МИГРАЦИЯ С ORACLE НА POSTGRESQL С ИСПОЛЬЗОВАНИЕМ АВТОМАТИЧЕСКОГО КОНВЕРТЕРА

Александр Кварацхелия, Александр Чирков

БАРС Груп, г.Казань

ТЕХ.СТЕК ИНФОРМАЦИОННОЙ СИСТЕМЫ

PHP

~1Гб Исходные коды

9250 Формы

1260 Динамические формы

ORACLE

3750 Таблицы

3450 Вьюхи

2730 Триггеры

620 Процедуры, функции

2810 Пакеты

PHP ФОРМЫ

```
<component cmptype="Action" name="InsertAction" mode="post">
    declare col2_id NUMBER(17);
            Nval_id NUMBER(17);
            nDataType NUMBER(17);
    begin
      :iddduser := :EMPLOYER;
       /*добавляем видимое значение*/
        d_pkg_add_dir_rows.add(pnd_insert_id => :pnd_insert_id_rows,
                               pnlpu => :pnlpu,
                               pnpid => :pnpid,
                               pnhid => :pnhid,
                               pnr_order => null);
       d_pkg_add_dir_values.add(pnd_insert_id => :pnd_insert_id_values,
                               pnlpu => :pnlpu,
                               pnd_column => :pnd_column,
                               pnd_row => :pnd_insert_id_rows,
                               psstr_value => :psstr_value,
                               pnnum_value => null,
                               pddat_value => null,
                               pblblob_value => null);

        /*заполнение невидимого значения*/
       if :psNstr_value is not null then
          begin
             if :COLUMNID_N is null then
                /*Если нет невидимого столбца в словаре, добавляем столбец с типом VA
                begin
                   begin
                      select ad.ID
                        into nDataType
                        from D_V_ADD_DATATYPES ad
                       where ad.DT_TYPE=0
                         and ad.STR_LENGTH=4000
                         and ad.version=d_pkg_versions.get_version_by_lpu(1,:pnlpu,
                         and rownum=1
                          order by id;
                      exception when NO_DATA_FOUND then
                         /*если нет подходящего типа данных- создаем*/
                         d_PKG_ADD_DATATYPES.add(nDataType, :pnlpu,
                                                 'VARCHAR2(4000)',        --Наимен
                                                 0,          --Тип данных
```

```
<component cmptype="DataSet" name="DS_ANALITIC" activateoncreate="false" com
    <![CDATA[
        select t.*
        from D_V_REP_PMC_INF_AGREEMENT t
        where t.CREATE_DATE<=to_date(:D_END, 'dd.mm.yyyy')
        and t.CREATE_DATE>=to_date(:D_BEGIN, 'dd.mm.yyyy')
        and t.LPU=:LPU
        @if(:P_NAME){
            and (upper(t.PAT_FIO) like upper(:P_NAME||'%'))
        @}
        @if(:P_CARD){
            and (upper(t.CARD_NUMB) like upper(:P_CARD||'%'))
        @}
        @if(:C_LOC){
            and t.CARD_LOCATION=:C_LOC
        @}
        @if(:AGREE!=2){
            and t.IA_PRINTED=:AGREE
        @}
        @if(:DIV){
            and exists (select null
                from D_V_PMC_REGISTRATION pr
                where pr.PID = t.PAT_ID
                and pr.LPU = t.LPU
                and pr.BEGIN_DATE <= trunc(sysdate)
                and(pr.END_DATE >= trunc(sysdate) or pr.END_DATE is null)
                and pr.DIVISION_ID = :DIV)
        @}
        @if(:MARKER_FILTER){
            and exists(
              select COLUMN_VALUE
              from table(D_PKG_TOOLS.STR_SEPARATE(:MARKER_FILTER)) t
              where instr(';'||t.MARKER_ID||';',';'||t.COLUMN_VALUE||';') >
            )
        @}
    ]]>
    <component cmptype="Variable" name="LPU" src="LPU" get="lpu" srctype="se
    <component cmptype="Variable" name="D_BEGIN"       src="BEGIN_DATE"
    <component cmptype="Variable" name="D_END"         src="END_DATE"
```

ПЛАН!

Использовать грамматический парсер для получения AST-дерева для PL/SQL

Дерево для PL/SQL перерабатывать и создавать аналогичное дерево для PostgreSQL

Анонимные блоки из PHP конвертировать в реальном времени через специальный микросервис

Запаковать всё в докер контейнеры и автоматизировать сборку ИС на PostgreSQL

Научиться создавать тестовые сценарии и запускать массированные тесты

ИС на PostgreSQL

PHP

Коннектор к PostgreSQL

Elasticsearch

Логи + отладочная инфа

Микросервис

PL/SQL -> PL/pgSQL

история конвертации

PostgreSQL

Сконвертированная схема ИС

ANTLR4

ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks.

From a grammar, ANTLR generates a parser that can build and walk parse trees.

```
select t.UNITCODE
  from D_UNITLIST t
where t.PARENTUNITCODE is null
      connect by t.UNITCODE = prior t.PARENTUNITCODE
      start with t.UNITCODE = fsUNITCODE
```

```
WITH RECURSIVE tmp AS (
    (SELECT t.*, 1 as level FROM D_UNITLIST t WHERE t.UNITCODE = fsUNITCODE)
UNION
    (SELECT t.*, tmp.level + 1 as level FROM D_UNITLIST t JOIN tmp ON tmp.PARENTUNITCODE = t.UNITCODE)
)
SELECT UNITCODE FROM tmp WHERE nullif2(PARENTUNITCODE) is null
```

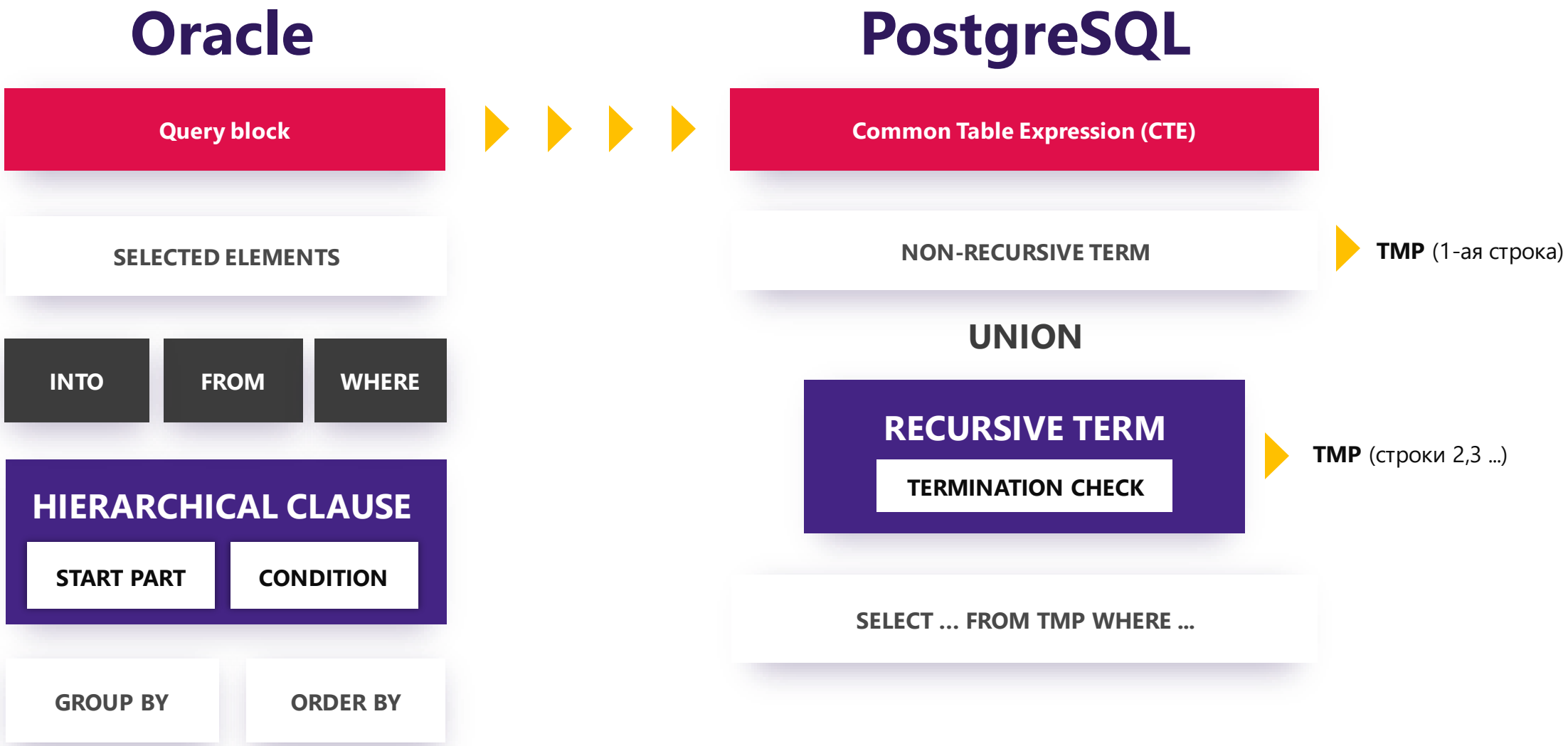
ГРАММАТИКА PL/SQL (ANTLR4)

```
query_block
    : SELECT (DISTINCT | UNIQUE | ALL)? (ASTERISK | selected_element (',' selected_element)*)
      into_clause? from_clause where_clause? hierarchical_query_clause? group_by_clause? model_clause?
    ;

selected_element
    : select_list_elements column_alias?
    ;

from_clause
    : FROM table_ref_list
    ;

select_list_elements
    : tableview_name '.' ASTERISK
    | (regular_id '.')? expressions
    ;

table_ref_list
    : (','? table_ref)+
    ;
```

```
hierarchical_query_clause
    : CONNECT BY NOCYCLE? condition start_part?
    | start_part CONNECT BY NOCYCLE? condition
    ;

start_part
    : START WITH condition
    ;
```

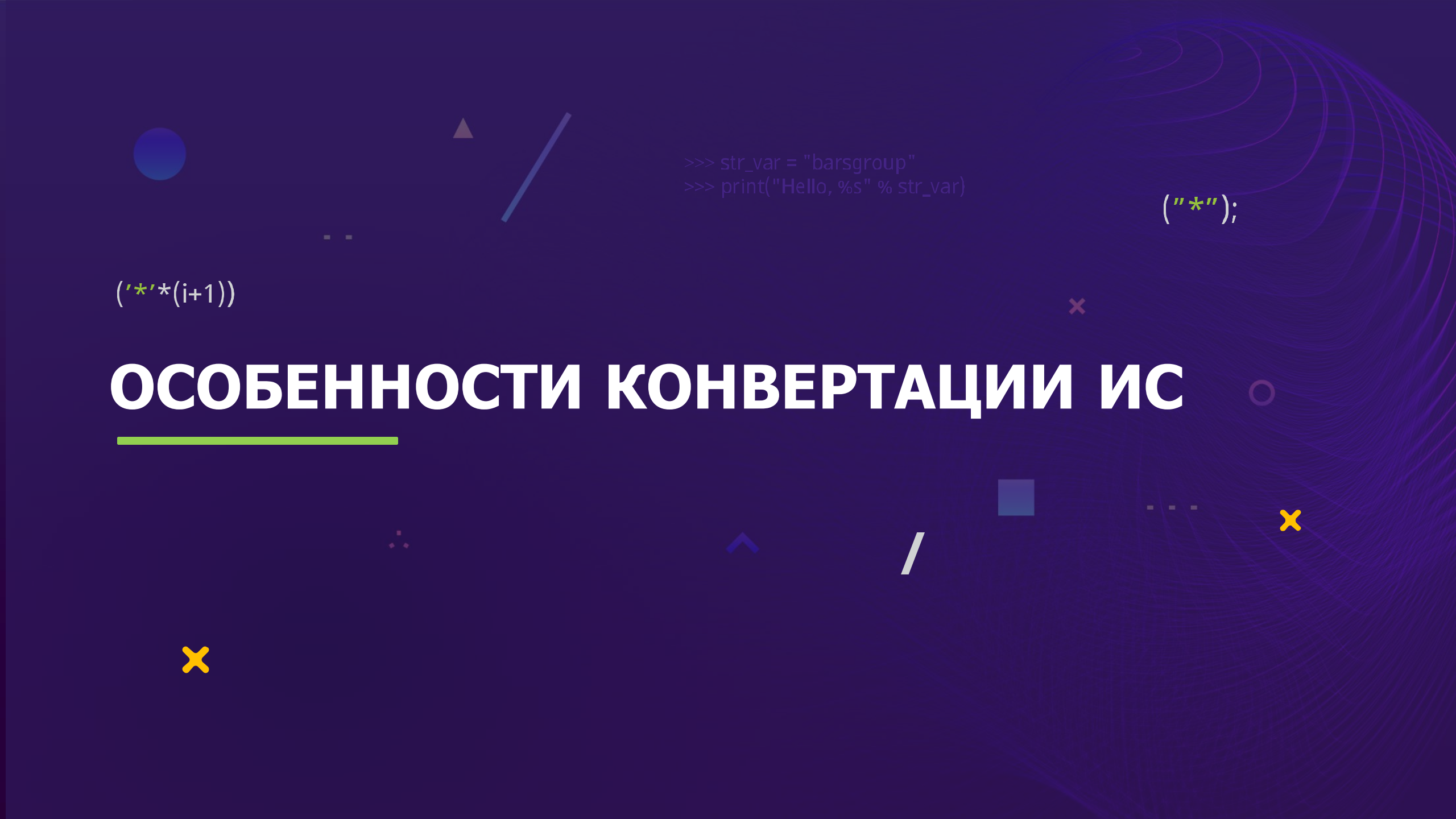

ctx = {PlSqlParser$Query_blockContext@1097} "[8851 8844 8756 8586 1536 1476]"
children = {ArrayList@1247} size = 5
0 = {TerminalNodeImpl@1254} "select"
1 = {PlSqlParser$Selected_elementContext@1255} "[8873 8851 8844 8756 8586 1536 1476]"
2 = {PlSqlParser$From_clauseContext@1256} "[8886 8851 8844 8756 8586 1536 1476]"
3 = {PlSqlParser$Where_clauseContext@1257} "[8887 8851 8844 8756 8586 1536 1476]"
4 = {PlSqlParser$Hierarchical_query_clauseContext@1258} "[8890 8851 8844 8756 8586 1536 14
children = {ArrayList@1264} size = 4
0 = {TerminalNodeImpl@1270} "connect"
1 = {TerminalNodeImpl@1271} "by"
2 = {PlSqlParser$ConditionContext@1272} "[9141 8890 8851 8844 8756 8586 1536 1476]"
3 = {PlSqlParser$Start_partContext@1273} "[9142 8890 8851 8844 8756 8586 1536 1476]"
children = {ArrayList@1284} size = 3
0 = {TerminalNodeImpl@1290} "start"
1 = {TerminalNodeImpl@1291} "with"
2 = {PlSqlParser$ConditionContext@1292} "[9157 9142 8890 8851 8844 8756 8586 15
children = {ArrayList@1296} size = 1
0 = {PlSqlParser$ExpressionContext@1302} "[9769 9157 9142 8890 8851 8844 8
children = {ArrayList@1304} size = 1
0 = {PlSqlParser$Logical_expressionContext@1309} "[9780 9769 9157 914
start = {CommonToken@1297} "[@43,144:144='t',<2252>,5:18]"
stop = {CommonToken@1249} "[@49,157:166='fsUNITCODE',<2252>,5:31]"

Трансформация AST-дерева (иерархические запросы)



```
select t.UNITCODE
  from D_UNITLIST t
where t.PARENTUNITCODE is null
      connect by t.UNITCODE = prior t.PARENTUNITCODE
      start with t.UNITCODE = fsUNITCODE
```

```
WITH RECURSIVE tmp AS (
    (SELECT t.*, 1 as level FROM D_UNITLIST t WHERE t.UNITCODE = fsUNITCODE)
UNION
    (SELECT t.*, tmp.level + 1 as level FROM D_UNITLIST t JOIN tmp ON tmp.PARENTUNITCODE = t.UNITCODE)
)
SELECT UNITCODE FROM tmp WHERE nullif2(PARENTUNITCODE) is null
```



ОСОБЕННОСТИ КОНВЕРТАЦИИ ИС

Структура базы при выгрузке Oracle Developer

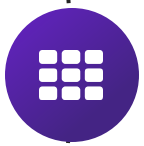
- BEFORE
- TABLES
- DATA
- INDEXES
- CONSTRAINTS
- REF_CONSTRAINTS
- SEQUENCES
- TYPES
- FIRST_CYCLE
- PACKAGES
- PACKAGE_BODIES
- PROCEDURES
- FUNCTIONS
- VIEWS
- TRIGGERS
- MATERIALIZED_VIEWS
- AFTER

BEFORE

- Отказались от установки orafce
- Касты
- Операторы
- Аналоги системных таблиц (user_objects, user_triggers и т.д.)
- Аналоги функций и процедур (regexp_count, to_char…)
- Дополнительные функции (для работы table of, xml и др.)
- Аналоги пакетов (dbms_sql, dbms_xmldom, dbms_lob…)

TABLES

Создание таблиц и комментариев



Получение сведений о типе данных столбцов (касты, замена конструкции table.field%type в типе record)

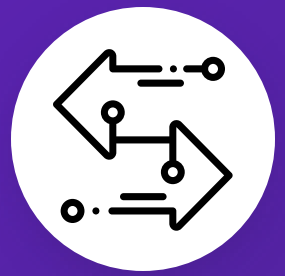
Генерация кода процедуры, которая создает временные таблицы

Пример конвертации глобальных таблиц

```
CREATE GLOBAL TEMPORARY TABLE "global_temp_table"
(
    "ID" NUMBER(8,0),
    "TITLE" VARCHAR2(36)
);

CREATE OR REPLACE FUNCTION test_global_temp_table
(
  param in NUMBER
) return VARCHAR2(10)
is
    var1 global_temp_table.id%type
begin
  SELECT id, title FROM global_temp_table;
  return 'ok';
end test_global_temp_table;
```

```
CREATE OR REPLACE PROCEDURE proc_create_temp_table_global_temp_table()
LANGUAGE 'plpgsql' AS
$$
BEGIN
    CREATE TEMPORARY UNLOGGED TABLE IF NOT EXISTS global_temp_table
    (
        "id" NUMERIC(8, 0),
        "title" VARCHAR(36)
    );
END
$$;

CREATE TABLE IF NOT EXISTS global_temp_table
(
    "id" NUMERIC(8, 0),
    "title" VARCHAR(36)
);


CREATE OR REPLACE FUNCTION test_global_temp_table(
    param NUMERIC
)
RETURNS VARCHAR(10)
LANGUAGE 'plpgsql' AS
$$
DECLARE
    var1 global_temp_table.id%TYPE;
BEGIN
    CALL proc_create_temp_table_global_temp_table();
    SELECT id, title FROM global_temp_table;
    RETURN 'ok';
END
$$ VOLATILE;
```

DATA, INDEXES, CONSTRAINTS, SEQUENCES



Переносятся без особых проблем и на работу других сущностей внутри конвертера не влияют

TYPES

```
CREATE OR REPLACE TYPE "D_TP_SSS" as object
(
  STR1                  VARCHAR2(4000),
  STR2                  VARCHAR2(4000)
);

CREATE OR REPLACE TYPE "D_CL_SSS" as table of D_TP_SSS;

CREATE OR REPLACE PROCEDURE TEST
is
  cRES D_CL_SSS := D_CL_SSS();
begin
    cRES.EXTEND;
    cRES(cRES.LAST) := D_TP_SSS('строка1', 'строка2');
    cRES.EXTEND;
    SELECT D_TP_SSS(1, 2) INTO cRES(cRES.LAST) FROM
dual;
end TEST;
```

```
do $anonym$
begin
    CREATE TYPE d_tp_sss AS (str1 varchar(4000), str2 varchar(4000));
    EXCEPTION WHEN duplicate_object
        THEN return;
end
$anonym$;

do $anonym$
begin
    CREATE DOMAIN d_cl_sss AS dictionary_type;
    EXCEPTION WHEN duplicate_object
        THEN return;
end
$anonym$;
...
Создание функций для работы с TABLE OF disctionary_set, dictionary_get, collect_bulk
и т.д.
...
CREATE OR REPLACE PROCEDURE test()
LANGUAGE 'plpgsql' AS
$$
DECLARE
    cres d_cl_sss := DICTIONARY_CREATE_INT();
    or2pgTmpVar0_0 d_tp_sss;
BEGIN
    cres := dictionary_extend(cres, 1);
    cres := dictionary_set(cres, row('строка1','строка2')::d_tp_sss,
dictionary_last(cres));
    cres := dictionary_extend(cres, 1);
    SELECT row(1, 2)::d_tp_sss INTO or2pgTmpVar0_0 FROM dual;
    cres := dictionary_set(cres, or2pgTmpVar0_0, dictionary_last(cres));
END
$$;
```

FIRST_CYCLE

Дубль папок процедур и функций с той целью, чтобы перед обработкой пакетов, пакеты уже знали информацию о наличии таких функций и их параметрах

PACKAGES

Создание схем

Сбор информации о функциях, процедурах и типах параметров

Обработка глобальных переменных

Пример преобразования глобальных переменных

```
CREATE OR REPLACE PACKAGE BODY "GV_PKG" is
VERSION NUMBER := 1;
procedure TEST
is
    x NUMBER := 2;
begin
    IF x < VERSION
        THEN x := 1;
    END IF;
    VERSION := VERSION + 1;
    SELECT 3 INTO VERSION FROM dual;
end TEST;
end GV_PKG;
```

```
CREATE SCHEMA IF NOT EXISTS gv_pkg;
CREATE OR REPLACE PROCEDURE gv_pkg.test()
LANGUAGE 'plpgsql' AS
$$
DECLARE
    x NUMERIC := 2;
    or2pgTmpVar0_0 NUMERIC;
BEGIN
    IF x < (gv_pkg.get_version()) THEN
        x := 1;
    END IF;
    CALL gv_pkg.set_version(((gv_pkg.get_version()) + 1)::NUMERIC);
    or2pgTmpVar0_0 := (gv_pkg.get_version());
    SELECT 3 "3" INTO or2pgtmpvar0_0 FROM dual;
    CALL gv_pkg.set_version((or2pgTmpVar0_0)::NUMERIC);
END
$$;

gv_pkg.set_version - set_config('gv_pkg.version', val::varchar, false);
gv_pkg.get_version - current_setting('gv_pkg.version', true);
```

PACKAGE BODIES, FUNCTIONS, PROCEDURES

1

Обработка параметров

Изменение типов

Изменение значений по умолчанию

2

Обработка declare

Изменение типов переменных

Создание типов и подтипов на уровне пакетов

Обработка внутренних функций и процедур

3

Обработка body

Добавление в declare блок переменных циклов

Добавление в declare блок «промежуточных» переменных

Обработка исключений

connect by … start with -> with recursive

(+) заменяется на LEFT OUTER JOIN

Замена rownum

Конкатенация A || B заменяется на функцию concat

Проверка на IS NULL с учетом пустых строк

Замена функции decode на конструкцию case

Касты в конструкциях case, coalesce, union

Меняется поведение некоторых стандартных функций (substr, replace)

Сборная солянка

```
CREATE OR REPLACE PACKAGE BODY "TEST_PKG" is
PROCEDURE TEST
(
    psDef in VARCHAR2 default NULL,
    psNonDef in VARCHAR2
)
is
    iCURSOR                INTEGER;

    procedure REPL (psSTR OUT VARCHAR2) is
    begin
        psSTR := replace(psSTR, '1','2');
    end REPL;
begin
    iCURSOR := dbms_sql.open_cursor;
    begin
        SELECT decode(rownum, 2, 1, REPL('STR'))
            INTO RN
        FROM
            t1 t1, t2 t2
        WHERE
            t1.id = t2.id(+) AND
            ROWNUM = 1 AND t1.nl is null AND
            (t1.id || t1.pid) = '15' AND substr(t1.fio, 2,
-3) = '15';
        EXCEPTION WHEN OTHERS THEN
            return;
    end;
end TEST;
end TEST_PKG;
```

```
CREATE SCHEMA IF NOT EXISTS test_pkg;

CREATE OR REPLACE PROCEDURE test_pkg.test(
    psdef varchar = NULL,
    psnondef varchar = NULL
)
LANGUAGE 'plpgsql' AS
$$
DECLARE
    iCURSOR dbms_sql.record;
BEGIN
    icursor := dbms_sql.open_cursor();
    BEGIN
        SELECT(CASE
                WHEN (row_number() over () = 2) OR (check_null((row_number()
over ()),2)) THEN 1
                ELSE test_pkg.test_repl('STR', iCURSOR, psdef, psnondef)
            END)
        INTO STRICT rn
        FROM
            t1 t1
            LEFT OUTER JOIN t2 t2 ON t1.id = t2.id
        WHERE
            nullif2(t1.nl) IS NULL AND
            (concat(t1.id, t1.pid)) = '15' AND
            substr2((t1.fio)::varchar, 2, (-3)::numeric) = '15'
        LIMIT 1;
        EXCEPTION WHEN others THEN return;
    END;
END
$$;
```

VIEWS

Создание views с учетом зависимостей
(т.к. нет конструкции CREATE OR REPLACE FORCE)

TRIGGERS

- Разбивается на две части (создание функции + создание триггера)
- Замена inserting, updating, deleting на TP_OP = ‘INSERT’ …
- В конце блок IF с возвратом NEW ИЛИ OLD

```
CREATE OR REPLACE TRIGGER "TEST_TRIGGER"
  after insert or update or delete on TEST_TABLE
  for each row
begin
  if inserting then
    LOG_ADD('TEST_TABLE', 'I', :new.ID);
  end if;
  if updating then
     REG('TEST_TABLE', 'CODE', 'VARCHAR2', :old.CODE, :new.CODE);
end if;
  if deleting then
    LOG_ADD('TEST_TABLE', 'D', :old.ID);
  end if;
end TEST_TRIGGER;
```

```
CREATE OR REPLACE FUNCTION trigger_fct_test_trigger() RETURNS trigger AS $BODY$
BEGIN
    IF TG_OP = 'INSERT' THEN
        CALL log_add('TEST_TABLE', 'I', NEW.ID);
    END IF;
    IF TG_OP = 'UPDATE' THEN
        CALL reg('TEST_TABLE', 'CODE', 'VARCHAR2', OLD.CODE, NEW.CODE);
    END IF;
    IF TG_OP = 'DELETE' THEN
        CALL log_add('TEST_TABLE', 'D', OLD.ID);
    END IF;
    IF TG_OP = 'INSERT' THEN
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        RETURN NEW;
    ELSE
        RETURN OLD;
    END IF;
END
$BODY$ LANGUAGE 'plpgsql';

DROP TRIGGER IF EXISTS test_trigger ON test_table CASCADE;
CREATE TRIGGER test_trigger
after insert or update or delete on test_table for each row
EXECUTE PROCEDURE trigger_fct_test_trigger();
```

AFTER

Переопределенные сущности, которые не удается преобразовать с помощью конвертера

Функции которые будут работать на порядок быстрее

Функции которые проще переписать руками, чем доработать конвертер

Что изменилось в backend части?

- Селекты и анонимные блоки (касты)
- 4 новых класса (log, cache, lob, postgre)
- Возможность поддерживать 2 версии

Ручная работа

Функции и процедуры с количеством аргументов > 100

Как временное решение - собрать PostgreSQL из исходников с другим лимитом на количество аргументов, а параллельно переделать все такие функции

Функции и процедуры с одинаковым именем и одинаковыми параметрами

Проблемы из-за особенностей статического анализа кода

Алиасы таблиц в подзапросах

```
select t.ID, t.TIMER, ...
  case when (
    select null from D_V_AGENT_COMP_DISEASES t
      where t.PID = t.PATIENT_AGENT) ...
from D_V_DIR_SERV_FOR_DIARY t
```

```
select t.ID, t.TIMER, ...
  case when (
    select null from D_V_AGENT_COMP_DISEASES ta
      where ta.PID = t.PATIENT_AGENT) ...
from D_V_DIR_SERV_FOR_DIARY t
```

Функции с OUT параметром, которые возвращают значение

СОЗДАЁМ ТЕХНОЛОГИИ. МЕНЯЕМ ЖИЗНЬ

Email: akvarats@bars.group
Telegram: @akvarats