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

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.

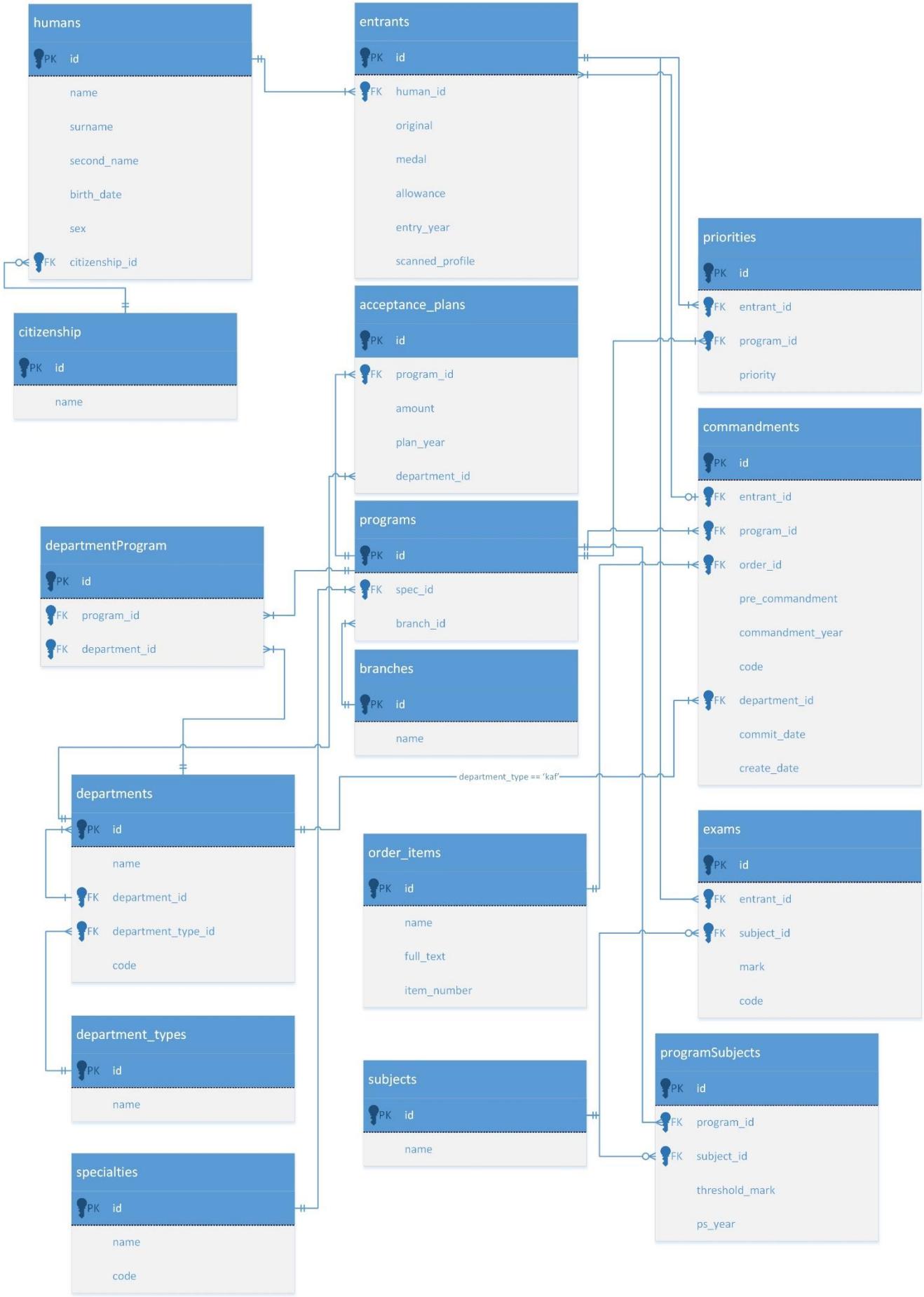
Разработка базы данных абитуриентов поможет приемной комиссии большую часть работы автоматизировать. В отличии от стеллажей с огромным количеством личных дел, БД позволяет осуществлять мгновенный доступ к информации о любом абитуриенте, а также без ошибок формировать списки людей по различным критериям.

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%')); --'Pocc%'
```

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,DEP
ARTMENT_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/