Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Отчет по лабораторной работе №3

«Основы языка PL/SQL»

Выполнил: студент гр. 953501
Кореневский С. А.

Поверил: Чащин С.В.

Минск 2022

Цель работы

Получить общее представление о PL/SQL и познакомиться с основными понятиями языка. Изучить реляционные свойства PL/SQL, включая синтаксис языка, типы данных, способы использования SQL, инструкции DML, а также особенности работы с транзакциями. Рассмотреть синтаксис создания подпрограмм (процедур и функций), которые могут храниться и выполнятся на стороне сервера БД.

Задачи

1. Написать процедуру/функцию на вход которой подаются два текстовых параметра (dev_schema_name, prod_schema_name), которые являются названиями схем баз данных (условно схема для разработки(Dev) и промышленная схема(Prod)), на выход процедура должна предоставить перечень таблиц, которые есть в схеме Dev, но нет в Prod, либо в которых различается структура таблиц. Наименования таблиц должны быть отсортированы в соответствии с очередностью их возможного создания в схеме prod (необходимо учитывать foreign key в схеме). В случае закольцованных связей выводить соответствующее сообщение

2. Доработать предыдущий скрипт с учетом возможности сравнения не только таблиц, но и процедур, функций, индексов пакетов

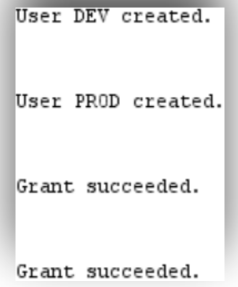
3. Доработать предыдущий скрипт с генерацией ddl-скрипта на обновление объектов, а также с учетом необходимости удаления в схеме prod объектов, отсутствующих в схеме dev.

Результаты выполнения

Написать процедуру/функцию на вход которой подаются два текстовых параметра (dev_schema_name, prod_schema_name), которые являются названиями схем баз данных (условно схема для разработки(Dev) и промышленная схема(Prod)), на выход процедура должна предоставить перечень таблиц, которые есть в схеме Dev, но нет в Prod, либо в которых различается структура таблиц. Наименования таблиц должны быть отсортированы в соответствии с очередностью их возможного создания в схеме prod (необходимо учитывать foreign key в схеме). В случае закольцованных связей выводить соответствующее сообщение.

- Создание пользователей:

```
CREATE USER dev IDENTIFIED BY devpsw;
CREATE USER prod IDENTIFIED BY prodpsw;
GRANT ALL PRIVILEGES TO dev;
GRANT ALL PRIVILEGES TO prod;
```

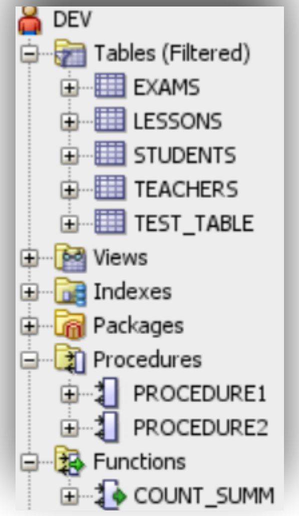


```
User DEV created.

User PROD created.

Grant succeeded.

Grant succeeded.
```

- Заполнение схемы DEV:

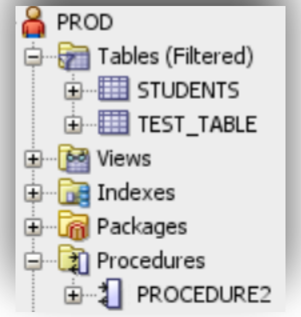
```
CREATE TABLE DEV.students(
      id NUMBER NOT NULL CONSTRAINT PK_STUDENTS PRIMARY KEY,
      name VARCHAR2(100)
);

CREATE TABLE DEV.lessons (
      id NUMBER NOT NULL CONSTRAINT PK_LESSONS PRIMARY KEY,
      name VARCHAR2(100)
);


CREATE TABLE DEV.exams(
      id NUMBER NOT NULL CONSTRAINT PK_EXAMS PRIMARY KEY,
      lesson_id NUMBER,
      CONSTRAINT FK_LESSONS FOREIGN KEY (lesson_id)
      REFERENCES DEV.lessons(id),
      datetime DATE
);

CREATE TABLE DEV.teachers(
      id NUMBER NOT NULL CONSTRAINT TEACHERS_PK PRIMARY KEY,
      name VARCHAR2(100));
```

```
CREATE TABLE DEV.test_table(
      id NUMBER,
      count NUMBER,
      testing VARCHAR2(50)
);

CREATE OR REPLACE PROCEDURE DEV.Procedure1
AS
BEGIN
      dbms_output.put_line('Procedure1 dev user');
END;

CREATE OR REPLACE PROCEDURE DEV.Procedure2
AS
BEGIN
      dbms_output.put_line('Procedure2 dev user');
END;

CREATE OR REPLACE FUNCTION DEV.count_summ(
      a NUMBER,
      b NUMBER
)
RETURN NUMBER
IS
BEGIN
      RETURN a + b;
END;
```



- Заполнение схемы PROD:

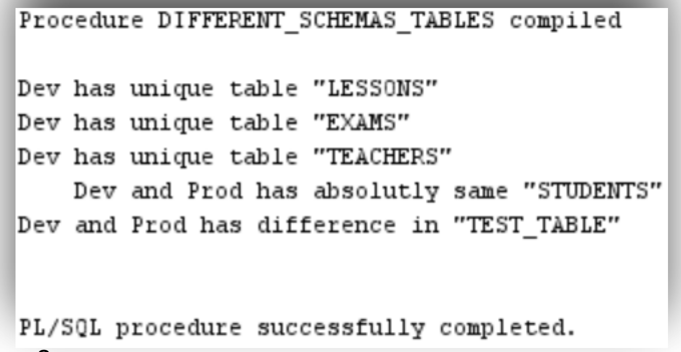
```
CREATE TABLE PROD.students(
      id NUMBER NOT NULL CONSTRAINT PK_STUDENTS PRIMARY KEY,
      name VARCHAR2(100)
);


CREATE TABLE PROD.test_table(
      id NUMBER,
      testing VARCHAR2(50)
);

CREATE OR REPLACE PROCEDURE PROD.Procedure2
AS
BEGIN
    bms_output.put_line('Procedure2 prod user');
END
```



• Процедура сравнения таблиц в схемах:

```
-- SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE different_schemas_tables (
dev_schema_name VARCHAR2, prod_schema_name VARCHAR2) AUTHID CURRENT_USER
AS
    TYPE tables_names_arr IS TABLE OF VARCHAR2(100);
    different_t tables_names_arr := tables_names_arr();
    dev_t tables_names_arr;
    prod_t tables_names_arr;
    same_t tables_names_arr;
    not_prod_t tables_names_arr;
    current_table VARCHAR2(100);
    recursion_level INTEGER;
    i INTEGER;

    PROCEDURE add_table(name_t VARCHAR2)
    AS
        parent_tables tables_names_arr := tables_names_arr();
        cycle_error EXCEPTION;
        i INT;
    BEGIN
        IF (recursion_level > 100) THEN
            dbms_output.put_line('Cycle in ' || name_t);
            RAISE cycle_error;
        END IF;
        IF (name_t MEMBER OF different_t
                OR name_t NOT MEMBER OF not_prod_t) THEN
            RETURN;
        END IF;
        SELECT c_pk.table_name
            BULK COLLECT INTO parent_tables
            FROM all_cons_columns a
            JOIN all_constraints c
                ON a.OWNER = c.OWNER
                AND a.constraint_name = c.constraint_name
            JOIN all_constraints c_pk
                ON c.r_owner = c_pk.OWNER
                AND c.r_constraint_name = c_pk.constraint_name
            WHERE
                c.constraint_type = 'R'
                AND a.table_name = name_t
                AND a.OWNER = dev_schema_name;
        IF (parent_tables.COUNT > 0) THEN
            i := parent_tables.FIRST;
            WHILE (i IS NOT NULL)
            LOOP
                recursion_level := recursion_level + 1;
                add_table(parent_tables(i));
                recursion_level := recursion_level - 1;
                i := parent_tables.NEXT(i);
            END LOOP;
        END IF;
        different_t.EXTEND;
        different_t(different_t.COUNT) := name_t;
        dbms_output.put_line('Dev has unique table "'
                             || name_t || '"');
    END;
```

```
BEGIN
    SELECT table_name BULK COLLECT INTO dev_t
        FROM all_tables WHERE OWNER=dev_schema_name;
    SELECT table_name BULK COLLECT INTO prod_t
        FROM all_tables WHERE OWNER=prod_schema_name;

    not_prod_t := dev_t MULTISET EXCEPT prod_t;
    i := not_prod_t.FIRST;
    WHILE i IS NOT NULL
    LOOP
        current_table := not_prod_t(i);

        IF (current_table MEMBER OF different_t) THEN
            i := not_prod_t.NEXT(i);
            CONTINUE;
        END IF;

        recursion_level := 0;
        add_table(current_table);
        i := not_prod_t.NEXT(i);
    END LOOP;

    same_t := dev_t MULTISET INTERSECT prod_t;
    i := same_t.FIRST;

    WHILE i IS NOT NULL
    LOOP
        current_table := same_t(i);

        IF (dbms_metadata_diff.compare_alter(
            'TABLE', current_table, current_table,
            dev_schema_name, prod_schema_name
        ) = EMPTY_CLOB() )
        THEN
            dbms_output.put_line('Dev and Prod has absolutly same "'
                                 || current_table || '"');
        ELSIF (dbms_metadata_diff.compare_alter(
            'TABLE', current_table, current_table,
            dev_schema_name, prod_schema_name
        ) IS NOT NULL)
        THEN
            different_t.EXTEND;
            different_t(different_t.COUNT) := current_table;
            dbms_output.put_line('Dev and Prod has difference in "'
                                 || current_table || '"');
        END IF;

        i:= same_t.NEXT(i);
    END LOOP;

END;

BEGIN
    different_schemas_tables('DEV', 'PROD');
END;
```



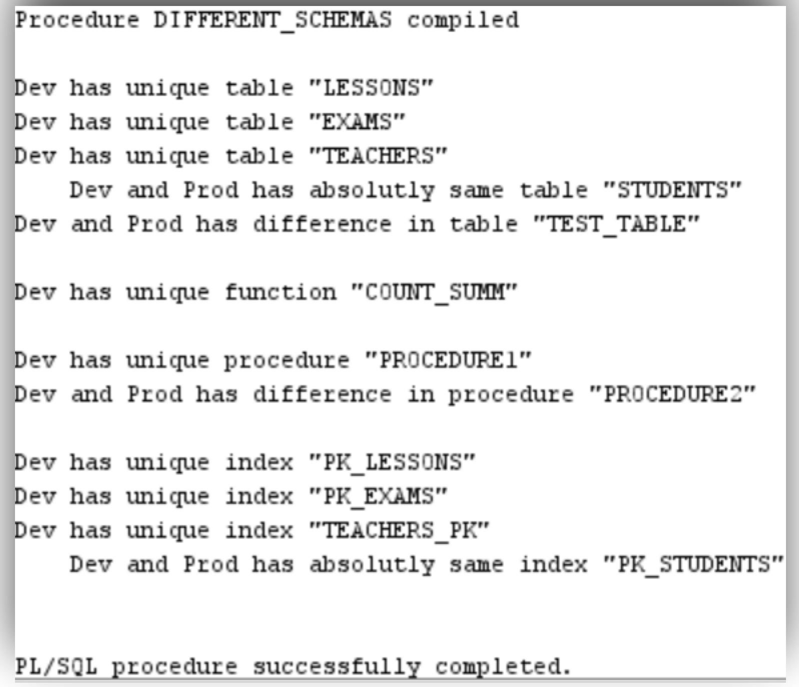
```
Procedure DIFFERENT_SCHEMAS_TABLES compiled

Dev has unique table "LESSONS"
Dev has unique table "EXAMS"
Dev has unique table "TEACHERS"
    Dev and Prod has absolutly same "STUDENTS"
Dev and Prod has difference in "TEST_TABLE"


PL/SQL procedure successfully completed.
```

2. Доработать предыдущий скрипт с учетом возможности сравнения не только таблиц, но и процедур, функций, индексов пакетов.

```
CREATE OR REPLACE PROCEDURE different_schemas (
    dev_schema_name VARCHAR2,
    prod_schema_name VARCHAR2) AUTHID CURRENT_USER
AS
    TYPE names_arr IS TABLE OF VARCHAR2(256);
    different names_arr := names_arr();
    recursion_level INTEGER;

    PROCEDURE add_table(name_t VARCHAR2, table_items names_arr)
    AS
        parent_tables names_arr := names_arr();
        cycle_error EXCEPTION;
        i INT;
    BEGIN
        IF (recursion_level > 100) THEN
            dbms_output.put_line('Cycle in ' || name_t);
            RAISE cycle_error;
        END IF;
        IF (name_t MEMBER OF different
                OR name_t NOT MEMBER OF table_items) THEN
            RETURN;
        END IF;

        SELECT c_pk.table_name
            BULK COLLECT INTO parent_tables
            FROM all_cons_columns a
            JOIN all_constraints c
                ON a.OWNER=c.OWNER
                AND a.constraint_name = c.constraint_name
            JOIN  all_constraints c_pk
                ON c.r_owner=c_pk.OWNER
                AND c.r_constraint_name = c_pk.constraint_name
            WHERE
                c.constraint_type = 'R'
                AND a.table_name = name_t
                AND a.OWNER=dev_schema_name;

        IF (parent_tables.COUNT > 0) THEN
            i := parent_tables.FIRST;
            WHILE (i IS NOT NULL)
            LOOP
                recursion_level := recursion_level + 1;
                add_table(parent_tables(i), table_items);
                recursion_level := recursion_level - 1;
                i := parent_tables.NEXT(i);
            END LOOP;
        END IF;

        different.EXTEND;
        different(different.COUNT) := name_t;
        dbms_output.put_line('Dev has unique table "'
                             || name_t || '"');
    END;


    PROCEDURE get_items_of_type(item_type VARCHAR2)
    AS
        dev_items names_arr;
        prod_items names_arr;

        not_prod_items names_arr;

        same_items names_arr;

        lines names_arr;

        current_item VARCHAR2(100);

        i INTEGER;

    BEGIN

        CASE item_type

        WHEN 'TABLE' THEN

            SELECT table_name

                BULK COLLECT INTO dev_items

                FROM all_tables

                WHERE OWNER=dev_schema_name;

            SELECT table_name

                BULK COLLECT INTO prod_items

                FROM all_tables

                WHERE OWNER = prod_schema_name;

        WHEN 'PROCEDURE' THEN

            SELECT object_name

                BULK COLLECT INTO dev_items

                FROM all_procedures

                WHERE OWNER=dev_schema_name;

            SELECT object_name

                BULK COLLECT INTO prod_items

                FROM all_procedures

                WHERE OWNER=prod_schema_name;

        WHEN 'FUNCTION' THEN

            SELECT object_name

                BULK COLLECT INTO dev_items

                FROM all_objects

                WHERE OWNER=dev_schema_name
```

```
          AND object_type = 'FUNCTION';
    SELECT object_name
        BULK COLLECT INTO prod_items
        FROM all_objects
        WHERE OWNER=prod_schema_name
          AND object_type = 'FUNCTION';
WHEN 'INDEX' THEN
    SELECT index_name
        BULK COLLECT INTO dev_items
        FROM all_indexes
        WHERE OWNER=dev_schema_name;
    SELECT index_name
        BULK COLLECT INTO prod_items
        FROM all_indexes
        WHERE OWNER=prod_schema_name;
END CASE;

not_prod_items := dev_items MULTISET EXCEPT prod_items;
i := not_prod_items.FIRST;

WHILE i IS NOT NULL
LOOP
    current_item := not_prod_items(i);

    IF (current_item MEMBER OF different) THEN
        i := not_prod_items.NEXT(i);
        CONTINUE;
    END IF;
    IF (item_type = 'TABLE') THEN
        recursion_level := 0;
        add_table(current_item, not_prod_items);
        i := not_prod_items.NEXT(i);
        CONTINUE;
    END IF;

    different.EXTEND;
    different(different.COUNT) := current_item;

    dbms_output.put_line('Dev has unique '
                      || LOWER(item_type)
                      || ' "' || current_item || '"');
END LOOP;

same_items := dev_items MULTISET INTERSECT prod_items;
i := same_items.FIRST;

WHILE i IS NOT NULL
LOOP
    current_item := same_items(i);

    IF (item_type IN ('TABLE', 'INDEX'))
    THEN
        IF (dbms_metadata_diff.compare_alter(
                item_type, current_item,
                current_item, dev_schema_name,
                prod_schema_name
        ) = EMPTY_CLOB() )
        THEN
            dbms_output.put_line(
                'Dev and Prod has absolutly same '
                || LOWER(item_type) || ' "'
                || current_item || '"');
        ELSIF (dbms_metadata_diff.compare_alter(
            item_type, current_item, current_item,
            dev_schema_name, prod_schema_name
        )IS NOT NULL)
        THEN
            different.EXTEND;
            different(different.COUNT) := current_item;
            dbms_output.put_line(
                'Dev and Prod has difference in '
                || LOWER(item_type) || ' "'
                || current_item || '"');
        END IF;

    ELSIF (item_type IN ('PROCEDURE', 'FUNCTION')) THEN
        SELECT nvl(s1.text, s2.text)
            BULK COLLECT INTO lines
            FROM
                (SELECT text FROM all_source
                 WHERE type = current_item
                 AND OWNER = dev_schema_name) s1
            FULL OUTER JOIN
                (SELECT text FROM all_source
                 WHERE type = current_item
                 AND OWNER = prod_schema_name) s2
            ON s1.text = s2.text
            WHERE
                s1.text IS NULL OR s2.text IS NULL;

            IF (lines IS NOT NULL) THEN
                different.EXTEND;
```

```
                        different(different.COUNT) := current_item;
                        dbms_output.put_line(
                            'Dev and Prod has difference in '
                            || LOWER(item_type) || ' "'
                            || current_item || '"');
                    END IF;
            END IF;

            i:= same_items.NEXT(i);

        END LOOP;
    END;
BEGIN
    get_items_of_type('TABLE');
    dbms_output.put_line('');
    get_items_of_type('FUNCTION');
    dbms_output.put_line('');
    get_items_of_type('PROCEDURE');
    dbms_output.put_line('');
    get_items_of_type('INDEX');
END;

BEGIN
    different_schemas('DEV','PROD');
END;
```



```
Procedure DIFFERENT_SCHEMAS compiled

Dev has unique table "LESSONS"
Dev has unique table "EXAMS"
Dev has unique table "TEACHERS"
    Dev and Prod has absolutly same table "STUDENTS"
Dev and Prod has difference in table "TEST_TABLE"

Dev has unique function "COUNT_SUMM"

Dev has unique procedure "PROCEDURE1"
Dev and Prod has difference in procedure "PROCEDURE2"

Dev has unique index "PK_LESSONS"
Dev has unique index "PK_EXAMS"
Dev has unique index "TEACHERS_PK"
    Dev and Prod has absolutly same index "PK_STUDENTS"


PL/SQL procedure successfully completed.
```

1. Доработать предыдущий скрипт с генерацией ddl-скрипта на обновление объектов, а также с учетом необходимости удаления в схеме prod объектов, отсутствующих в схеме dev.

```
CREATE OR REPLACE PROCEDURE different_schemas_ddl(

    dev_schema_name VARCHAR2,

    prod_schema_name VARCHAR2) AUTHID CURRENT_USER

AS

    TYPE code_t IS TABLE OF CLOB;

    TYPE names_arr IS TABLE OF VARCHAR2(256);

    ddl_statements code_t := code_t();

    different names_arr := names_arr();

    recursion_level INTEGER;

    i INTEGER;


    PROCEDURE add_table(name_t VARCHAR2,

                        table_items names_arr,

                        owner_shema VARCHAR2)
```

```
AS
    parent_tables names_arr := names_arr();
    cycle_error EXCEPTION;
    i INT;
BEGIN
    IF (recursion_level > 100) THEN
        dbms_output.put_line('Cycle in ' || name_t);
        RAISE cycle_error;
    END IF;
    IF (name_t MEMBER OF different
        OR name_t NOT MEMBER OF table_items) THEN
        RETURN;
    END IF;

    SELECT c_pk.table_name
        BULK COLLECT INTO parent_tables
        FROM all_cons_columns a
        JOIN all_constraints c
            ON a.OWNER = c.OWNER
            AND a.constraint_name = c.constraint_name
        JOIN all_constraints c_pk
            ON c.r_owner = c_pk.OWNER
            AND c.r_constraint_name = c_pk.constraint_name
        WHERE
            c.constraint_type = 'R'
            AND a.table_name = name_t
            AND a.OWNER = owner_shema;

    IF (parent_tables.COUNT > 0) THEN
        i := parent_tables.FIRST;
        WHILE (i IS NOT NULL)
        LOOP
            recursion_level := recursion_level + 1;
            add_table(parent_tables(i),
                      table_items, owner_shema);
            recursion_level := recursion_level - 1;
            i := parent_tables.NEXT(i);
        END LOOP;
    END IF;
    different.EXTEND;
    different(different.COUNT) := name_t;
    dbms_output.put_line(INITCAP(owner_shema)
                         || ' has unique table "'
                         || name_t || '"');
END;

PROCEDURE add_table2(name_t VARCHAR2,
                     table_items names_arr,
                     owner_shema VARCHAR2)
AS
    children_tables names_arr := names_arr();
    cycle_error EXCEPTION;
    i INT;
BEGIN
    if(recursion_level > 100) THEN
        dbms_output.put_line('Cycle in ' || name_t);
        RAISE cycle_error;
    END IF;

    IF (name_t MEMBER OF different
        OR name_t NOT MEMBER OF table_items) THEN
        RETURN;
    END IF;

    SELECT c.table_name
        BULK COLLECT INTO children_tables
        FROM all_cons_columns a
        JOIN all_constraints c_pk
            ON a.OWNER =  c_pk.OWNER
            AND a.constraint_name = c_pk.constraint_name
        JOIN all_constraints c
            ON c.r_owner =  c_pk.OWNER
            AND c.r_constraint_name = c_pk.constraint_name
        WHERE
            c.constraint_type='R'
            AND a.table_name = name_t
            AND a.OWNER = owner_shema;

    IF (children_tables.COUNT > 0) THEN
        i := children_tables.FIRST;
        WHILE (i IS NOT NULL)
        LOOP
            recursion_level := recursion_level + 1;
            add_table2(children_tables(i),
                       table_items, owner_shema);
            recursion_level := recursion_level - 1;
            i := children_tables.NEXT(i);
        END LOOP;
    END IF;

    different.EXTEND;
```

```
    different(different.COUNT) := name_t;

    dbms_output.put_line(INITCAP(owner_shema)

                         || ' has unique table "'

                         || name_t || '"');

END;




PROCEDURE get_items_of_type(item_type VARCHAR2)

AS

    dev_items names_arr;

    prod_items names_arr;

    not_prod_items names_arr;

    not_dev_items names_arr;

    same_items names_arr;

    lines names_arr;

    current_item VARCHAR2(100);

    i INTEGER;


BEGIN

    CASE item_type

    WHEN 'TABLE' THEN

        SELECT table_name

            BULK COLLECT INTO dev_items

            FROM all_tables

            WHERE OWNER = dev_schema_name;

        SELECT table_name

            BULK COLLECT INTO prod_items

            FROM all_tables

            WHERE OWNER = prod_schema_name;
```

```
WHEN 'FUNCTION' THEN
    SELECT object_name
        BULK COLLECT INTO dev_items
        FROM all_objects
        WHERE OWNER = dev_schema_name
          AND object_type = 'FUNCTION';
    SELECT object_name
        BULK COLLECT INTO prod_items
        FROM all_objects
        WHERE OWNER = prod_schema_name
          AND object_type = 'FUNCTION';
WHEN 'PROCEDURE' THEN
    SELECT object_name
        BULK COLLECT INTO dev_items
        FROM all_objects
        WHERE OWNER = dev_schema_name
          AND object_type = 'PROCEDURE';
    SELECT object_name
        BULK COLLECT INTO prod_items
        FROM all_procedures
        WHERE OWNER = prod_schema_name;
WHEN 'INDEX' THEN
    SELECT index_name
        BULK COLLECT INTO dev_items
        FROM all_indexes
        WHERE OWNER = dev_schema_name;
    SELECT index_name
        BULK COLLECT INTO prod_items
        FROM all_indexes
        WHERE OWNER = prod_schema_name;
END CASE;
```

```
not_prod_items := dev_items
    MULTISET EXCEPT prod_items;
i := not_prod_items.FIRST;
IF i IS NOT NULL THEN
    dbms_output.put_line('Add ' || LOWER(item_type)
                         || '(s)' || chr(10)
                         || '----------------------------');
END IF;

WHILE i IS NOT NULL
LOOP
    current_item := not_prod_items(i);
    IF (current_item MEMBER OF different) THEN
        i := not_prod_items.NEXT(i);
        CONTINUE;
    ELSIF (item_type = 'TABLE') THEN
        recursion_level := 0;
        add_table(current_item,
                  not_prod_items,
                  dev_schema_name);
        i := not_prod_items.NEXT(i);
        CONTINUE;
    END IF;
    different.EXTEND;
    different(different.COUNT) := current_item;
    dbms_output.put_line('Dev has unique '
                         || LOWER(item_type)
                         || ' "' || current_item
                         || '"');
    i := not_prod_items.NEXT(i);
```

```
END LOOP;


i := different.FIRST;
WHILE i IS NOT NULL
LOOP
    current_item := different(i);
    ddl_statements.EXTEND;
    SELECT dbms_metadata.get_ddl(item_type,
                                 current_item,
                                 dev_schema_name)
        INTO ddl_statements (ddl_statements.COUNT)
        FROM dual;
    i:= different.NEXT(i);
END LOOP;


different := names_arr();


same_items := dev_items
    MULTISET INTERSECT prod_items;
i:= same_items.FIRST;
IF i IS NOT NULL THEN
    dbms_output.put_line(chr(10) || 'Update  '
                         || LOWER(item_type)
                         || '(s)' || chr(10)
                         || '----------------------------');
END IF;
WHILE i IS NOT NULL
LOOP
    current_item := same_items(i);
```

```
IF (item_type IN ('TABLE', 'INDEX')) THEN
     IF (dbms_metadata_diff.compare_alter(
        item_type, current_item,
        current_item, dev_schema_name,
        prod_schema_name
    ) = EMPTY_CLOB() ) THEN
        dbms_output.put_line(
            '(no update) absolutly same '
            || LOWER(item_type) || ' "'
            || current_item || '"');
    ELSIF (dbms_metadata_diff.compare_alter(
        item_type, current_item, current_item,
        dev_schema_name, prod_schema_name
    ) IS NOT NULL) THEN
        different.EXTEND;
        different(different.COUNT) := current_item;
        dbms_output.put_line(
            'Dev and Prod has difference in '
            || LOWER(item_type) || ' "'
            || current_item || '"');
    END IF;
ELSIF (item_type IN ('PROCEDURE',
       'FUNCTION')) THEN
    SELECT nvl(s1.text, s2.text)
        BULK COLLECT INTO lines
        FROM
            (SELECT text
             FROM all_source
             WHERE type = current_item
               AND OWNER = dev_schema_name) s1
        FULL OUTER JOIN
```

```
                (SELECT text
                 FROM all_source
                 WHERE type = current_item
                   AND OWNER = prod_schema_name) s2
            ON s1.text = s2.text
            WHERE
                s1.text IS NULL OR s2.text IS NULL;

        IF (lines IS NOT NULL) THEN
            different.EXTEND;
            different(different.COUNT) := current_item;
            dbms_output.put_line(
               'Dev and Prod has difference in '
               || LOWER(item_type) || ' "'
               || current_item || '"');
        END IF;
    END IF;
    i := same_items.NEXT(i);
END LOOP;

i := different.FIRST;
WHILE i IS NOT NULL
LOOP
    current_item := different(i);
    ddl_statements.EXTEND;
    IF (item_type = 'TABLE'
        OR item_type = 'INDEX') THEN
        SELECT dbms_metadata_diff.compare_alter(
            item_type, current_item, current_item,
            prod_schema_name, dev_schema_name)
        INTO ddl_statements(ddl_statements.COUNT)
```

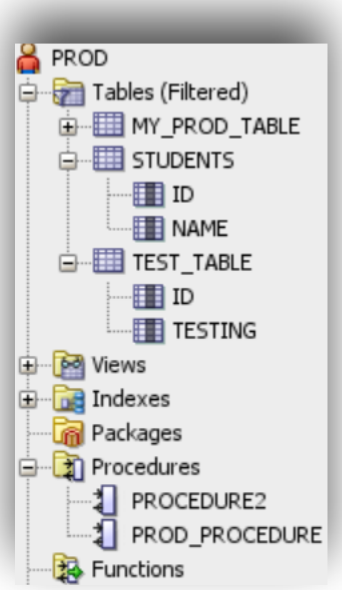
```
            FROM dual;
        ELSE
            ddl_statements(ddl_statements.COUNT) :=
                'DROP  ' || item_type || ' '
                || current_item || ';';
            ddl_statements.EXTEND;
            SELECT dbms_metadata.get_ddl(item_type,
                                         current_item,
                                         dev_schema_name)
            INTO ddl_statements (ddl_statements.COUNT)
            FROM dual;
        END IF;
        i:= different.NEXT(i);
    END LOOP;

    different:= names_arr();
    not_dev_items := prod_items MULTISET EXCEPT dev_items;
    i := not_dev_items.FIRST;

    IF i IS NOT NULL THEN
        dbms_output.put_line(chr(10) || 'Delete '
                             || LOWER(item_type) || '(s)'
                             || chr(10) ||
'---------------------------');
    END IF;

    WHILE i IS NOT NULL
    LOOP
        current_item := not_dev_items(i);
        IF (current_item MEMBER OF different) THEN
            i:= not_dev_items.NEXT(i);
            CONTINUE;
        END IF;
        IF (item_type='TABLE') THEN
            recursion_level := 0;
            add_table2(current_item,
                       not_dev_items, prod_schema_name);
            i:= not_dev_items.NEXT(i);
            CONTINUE;
        END IF;
        different.EXTEND;
        different(different.COUNT) := current_item;
        dbms_output.put_line('Prod has unique '
                             || LOWER(item_type)
                             || ' "' || current_item
                             || '"');
        i := not_dev_items.NEXT(i);
    END LOOP;

    i:= different.FIRST;
    WHILE i IS NOT NULL
    LOOP
        current_item := different(i);
        ddl_statements.EXTEND;
        ddl_statements(ddl_statements.count) :=
            'DROP ' || item_type || ' PROD.'
            || current_item || ';';
        i := different.NEXT(i);
    END LOOP;

    different := names_arr();
END;
```

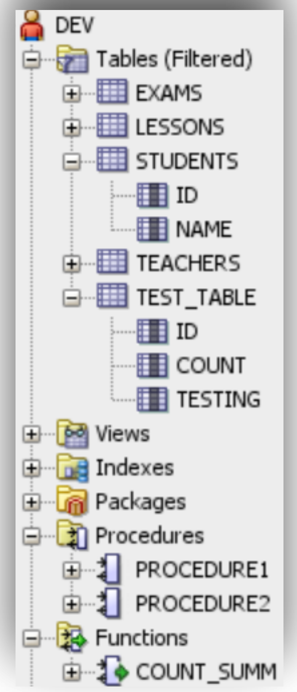
```
BEGIN
    get_items_of_type('TABLE');
    dbms_output.put_line('');
    get_items_of_type('FUNCTION');
    dbms_output.put_line('');
    get_items_of_type('PROCEDURE');
    dbms_output.put_line('');
    get_items_of_type('INDEX');

    i := ddl_statements.FIRST;
    WHILE i IS NOT NULL
    LOOP
        dbms_output.put_line(REPLACE(ddl_statements(i),
                                     'DEV', 'PROD'));
        i := ddl_statements.NEXT(i);
    END LOOP;
END;

BEGIN
    different_schemas_ddl('DEV','PROD');
END;
```

Состояние схем на момент выполнения процедуры:

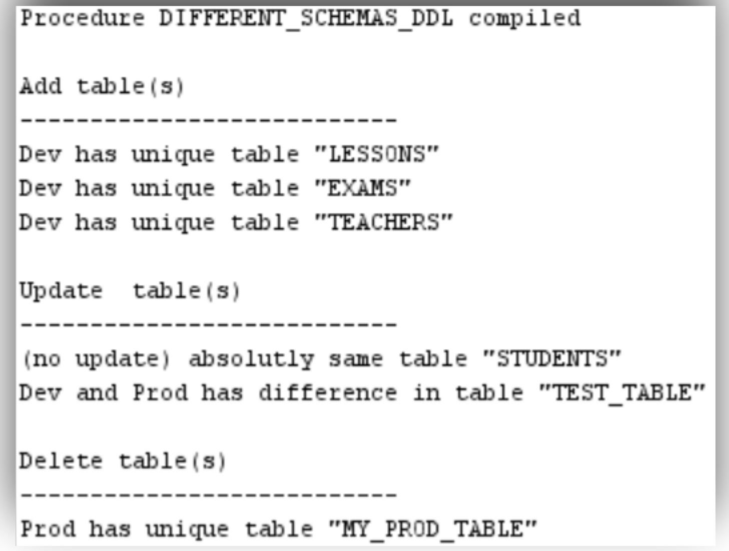
```
CREATE TABLE PROD.my_prod_table(
    id NUMBER,
    str VARCHAR2(100)
);

CREATE OR REPLACE PROCEDURE PROD.Procedure2
```

```
AS

BEGIN

    dbms_output.put_line('Procedure2 prod user');

END;
```

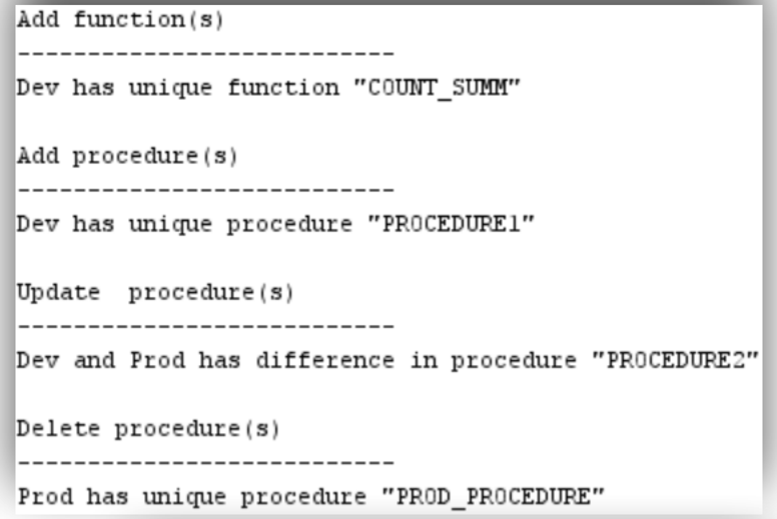




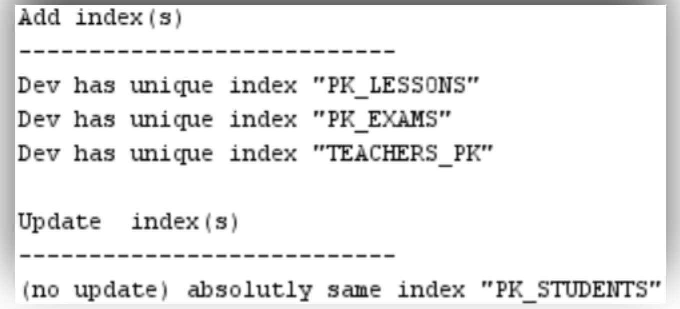
Результат работы процедуры:



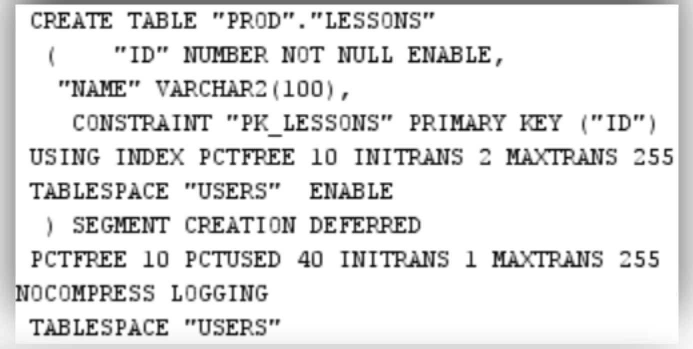
```
Procedure DIFFERENT_SCHEMAS_DDL compiled

Add table(s)
---------------------------
Dev has unique table "LESSONS"
Dev has unique table "EXAMS"
Dev has unique table "TEACHERS"

Update  table(s)
---------------------------
(no update) absolutly same table "STUDENTS"
Dev and Prod has difference in table "TEST_TABLE"

Delete table(s)
---------------------------
Prod has unique table "MY_PROD_TABLE"
```



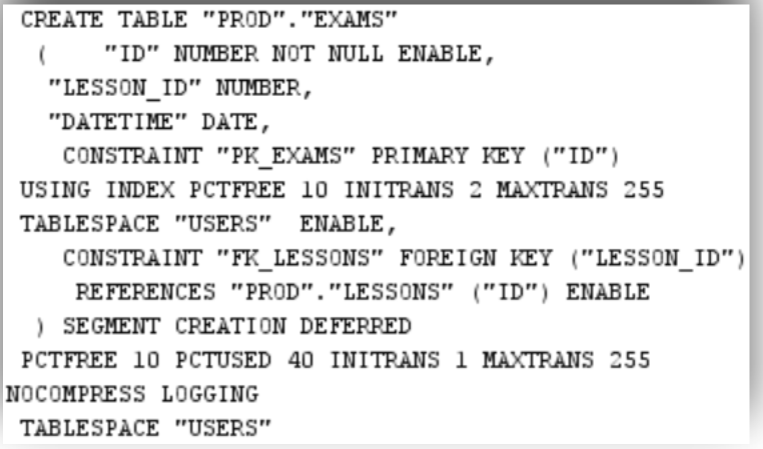
```
Add function(s)
---------------------------
Dev has unique function "COUNT_SUMM"

Add procedure(s)
---------------------------
Dev has unique procedure "PROCEDURE1"

Update  procedure(s)
---------------------------
Dev and Prod has difference in procedure "PROCEDURE2"

Delete procedure(s)
---------------------------
Prod has unique procedure "PROD_PROCEDURE"
```



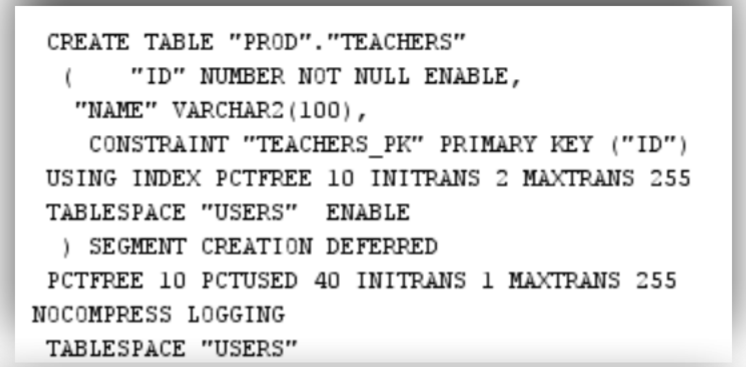
```
Add index(s)
---------------------------
Dev has unique index "PK_LESSONS"
Dev has unique index "PK_EXAMS"
Dev has unique index "TEACHERS_PK"

Update  index(s)
---------------------------
(no update) absolutly same index "PK_STUDENTS"
```



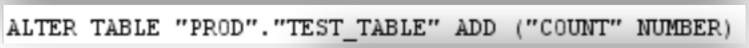
```
 CREATE TABLE "PROD"."LESSONS"
  (    "ID" NUMBER NOT NULL ENABLE,
   "NAME" VARCHAR2(100),
    CONSTRAINT "PK_LESSONS" PRIMARY KEY ("ID")
 USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
 TABLESPACE "USERS"  ENABLE
  ) SEGMENT CREATION DEFERRED
 PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
 TABLESPACE "USERS"
```



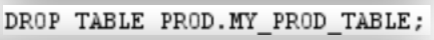
```
 CREATE TABLE "PROD"."EXAMS"
  (    "ID" NUMBER NOT NULL ENABLE,
   "LESSON_ID" NUMBER,
   "DATETIME" DATE,
    CONSTRAINT "PK_EXAMS" PRIMARY KEY ("ID")
 USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
 TABLESPACE "USERS"  ENABLE,
    CONSTRAINT "FK_LESSONS" FOREIGN KEY ("LESSON_ID")
     REFERENCES "PROD"."LESSONS" ("ID") ENABLE
  ) SEGMENT CREATION DEFERRED
 PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
 TABLESPACE "USERS"
```



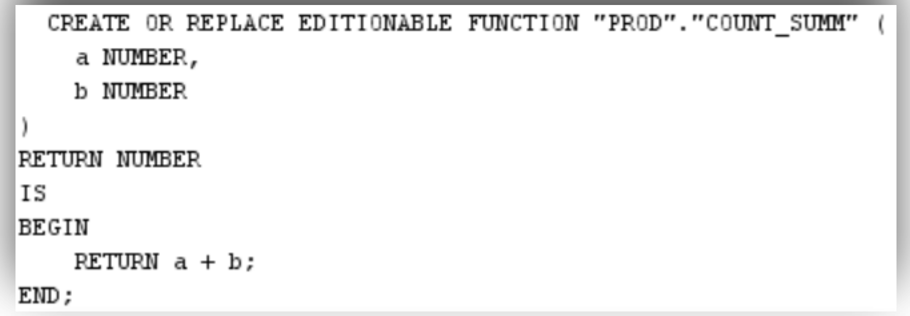
```
 CREATE TABLE "PROD"."TEACHERS"
  (    "ID" NUMBER NOT NULL ENABLE,
   "NAME" VARCHAR2(100),
    CONSTRAINT "TEACHERS_PK" PRIMARY KEY ("ID")
 USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
 TABLESPACE "USERS"  ENABLE
  ) SEGMENT CREATION DEFERRED
 PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
 TABLESPACE "USERS"
```



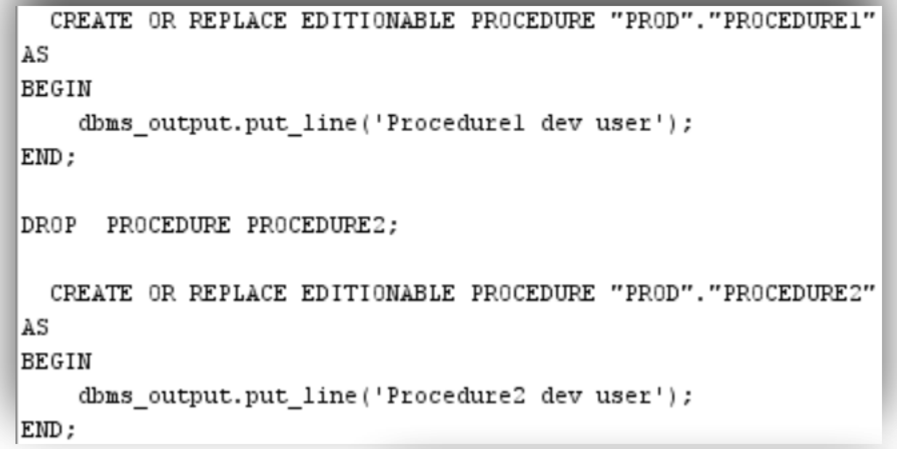
```
ALTER TABLE "PROD"."TEST_TABLE" ADD ("COUNT" NUMBER)
```



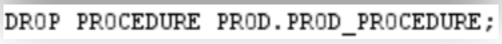
```
DROP TABLE PROD.MY_PROD_TABLE;
```



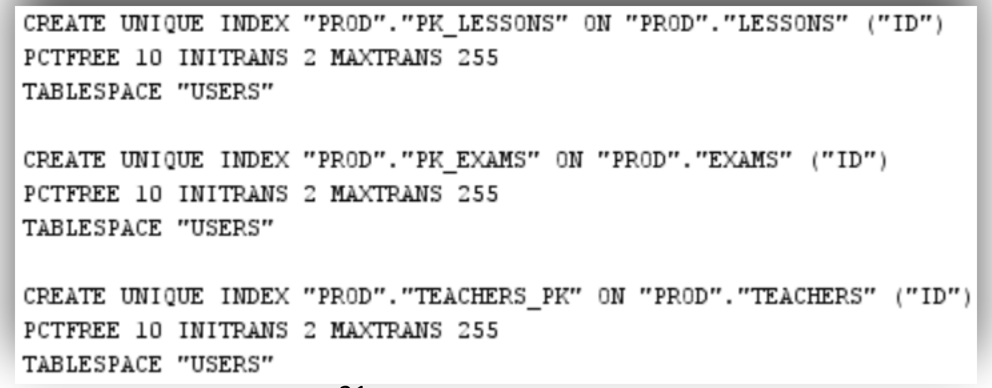
```
  CREATE OR REPLACE EDITIONABLE FUNCTION "PROD"."COUNT_SUMM" (
    a NUMBER,
    b NUMBER
)
RETURN NUMBER
IS
BEGIN
    RETURN a + b;
END;
```



```
  CREATE OR REPLACE EDITIONABLE PROCEDURE "PROD"."PROCEDURE1"
AS
BEGIN
    dbms_output.put_line('Procedure1 dev user');
END;

DROP  PROCEDURE PROCEDURE2;

  CREATE OR REPLACE EDITIONABLE PROCEDURE "PROD"."PROCEDURE2"
AS
BEGIN
    dbms_output.put_line('Procedure2 dev user');
END;
```



```
DROP PROCEDURE PROD.PROD_PROCEDURE;
```



```
CREATE UNIQUE INDEX "PROD"."PK_LESSONS" ON "PROD"."LESSONS" ("ID")
PCTFREE 10 INITRANS 2 MAXTRANS 255
TABLESPACE "USERS"

CREATE UNIQUE INDEX "PROD"."PK_EXAMS" ON "PROD"."EXAMS" ("ID")
PCTFREE 10 INITRANS 2 MAXTRANS 255
TABLESPACE "USERS"

CREATE UNIQUE INDEX "PROD"."TEACHERS_PK" ON "PROD"."TEACHERS" ("ID")
PCTFREE 10 INITRANS 2 MAXTRANS 255
TABLESPACE "USERS"
```