PG BootCamp Russia 2025 Ekaterinburg

Превращаем PostgreSQL в RestFull API Server

Константин Ващенков
kv@xsquare.ru
xsquare.ru

@XSQUARE365

Разработка приложений

Web

HTTP SERVER

APPLICATION SERVER

DB SERVER

Разработка приложений

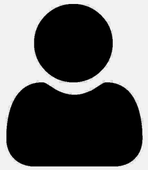
Web

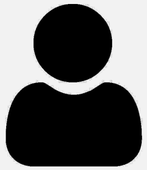
HTTP SERVER

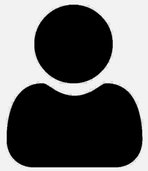
APPLICATION SERVER

DB SERVER

Разработка:





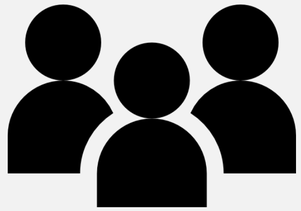


Разработка web-приложений

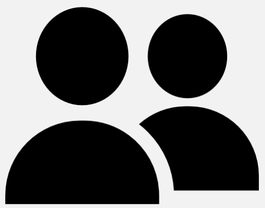
Web

Разработка:

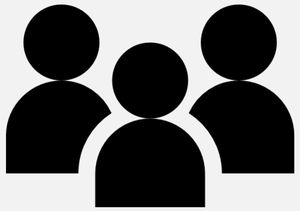
HTTP SERVER



APPLICATION SERVER



DB SERVER



PostgreSQL и внешний мир

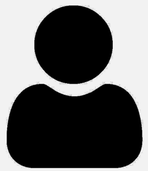
Web

API

Другие DB

Как подключать

PostgreSQL SERVER



Типовые задачи OLTP-приложения

Базы интеграция

База данных

ORACLE | MySQL | Firebird | Microsoft | PostgreSQL | IBM

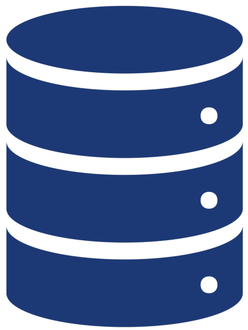
Отчетность/Репликация

LDAP | Auth

HTTP Client | Cripto/CryptoPro

Планировщики

PostgreSQL



Очереди

Работа с файловой системой

Отчетность DOCx/XLSx/PDF

Работа с геоданными | Графическая визуализация

API для внутр. систем | API для ЛК/Сайт

email | sms

Расширения. Сила PostgreSQL в расширяемости!

Миграция
- ora2pg
- orafce
- ora_migrator

Планировщики
- pg_cron
- pg_timetable

Расширения
- http
- pgmail
- pgcrypto
- postGIS
- uuid-ossp
- hstore
- pg_trgm
- pg_stat_statements

и др.

Обёртки внешних данных
- dblink_fdw
- oracle_fdw
- tds_fdw

Загружаемые процедурные языки
- PL/Python
- PL/Perl
- PL/Tcl

Расширения. Проблема №1

Обёртки внешних данных

- dblink_fdw
- oracle_fdw
- tds_fdw
- и т.д.

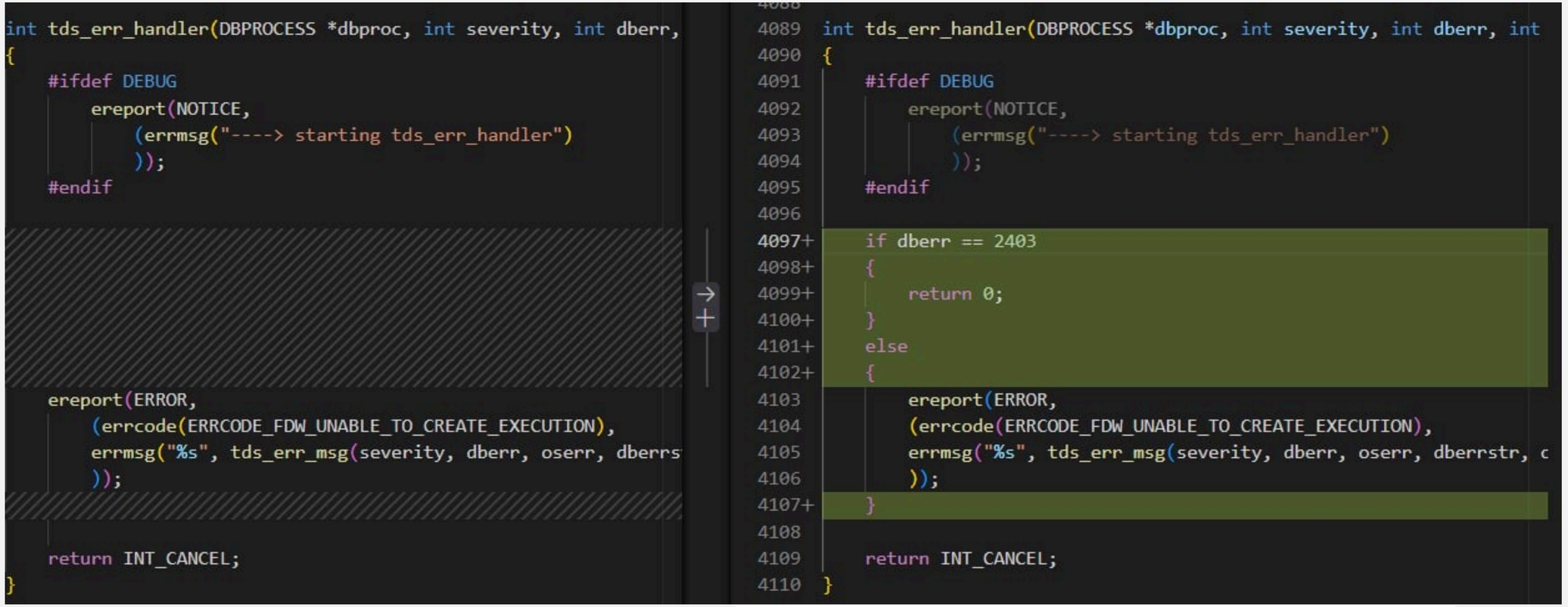
1. Базы все разные

2. Зависимость от библиотек
Просто не можем обновиться.
Нужно компилировать снова и снова

3. Отсутствие пула соединений

4. Кодировки

tds_FDW for MS SQL. Кодировки



```
int tds_err_handler(DBPROCESS *dbproc, int severity, int dberr,
{
    #ifdef DEBUG
        ereport(NOTICE,
            (errmsg("----> starting tds_err_handler")
            ));
    #endif

    ereport(ERROR,
        (errcode(ERRCODE_FDW_UNABLE_TO_CREATE_EXECUTION),
        errmsg("%s", tds_err_msg(severity, dberr, oserr, dberrs
        ));

    return INT_CANCEL;
}
```

```
4089 int tds_err_handler(DBPROCESS *dbproc, int severity, int dberr, int
4090 {
4091     #ifdef DEBUG
4092         ereport(NOTICE,
4093             (errmsg("----> starting tds_err_handler")
4094             ));
4095     #endif
4096
4097+    if dberr == 2403
4098+    {
4099+        return 0;
4100+    }
4101+    else
4102+    {
4103         ereport(ERROR,
4104         (errcode(ERRCODE_FDW_UNABLE_TO_CREATE_EXECUTION),
4105         errmsg("%s", tds_err_msg(severity, dberr, oserr, dberrstr, c
4106         ));
4107+    }
4108
4109     return INT_CANCEL;
4110 }
```

Расширения PL/Языки. Проблема №2

PL/Python – Все может. Но безумно медленный и ресурсоемкий.

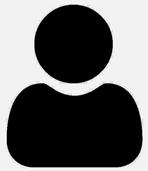
PL/Perl – Все может. Работает на уровне ОС и опасен

PostgreSQL и JSON

Web

от SQL к

JSON



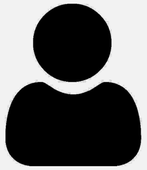
Postgres

Расширения. Решение №1

Postgres

решаем транспортную задачу
исходящего трафика

pgsql-http



только SQL+PL/pgSQL

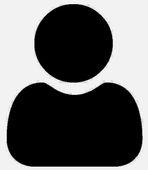
API

Расширения. Решение №2

Postgres

решаем транспортную задачу
входящего трафика

xDAC



только SQL+PL/pgSQL

API

Публикуем данные. Пример №1

```
-- Функция в PostgreSQL
CREATE OR REPLACE FUNCTION api.f_hello()
 RETURNS json LANGUAGE plpgsql
AS $function$
BEGIN
    RETURN '{"GET_RETURN": "Hello world!!"}'::json;
END;
$function$;
```

```
curl "http://127.0.0.1:8887/xdac/rpc/f_hello" -X GET
```

Публикуем данные. Пример №2

```
--Функция в PostgreSQL с передачей параметров
CREATE OR REPLACE FUNCTION api.f_concat_str(p_first_str character varying DEFAULT '',
                                            p_second_str character varying DEFAULT '')
 RETURNS json LANGUAGE plpgsql
AS $function$
DECLARE
    l_result VARCHAR;
BEGIN
    l_result := CONCAT(p_first_str,p_second_str);

    RETURN ('{"CONCAT_RESULT": "'||l_result||'"}')::JSON;
END;
$function$;
```

```
curl -H "Content-Type: application/json" -d '{"p_first_str": "", "p_second_str": "" }' "http://127.0.0.1:8887/xdac/rpc/f_concat_str"
```

Публикуем данные. Пример №3

```
-- Функция в PostgreSQL с передачей JSON или JSONB
CREATE OR REPLACE FUNCTION api.f_json_parse(p_json json)
 RETURNS json LANGUAGE plpgsql
AS $function$
DECLARE
BEGIN
    RETURN ('{"JSON_OUTPUT": '||p_json::TEXT||'}')::JSON;

EXCEPTION WHEN OTHERS
THEN
    RETURN 'Некорректный формат JSON!';
END;
$function$;
```

```
curl -H "Content-Type: application/json" -H "Prefer: params=single-object"
-d '{"first_value": 1, "second_value": "two", "third_value": [1, "two", "3"]}' "http://127.0.0.1:8887/xdac/rpc/f_json_parse
```

Публикуем данные. Пример №4

```
--Функция в PostgreSQL. Получаем в JSON сведения о клиенте по ID из списка клиентов


CREATE OR REPLACE FUNCTION api.f_get_client(p_id bigint)
 RETURNS json LANGUAGE plpgsql
AS $function$
    DECLARE
        l_client_info json;
    BEGIN
        SELECT json_build_object('fi',            CONCAT(c.last_name,' ',c.first_name),
                                 'date_of_birth', to_char(c.date_of_birth, 'dd.mm.yyyy'),
                                 'phone',         c.phone,
                                 'email',         c.email)
        INTO l_client_info
        FROM cli.t_clients c
        WHERE c.id = p_id;

        RETURN l_client_info;
    END;
$function$;
```

curl "http://127.0.0.1:8887/xdac/rpc/f_get_client?p_id=2" -X GET

Публикуем данные. Пример №5

```
-- Функция в PostgreSQL.  Пример массовой выгрузки из таблицы
CREATE OR REPLACE FUNCTION api.f_get_mass()
    RETURNS json LANGUAGE plpgsql
AS $function$
    DECLARE
        l_mass_clients JSON;
    BEGIN

        WITH tbl AS (
            SELECT c.last_name, c.first_name, c.date_of_birth, c.email, c.phone
            FROM cli.t_clients c
            LIMIT 3
        )
        SELECT JSON_AGG(tbl.*) as mass_clients
        INTO l_mass_clients
        FROM tbl;

        RETURN l_mass_clients;

    END;
$function$;
```

```
curl "http://127.0.0.1:8887/xdac/rpc/f_get_mass" -X GET
```

Публикуем данные. Пример №6 или для ленивых

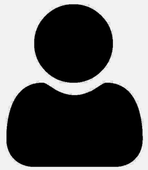
```
-- Функция в PostgreSQL или для ленивых
create or replace function api.f_get_user_xml() returns bytea
    language plpgsql
as
$$
declare
    L_mass_clients text;
begin
    select query_to_xml('select * from users.t_user_auth_log LIMIT 8',false,false,'')::text into L_mass_clients;
    return L_mass_clients::bytea;
end;
$$;
```

curl "http://127.0.0.1:8887/xdac/rpc/ f_get_user_xml" -X GET

Получаем данные. Пример №1

решаем транспортную задачу
исходящего трафика. HTTP **GET**

pgsql-http



```
select content::json data
  from
http_get('http://my_ip/xdac/f_hello') t
```

Получаем данные. Пример №2

Метод **POST**
и
Timeout

```
create function f_get_message_status(p_message_id character varying) returns json
    language plpgsql
as
$$
DECLARE
  l_session varchar(500);
  l_data    varchar;
  l_resp    http_response;
  l_url     varchar;
BEGIN
    l_session := sms.f_login();
    l_data := 'sessionId='||l_session||
              '&messageId='||(p_message_id::json->>0);
    l_url := pref.f$c1( p_code 'SMS_ADDRESS')||'/Sms/State';

    perform http_set_curlopt('CURLOPT_TIMEOUT','300');
    l_resp := http_post(l_url,l_data,'application/x-www-form-urlencoded');
    return l_resp.content::json;
END;
$$;
```

Получаем данные. Пример №3

```
root@lcdp4-edu.xsquare:~# curl -X POST -H "X-Forwarded-For: 127.0.0.1" -
H "Cookie: session=123456789" -H 'Content-Type: application/json' --data
 '{
  "id": "00000000-0000-0000-0000-000000000000",
  "fileName": "example.txt",
  "fileContent": "UEsDBBQACAgIA...",
  "mimeType": "text/plain",
  "type": "File"
}' http://localhost:9000/add_file
```

HTTP-сервис принимает запрос
и устанавливает cookies

```
l_header := current_setting('request.headers', true)::json;

l_ext := l_header->>'X-Forwarded-For';

l_arr := string_to_array(l_headers->>'Cookie',';');

FOREACH l_entry IN ARRAY l_arr

LOOP
  l_arr_ent := string_to_array(l_entry,'=');
  l_cookie := l_cookie||hstore(trim(l_arr_ent[1]),trim(l_arr_ent[2]));
END LOOP;
```

Передаёт **хедеры**
через параметры конфигурации

```
CALL files.add_file(p_id           =>  (p_json->>'id')::uuid,
                    p_file_name    =>  (p_json->>'fileName'),
                    p_file_content =>  decode((p_json->>'fileContent')
                                          ,'base64'
                                          ),
                    p_mime_type    =>  (p_json->>'mimeType'),
                    p_file_type    =>  (p_json->>'type'),
                    p_id_user      =>  (l_cookie->'session'));
```

Тело запроса
как параметр вызова функции

FWD. Кандидат на вылет

Postgres

~~fdw~~

Postgres

Oracle

DB2

MS SQL

MySQL

MS Excel

FWD. Postgre select * from Oracle

```sql
-- превращаем Oracle а таблицу в Postgres
WITH json_data AS (
  select content::json data
    from http_get( uri 'http://10.150.117.205:8890/xdac/rpc/F_GET_ORA_CLIENTS') t
)
SELECT
  elem.value->>'id',
  elem.value->>'last_name',
  elem.value->>'first_name',
  elem.value->>'email'
FROM
  json_data,
  json_array_elements(data) elem;
```

```sql
-- Делаем таблицу в Oracle в виде JSON
create or replace function xdac_demo.f_get_ora_clients return clob as
        l_client_info clob;
    begin
        -- FUNCTION IN ORACLE

        select JSON_ARRAYAGG(
               json_object(
                       key 'id' value to_char(id),
                       key 'first_name' value first_name,
                       key 'last_name' value last_name||' tantor',
                   key 'email' value email
--                         returning CLOB
               )
        )
        into l_client_info
        from t_clients;


    return l_client_info;
end;
```

curl "http://127.0.0.1:8887/xdac/rpc/ f_get_ora_clients" -X GET

FWD. Postgre select * from Postgre

```sql
WITH json_data AS (
  select content::json data
    from http_get( uri 'http://10.150.117.205:8890/xdac/rpc/F_GET_ORA_CLIENTS') t
)
SELECT
  elem.value->>'id',
  elem.value->>'last_name',
  elem.value->>'first_name',
  elem.value->>'email'
FROM
  json_data,
  json_array_elements(data) elem;
```

f_get_user_xml

```sql
-- Функция в PostgreSQL или для ленивых
create or replace function api.f_get_user_xml() returns bytea
    language plpgsql
    select query_to_xml('select * from users.t_user_auth_log LIMIT 8',false,false,'')::text into L_mass_clients;
    return L_mass_clients::bytea;
end;
```

FWD. select * from Postgre

```
WITH json_data AS (
  select content::json data
    from http_get( uri 'http://10.150.117.205:8890/xdac/rpc/F_GET_ORA_CLIENTS') t
)
SELECT
  elem.value->>'id',
  elem.value->>'last_name',
  elem.value->>'first_name',
  elem.value->>'email'
FROM
  json_data,
  json_array_elements(data) elem;
```

f_get_user_xml

```
-- Функция в PostgreSQL или для ленивых
create or replace function api.f_get_user_xml() returns bytea
    language plpgsql
    select query_to_xml('select * from users.t_user_auth_log LIMIT 8',false,false,'')::text into L_mass_clients;
    return L_mass_clients::bytea;
end;
```

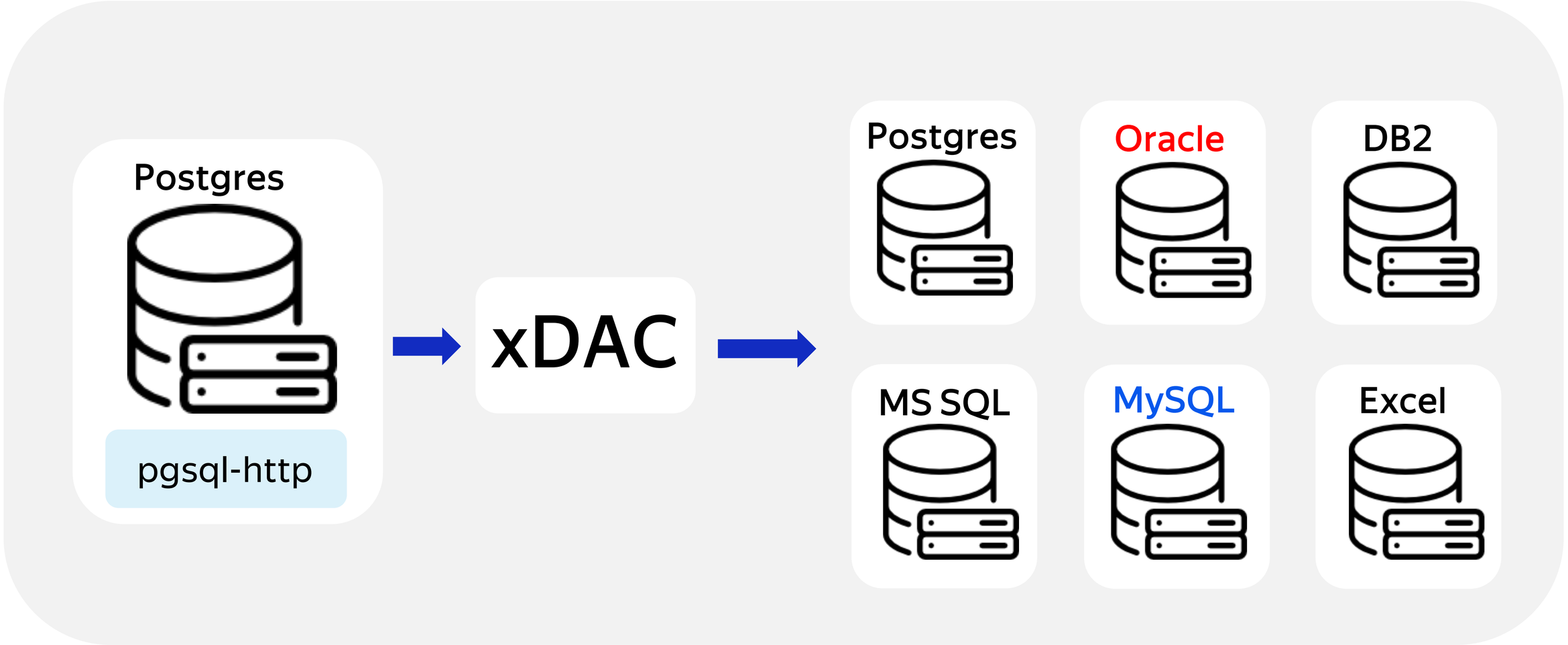
FWD. Поможем Oracle)))

SQL Output Statistics

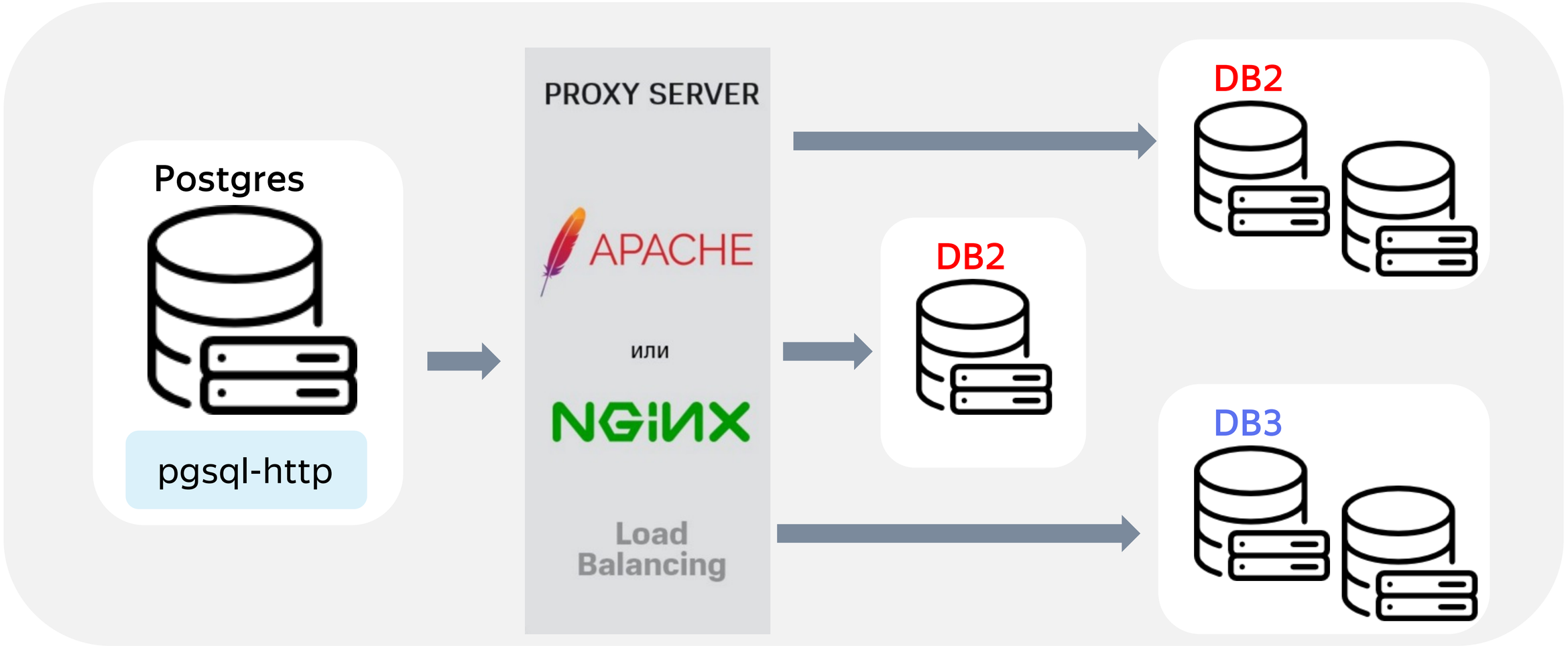
```
-- превращаем Postgres в таблицу в PostgreSQL
select x.*
from xmltable (
                '/table/row'
                passing xmltype(httpuritype.createUri('http://10.150.117.205:8887/xdac/rpc/f_get_user_xml').getblob(),nls_charset_id('AL32UTF8'))
     columns
      id varchar2(100) path 'id'
      ,id_user varchar2(100) path 'id_user'
      ,date_created varchar2(100) path 'date_created'
      ,id_addr varchar2(100) path 'id_addr'
) x
;
```

```
1   -- Функция в PostgreSQL или для ленивых
2   create or replace function api.f_get_user_xml() returns bytea
3       language plpgsql
9       select query_to_xml('select * from users.t_user_auth_log LIMIT 8',false,false,'')::text into L_mass_clients;
10      return L_mass_clients::bytea;
11  end;
```

FWD. Один ко многим

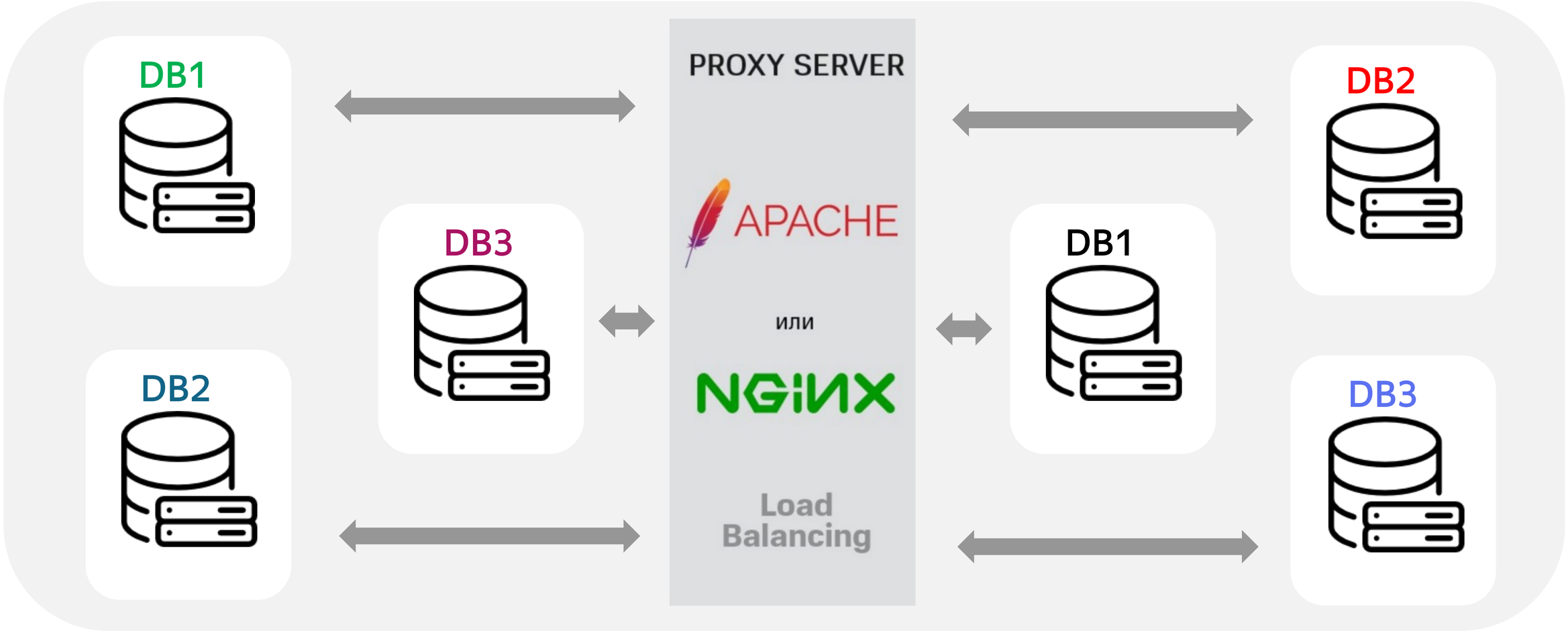


FWD. Один ко многим с балансировкой



FWD. Многие ко многим с балансировкой.

или как мы победили шардининг

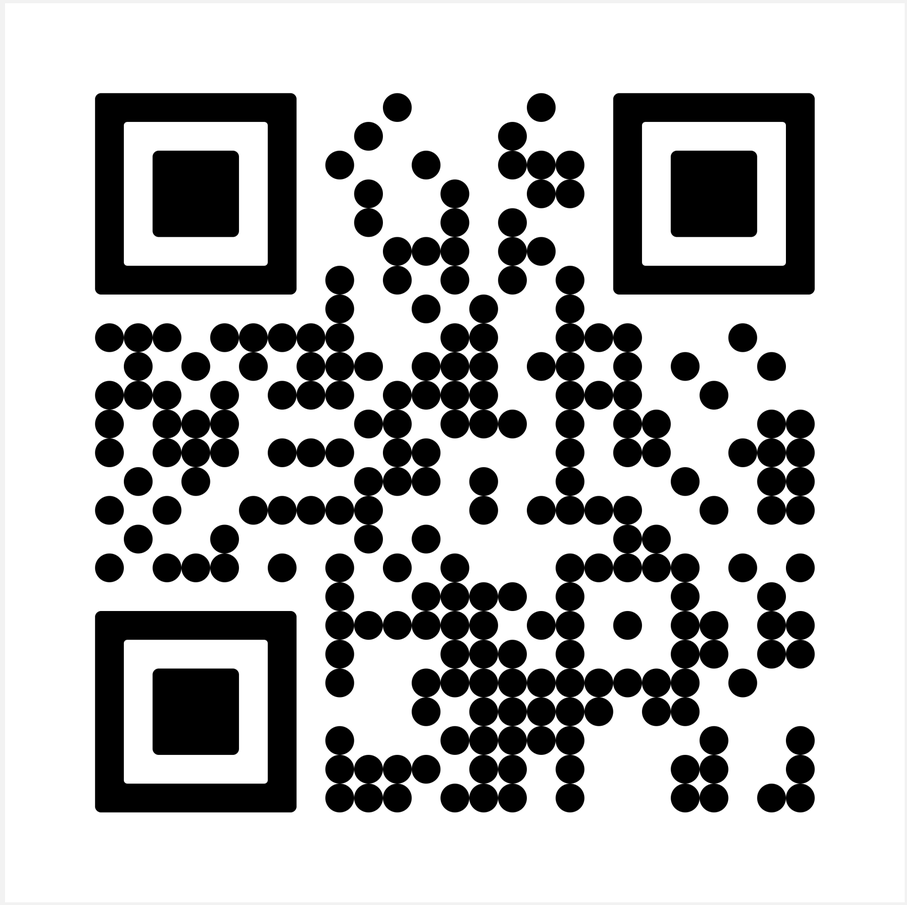


Практика. Подключаем Гос Услуги/ЕСИА

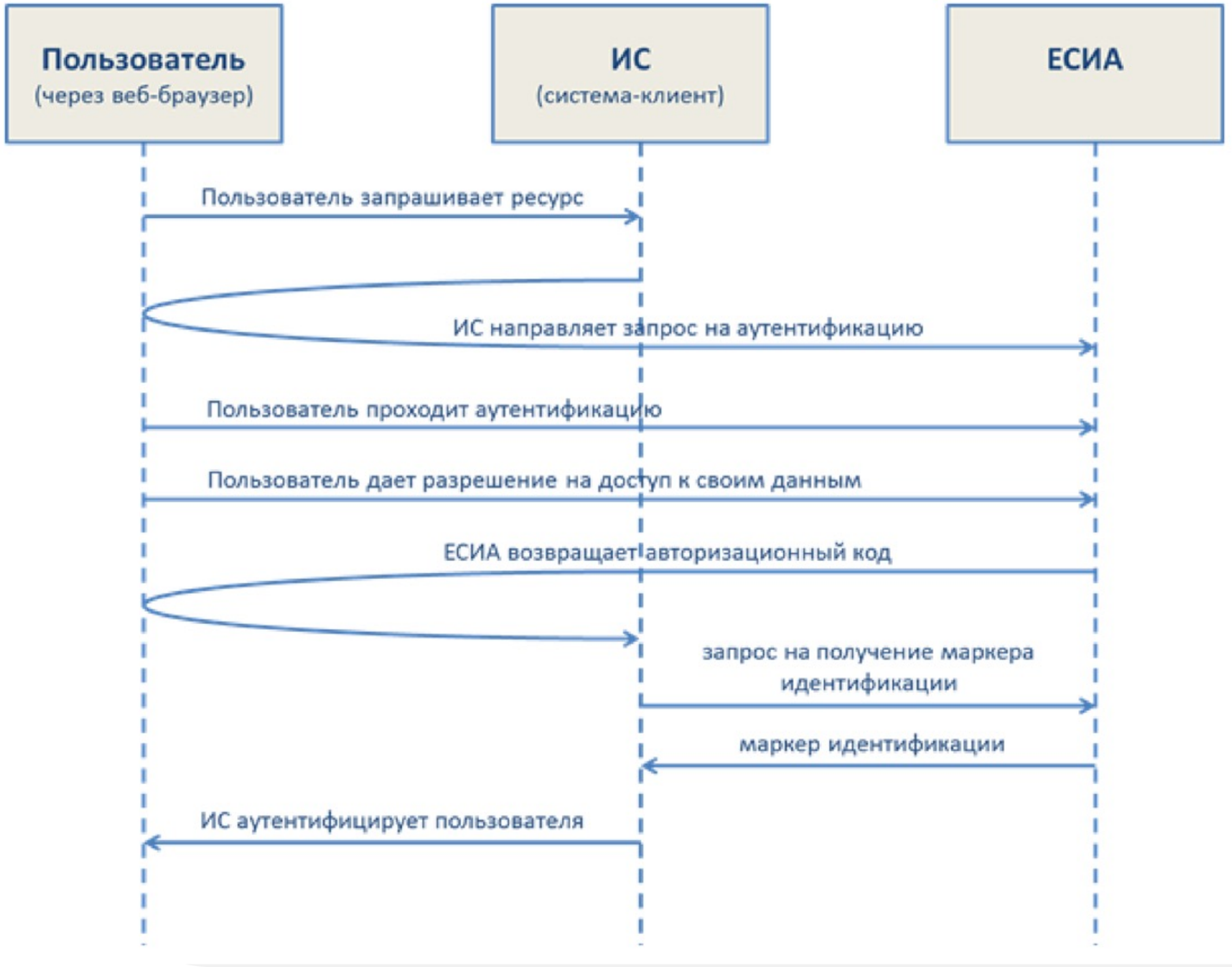
ЕДИНАЯ СИСТЕМА
ИДЕНТИФИКАЦИИ И АУТЕНТИФИКАЦИИ

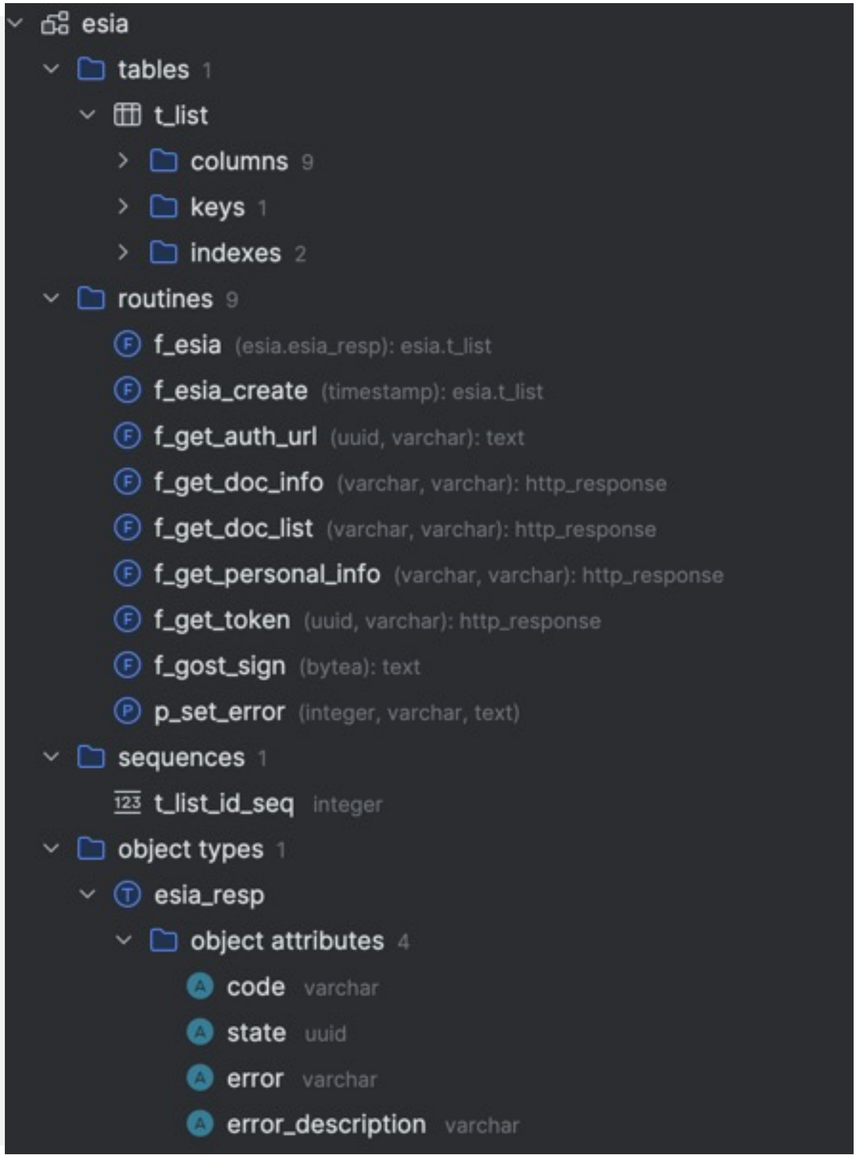
Методические рекомендации
по использованию
Единой системы идентификации и аутентификации
Версия 2.84

Практика. Раскрываем коды клиента к ЕСИА



Практика. Протокол. Стр. 47





Практика. ЕСИА. AUTH

```
create function f_get_auth_url(p_state uuid, p_url character varying DEFAULT NULL::character varying) returns text
    language plpgsql
as
$$
DECLARE
  l_timestamp   VARCHAR(25);
  l_secret      TEXT;
  l_ret         TEXT;
BEGIN
  l_timestamp := TO_CHAR(CURRENT_TIMESTAMP,'YYYY.MM.DD HH24:MI:SS TZHTZM');
  l_secret := esia.f_gost_sign( p_data (CONCAT(  pref.f$c1( p_code 'ESIA_SCOPE'),
                                                 l_timestamp,pref.f$c1( p_code 'ESIA_CLIENT_ID')
                                                 ,p_state::VARCHAR))::BYTEA);
  l_ret := CONCAT(pref.f$c1( p_code 'ESIA_AUTH')
                 ,'?scope=',urlencode(pref.f$c1( p_code 'ESIA_SCOPE'))
                 ,'&redirect_uri=',urlencode(COALESCE(p_url,pref.f$c1( p_code 'ESIA_RETURN_URL')))
                 ,'&client_id=',pref.f$c1( p_code 'ESIA_CLIENT_ID')
                 ,'&timestamp=',urlencode(l_timestamp)
                 ,'&state=',p_state::VARCHAR
                 ,'&response_type=code'
                 ,'&client_secret=',urlencode(l_secret)
                 ,'&access_type=online');
  RETURN l_ret;
EXCEPTION
.............
END;
$$;
```

Практика. ЕСИА. get_Token

```
create function f_get_token(p_state uuid, p_code character varying) returns http_response
BEGIN
  l_timestamp := TO_CHAR(CURRENT_TIMESTAMP,'YYYY.MM.DD HH24:MI:SS TZHTZM');
  l_secret := esia.f_gost_sign( p_data (CONCAT(pref.f$c1('ESIA_SCOPE'),l_timestamp,pref.f$c1('ESIA_CLIENT_ID'),p_state::VARCHAR))::BYTEA);

  l_body := CONCAT('client_id=',pref.f$c1('ESIA_CLIENT_ID')
                  ,'&code=',p_code
                  ,'&grant_type=authorization_code'
                  ,'&client_secret=',urlencode(l_secret)
                  ,'&state=',p_state::VARCHAR
                  ,'&redirect_uri=',urlencode(pref.f$c1('ESIA_RETURN_URL'))
                  ,'&scope=',urlencode(pref.f$c1('ESIA_SCOPE'))
                  ,'&timestamp=',urlencode(l_timestamp)
                  ,'&token_type=Bearer'
                  ,'&access_type=online');

  PERFORM http_set_curlopt('CURLOPT_TIMEOUT','300');
  PERFORM http_set_curlopt('CURLOPT_CONNECTTIMEOUT','5');
  l_req.method  := 'POST';
  l_req.uri     := pref.f$c1('ESIA_TOKEN');

  l_req.content_type := 'application/x-www-form-urlencoded';
  l_req.content := l_body;
  l_resp := http(l_req);
  RETURN l_resp;
```

Практика. ЕСИА. get_Personal_Info

```
create function f_get_personal_info(p_token character varying, p_client_id character varying) returns http_response
DECLARE
  l_req         http_request;
  l_resp        http_response;
BEGIN
  PERFORM http_set_curlopt('CURLOPT_TIMEOUT','300');
  PERFORM http_set_curlopt('CURLOPT_CONNECTTIMEOUT','5');

  l_req.method  := 'GET';
  l_req.uri     := pref.f$c1('ESIA_PRNS')||'/'||p_client_id;

  l_req.headers := array_append(l_req.headers
                               ,http_header('authorization','Bearer '||p_token));
  l_resp := http(l_req);
  RETURN l_resp;
```

PG BootCamp Russia 2025 Ekaterinburg

Спасибо за внимание!

Константин Ващенков
kv@xsquare.ru
xsquare.ru

@XSQUARE365