jsonb) RETURNS TABLE(erfolg boolean)
AS $$
 BEGIN
 PERFORM username FROM user_feedback WHERE username = json_->>'username';
 IF found THEN
    UPDATE user_feedback
    SET feedback = (COALESCE(feedback,'{}'::jsonb)||(json_->>'feedback')::jsonb)::jsonb
    WHERE username = json_->>'username';
 ELSE
    INSERT INTO ru_user_words_feedback(username,feedback)
      SELECT json_->>'username',(json_->>'feedback')::jsonb;
 END IF;
 RETURN QUERY SELECT found;
END;
$$ LANGUAGE plpgsql
;

SELECT * FROM user_feedback_insert(E'{"username":"Otto","feedback":{"2":1}}'::jsonb);
```

# Perl “tie” к базе данных? => триггер: рабочий процесс в фоновом режиме

```
| TG_OP                | TG_WHEN     |     UPDATE [ OF column_name [, ... ] ]
|----------------------+-------------|
| INSERT/UPDATE/DELETE | BEFORE      |
|----------------------+-------------|
| INSERT/UPDATE/DELETE | AFTER       |
|----------------------+-------------|
| INSERT/UPDATE/DELETE | INSTEAD OF  |
|----------------------+-------------|

DROP FUNCTION public.user_feedback_trigger CASCADE;
CREATE OR REPLACE FUNCTION public.user_feedback_trigger() RETURNS trigger AS
$$
BEGIN
IF (TG_OP = 'UPDATE') THEN
  IF (TG_WHEN = 'BEFORE') THEN -- new.update_date = now();
      RETURN new;
  END IF;
  IF (TG_WHEN = 'AFTER') THEN  -- INSERT INTO public.user_feedback_hist VALUE
      RETURN NULL;
  END IF;
ELSIF (TG_OP = 'INSERT') THEN  -- new.update_date = now();
      RETURN new;
END IF;
END
$$ LANGUAGE plpgsql VOLATILE COST 100;

CREATE TRIGGER user_feedback_trigger_before
  BEFORE INSERT OR UPDATE ON public.user_feedback
     FOR EACH ROW  EXECUTE FUNCTION public.user_feedback_trigger();

CREATE TRIGGER user_feedback_trigger_after
  AFTER UPDATE  ON public.user_feedback
     FOR EACH ROW  EXECUTE FUNCTION public.user_feedback_trigger();
```

*perldoc perltie*

The tie() function binds a variable to a class (package) that will provide the implementation for access methods (FETCH, STORE, ...) for that variable.

*Trigger*

Триггер привязывает функции к таблице, зависящей от TG_WHEN.
TG_OP позволяет идентифицировать фактическое действие над таблицей.

(- - INSERT INTO public.user_feedback_hist VALUES(old.*); )

# Резюме/заключение: Стиль работы, ориентированный на “hash-ref”

Работая напрямую с “SQL” вместо DBI(x)::Class,

=> Я могу распространить свой стиль работы “hash-ref” на PostgreSQL.

- hash-ref как параметр для subs в Perl

  => “созданные” данные, вставленные в хэш-ссылку

- хэш в формате jsonb в качестве параметра для функций в PostgreSQL

  => sql-данные в виде “возвращаемой таблицы”, вставленные в структуру hash-ref

Для меня это приводит к

1. сосредоточьтесь на логике & больше гибкости / контроля / элегантности
2. больше удовольствия / фитнес-тренировка для моего мозга: “print Dumper($h)” очень помогает

Я беру на себя ответственность за контент и структуры внутри PostgreSQL.

Спасибо!

Давайте обсудим ...