Опыт автоматизированной миграции СДИ БАЗИС с Oracle на Postgres Pro

Александр Лаврентьев, Максим Шарлаев

Февраль, 2024

Характеристика СДИ БАЗИС

Предыстория миграции

Организация проекта

Архитектура

Результаты

Примеры

// Характеристика СДИ БАЗИС

Система технического учета

- Документирование инженерной, телеком- и ит-инфраструктуры
- Учет, планирование, визуализация, навигация

История

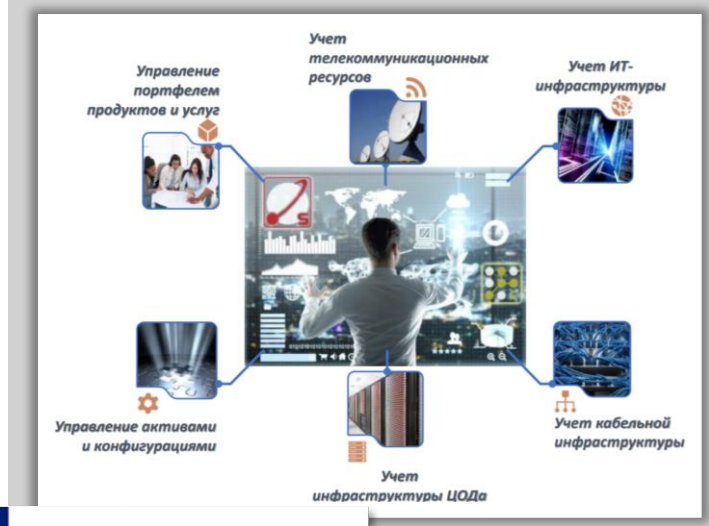
- С 1994 по 2019 FNT Command, 600+ клиентов в мире, входит в десятку лидеров DCIM
- В 2019 в результате многолетнего успешного партнерства FNT и СДИ код российской версии Command юридически передан СДИ и стал БАЗИС'ом
- С 2020 независимый отечественный продукт, включен в реестр Минкомсвязи России
- 20+ клиентов в РФ и СНГ

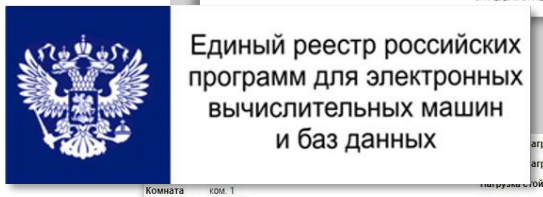
Технологическая эволюция

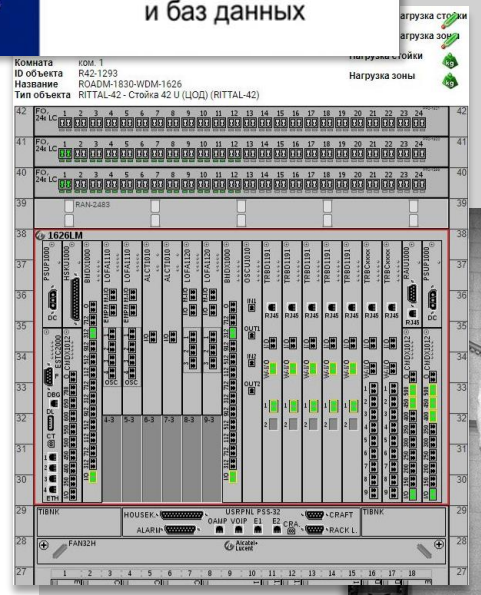
- В 90х: САПР СКС с толстым клиентом и алгоритмическим ядром на PL/SQL
- В 00х: технологический акцент смещен на Java и web
- В 10х: логика новых модулей реализована на Java, но алгоритмы ядра и унаследованных модулей, объемом около полумиллиона строк остались на PL/SQL

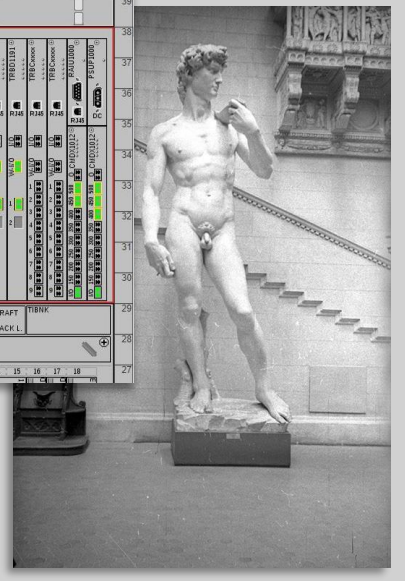
Особенности унаследованного кода PL/SQL

- Богатая система типов (330 объектных типов, 135 пакетов)
- Сложные вычислительные алгоритмы объемом 1000+ строк
- Широкое использование коллекций Oracle, включая ассоциативные
- Динамический SQL как в хранимых процедурах, так и в Java
- Сотни человеко-лет потрачены на отладку и оптимизацию









// Предыстория миграции на Postgres

Стратегическая задача стоит не в смене PL/SQL на иной *SQL диалект, а в переносе унаследованного кода Oracle на уровень Java сервисов

- Инкапсуляция функций ядра в драйвере СУБД
- Создание DAO слоя для сложной бизнес-логики

С 2020г выполнялись подготовительные работы в фоновом режиме

- Поиск и удаление мертвого кода, сужение функциональных рамок, очистка зависимостей: -50% кода PL/SQL
- Консультации с PostgresPro, поиск внешних компаний на перенос PL/SQL кода в PL/pgSQL: оценки большие, сравнимые с оценкой перевода на Java
- Реализовали PoC с прокси-драйвером и переписали вручную пару пакетов. Пробовали доработать ora2pg, чтобы упростить последующую правку кода PL/pgSQL. Экспериментировали с коммерческими транспайлерами (Strumenta, Ispirer)

PostgresPro

STRUMENTA

Уход Oracle из России в 2022г актуализировал тему Postgres

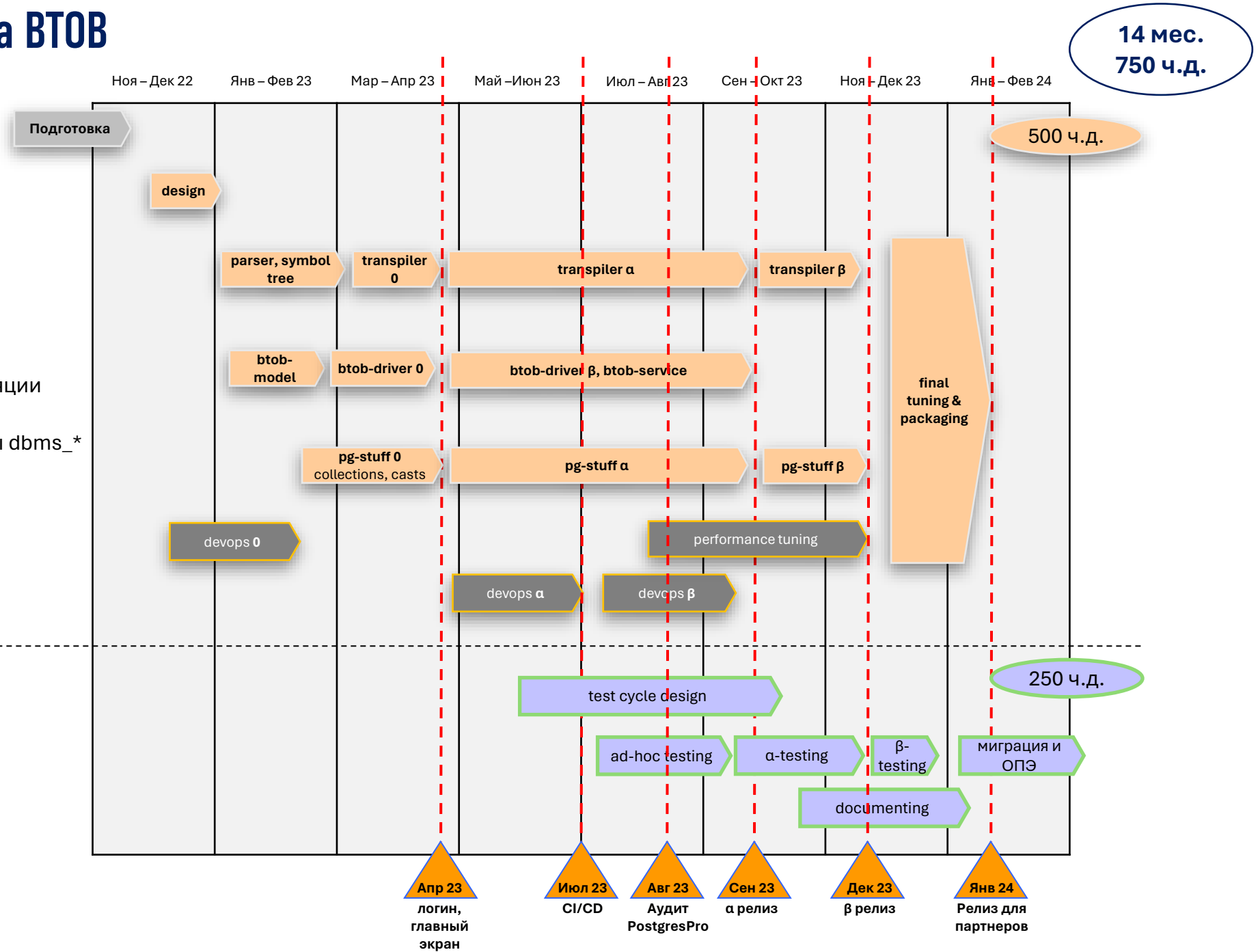
- Возможно, мы бы не решились самостоятельно разрабатывать транспайлер PL/SQL, если бы не нашли в архиве PgConf 2020 доклад архитекторов АК БАРС:
  “Миграция с Oracle на PostgreSQL с использованием автоматического конвертера”
- К сожалению, АК БАРС не стал выводить это решение на рынок, но доклад убедил нас пойти тем же путем
- В декабре 2022г открыли проект трансляции и миграции с помощью парсера ANTLR4
- В январе 2024г выпустили релиз СДИ БАЗИС 14.0 для Postgres Pro и vanilla

ANTLR

// Организация проекта BTOB

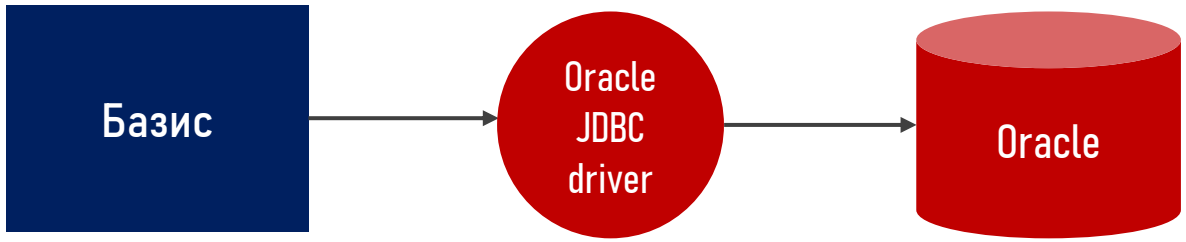
- Выделение минимального этапа
- Ресурсы, план-график
  - Архитектура, декомпозиция
  - Спецификация интерфейсов
- Парсер, таблица символов
- Генерация кода
- Модульные тесты
  - JDBC прокси-драйвер
  - Сервис динамической трансляции
- Коллекции, касты, служебные пакеты dbms_*
- Override функции
  - Тестовые и проектные среды
  - Мониторинг и настройка производительности
  - Jenkins, автоматизация

- 1500+ функциональных сценариев
- 300+ ошибок найдено и закрыто

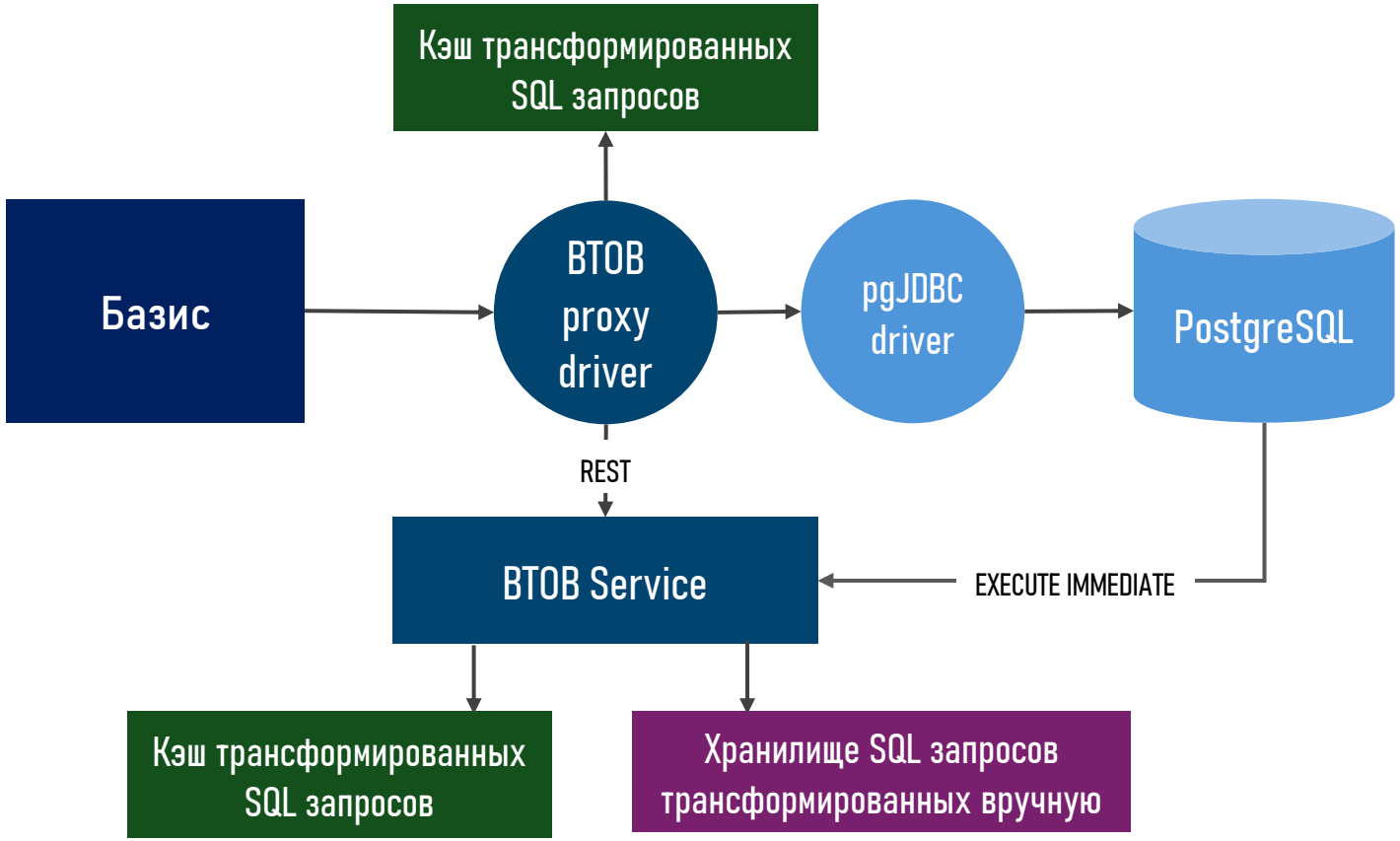


// Архитектура времени выполнения (runtime)

Было



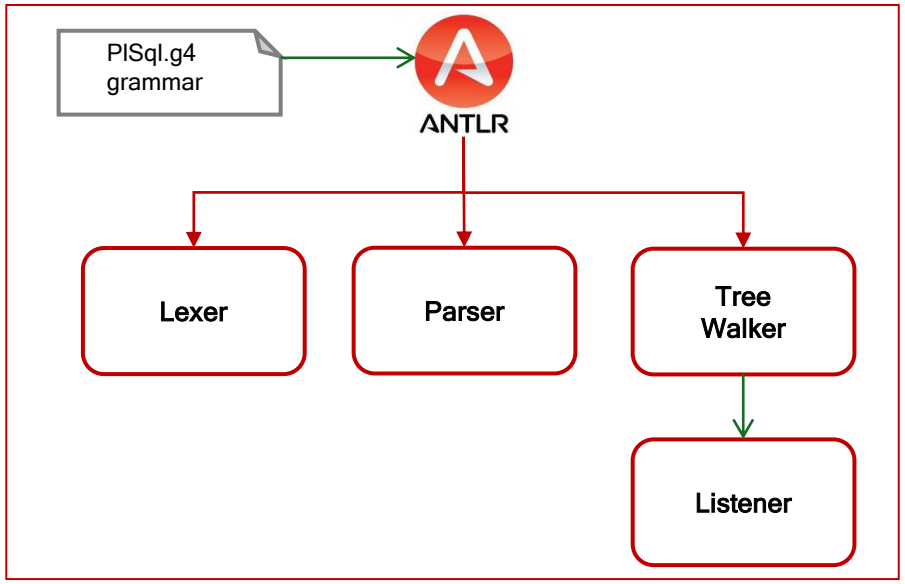
Стало



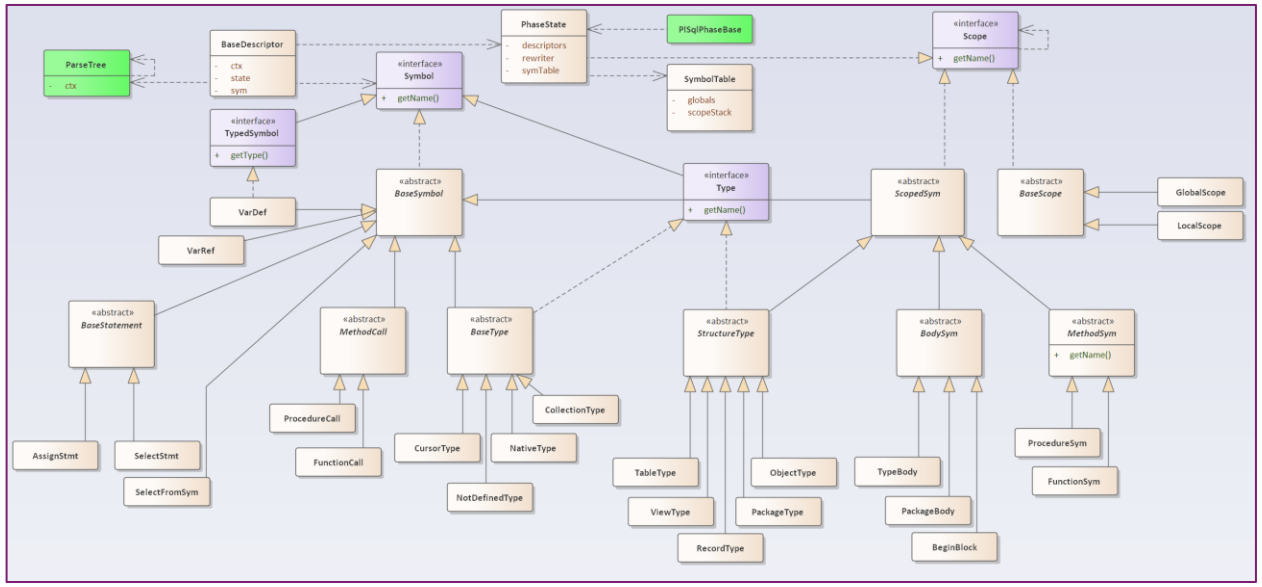
Функции реализованные в BTOB Driver

- Обращение к BTOB Service для получения трансформированных SQL-запросов
- Кэш трансформированных запросов для оптимизации количества обращений к сервису трансформации
- Передача структурных типов в/из PostgreSQL. В нативном драйвере работа с java.sql.Struct не реализована.
- Работа с массивами с учетом собственной реализации коллекций
- Работа с BLOB-полями. Своя реализация java.sql.Blob.
- Работа с CLOB-полями. Своя реализация java.sql.Clob. В PostgreSQL специализированного типа CLOB нет, используется тип text или varchar
- Работа с числовыми типами по аналогии с Oracle JDBC Driver
- В SQLException значения поля SQLState переносится в поле vendorCode
- Прочие мелкие фиксы для того, чтобы поведение было максимально близко к Oracle JDBC Driver

// Основные компоненты транспайлера PL/SQL



1. Компоненты, сгенерированные ANTLR

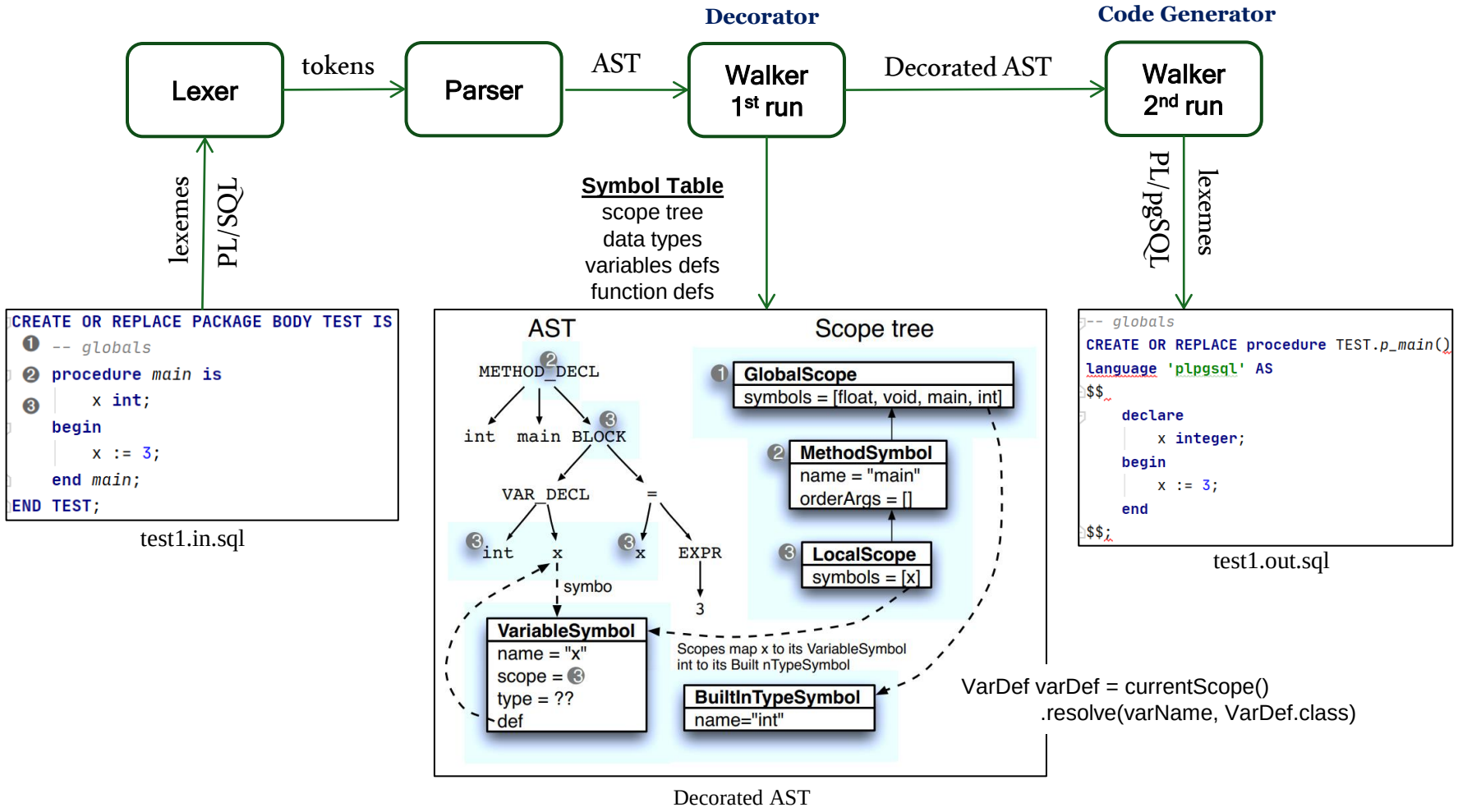


2. Фрагмент диаграммы классов таблицы символов PL/SQL

```
MethodParameterDef(param_def) ::= <<
%param_def.inOut% %param_def.name% %param_def.type.pgName%%if(param_def.defaultValue)% DEFAULT %param_def.defaultValue%%endif%
>>

MethodParameterDefNoDef(param_def) ::= <<
%param_def.inOut% %param_def.name% %param_def.type.pgName%
>>

GetGlobalVar(package_name, name, fields, type) ::= <<
%if(fields)%(%endif%%package_name%.get_%name%()%if(fields)%::%type%).%fields; separator="."%%endif%
>>

SetGlobalVar(package_name, name, value) ::= <<
CALL %package_name%.set_%name%(%value%)
>>
```

3. Фрагмент библиотеки шаблонов (String Templates) генерации кода PL/pgSQL

// Схема трансляции PL/SQL



- Парсер строит AST (abstract syntax tree) для последующих обходов
- На первом обходе строится Symbol Table – иерархическая структура областей видимости (Scope). Область видимости содержит карту символов (Symbols), определенных в ней. Scope умеет резолвить символы в своем и окружающем скоупе, а также среди своих членов (в случае структур типа record). Соответствие между узлами AST и символами сохраняется в карте дескрипторов
- На втором обходе для каждого узла AST обходчик проверяет определен ли для него дескриптор и вызывает на нем render()
- Дескриптор в методе render() либо сам модифицирует представление узла дерева, либо делегирует трансформацию символу узла. Далее дескриптор либо полностью заменяет представление узла на трансформацию, либо добавляет новые элементы до или после него

// Некоторые особенности трансляции

Глобальный контекст (объектные типы, заголовки пакетов) транслируется один раз при миграции и сериализуется. Сервис динамической трансляции при старте десериализует глобальный контекст и далее корректно определяет его символы в JIT режиме EXECUTE IMMEDIATE

Трансляция базовых типов

- Не используются числовые типы переменного размера (numeric, decimal)[1]
- Ряд операторов перегружены для базовых типов (арифметика varchar + double, сравнение varchar с date)

Автономные транзакции используются только для логирования

- PostgresPro: из коробки
- PostgreSQL: только логирование критичных событий через dblink

Ограничения транслятора

- Версия PL/SQL: 11.2
- Директивы компилятора игнорируются
- Переписаны вручную редкие конструкции: GOTO, xmlagg, xmlsequence, bulk_exceptions и другие
- Объем оттранслированного кода в среднем на 20% больше оригинала[2]

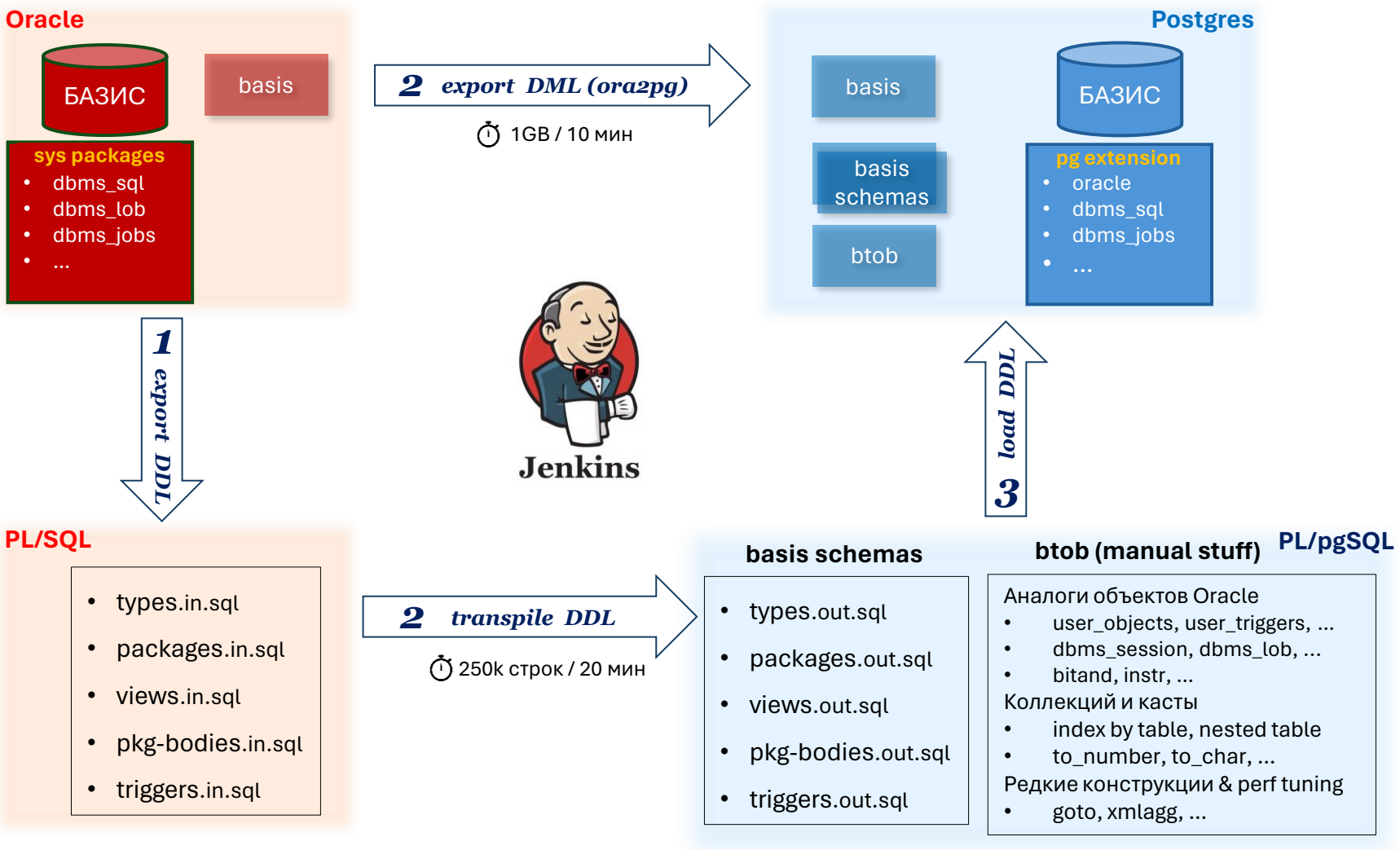
Алгоритмы трансляции по своей природе рекурсивны и полиморфны, потому компактны, но не интуитивны. Важно контролировать регресс с помощью тестов

---

[1] Исключение составляют адреса IPv6, внутреннее представление которых не помещается в 8 байт нативных типов Postgres

[2] Использование поддержки пакетов PostgresPro сделает код более компактным

// Схема CI/CD цикла трансляции и миграции



Контроль регресса

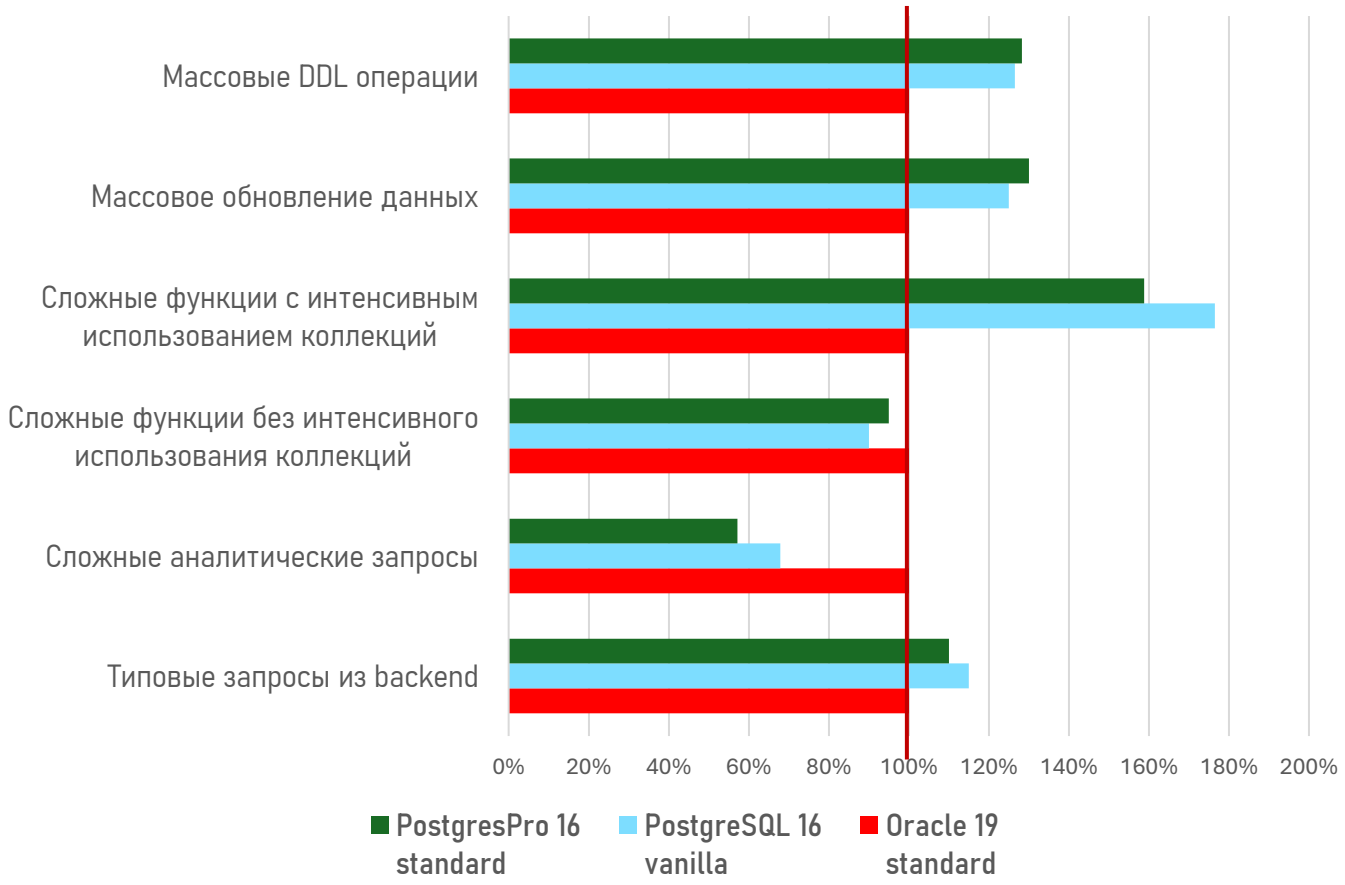
- Полный CI/CD цикл с пересозданием стендов тестирования: ~40 минут
- Быстрый цикл с тестами на коммит: ~7-10 минут
- Коммит автоматически отклоняется, если тесты падают

Тесты

- Синтаксические на контроль отдельных трансформаций
- Функциональные (pgTap)
- На производительность по группам типовых операций

Производительность

Нормированное время выполнения групп тестов на производительность[1]



В среднем пользователи не видят разницы в работе стендов БАЗИС Oracle vs Postgres

Исключением являются операции с интенсивным созданием и обновлением многоуровневых коллекций:

- вычисление ACL первого логина пользователя
- вычисление ACL при открытии объекта во внешнем модуле

[1] Ресурсы виртуальных машин одинаковы для всех типов СУБД: 1 virtual processor, RAM 4GB, SSD

// Результаты

Подготовили и выпустили релиз БАЗИС для СУБД Postgres Pro и PostgreSQL в течение 14 месяцев

Полное тестирование проводилось для версий:

- Postgres Pro 15 и 16 Standard
- PostgreSQL vanilla 15 и 16
- Pro Enterprise Certified[1]

Производительность БАЗИС для Postgres Pro и PostgreSQL сравнима с версией для Oracle

Объем оттранслированного кода составил 250k строк PL/SQL

Объем созданного инструментария составил 36k строк (75% Java, 25% PL/pgSQL):

- транспайлер (таблица символов, декоратор, генератор кода, тесты): 25k строк
- прокси-драйвер и сервис динамической трансляции: 5k строк
- функции plpgsql (оптимизированные вручную, коллекции, касты): 5k строк
- DevOps, автоматизация сборки и миграции данных: 1k строк

Полные трудозатраты составили 750 чел.дней:

- 500 чел.дней на разработку, включая unit tests и DevOps
- 250 чел.дней на QE (функциональные сценарии и ручное тестирование) и документирование

[1] На момент подготовки презентации тестирование для версии Postgres Pro Enterprise Certified не закончено

// Что понравилось и не понравилось в Postgres

ПОНРАВИЛОСЬ

- Удобный и легкий инструментарий администрирования и мониторинга[1]
- Апгрейд Postgres Pro без остановки сервера
- Производительность
- Открытость и современность, соответствие стандартам ANSI вплоть до SQL:2011, частичная поддержка SQL:2016 (JSON)
- Хорошая документация и community
- Большое количество полезных открытых расширений
- Улучшенная поддержка миграции с Oracle в Postrges Pro
- Хорошая интеграция со множеством языков программирования и платформ разработок
- Возможность перегрузки операторов
- Приятные limit, offset, serial, raise notice record

НЕ ПОНРАВИЛОСЬ

- Не поняли как апгрейдить ванильный Postgres с учетом зависимостей от расширений, поэтому пока устанавливаем с нуля и переносим бэкап
- Нет готовой поддержки контейнерных баз (Multitenant)
- Отсутствие джобов на стороне сервера в ванильной версии и в Pro Standard (есть в Pro Enterprise)
- Отсутствие в ванильной версии поддержки ассоциативных коллекций, пакетов, автономных транзакций (есть в Pro Standard)
- Громоздкая конструкция иерархического запроса (with…)
- Нет поддержки интерфейсов java.sql.Struct, java.sql.SQLData в pgJDBC
- Слишком уж незатейливый алгоритм приведения типов при поиске перегруженных функций (overloading)
- Каст numeric и decimal к varchar порождает “.0”
- Нет проверки зависимостей при компиляции функций

[1] Было бы не плохо иметь из коробки несколько профилей конфигурации СУБД: для массовых ETL операций, для OLTP, для OLAP

// Заключение

Автоматизированная трансляция принципиально эффективнее ручной миграции[1]

- Заменяет редкую экспертизу предметной области на распространенную техническую с хорошей поддержкой IDE
- Заменяет логические ошибки переписанных вручную алгоритмов на более простые технические ошибки генерации кода
- Дает бонусный инструментарий рефакторинга, оптимизации и последующего перевода кода на другой SQL диалект или язык общего назначения

Удачные практики

- Ранняя реализация CI/CD цикла с авто-тестами минимизирует потери на регресс
- Использование расширений Postgres (orafce, pg_variables, dbms_jobs и др.) существенно упрощает трансляцию
- Небольшой рефакторинг исходного кода или формальной грамматики порой выгоднее, чем усложнение транспайлера
- Ручная оптимизация наиболее ресурсоемких функций и уникальных конструкций PL/SQL[2]

От чего отказались

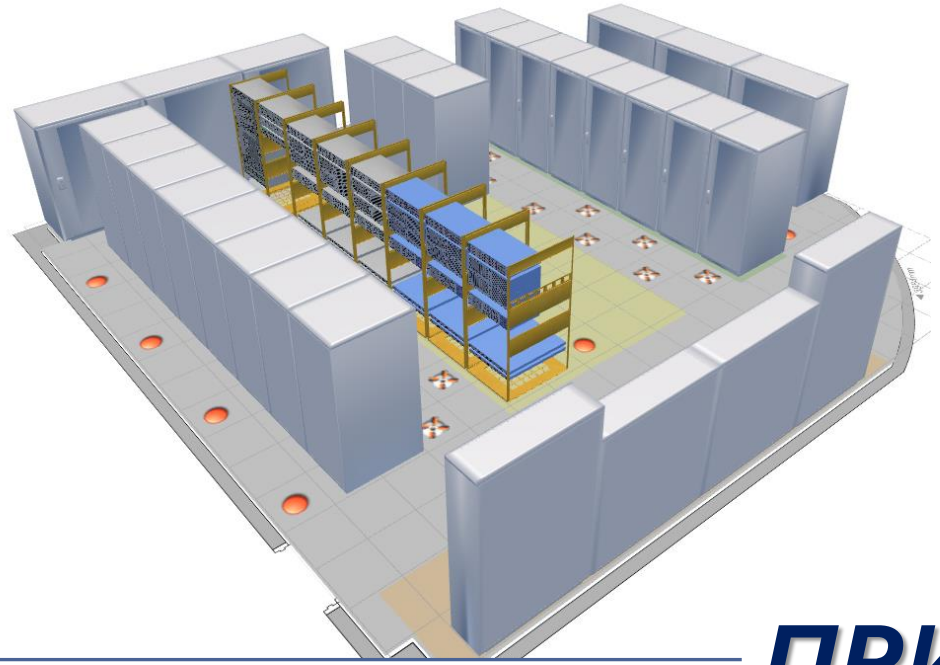
- Одновременная работа сервера приложений с двумя БД для авто-тестирования себя не оправдала, т.к. требует реализации не простых алгоритмов сравнения результатов транзакций с учетом особенностей разных СУБД. Такой инструмент может себя окупить, если необходимо транслировать несколько проектов.

Возможно лучшим подходом со временем станет встроенная поддержка популярных конструкций Oracle в PostgresPro на уровне ядра и библиотек

---

[1] Для унаследованного PL/SQL кода существенного объема > 50k строк

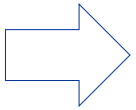
[2] В случае БАЗИС объем ручной доработки составил ~5% от объема исходного кода



ПРИМЕРЫ ТРАНСЛЯЦИИ PL/SQL

// Переменные уровня пакета

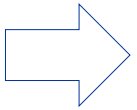
```
CREATE OR REPLACE PACKAGE BODY TEST is
    VERSION NUMBER := 1;
    globalVar STYCLI_STSIGPATH;

    PROCEDURE test_package_global_vars IS
        v varchar2;
    BEGIN
        VERSION := VERSION + 1;

        SELECT 3 INTO VERSION FROM dual;

        if globalVar.DEV1_LOCATION_INFO.Count() > 0 then
            v := globalVar.DEV1_LOCATION_INFO.ZONE_TABLE;
        end if;
    END test_package_global_vars;
END TEST;
```



```
create or replace function TEST.get_VERSION() returns double precision
language 'plpgsql' as $$
begin
    if not pgv_exists('TEST', 'VERSION') then
        perform pgv_set('TEST', 'VERSION', 1::double precision);
    end if;
    return pgv_get('TEST', 'VERSION', null::double precision);
end $$;

create or replace procedure TEST.set_VERSION(v double precision)
language 'plpgsql' as $$
begin
    perform pgv_set('TEST', 'VERSION', v::double precision);
end $$;


create or replace function TEST.get_globalVar() returns basis.STYCLI_STSIGPATH
language 'plpgsql' as $$
begin
    if not pgv_exists('TEST', 'globalVar') then
        perform pgv_set('TEST', 'globalVar', null::basis.STYCLI_STSIGPATH);
    end if;
    return pgv_get('TEST', 'globalVar', null::basis.STYCLI_STSIGPATH);
end $$;

create or replace procedure TEST.set_globalVar(v basis.STYCLI_STSIGPATH)
language 'plpgsql' as $$
begin
    perform pgv_set('TEST', 'globalVar', v::basis.STYCLI_STSIGPATH);
end $$;


create or replace procedure TEST.p_test_package_global_vars()
language 'plpgsql' as $$
declare
    or2pgTmpVar22 double precision;
    v varchar;
begin
    call TEST.set_VERSION(TEST.get_VERSION() + 1);

    select 3 into strict or2pgTmpVar22 from dual;
    call TEST.set_VERSION(or2pgTmpVar22);

    if dictionary_count((TEST.get_globalVar()::basis
                        .STYCLI_STSIGPATH)
                        .DEV1_LOCATION_INFO) > 0 then
        v := ((TEST.get_globalVar()::basis.STYCLI_STSIGPATH)
            .DEV1_LOCATION_INFO::basis.STYCBA_LOCATION_INFO)
            .ZONE_TABLE;
    end if;
end $$;
```

Иерархические запросы

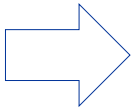
```
-- 1
SELECT staff_id, name, CONNECT_BY_ROOT name as "Manager"
    FROM staff_table
    START WITH staff_id = '1001'
    CONNECT BY PRIOR staff_id = manager_id;

-- 2
SELECT staff_id, name, CONNECT_BY_ISLEAF
    FROM staff_table
    START WITH staff_id = '1001'
    CONNECT BY PRIOR staff_id = manager_id;

-- 3
SELECT staff_id, name, manager_id,
    SYS_CONNECT_BY_PATH(name,'/') AS PATH
    FROM staff_table
    START WITH staff_id = '1001'
    CONNECT BY PRIOR staff_id = manager_id;

-- 4
WITH dummy AS (select 1 from dual)
    SELECT staff_id, name, manager_id, LEVEL
    FROM staff_table
    START WITH staff_id = '1001'
    CONNECT BY PRIOR staff_id = manager_id;
```



```
-- 1
with recursive or2pgTmpQuery0 as (
    (select name as "Manager", tbl_tmp.*
        from staff_table tbl_tmp where staff_id = '1001')
    union all
    (select or2pgTmpQuery0."Manager", tbl_tmp.*
        from staff_table tbl_tmp join or2pgTmpQuery0
        on or2pgTmpQuery0.staff_id = tbl_tmp.manager_id)
)
select staff_id, name, "Manager"
from or2pgTmpQuery0 tbl_tmp;

-- 2
with recursive or2pgTmpQuery1 as (
    (select tbl_tmp.* from staff_table tbl_tmp where staff_id = '1001')
    union all
    (select tbl_tmp.* from staff_table tbl_tmp join or2pgTmpQuery1 on
            or2pgTmpQuery1.staff_id = tbl_tmp.manager_id)
)
select staff_id, name,
    case when exists(select 1 from or2pgTmpQuery1 isleaf_tmp
            where tbl_tmp.staff_id = isleaf_tmp.manager_id)
        then 0 else 1
    end as or2pgTmpColAlias7
from or2pgTmpQuery1 tbl_tmp;

-- 3
with recursive or2pgTmpQuery2 as (
    (select concat('/', tbl_tmp.name) as PATH_path, tbl_tmp.*
        from staff_table tbl_tmp where staff_id = '1001')
    union all
    (select
        concat(or2pgTmpQuery2.PATH_path, '/', tbl_tmp.name) as PATH_path,
        tbl_tmp.*
        from staff_table tbl_tmp join or2pgTmpQuery2 on
            or2pgTmpQuery2.staff_id = tbl_tmp.manager_id)
)
select staff_id, name, manager_id, PATH_path as path
from or2pgTmpQuery2 tbl_tmp;

-- 4
with recursive or2pgTmpQuery3 as (
    (select 1 as level, tbl_tmp.*
        from staff_table tbl_tmp where staff_id = '1001')
    union all
    (select or2pgTmpQuery3.level + 1 as level, tbl_tmp.*
        from staff_table tbl_tmp join or2pgTmpQuery3 on
            or2pgTmpQuery3.staff_id = tbl_tmp.manager_id)
), dummy AS (select 1 from dual)
select staff_id, name, manager_id, level
from or2pgTmpQuery3 tbl_tmp;
```

// Dynamic SQL with bulk collect

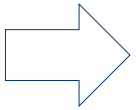
```
CREATE OR REPLACE PACKAGE BODY TEST is
    FUNCTION test_execute_bulk_collect
    RETURN stymod_varchar2tab AS
        vSql varchar2(100 char);
        vValues stymod_varchar2tab := stymod_varchar2tab();
        param varchar2(10 char);
    BEGIN
        vSql := 'select :1 from svidev_device';
        EXECUTE IMMEDIATE vSql BULK COLLECT
            INTO vValues USING param;
        RETURN vValues;
    END;
END;
```



```
create or replace function TEST.f_test_execute_bulk_collect()
returns basis.STYMOD_VARCHAR2TAB language 'plpgsql' AS $$
declare
    or2pgTmpVar377 varchar(1024);
    or2pgTmpCur7 refcursor;
    vSql varchar(100);
    vValues basis.STYMOD_VARCHAR2TAB := dictionary_create();
    param varchar(10);
begin
    vSql := 'select :1 from svidev_device';
    open or2pgTmpCur7 for execute  transpile(vSql)
        using param;
    vValues := dictionary_create();
    loop
        fetch or2pgTmpCur7 into or2pgTmpVar377;
        exit when not found;
        vValues := dictionary_extend(vValues, 1);
        vValues := dictionary_set(vValues, or2pgTmpVar377,
            dictionary_last(vValues));
    end loop;
    close or2pgTmpCur7;
    return vValues;
end $$;
```

// Functions with OUT parameters

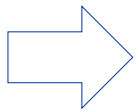
```
CREATE OR REPLACE PACKAGE BODY TEST AS
    FUNCTION getZoneElidTest(
                                pLinkType  in out nocopy varchar2,
                                pHierarchy in out nocopy varchar2)
    RETURN varchar2 IS
        vZoneElid   varchar2 := 'VNIJYQC0OQGPR1';
        vLinkType   varchar2 := 'vLinkType';
        vHierarchy  varchar2 := 'vHierarchy';
    BEGIN
        if pLinkType is null then
            pLinkType := vLinkType;
        end if;
        if pHierarchy is null then
            pHierarchy := vHierarchy;
        end if;
        RETURN vZoneElid;
    END getZoneElidTest;

    FUNCTION getEquipLocationInfo(pElid in varchar2)
    RETURN varchar2 IS
        vResult     varchar2;
        vLinkType   varchar2 := 'vLinkType_Orig';
        vHierarchy  varchar2 := NULL;
    BEGIN
        vResult := getZoneElidTest(vLinkType, vHierarchy,
            vPlanHierarchy);
        dbms_output.put_line('vResult=' || vResult);
        dbms_output.put_line('vLinkType=' || vLinkType);
        dbms_output.put_line('vHierarchy=' || vHierarchy);
        RETURN vResult;
    END GetEquipLocationInfo;
END TEST;
```



```
create or replace function TEST.f_getZoneElidTest(
    in out pLinkType varchar,
    in out pHierarchy varchar,
    in out return_out varchar default null) returns record
language 'plpgsql' as $$
declare
    vZoneElid      varchar := 'VNIJYQC0OQGPR1';
    vLinkType      varchar := 'vLinkType';
    vHierarchy     varchar := 'vHierarchy';
begin
    if is_null(pLinkType) then
        pLinkType := vLinkType;
    end if;
    if is_null(pHierarchy) then
        pHierarchy := vHierarchy;
    end if;
    return_out := vZoneElid;
end $$;

create or replace function TEST.f_getEquipLocationInfo( pElid varchar)
        returns varchar
language 'plpgsql' as $$
declare
    return_out_record record;
    vResult     varchar;
    vLinkType   varchar := 'vLinkType_Orig';
    vHierarchy  varchar := NULL;
begin
    return_out_record := TEST.f_getZoneElidTest(vLinkType, vHierarchy,
        vPlanHierarchy::varchar);
    vLinkType := return_out_record.pLinkType;
    vHierarchy := return_out_record.pHierarchy;
    vPlanHierarchy := return_out_record.return_out;
    vResult := return_out_record.return_out;
    raise notice '%', concat('vResult=', vResult);
    raise notice '%', concat('vLinkType=', vLinkType);
    raise notice '%', concat('vHierarchy=', vHierarchy);
    return vResult;
end $$;
```

// "Index by table" with bulk collect

```
CREATE OR REPLACE PACKAGE BODY TEST as
    TYPE INVENTORY is record (
        ID NUMBER,
        NAME VARCHAR2(123 CHAR)
    );

    TYPE INVENTORY_CL IS TABLE OF INVENTORY INDEX BY VARCHAR;
    cRES INVENTORY_CL;

    PROCEDURE TEST_ASSOCIATIVE_ARR IS
    BEGIN
        cRES(1) := INVENTORY(1, 'abc');
        IF cRES.count > 0 THEN
            FOR i IN cRES.FIRST .. cRES.LAST
            LOOP
                IF cRES(i).NAME = '1' THEN
                    EXIT;
                END IF;
            END LOOP;
        END IF;

        SELECT i_number, i_name
        BULK COLLECT
        INTO cRES
        FROM btob.inventory_table
        WHERE i_number > 110;

        IF cRES.EXISTS(1) THEN
            cRES(cRES.FIRST) := INVENTORY(1, 'abc');
        END IF;
    END;
END;
```



```
do $anonym$ begin
create type TEST.INVENTORY as (
    ID double precision,
    NAME varchar(123)
);
exception when duplicate_object then return; end $anonym$;

do $anonym$ begin
create domain TEST.INVENTORY_CL as associative_type;
exception when duplicate_object then return; end $anonym$;

create or replace procedure TEST.set_cRES(v TEST.INVENTORY_CL) language 'plpgsql' as $$
begin
    perform pgv_set('TEST', 'cRES', v::TEST.INVENTORY_CL);
end $$;

create or replace function TEST.get_cRES() returns TEST.INVENTORY_CL language 'plpgsql' as $$
begin
    if not pgv_exists('TEST', 'cRES') then
        perform pgv_set('TEST', 'cRES', dictionary_create_char()::TEST.INVENTORY_CL);
    end if;
    return pgv_get('TEST', 'cRES', null::TEST.INVENTORY_CL)::TEST.INVENTORY_CL;
end $$;

create or replace procedure TEST.p_test_associative_arr() language 'plpgsql' AS $$
declare
    or2pgTmpArr2 TEST.INVENTORY[];
begin
    call TEST.set_cRES(dictionary_set(TEST.get_cRES(), (row(1, 'abc')::TEST.INVENTORY)::TEST.INVENTORY,
        (1)::varchar));
    if dictionary_count(TEST.get_cRES()) > 0 then
        for i in dictionary_first(TEST.get_cRES()) .. dictionary_last(TEST.get_cRES()) loop
            if (dictionary_get(TEST.get_cRES(), null::TEST.INVENTORY, i)::TEST.INVENTORY).NAME = '1' then
                exit;
            end if;
        end loop;
    end if;

    select array_agg(row(i_number, i_name)::TEST.INVENTORY)
    into or2pgTmpArr2
    from btob.inventory_table
    where i_number > 110;
    call TEST.set_cRES(collect_bulk(TEST.get_cRES(), or2pgTmpArr2));

    if dictionary_exists(TEST.get_cRES(), 1) then
        call TEST.set_cRES(dictionary_set(TEST.get_cRES(), (row(1, 'abc')::TEST.INVENTORY)::TEST.INVENTORY,
            (dictionary_first(TEST.get_cRES()))::varchar));
    end if;
end $$;
```

Спасибо за внимание!

«СДИ Софт»

Россия, 107045, г. Москва,
ул. Трубная, д.12
Телефон: +7 (499) 495-10-42
Интернет: www.sdisoft.ru
Электронная почта: info@sdisoft.ru