Университет ИТМО

**Курсовая работа**

**по дисциплине: «Системы баз данных»**

Выполнил:

студент III курса

группы 3125

Припадчев Артём

Проверит:

Беликов П.А.

Санкт-Петербург

2014

Оглавление

[Описание предметной области 3](#_Toc405734075)

[ER-модель базы данных 4](#_Toc405734076)

[Инфологическая модель 5](#_Toc405734077)

[Создание таблиц, триггеров и последовательностей 6](#_Toc405734078)

[Программа генерации набора данных для базы данных 12](#_Toc405734079)

[Зачетные задачи 16](#_Toc405734080)

[Задача 1. 16](#_Toc405734081)

[Задача 2. 17](#_Toc405734082)

[Задача 3. 21](#_Toc405734083)

[Задача 4 26](#_Toc405734084)

[Использованная литература 27](#_Toc405734085)

# Описание предметной области

Задачей построения базы данных является зачисление абитуриентов в ВУЗы.

До того, как ВУЗ начинает прием документов, он формирует список документов, которые регламентируют прием. Правила приема могут изменяться ежегодно, т.к. список специальностей обучения динамичен, также изменяются законы РФ в части предоставления льгот гражданам при поступлении в ВУЗы.

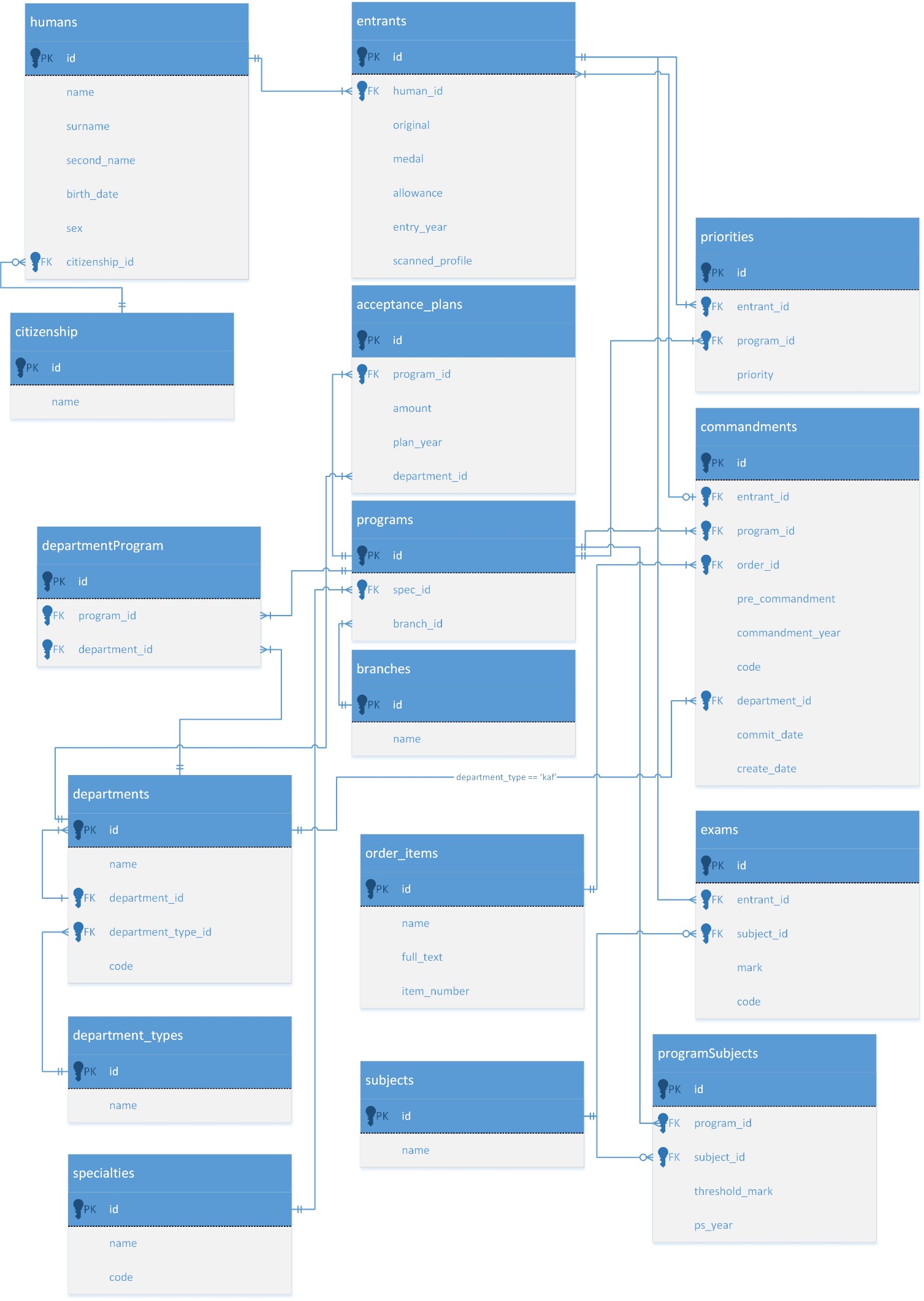
Разработка базы данных абитуриентов поможет приемной комиссии большую часть работы автоматизировать. В отличии от стеллажей с огромным количеством личных дел, БД позволяет осуществлять мгновенный доступ к информации о любом абитуриенте, а также без ошибок формировать списки людей по различным критериям.

Процесс зачисления в ВУЗ осуществляется в несколько этапов. Начинается всё с приема документов у абитуриента. При приеме документов абитуриент заявляет о своём желании участвовать в конкурсе на конкретную специальность, а если точнее, то на учебную программу. Выбирать специальностей можно несколько, выставляя для каждой свой приоритет. В таком случае, если студент не проходит по конкурсу на специальность с 1 приоритетом, то начинает участвовать в конкурсе на специальности со 2 приоритетом и так далее.

Помимо информации о выбранных специальностях и экзаменах, абитуриент предоставляет некоторые дополнительные сведения (ФИО, паспортные данные, гражданство и так далее), а также (при наличии) документы, которые подтверждают право на льготы.

Следующий этап – зачисление абитуриентов с самым высоким рейтингом. В связи с тем, что по законодательству России выделяются разные категории граждан, имеющих льготы при поступлении, приемная комиссия формирует несколько приказов со списками абитуриентов, зачисленных по определенному нормативному документу. Однако основная часть поступающих не имеет дополнительных льгот и участвует в общем конкурсе. Именно эта категория требует автоматизированной обработки.

# ER-модель базы данных



# Инфологическая модель

**humans (люди)**

**id –** ИД – NUMBER

**name** – имя – VARCHAR2

**surname** – фамилия – VARCHAR2

**second\_name –** отчество – VARCHAR2

**birth\_date –** дата рождения – DATE

**sex** – пол – CHAR

**citizenship\_id**  - ИД гражданства – NUMBER

**citizenship (гражданство)**

**id –** ИД – NUMBER

**name –** название – VARCHAR2

**entrants (абитуриенты)**

**id –** ИД – NUMBER

**human\_id** – ИД человека – NUMBER

**original –** наличие оригинала – CHAR(1)

**medal –** наличие медали – CHAR(1)

**allowance –** наличие льготы – CHAR(1)

**entry\_year** – год поступления – NUMBER

**scanned\_profile –** скан анкеты – BLOB

**acceptance\_plans (план приема абит-ов)**

**id –** ИД - NUMBER

**program\_id** – ИД программы – NUMBER

**department\_id –** ИД кафедры - NUMBER

**amount –** кол-во людей – NUMBER

**plan\_year** – год – NUMBER

**priorities (приоритеты)**

**id –** ИД - NUMBER

**entrant\_id –** ИД абитуриента - NUMBER

**program\_id –** ИД программы – NUMBER

**priority –** приоритет – NUMBER

**commandments (приказы)**

**id –**ИД – NUMBER

**entrant\_id –** ИД абитуриента – NUMBER

**program\_id –** ИД программы – NUMBER

**order\_id –** ИД пункта приказа – NUMBER

**pre\_commandment –** предварительный приказ – CHAR(1)

**commandment\_year** – год – NUMBER

**code –** номер приказа - NUMBER

**department\_id** – ИД кафедры - NUMBER

**commit\_date –** дата принятия – DATE

**create\_date** – дата создания – DATE

**programs (программы)**

**id –** ИД - NUMBER

**spec\_id –** ИД специальности - NUMBER

**branch\_id –** ИД отделения – NUMBER

**branches (отделения)**

**id –** ИД – NUMBER

**name –** название – VARCHAR2

**departmentProgram (программы на кафедрах)**

**id –** ИД – NUMBER

**program\_id** – ИД программы - NUMBER

**department\_id** – ИД кафедры - NUMBER

**departments (кафедры/факультеты)**

**id –** ИД – NUMBER

**name –** название – VARCHAR2

**department\_id –** ИД принадлежности - NUMBER

**department\_type\_id –** ИД типа отделения – NUMBER

**code –** номер – NUMBER

**department\_types (типы отделов)**

**id –** ИД – NUMBER

**name –** название – VARCHAR2

**specialties (специальности)**

**id –** ИД – NUMBER

**name –** название – VARCHAR2

**code –** номер – NUMBER

**orders\_items (пункты приказа)**

**id –** ИД – NUMBER

**name –** название – VARCHAR2

**full\_text –** содержание пункта – CLOB

**item\_number –** номер пункта – NUMBER

**subjects (предметы)**

**id –** ИД – NUMBER

**name –** название – VARCHAR2

**exams (экзамены)**

**id –** ИД – NUMBER

**entrant\_id** – ИД абитуриента – NUMBER

**subject\_id –** ИД предмета – NUMBER

**mark –** балл – NUMBER

**code –** номер сертификата – NUMBER

**programSubjects (предметы для поступления на программу)**

**id –** ИД – NUMBER

**program\_id –** ИД программы – NUMBER

**subject\_id** – ИД предмета – NUMBER

**threshold\_mark –** пороговый балл – NUMBER

**ps\_year –** год - DATE

# Создание таблиц, триггеров и последовательностей

CREATE TABLE CITIZENSHIP

(

ID NUMBER(6,0) NOT NULL CONSTRAINT "CITIZENSHIP\_PK" PRIMARY KEY,

NAME VARCHAR2(50) NOT NULL,

CONSTRAINT "CITIZENSHIP\_UK" UNIQUE (NAME)

)

/

CREATE SEQUENCE CITIZENSHIP\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER CITIZENSHIP\_BIR

BEFORE INSERT ON CITIZENSHIP

FOR EACH ROW

BEGIN

SELECT CITIZENSHIP\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

:NEW.NAME := INITCAP(:NEW.NAME);

END CITIZENSHIP\_BIR;

/

CREATE TABLE HUMANS

(

ID NUMBER(6,0) NOT NULL CONSTRAINT "HUMANS\_PK" PRIMARY KEY,

NAME VARCHAR2(50) NOT NULL,

SURNAME VARCHAR2(50) NOT NULL,

SECOND\_NAME VARCHAR(50) NOT NULL,

BIRTH\_DATE DATE NOT NULL,

SEX CHAR(1) NOT NULL CONSTRAINT "M/F"

CHECK (SEX IN ('M','F')),

CITIZENSHIP\_ID NUMBER(6,0) NOT NULL CONSTRAINT "CITIZENSHIP\_FK"

REFERENCES CITIZENSHIP(ID),

CONSTRAINT "HUMANS\_UK" UNIQUE (SURNAME,NAME,SECOND\_NAME,BIRTH\_DATE,SEX)

);

CREATE SEQUENCE HUMANS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER HUMANS\_BIR

BEFORE INSERT ON HUMANS

FOR EACH ROW

BEGIN

SELECT HUMANS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

:NEW.SURNAME := INITCAP(:NEW.SURNAME);

:NEW.NAME := INITCAP(:NEW.NAME);

:NEW.SECOND\_NAME := INITCAP(:NEW.SECOND\_NAME);

IF :NEW.BIRTH\_DATE IS NULL THEN

:NEW.BIRTH\_DATE := TO\_DATE('11.11.1111','DD.MM.YYYY');

END IF;

END HUMANS\_BIR;

/

CREATE TABLE ENTRANTS

(

ID NUMBER(6,0) NOT NULL CONSTRAINT "ENTRANTS\_PK" PRIMARY KEY,

HUMAN\_ID NUMBER(6,0) NOT NULL CONSTRAINT "ENTRANTS\_FK"

REFERENCES HUMANS(ID),

ORIGINAL CHAR(1) NOT NULL CONSTRAINT "ORIGINAL IS Y/N"

CHECK (ORIGINAL IN ('Y','N')),

MEDAL CHAR(1) NOT NULL CONSTRAINT "MEDAL IS Y/N"

CHECK (MEDAL IN ('Y','N')),

ALLOWANCE CHAR(1) NOT NULL CONSTRAINT "ALLOWANCE IS Y/N"

CHECK (ALLOWANCE IN ('Y','N')),

ENTRY\_YEAR NUMBER(4,0) NOT NULL,

SCANNED\_PROFILE BLOB,

CONSTRAINT "ENTRANTS\_UK" UNIQUE (HUMAN\_ID,ENTRY\_YEAR)

);

CREATE SEQUENCE ENTRANTS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER ENTRANTS\_BIR

BEFORE INSERT ON ENTRANTS

FOR EACH ROW

BEGIN

SELECT ENTRANTS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

END ENTRANTS\_BIR;

/

CREATE TABLE BRANCHES

(

ID NUMBER(3,0) NOT NULL CONSTRAINT "BRANCHES\_PK" PRIMARY KEY,

NAME VARCHAR2(50) NOT NULL,

CONSTRAINT "BRANCHES\_UK" UNIQUE(NAME)

);

CREATE SEQUENCE BRANCHES\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER BRANCHES\_BIR

BEFORE INSERT ON BRANCHES

FOR EACH ROW

BEGIN

SELECT BRANCHES\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

:NEW.NAME := INITCAP(:NEW.NAME);

END BRANCHES\_BIR;

/

CREATE TABLE SPECIALTIES

(

ID NUMBER(3,0) NOT NULL CONSTRAINT "SPECIALTIES\_PK" PRIMARY KEY,

NAME VARCHAR2(50) NOT NULL,

CODE VARCHAR2(10) NOT NULL,

CONSTRAINT "SPECIALTIES\_UK" UNIQUE (CODE)

);

CREATE SEQUENCE SPECIALTIES\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER SPECIALTIES\_BIR

BEFORE INSERT ON SPECIALTIES

FOR EACH ROW

BEGIN

SELECT SPECIALTIES\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

:NEW.NAME := UPPER(:NEW.NAME);

END SPECIALTIES\_BIR;

/

CREATE TABLE PROGRAMS

(

ID NUMBER(3,0) NOT NULL CONSTRAINT "PROGRAMS\_PK" PRIMARY KEY,

SPEC\_ID NUMBER(3,0) NOT NULL CONSTRAINT "SPECIALTIES\_FK"

REFERENCES SPECIALTIES(ID),

BRANCH\_ID NUMBER(3,0) NOT NULL CONSTRAINT "BRANCHES\_FK"

REFERENCES BRANCHES(ID),

CONSTRAINT "PROGRAMS\_UK" UNIQUE (SPEC\_ID,BRANCH\_ID)

);

CREATE SEQUENCE PROGRAMS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER PROGRAMS\_BIR

BEFORE INSERT ON PROGRAMS

FOR EACH ROW

BEGIN

SELECT PROGRAMS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

END PROGRAMS\_BIR;

/

CREATE TABLE DEPARTMENT\_TYPES

(

ID NUMBER(3,0) NOT NULL CONSTRAINT "DEPARTMENT\_TYPES\_PK" PRIMARY KEY,

NAME VARCHAR2(50) NOT NULL,

CONSTRAINT "DEPARTMENT\_TYPES\_UK" UNIQUE(NAME)

);

CREATE SEQUENCE DEPARTMENT\_TYPES\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER DEPARTMENT\_TYPES\_BIR

BEFORE INSERT ON DEPARTMENT\_TYPES

FOR EACH ROW

BEGIN

SELECT DEPARTMENT\_TYPES\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

:NEW.NAME := INITCAP(:NEW.NAME);

END DEPARTMENT\_TYPES\_BIR;

/

CREATE TABLE DEPARTMENTS

(

ID NUMBER (3,0) NOT NULL CONSTRAINT "DEPARTMENTS\_PK" PRIMARY KEY,

NAME VARCHAR2(100) NOT NULL,

DEPARTMENT\_ID NUMBER (3,0) CONSTRAINT "DEPARTMENTS\_FK"

REFERENCES DEPARTMENTS(ID),

DEPARTMENT\_TYPE\_ID NUMBER (3,0) NOT NULL CONSTRAINT "DEPARTMENT\_TYPES\_FK" REFERENCES DEPARTMENT\_TYPES(ID),

CODE NUMBER(3,0) NOT NULL,

ABBR VARCHAR2(15) NOT NULL,

CONSTRAINT "DEPARTMENTS\_UK" UNIQUE(CODE)

);

CREATE SEQUENCE DEPARTMENTS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER DEPARTMENTS\_BIR

BEFORE INSERT ON DEPARTMENTS

FOR EACH ROW

BEGIN

SELECT DEPARTMENTS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

:NEW.NAME := UPPER(:NEW.NAME);

END DEPARTMENTS\_BIR;

/

CREATE TABLE ACCEPTANCE\_PLANS

(

ID NUMBER(3,0) NOT NULL CONSTRAINT "ACCEPTANCE\_PLANS\_PK" PRIMARY KEY,

PROGRAM\_ID NUMBER(3,0) NOT NULL CONSTRAINT "PLANS\_PROGRAMS\_FK"

REFERENCES PROGRAMS(ID),

DEPARTMENT\_ID NUMBER(3,0) NOT NULL CONSTRAINT "PLANS\_DEPARTMENTS\_FK"

REFERENCES DEPARTMENTS(ID),

AMOUNT NUMBER(3,0) NOT NULL,

PLAN\_YEAR NUMBER (4,0) NOT NULL,

CONSTRAINT "ACCEPTANCE\_PLANS\_UK" UNIQUE(PROGRAM\_ID,DEPARTMENT\_ID)

);

CREATE SEQUENCE ACCEPTANCE\_PLANS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER ACCEPTANCE\_PLANS\_BIR

BEFORE INSERT ON ACCEPTANCE\_PLANS

FOR EACH ROW

BEGIN

SELECT ACCEPTANCE\_PLANS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

END ACCEPTANCE\_PLANS\_BIR;

/

CREATE TABLE DEPARTMENTPROGRAM

(

ID NUMBER(3,0) NOT NULL CONSTRAINT "DEPARTMENTPROGRAM\_PK" PRIMARY KEY,

PROGRAM\_ID NUMBER(3,0) NOT NULL CONSTRAINT "DP\_PROGRAMS\_FK" REFERENCES PROGRAMS(ID),

DEPARTMENT\_ID NUMBER(3,0) NOT NULL CONSTRAINT "DP\_DEPARTMENT\_FK" REFERENCES DEPARTMENTS(ID),

CONSTRAINT "DEPARTMENTPROGRAM\_UK" UNIQUE(PROGRAM\_ID,DEPARTMENT\_ID)

);

CREATE SEQUENCE DEPARTMENTPROGRAM\_SEQ INCREMENT BY 1 START WITH 5;

CREATE OR REPLACE TRIGGER DEPARTMENTPROGRAM\_BIR

BEFORE INSERT ON DEPARTMENTPROGRAM

FOR EACH ROW

BEGIN

SELECT DEPARTMENTPROGRAM\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

END DEPARTMENTPROGRAM\_BIR;

/

CREATE TABLE SUBJECTS

(

ID NUMBER(3,0) NOT NULL CONSTRAINT "SUBJECTS\_PK" PRIMARY KEY,

NAME VARCHAR2(50) NOT NULL,

CONSTRAINT "SUBJECTS\_UK" UNIQUE(NAME)

);

CREATE SEQUENCE SUBJECTS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER SUBJECTS\_BIR

BEFORE INSERT ON SUBJECTS

FOR EACH ROW

BEGIN

SELECT SUBJECTS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

END SUBJECTS\_BIR;

/

CREATE TABLE ORDERS\_ITEMS

(

ID NUMBER(3,0) NOT NULL CONSTRAINT "ORDERS\_ITEMS" PRIMARY KEY,

NAME VARCHAR2(50) NOT NULL,

FULL\_TEXT CLOB,

ITEM\_NUMBER NUMBER(2,0) NOT NULL,

CONSTRAINT "ORDERS\_ITEMS\_UK" UNIQUE(NAME)

);

CREATE SEQUENCE ORDERS\_ITEMS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER ORDERS\_ITEMS\_BIR

BEFORE INSERT ON ORDERS\_ITEMS

FOR EACH ROW

BEGIN

SELECT ORDERS\_ITEMS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

:NEW.NAME := INITCAP(:NEW.NAME);

END ORDERS\_ITEMS\_BIR;

/

CREATE TABLE PRIORITIES

(

ID NUMBER(6,0) NOT NULL CONSTRAINT "PRIORITIES\_PK" PRIMARY KEY,

ENTRANT\_ID NUMBER(6,0) NOT NULL CONSTRAINT "PR\_ENTRANTS\_FK" REFERENCES ENTRANTS(ID),

PROGRAM\_ID NUMBER(6,0) NOT NULL CONSTRAINT "PR\_PROGRAMS\_FK" REFERENCES DEPARTMENTPROGRAM(ID),

PRIORITY NUMBER(1,0) NOT NULL CONSTRAINT "Приоритет не больше 3" CHECK(PRIORITY IN (1,2,3)),

CONSTRAINT "PRIORITIES\_UK" UNIQUE(ENTRANT\_ID,PRIORITY)

);

CREATE SEQUENCE PRIORITIES\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER PRIORITIES\_BIR

BEFORE INSERT ON PRIORITIES

FOR EACH ROW

BEGIN

SELECT PRIORITIES\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

END PRIORITIES\_BIR;

/

CREATE TABLE EXAMS

(

ID NUMBER(6,0) NOT NULL CONSTRAINT "EXAMS\_PK" PRIMARY KEY,

ENTRANT\_ID NUMBER (6,0) NOT NULL CONSTRAINT "EXAMS\_ENTRANTS\_FK" REFERENCES ENTRANTS(ID),

SUBJECT\_ID NUMBER(3,0) NOT NULL CONSTRAINT "EXAMS\_SUBJECTS\_FK" REFERENCES SUBJECTS(ID),

MARK NUMBER(3,0) NOT NULL,

CODE NUMBER(9,0) NOT NULL,

CONSTRAINT "EXAMS\_UK" UNIQUE(CODE)

);

CREATE SEQUENCE EXAMS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER EXAMS\_BIR

BEFORE INSERT ON EXAMS

FOR EACH ROW

BEGIN

SELECT EXAMS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

END EXAMS\_BIR;

/

CREATE TABLE PROGRAMSUBJECTS

(

ID NUMBER(6,0) NOT NULL CONSTRAINT "PROGRAMSUBJECTS\_PK" PRIMARY KEY,

PROGRAM\_ID NUMBER(3,0) NOT NULL CONSTRAINT "PS\_PROGRAMS\_FK" REFERENCES PROGRAMS(ID),

SUBJECT\_ID NUMBER(3,0) NOT NULL CONSTRAINT "PS\_SUBJECTS\_FK" REFERENCES SUBJECTS(ID),

THRESHOLD\_MARK NUMBER(3,0) NOT NULL,

PS\_YEAR NUMBER(4,0) NOT NULL,

CONSTRAINT "PROGRAMSUBJECTS\_UK" UNIQUE(PROGRAM\_ID,SUBJECT\_ID,PS\_YEAR)

);

CREATE SEQUENCE PROGRAMSUBJECTS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER PROGRAMSUBJECTS\_BIR

BEFORE INSERT ON PROGRAMSUBJECTS

FOR EACH ROW

BEGIN

SELECT PROGRAMSUBJECTS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

END PROGRAMSUBJECTS\_BIR;

/

CREATE TABLE COMMANDMENTS

(

ID NUMBER(6,0) NOT NULL CONSTRAINT "COMMANDMENTS\_PK" PRIMARY KEY,

ENTRANT\_ID NUMBER(6,0) NOT NULL CONSTRAINT "COMM\_ENTRANTS\_FK" REFERENCES ENTRANTS(ID),

PROGRAM\_ID NUMBER(3,0) NOT NULL CONSTRAINT "COMM\_PROGRAMS\_FK" REFERENCES PROGRAMS(ID),

ORDER\_ID NUMBER(3,0) NOT NULL CONSTRAINT "COMM\_ORDERS\_ITEMS\_FK" REFERENCES ORDERS\_ITEMS,

PRE\_COMMANDMENT CHAR(1) NOT NULL CONSTRAINT "PRE\_COMMANDMENT IS Y/N"

CHECK (PRE\_COMMANDMENT IN ('Y','N')),

COMMANDMENT\_YEAR NUMBER(4,0) NOT NULL,

CODE NUMBER(9,0) NOT NULL,

DEPARTMENT\_ID NUMBER(3,0) NOT NULL CONSTRAINT "COMM\_DEPARTMENTS\_FK" REFERENCES DEPARTMENTS(ID),

COMMIT\_DATE DATE,

CREATE\_DATE DATE,

CONSTRAINT "COMMANDMENTS\_UK" UNIQUE(CODE)

);

CREATE SEQUENCE COMMANDMENTS\_SEQ INCREMENT BY 1 START WITH 1;

CREATE OR REPLACE TRIGGER COMMANDMENTS\_BIR

BEFORE INSERT ON COMMANDMENTS

FOR EACH ROW

BEGIN

SELECT COMMANDMENTS\_SEQ.NEXTVAL INTO :NEW.ID FROM DUAL;

:NEW.CREATE\_DATE := SYSDATE;

END COMMANDMENTS\_BIR;

/

# Программа генерации набора данных для базы данных

class Program

{

static readonly Random rndGen = new Random();

static readonly Random gen = new Random();

static void Main()

{

/\*string[] array = {"Русский язык", "Математика", "Физика", "Информатика", "Английский язык", "История", "Обществознание",

"Химия", "Биология", "Литература"};

string[] array = { "Дневное", "Вечернее", "Заочное" };

string[] array = {"Прикладная математика и информатика", "Информатика и вычислительная техника",

"Прикладная информатика", "Программная инженерия", "Информационная безопасность",

"Приборостроение", "Оптотехника", "Техническая физика", "Менеджмент",

"Интеллектуальные системы в гуманитарной сфере"};

string[] array2 = {"01.03.02", "09.03.01", "09.03.03", "09.03.04", "10.03.01", "12.03.01", "12.03.02",

"16.03.01", "38.03.02", "45.03.04"};

\*/

Console.WriteLine("Complite");

Console.ReadLine();

}

static void GetExams()

{

using (StreamWriter sw = new StreamWriter(@"insert\_exams.sql", false, Encoding.UTF8))

{

sw.WriteLine("DECLARE");

sw.WriteLine(" V\_ID ENTRANTS.ID%TYPE;");

sw.WriteLine(" CURSOR GET\_ENTRANTS\_ID IS");

sw.WriteLine(" SELECT ID FROM ENTRANTS;");

sw.WriteLine("BEGIN");

sw.WriteLine(" OPEN GET\_ENTRANTS\_ID;");

for (int i = 0; i < 10000; i++)

{

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,1," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,2," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,3," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,4," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,5," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,6," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,7," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,8," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,9," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO EXAMS(ENTRANT\_ID, SUBJECT\_ID, MARK, CODE)");

sw.WriteLine("VALUES(V\_ID,10," + rndGen.Next(30, 101) + "," + rndGen.Next(1, 999999999) + ");");

}

sw.WriteLine("END;");

sw.WriteLine("/");

}

}

static void GetPriorities()

{

using (StreamWriter sw = new StreamWriter(@"insert\_priorityes.sql", false, Encoding.Default))

{

sw.WriteLine("DECLARE");

sw.WriteLine(" V\_ID ENTRANTS.ID%TYPE;");

sw.WriteLine(" CURSOR GET\_ENTRANTS\_ID IS");

sw.WriteLine(" SELECT ID FROM ENTRANTS;");

sw.WriteLine("BEGIN");

sw.WriteLine(" OPEN GET\_ENTRANTS\_ID;");

int first = 0;

int second = 0;

int third = 0;

for (int i = 0; i < 10000; i++)

{

first = rndGen.Next(5, 32);

second = rndGen.Next(5, 32);

while (second == first)

{

second = rndGen.Next(5, 32);

}

third = rndGen.Next(5, 32);

while ((third == first) || (third == second))

{

third = rndGen.Next(5, 32);

}

sw.WriteLine("FETCH GET\_ENTRANTS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO PRIORITIES(ENTRANT\_ID,PROGRAM\_ID,PRIORITY)");

sw.WriteLine("VALUES (V\_ID," + first + ",1);");

sw.WriteLine("INSERT INTO PRIORITIES(ENTRANT\_ID,PROGRAM\_ID,PRIORITY)");

sw.WriteLine("VALUES (V\_ID," + second + ",2);");

sw.WriteLine("INSERT INTO PRIORITIES(ENTRANT\_ID,PROGRAM\_ID,PRIORITY)");

sw.WriteLine("VALUES (V\_ID," + third + ",3);");

sw.WriteLine();

}

sw.WriteLine("END;");

sw.WriteLine("/");

}

}

static void GetEntrants()

{

const string yn = "YN";

StringBuilder original = new StringBuilder();

StringBuilder medal = new StringBuilder();

StringBuilder allowance = new StringBuilder();

DateTime year;

StringBuilder entry\_year = new StringBuilder();

using (StreamWriter sw = new StreamWriter(@"insert\_entrants.sql", false, Encoding.Default))

{

sw.WriteLine("DECLARE");

sw.WriteLine(" V\_ID HUMANS.ID%TYPE;");

sw.WriteLine(" CURSOR GET\_HUMANS\_ID IS");

sw.WriteLine(" SELECT ID FROM HUMANS;");

sw.WriteLine("BEGIN");

sw.WriteLine(" OPEN GET\_HUMANS\_ID;");

for (int i = 0; i < 10000; i++)

{

original.Clear();

medal.Clear();

allowance.Clear();

entry\_year.Clear();

original.Append(GetRandomPassword(yn, 1));

medal.Append(GetRandomPassword(yn, 1));

allowance.Append(GetRandomPassword(yn, 1));

year = RandomDay();

entry\_year.Append(year.Year.ToString());

sw.WriteLine("FETCH GET\_HUMANS\_ID INTO V\_ID;");

sw.WriteLine("INSERT INTO ENTRANTS(HUMAN\_ID,ORIGINAL,MEDAL,ALLOWANCE,ENTRY\_YEAR)");

sw.WriteLine("VALUES(V\_ID,'" + medal + "','" + allowance + "','" + original + "'," + entry\_year + ");");

sw.WriteLine();

}

sw.WriteLine("END;");

sw.WriteLine("/");

}

}

static void GetHumans()

{

const string rc = "qwertyuiopasdfghjklzxcvbnm";

StringBuilder name = new StringBuilder();

StringBuilder surname = new StringBuilder(); ;

StringBuilder second\_name = new StringBuilder(); ;

DateTime bd;

StringBuilder birth\_date = new StringBuilder(); ;

string sx = "МЖ";

StringBuilder sex = new StringBuilder();

StringBuilder line = new StringBuilder();

using (StreamWriter sw = new StreamWriter(@"insert\_humans.sql", false, Encoding.Default))

{

for (int i = 0; i < 10000; i++)

{

name.Clear();

surname.Clear();

second\_name.Clear();

birth\_date.Clear();

sex.Clear();

line.Clear();

name.Append(GetRandomPassword(rc, 10));

surname.Append(GetRandomPassword(rc, 10));

second\_name.Append(GetRandomPassword(rc, 10));

bd = RandomDay();

birth\_date.Append(bd.Day + "." + bd.Month + "." + bd.Year);

sex.Append(GetRandomPassword(sx, 1));

line.AppendLine("INSERT INTO HUMANS(NAME, SURNAME,SECOND\_NAME,BIRTH\_DATE,SEX,CITIZENSHIP\_ID)");

line.AppendLine("VALUES('" + name + "','" + surname + "','" + second\_name + "',TO\_DATE('" + birth\_date + "','DD.MM.YYYY'),");

line.AppendLine("'" + sex + "',(SELECT ID FROM CITIZENSHIP WHERE NAME='Россия'));");

sw.WriteLine(line);

}

}

}

static string GetRandomPassword(string ch, int pwdLength)

{

char[] pwd = new char[pwdLength];

for (int i = 0; i < pwd.Length; i++)

pwd[i] = ch[rndGen.Next(ch.Length)];

return new string(pwd);

}

static DateTime RandomDay()

{

DateTime start = new DateTime(1997, 1, 1);

DateTime end = new DateTime(2014, 1, 1);

int range = (end - start).Days;

return start.AddDays(gen.Next(range));

}

}

# Зачетные задачи

## Задача 1.

Необходимо реализовать 1 SQL-запрос, каждая строка результата которого должна содержать строку из стержневой сущности и, объединенные агрегационными функциями, данные из характеристических сущностей. В запросе должен быть обеспечен поиск по строковым атрибутам (на маску *«%поисковый запрос%»*, с возможностью использования неправильной раскладки клавиатуры.

Для обеспечения использования неправильной раскладки клавиатуры была реализована функция F\_TRANSLATE:

CREATE OR REPLACE FUNCTION F\_TRANSLATE(WORD IN VARCHAR2)

RETURN VARCHAR2

IS

S VARCHAR2(20);

BEGIN

S:=UPPER(WORD);

S:=REPLACE(S,'F', 'А');

S:=REPLACE(S,',', 'Б');

S:=REPLACE(S,'D', 'В');

S:=REPLACE(S,'U', 'Г');

S:=REPLACE(S,'L', 'Д');

S:=REPLACE(S,'T', 'Е');

S:=REPLACE(S,'`', 'Ё');

S:=REPLACE(S,';', 'Ж');

S:=REPLACE(S,'P', 'З');

S:=REPLACE(S,'B', 'И');

S:=REPLACE(S,'Q', 'Й');

S:=REPLACE(S,'R', 'К');

S:=REPLACE(S,'K', 'Л');

S:=REPLACE(S,'V', 'М');

S:=REPLACE(S,'Y', 'Н');

S:=REPLACE(S,'J', 'О');

S:=REPLACE(S,'G', 'П');

S:=REPLACE(S,'H', 'Р');

S:=REPLACE(S,'C', 'С');

S:=REPLACE(S,'N', 'Т');

S:=REPLACE(S,'E', 'У');

S:=REPLACE(S,'A', 'Ф');

S:=REPLACE(S,'[', 'Х');

S:=REPLACE(S,'W', 'Ц');

S:=REPLACE(S,'X', 'Ч');

S:=REPLACE(S,'I', 'Ш');

S:=REPLACE(S,']', 'Ъ');

S:=REPLACE(S,'S', 'Ы');

S:=REPLACE(S,'M', 'Ь');

S:=REPLACE(S,'''', 'Э');

S:=REPLACE(S,'.', 'Ю');

S:=REPLACE(S,'Z', 'Я');

RETURN S;

END;

/

Запрос с объединением нескольких таблиц и использованием агрегационной функции:

SELECT SUR, EN\_PR, CIT FROM

(

SELECT SUR, EN\_PR, C.NAME CIT

FROM CITIZENSHIP C

JOIN

(

SELECT H.SURNAME SUR, EN\_PR, H.CITIZENSHIP\_ID CIT\_ID

FROM HUMANS H

JOIN

(

SELECT HUMAN\_ID H\_ID, EN\_PR FROM

ENTRANTS E JOIN

(

SELECT P.ENTRANT\_ID ENT\_ID, LISTAGG(P.PROGRAM\_ID,', ')

WITHIN GROUP (ORDER BY P.ENTRANT\_ID) AS EN\_PR

FROM PRIORITIES P

GROUP BY P.ENTRANT\_ID

)

ON E.ID = ENT\_ID

)

ON H.ID = H\_ID

)

ON C.ID = CIT\_ID

)

WHERE CIT LIKE INITCAP(F\_TRANSLATE('Hjcc%')); --'Росс%'

## Задача 2.

Необходимо реализовать алгоритмы Левенштейна, Джаро-Винклера, и любой метод нечеткого поиска.

В качестве третьего метода был выбран алгоритм Хемминга.

/\*Левенштейн\*/

CREATE OR REPLACE FUNCTION ld (

as\_src\_i IN VARCHAR2

, as\_trg\_i IN VARCHAR2

)

RETURN NUMBER

DETERMINISTIC

AS

ln\_src\_len PLS\_INTEGER := NVL(LENGTH(as\_src\_i), 0);

ln\_trg\_len PLS\_INTEGER := NVL(LENGTH(as\_trg\_i), 0);

ln\_hlen PLS\_INTEGER;

ln\_cost PLS\_INTEGER;

TYPE t\_numtbl IS TABLE OF PLS\_INTEGER INDEX BY BINARY\_INTEGER;

la\_ldmatrix t\_numtbl;

BEGIN

IF (ln\_src\_len = 0)

THEN

RETURN ln\_trg\_len;

ELSIF (ln\_trg\_len = 0)

THEN

RETURN ln\_src\_len;

END IF;

ln\_hlen := ln\_src\_len + 1;

FOR h IN 0 .. ln\_src\_len

LOOP

la\_ldmatrix(h) := h;

END LOOP;

FOR v IN 0 .. ln\_trg\_len

LOOP

la\_ldmatrix(v \* ln\_hlen) := v;

END LOOP;

FOR h IN 1 .. ln\_src\_len

LOOP

FOR v IN 1 .. ln\_trg\_len

LOOP

IF (SUBSTR(as\_src\_i, h, 1) = SUBSTR(as\_trg\_i, v, 1))

THEN

ln\_cost := 0;

ELSE

ln\_cost := 1;

END IF;

la\_ldmatrix(v \* ln\_hlen + h) :=

LEAST(

la\_ldmatrix((v - 1) \* ln\_hlen + h ) + 1

, la\_ldmatrix( v \* ln\_hlen + h - 1) + 1

, la\_ldmatrix((v - 1) \* ln\_hlen + h - 1) + ln\_cost

)

;

END LOOP;

END LOOP;

RETURN la\_ldmatrix(ln\_trg\_len \* ln\_hlen + ln\_src\_len);

END ld;

/

/\*Джаро-Винклер\*/

CREATE OR REPLACE FUNCTION jws

(p\_string1 IN VARCHAR2,

p\_string2 IN VARCHAR2)

RETURN NUMBER

DETERMINISTIC

AS

v\_closeness NUMBER := 0;

v\_temp VARCHAR2 (32767);

v\_comp1 VARCHAR2 (32767);

v\_comp2 VARCHAR2 (32767);

v\_matches NUMBER := 0;

v\_char VARCHAR2 (1);

v\_transpositions NUMBER := 0;

v\_d\_jaro NUMBER := 0;

v\_leading NUMBER := 0;

v\_d\_winkler NUMBER := 0;

v\_jws NUMBER := 0;

BEGIN

-- check for null strings:

IF p\_string1 IS NULL OR p\_string2 IS NULL THEN

RETURN 0;

END IF;

-- closeness:

v\_closeness := (GREATEST (LENGTH (p\_string1), LENGTH (p\_string2)) / 2) - 1;

-- find matching characters and transpositions within closeness:

v\_temp := p\_string2;

FOR i IN 1 .. LENGTH (p\_string1) LOOP

IF INSTR (v\_temp, SUBSTR (p\_string1, i, 1)) > 0 THEN

v\_char := SUBSTR (p\_string1, i, 1);

IF ABS (INSTR (p\_string1, v\_char) - INSTR (p\_string2, v\_char)) <= v\_closeness THEN

v\_comp1 := v\_comp1 || SUBSTR (p\_string1, i, 1);

v\_temp := SUBSTR (v\_temp, 1, INSTR (v\_temp, SUBSTR (p\_string1, i, 1)) - 1)

|| SUBSTR (v\_temp, INSTR (v\_temp, SUBSTR (p\_string1, i, 1)) + 1);

END IF;

END IF;

END LOOP;

v\_temp := p\_string1;

FOR i IN 1 .. LENGTH (p\_string2) LOOP

IF INSTR (v\_temp, SUBSTR (p\_string2, i, 1)) > 0 THEN

v\_char := SUBSTR (p\_string2, i, 1);

IF ABS (INSTR (p\_string2, v\_char) - INSTR (p\_string1, v\_char)) <= v\_closeness THEN

v\_comp2 := v\_comp2 || SUBSTR (p\_string2, i, 1);

v\_temp := SUBSTR (v\_temp, 1, INSTR (v\_temp, SUBSTR (p\_string2, i, 1)) - 1)

|| SUBSTR (v\_temp, INSTR (v\_temp, SUBSTR (p\_string2, i, 1)) + 1);

END IF;

END IF;

END LOOP;

-- check for null strings:

IF v\_comp1 IS NULL OR v\_comp2 IS NULL THEN

RETURN 0;

END IF;

-- count matches and transpositions within closeness:

FOR i IN 1 .. LEAST (LENGTH (v\_comp1), LENGTH (v\_comp2)) LOOP

IF SUBSTR (v\_comp1, i, 1) = SUBSTR (v\_comp2, i, 1) THEN

v\_matches := v\_matches + 1;

ELSE

v\_char := SUBSTR (v\_comp1, i, 1);

IF ABS (INSTR (p\_string1, v\_char) - INSTR (p\_string2, v\_char)) <= v\_closeness THEN

v\_transpositions := v\_transpositions + 1;

v\_matches := v\_matches + 1;

END IF;

END IF;

END LOOP;

v\_transpositions := v\_transpositions / 2;

-- check for no matches:

IF v\_matches = 0

THEN RETURN 0;

END IF;

-- Jaro:

v\_d\_jaro := ((v\_matches / LENGTH (p\_string1)) +

(v\_matches / LENGTH (p\_string2)) +

((v\_matches - v\_transpositions) / v\_matches))

/ 3;

-- count matching leading characters (up to 4):

FOR i IN 1 .. LEAST (LENGTH (p\_string1), LENGTH (p\_string2), 4) LOOP

IF SUBSTR (p\_string1, i, 1) = SUBSTR (p\_string2, i, 1) THEN

v\_leading := v\_leading + 1;

ELSE

EXIT;

END IF;

END LOOP;

-- Winkler:

v\_d\_winkler := v\_d\_jaro + ((v\_leading \* .1) \* (1 - v\_d\_jaro));

-- Jaro-Winkler similarity rounded:

v\_jws := ROUND (v\_d\_winkler \* 100);

RETURN v\_jws;

END jws;

/

-- Hamming Distance

CREATE OR REPLACE FUNCTION hd (

as\_src\_i IN VARCHAR2

, as\_trg\_i IN VARCHAR2

)

RETURN NUMBER

DETERMINISTIC

AS

ln\_src\_len PLS\_INTEGER := NVL(LENGTH(as\_src\_i), 0);

ln\_trg\_len PLS\_INTEGER := NVL(LENGTH(as\_trg\_i), 0);

ln\_distance PLS\_INTEGER := 0;

BEGIN

IF (ln\_src\_len <> ln\_trg\_len)

THEN

RETURN NULL;

END IF;

IF (ln\_src\_len = 0)

THEN

RETURN ln\_src\_len;

END IF;

FOR i IN 1..ln\_src\_len

LOOP

IF (SUBSTR(as\_src\_i, i, 1) <> SUBSTR(as\_trg\_i, i, 1))

THEN

ln\_distance := ln\_distance + 1;

END IF;

END LOOP;

RETURN ln\_distance;

END hd;

/

Пример запроса, с использованием алгоритма Джаро-Винклера:

SELECT SUR, EN\_PR, CIT FROM

(

SELECT SUR, EN\_PR, C.NAME CIT

FROM CITIZENSHIP C

JOIN

(

SELECT H.SURNAME SUR, EN\_PR, H.CITIZENSHIP\_ID CIT\_ID

FROM HUMANS H

JOIN

(

SELECT HUMAN\_ID H\_ID, EN\_PR FROM

ENTRANTS E JOIN

(

SELECT P.ENTRANT\_ID ENT\_ID, LISTAGG(P.PROGRAM\_ID,', ')

WITHIN GROUP (ORDER BY P.ENTRANT\_ID) AS EN\_PR

FROM PRIORITIES P

GROUP BY P.ENTRANT\_ID

)

ON E.ID = ENT\_ID

)

ON H.ID = H\_ID

)

ON C.ID = CIT\_ID

)

WHERE CIT IN

(

SELECT NAME FROM(

WITH COUNTRY\_NAMES AS

(SELECT NAME FROM CITIZENSHIP)

SELECT COUNTRY\_NAMES.NAME, jws(COUNTRY\_NAMES.NAME,'Rassiya') mjws

FROM COUNTRY\_NAMES

ORDER BY mjws DESC

)

WHERE ROWNUM = 1

);

## Задача 3.

Необходимо реализовать получение всех данных из БД при помощи конвейерных функций и объектов на языке PL/SQL, а также в формате XML.

Получение всех данных из БД в формате XML:

SELECT

XMLConcat(

XMLElement("CITIZENSHIP", (SELECT XMLAgg(XMLElement("COUNTRY", XMLForest(ID,NAME))) FROM CITIZENSHIP)),

XMLElement("BRANCHES", (SELECT XMLAgg(XMLElement("BRANCH", XMLForest(ID,NAME))) FROM CITIZENSHIP)),

XMLElement("SUBJECTS", (SELECT XMLAgg(XMLElement("SUBJECT", XMLForest(ID,NAME))) FROM SUBJECTS)),

XMLElement("SPECIALTIES", (SELECT XMLAgg(XMLElement("SPECIALTY", XMLForest(ID,NAME,CODE))) FROM SPECIALTIES)),

XMLElement("DEPARTMENT\_TYPES", (SELECT XMLAgg(XMLElement("DEPARTMENT\_TYPE", XMLForest(ID,NAME))) FROM DEPARTMENT\_TYPES)),

XMLElement("PROGRAMSUBJECTS", (SELECT XMLAgg(XMLElement("PROGRAMSUBJECT", XMLForest(ID,PROGRAM\_ID,SUBJECT\_ID,THRESHOLD\_MARK,PS\_YEAR))) FROM PROGRAMSUBJECTS)),

XMLElement("EXAMS", (SELECT XMLAgg(XMLElement("EXAM", XMLForest(ID,ENTRANT\_ID,SUBJECT\_ID,MARK,CODE))) FROM EXAMS WHERE ROWNUM <= 10)),

XMLElement("ORDERS\_ITEMS", (SELECT XMLAgg(XMLElement("ORDER\_ITEM", XMLForest(ID,NAME,ITEM\_NUMBER))) FROM ORDERS\_ITEMS)),

XMLElement("DEPARTMENTS", (SELECT XMLAgg(XMLElement("DEPARTMENT", XMLForest(ID,NAME,ABBR,DEPARTMENT\_ID,DEPARTMENT\_TYPE\_ID,CODE))) FROM DEPARTMENTS)),

XMLElement("PROGRAMS", (SELECT XMLAgg(XMLElement("PROGRAM", XMLForest(ID,SPEC\_ID,BRANCH\_ID))) FROM PROGRAMS)),

XMLElement("DEPARTMENTPROGRAM", (SELECT XMLAgg(XMLElement("DEPPROG", XMLForest(ID,PROGRAM\_ID,DEPARTMENT\_ID))) FROM DEPARTMENTPROGRAM)),

XMLElement("COMMANDMENTS", (SELECT XMLAgg(XMLElement("COMMANDMENT", XMLForest(ID,ENTRANT\_ID,PROGRAM\_ID,ORDER\_ID,PRE\_COMMANDMENT,COMMANDMENT\_YEAR,CODE,DEPARTMENT\_ID,COMMIT\_DATE,CREATE\_DATE))) FROM COMMANDMENTS)),

XMLElement("PRIORITIES", (SELECT XMLAgg(XMLElement("PRIORITY", XMLForest(ID,ENTRANT\_ID,PROGRAM\_ID,PRIORITY))) FROM PRIORITIES WHERE ROWNUM <= 10)),

XMLElement("ACCEPTANCE\_PLANS", (SELECT XMLAgg(XMLElement("ACCEPTANCE\_PLAN", XMLForest(ID,PROGRAM\_ID,AMOUNT,PLAN\_YEAR,DEPARTMENT\_ID))) FROM ACCEPTANCE\_PLANS)),

XMLElement("ENTRANTS", (SELECT XMLAgg(XMLElement("ENTRANT", XMLForest(ID,HUMAN\_ID,ORIGINAL,MEDAL,ALLOWANCE,ENTRY\_YEAR))) FROM ENTRANTS WHERE ROWNUM <= 10)),

XMLElement("HUMANS", (SELECT XMLAgg(XMLElement("HUMAN", XMLForest(ID,NAME,SURNAME,SECOND\_NAME,BIRTH\_DATE,SEX,CITIZENSHIP\_ID))) FROM HUMANS WHERE ROWNUM <= 10))

) FROM dual;

Получение всех данных из БД при помощи конвейерных функций и объектов на языке PL/SQL:

CREATE OR REPLACE PACKAGE ALLDATA IS

TYPE rowGetSubjects IS RECORD(

l\_id SUBJECTS.ID%TYPE,

l\_name SUBJECTS.NAME%TYPE

);

TYPE rowGetSpecialties IS RECORD(

l\_id SPECIALTIES.ID%TYPE,

l\_name SPECIALTIES.NAME%TYPE,

l\_code SPECIALTIES.CODE%TYPE

);

TYPE rowGetDepartmentTypes IS RECORD(

l\_id DEPARTMENT\_TYPES.ID%TYPE,

l\_name DEPARTMENT\_TYPES.NAME%TYPE

);

TYPE rowGetProgramSubjects IS RECORD(

l\_id PROGRAMSUBJECTS.ID%TYPE,

l\_program\_id PROGRAMSUBJECTS.PROGRAM\_ID%TYPE,

l\_subject\_id PROGRAMSUBJECTS.SUBJECT\_ID%TYPE,

l\_threshold\_mark PROGRAMSUBJECTS.THRESHOLD\_MARK%TYPE,

l\_ps\_year PROGRAMSUBJECTS.PS\_YEAR%TYPE

);

TYPE rowGetExams IS RECORD(

l\_id EXAMS.ID%TYPE,

l\_entrant\_id EXAMS.ENTRANT\_ID%TYPE,

l\_subject\_id EXAMS.SUBJECT\_ID%TYPE,

l\_mark EXAMS.MARK%TYPE,

l\_code EXAMS.CODE%TYPE

);

TYPE rowGetOrderItems IS RECORD(

l\_id ORDERS\_ITEMS.ID%TYPE,

l\_name ORDERS\_ITEMS.NAME%TYPE,

l\_full\_text ORDERS\_ITEMS.FULL\_TEXT%TYPE,

l\_item\_number ORDERS\_ITEMS.ITEM\_NUMBER%TYPE

);

TYPE rowGetDepartments IS RECORD(

l\_id DEPARTMENTS.ID%TYPE,

l\_name DEPARTMENTS.NAME%TYPE,

l\_department\_id DEPARTMENTS.DEPARTMENT\_ID%TYPE,

l\_department\_type\_id DEPARTMENTS.DEPARTMENT\_TYPE\_ID%TYPE,

l\_code DEPARTMENTS.CODE%TYPE,

l\_abbr DEPARTMENTS.ABBR%TYPE

);

TYPE rowGetDepartmentProgram IS RECORD(

l\_id DEPARTMENTPROGRAM.ID%TYPE,

l\_program\_id DEPARTMENTPROGRAM.PROGRAM\_ID%TYPE,

l\_department\_id DEPARTMENTPROGRAM.DEPARTMENT\_ID%TYPE

);

TYPE rowGetBranches IS RECORD(

l\_id BRANCHES.ID%TYPE,

l\_name BRANCHES.NAME%TYPE

);

TYPE rowGetPrograms IS RECORD(

l\_id PROGRAMS.ID%TYPE,

l\_spec\_id PROGRAMS.SPEC\_ID%TYPE,

l\_branch\_id PROGRAMS.BRANCH\_ID%TYPE

);

TYPE rowGetAcceptancePlans IS RECORD(

l\_id ACCEPTANCE\_PLANS.ID%TYPE,

l\_program\_id ACCEPTANCE\_PLANS.PROGRAM\_ID%TYPE,

l\_amount ACCEPTANCE\_PLANS.AMOUNT%TYPE,

l\_plan\_year ACCEPTANCE\_PLANS.PLAN\_YEAR%TYPE,

l\_department\_id ACCEPTANCE\_PLANS.DEPARTMENT\_ID%TYPE

);

TYPE rowGetPriorities IS RECORD(

l\_id PRIORITIES.ID%TYPE,

l\_entrant\_id PRIORITIES.ENTRANT\_ID%TYPE,

l\_program\_id PRIORITIES.PROGRAM\_ID%TYPE,

l\_priority PRIORITIES.PRIORITY%TYPE

);

TYPE rowGetCommandments IS RECORD(

l\_id COMMANDMENTS.ID%TYPE,

l\_entrant\_id COMMANDMENTS.ENTRANT\_ID%TYPE,

l\_program\_id COMMANDMENTS.PROGRAM\_ID%TYPE,

l\_order\_id COMMANDMENTS.ORDER\_ID%TYPE,

l\_pre\_commandment COMMANDMENTS.PRE\_COMMANDMENT%TYPE,

l\_commandment\_year COMMANDMENTS.COMMANDMENT\_YEAR%TYPE,

l\_code COMMANDMENTS.CODE%TYPE,

l\_department\_id COMMANDMENTS.DEPARTMENT\_ID%TYPE,

l\_commit\_date COMMANDMENTS.COMMIT\_DATE%TYPE,

l\_create\_date COMMANDMENTS.CREATE\_DATE%TYPE

);

TYPE rowGetEntrants IS RECORD(

l\_id ENTRANTS.ID%TYPE,

l\_human\_id ENTRANTS.HUMAN\_ID%TYPE,

l\_original ENTRANTS.ORIGINAL%TYPE,

l\_medal ENTRANTS.MEDAL%TYPE,

l\_allowance ENTRANTS.ALLOWANCE%TYPE,

l\_entry\_year ENTRANTS.ENTRY\_YEAR%TYPE,

l\_scanned\_profile ENTRANTS.SCANNED\_PROFILE%TYPE

);

TYPE rowGetHumans IS RECORD(

l\_id HUMANS.ID%TYPE,

l\_name HUMANS.NAME%TYPE,

l\_surname HUMANS.SURNAME%TYPE,

l\_second\_name HUMANS.SECOND\_NAME%TYPE,

l\_birth\_date HUMANS.BIRTH\_DATE%TYPE,

l\_sex HUMANS.SEX%TYPE,

l\_citizenship\_id HUMANS.CITIZENSHIP\_ID%TYPE

);

TYPE tblGetSubjects IS TABLE OF rowGetSubjects;

TYPE tblGetSpecialties IS TABLE OF rowGetSpecialties;

TYPE tblGetDepartmentTypes IS TABLE OF rowGetDepartmentTypes;

TYPE tblGetProgramSubjects IS TABLE OF rowGetProgramSubjects;

TYPE tblGetExams IS TABLE OF rowGetExams;

TYPE tblGetOrderItems IS TABLE OF rowGetOrderItems;

TYPE tblGetDepartments IS TABLE OF rowGetDepartments;

TYPE tblGetDepartmentProgram IS TABLE OF rowGetDepartmentProgram;

TYPE tblGetBranches IS TABLE OF rowGetBranches;

TYPE tblGetPrograms IS TABLE OF rowGetPrograms;

TYPE tblGetAcceptancePlans IS TABLE OF rowGetAcceptancePlans;

TYPE tblGetPriorities IS TABLE OF rowGetPriorities;

TYPE tblGetCommandments IS TABLE OF rowGetCommandments;

TYPE tblGetEntrants IS TABLE OF rowGetEntrants;

TYPE tblGetHumans IS TABLE OF rowGetHumans;

FUNCTION GetSubjects

(pCount NUMBER DEFAULT NULL)

RETURN tblGetSubjects

PIPELINED;

FUNCTION GetSpecialties

(pCount NUMBER DEFAULT NULL)

RETURN tblGetSpecialties

PIPELINED;

FUNCTION GetDepartmentTypes

(pCount NUMBER DEFAULT NULL)

RETURN tblGetDepartmentTypes

PIPELINED;

FUNCTION GetProgramSubjects

(pCount NUMBER DEFAULT NULL)

RETURN tblGetProgramSubjects

PIPELINED;

FUNCTION GetExams

(pCount NUMBER DEFAULT NULL)

RETURN tblGetExams

PIPELINED;

FUNCTION GetOrderItems

(pCount NUMBER DEFAULT NULL)

RETURN tblGetOrderItems

PIPELINED;

FUNCTION GetDepartments

(pCount NUMBER DEFAULT NULL)

RETURN tblGetDepartments

PIPELINED;

FUNCTION GetDepartmentProgram

(pCount NUMBER DEFAULT NULL)

RETURN tblGetDepartmentProgram

PIPELINED;

FUNCTION GetBranches

(pCount NUMBER DEFAULT NULL)

RETURN tblGetBranches

PIPELINED;

FUNCTION GetPrograms

(pCount NUMBER DEFAULT NULL)

RETURN tblGetPrograms

PIPELINED;

FUNCTION GetAcceptancePlans

(pCount NUMBER DEFAULT NULL)

RETURN tblGetAcceptancePlans

PIPELINED;

FUNCTION GetPriorities

(pCount NUMBER DEFAULT NULL)

RETURN tblGetPriorities

PIPELINED;

FUNCTION GetCommandments

(pCount NUMBER DEFAULT NULL)

RETURN tblGetCommandments

PIPELINED;

FUNCTION GetEntrants

(pCount NUMBER DEFAULT NULL)

RETURN tblGetEntrants

PIPELINED;

FUNCTION GetHumans

(pCount NUMBER DEFAULT NULL)

RETURN tblGetHumans

PIPELINED;

END ALLDATA;

/

CREATE OR REPLACE PACKAGE BODY ALLDATA IS

FUNCTION GetSubjects

(pCount NUMBER DEFAULT NULL)

RETURN tblGetSubjects

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM SUBJECTS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM SUBJECTS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetSubjects;

FUNCTION GetSpecialties

(pCount NUMBER DEFAULT NULL)

RETURN tblGetSpecialties

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM SPECIALTIES) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM SPECIALTIES WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetSpecialties;

FUNCTION GetDepartmentTypes

(pCount NUMBER DEFAULT NULL)

RETURN tblGetDepartmentTypes

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM DEPARTMENT\_TYPES) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM DEPARTMENT\_TYPES WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetDepartmentTypes;

FUNCTION GetProgramSubjects

(pCount NUMBER DEFAULT NULL)

RETURN tblGetProgramSubjects

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM PROGRAMSUBJECTS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM PROGRAMSUBJECTS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetProgramSubjects;

FUNCTION GetExams

(pCount NUMBER DEFAULT NULL)

RETURN tblGetExams

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM EXAMS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM EXAMS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetExams;

FUNCTION GetOrderItems

(pCount NUMBER DEFAULT NULL)

RETURN tblGetOrderItems

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM ORDERS\_ITEMS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM ORDERS\_ITEMS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetOrderItems;

FUNCTION GetDepartments

(pCount NUMBER DEFAULT NULL)

RETURN tblGetDepartments

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM DEPARTMENTS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM DEPARTMENTS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetDepartments;

FUNCTION GetDepartmentProgram

(pCount NUMBER DEFAULT NULL)

RETURN tblGetDepartmentProgram

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM DEPARTMENTPROGRAM) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM DEPARTMENTPROGRAM WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetDepartmentProgram;

FUNCTION GetBranches

(pCount NUMBER DEFAULT NULL)

RETURN tblGetBranches

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM BRANCHES) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM BRANCHES WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetBranches;

FUNCTION GetPrograms

(pCount NUMBER DEFAULT NULL)

RETURN tblGetPrograms

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM PROGRAMS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM PROGRAMS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetPrograms;

FUNCTION GetAcceptancePlans

(pCount NUMBER DEFAULT NULL)

RETURN tblGetAcceptancePlans

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM ACCEPTANCE\_PLANS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM ACCEPTANCE\_PLANS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetAcceptancePlans;

FUNCTION GetPriorities

(pCount NUMBER DEFAULT NULL)

RETURN tblGetPriorities

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM PRIORITIES) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM PRIORITIES WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetPriorities;

FUNCTION GetCommandments

(pCount NUMBER DEFAULT NULL)

RETURN tblGetCommandments

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM COMMANDMENTS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM COMMANDMENTS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetCommandments;

FUNCTION GetEntrants

(pCount NUMBER DEFAULT NULL)

RETURN tblGetEntrants

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM ENTRANTS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM ENTRANTS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetEntrants;

FUNCTION GetHumans

(pCount NUMBER DEFAULT NULL)

RETURN tblGetHumans

PIPELINED

IS

BEGIN

IF pCount IS NULL THEN

FOR curr IN

(SELECT \* FROM HUMANS) LOOP

PIPE ROW (curr);

END LOOP;

ELSE

FOR curr IN

(SELECT \* FROM HUMANS WHERE ROWNUM<=pCount) LOOP

PIPE ROW (curr);

END LOOP;

END IF;

END GetHumans;

END ALLDATA;

/

## Задача 4

В качестве дополнительного задания была сформирована следующая задача. Необходимо реализовать функцию, которая с использованием Яндекс.API возвращала координаты объекта по адресу.

Эта задача была разбита на несколько подзадач:

1) Функция для загрузки xml-документа по url

CREATE OR REPLACE Function load\_xml (p\_url IN VARCHAR2)

RETURN clob

AS

l\_http\_request UTL\_HTTP.req;

l\_http\_response UTL\_HTTP.resp;

l\_clob CLOB;

l\_text VARCHAR2(32767);

BEGIN

-- Initialize the CLOB.

DBMS\_LOB.createtemporary(l\_clob, FALSE);

-- Make a HTTP request and get the response.

l\_http\_request := UTL\_HTTP.begin\_request(p\_url);

l\_http\_response := UTL\_HTTP.get\_response(l\_http\_request);

-- Copy the response into the CLOB.

BEGIN

LOOP

UTL\_HTTP.read\_text(l\_http\_response, l\_text, 32766);

DBMS\_LOB.writeappend (l\_clob, LENGTH(l\_text), l\_text);

END LOOP;

EXCEPTION

WHEN UTL\_HTTP.end\_of\_body THEN

UTL\_HTTP.end\_response(l\_http\_response);

END;

-- Relase the resources associated with the temporary LOB.

RETURN l\_clob;

DBMS\_LOB.freetemporary(l\_clob);

EXCEPTION

WHEN OTHERS THEN

UTL\_HTTP.end\_response(l\_http\_response);

DBMS\_LOB.freetemporary(l\_clob);

RAISE;

END load\_xml;

/

2) Функция для возврата координат формата «долгота широта» по адресу.

Прим.: во входной строке все пробелы должны быть заменены символом «+»

CREATE OR REPLACE Function ya\_pos (addr IN VARCHAR2)

RETURN VARCHAR2

AS

TEXT CLOB;

POS VARCHAR2(50);

BEGIN

TEXT := load\_xml('geocode-maps.yandex.ru/1.x/?geocode='||addr);

SELECT

extractValue(

extract(

extract(

extract(VALUE(t),'/ymaps/GeoObjectCollection/\*','xmlns="http://maps.yandex.ru/ymaps/1.x"'),

'featureMember/\*','xmlns="http://www.opengis.net/gml"'),

'GeoObject/\*','xmlns="http://maps.yandex.ru/ymaps/1.x"'),

'Point/pos','xmlns="http://www.opengis.net/gml"')

INTO POS

FROM TABLE(XMLSequence(XMLType(TEXT))) t;

RETURN POS;

END;

/

3) Итоговая функция, которая преобразует входящий адрес к нужному виду и возвращает координаты объекта в формате «долгота широта»

CREATE OR REPLACE Function get\_pos (addr IN VARCHAR2)

RETURN VARCHAR2

AS

tmp VARCHAR2(500);

pos VARCHAR2(50);

BEGIN

tmp := REPLACE(addr,' ','+');

pos := ya\_pos(tmp);

RETURN pos;

END;

/

# Использованная литература

1. Кириллов В.В., Громов Г.Ю. Введение в реляционные базы данных ― СПб.: БХВ-Петербург, 2009. ― 464 с.: ил. + CD-ROM
2. <https://www.google.ru>
3. <http://www.cyberforum.ru/oracle/>