MINISTRY OF EDUCATION AND SCIENCE OF THE REPUBLIC OF KAZAKHSTAN

INTERNATIONAL INFORMATION TECHNOLOGY UNIVERSITY

COMPUTER ENGINEERING AND INFORMATION SECURITY DEPARTMENT

COURSE WORK

Program with PL/SQL

Done by student: Sheri Dosbol (Lonely Tiger🐯)

Group CSSE-1809-DA

Almaty 2020

1 SUBJECT AREA DESCRIPTION

Include a short description of your subject area.

Include a relational schema of your database

# **Introduction**

This report is about designing, creating and populating a database and making queries based on it. The DB is designed to hold information relating to getting all system information of libraries and readers.

1 DESIGNING DATABASE

* 1. Subject area analysis (part 1)

1. **List entities and attributes (in other words, tables and columns of the future database). Every entity should have an unique attribute (Primary key), underline it in the description:**

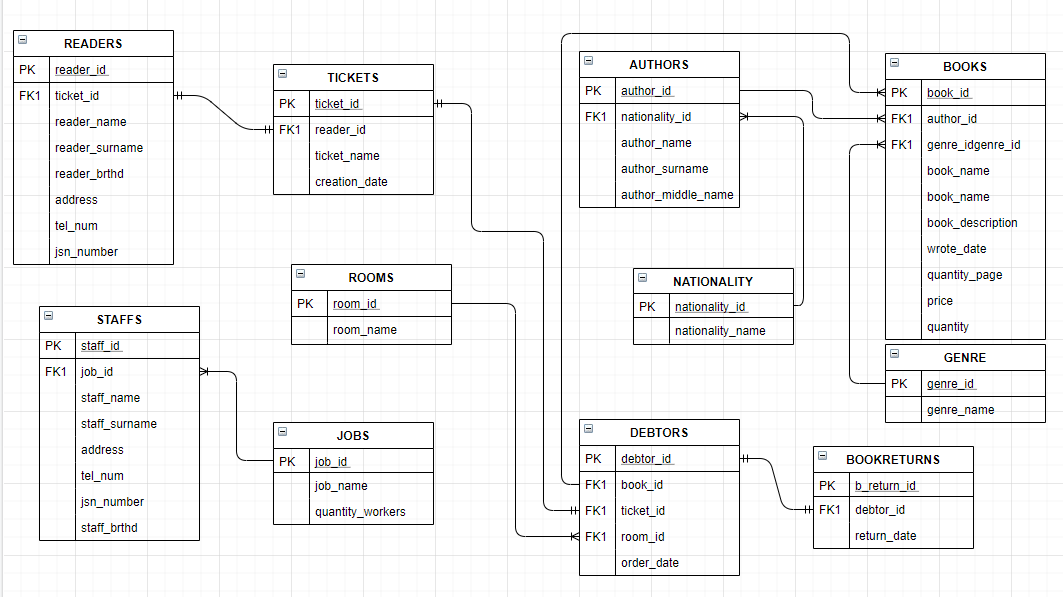
There are 11 entities in the DB:

* 1. **Readers(**reader\_id, reader\_name, reader\_surname, tel\_num, address, jsn\_number, ticket\_id**)**
  2. **GENRE**(genre\_id ,genre\_name )
  3. **ROOMS**(room\_id ,room\_name )
  4. **AUTHORS**(author\_id,author\_name,author\_surname,author\_middle\_name,nationality\_id )
  5. **TICKETS**(ticket\_id,ticket\_name,creation\_date,reader\_id )
  6. **READERS**(reader\_id, reader\_name ,reader\_surname, reader\_brthd, address, tel\_num, jsn\_number,ticket\_id )
  7. **BOOKS**(book\_id,book\_name, book\_description,author\_id,wrote\_date,genre\_id,
  8. quantity\_page,price,quantity )
  9. **JOBS**(job\_id, job\_name, quantity\_workers )
  10. **STAFFS**(staff\_id, staff\_name, staff\_surname, address, tel\_num, jsn\_number ,job\_id, staff\_brthd )
  11. **DEBTORS**(debtor\_id,book\_id,ticket\_id,room\_id,order\_date )
  12. **BOOKRETURNS**(b\_return\_id,debtor\_id,return\_date )
  13. **NATIONALITY**(nationality\_id,nationality\_name )

1. **Relationships between entities:**
   1. One reader have one ticket and in other side one ticket for one client; (1to1)
   2. One staff can work on one job and in other side in one job can be worked some staffs;(1toM)
   3. One book can have one author and in other side one author can have some books;(1toM)
   4. One rooms can have many debtors and in other side one debtor can visit one room;(1toM)
   5. One book can be enrolled only in one genre and from the other side one genre contains many books (N books); (1toM)
   6. One debtor can be one returner and other side one returner can be one debtor(1to1)
   7. One debtor can take many books and one book can be taken only one debtor(1toM)
   8. One reader have one ticket and in other side one ticket for one client; (1to1)
   9. One author can be born in one country and in other side in one country can was born many authors(1toM)
2. **Constraints:**
   * All the ID’s are unique;
   * One reader can have only one ticket;
   * Jsn number must be contains 12 digits;
   * Debtors must return books within a month;
   * Readers can`t take a book without permission of the staff;
   * Readers can`t take more than one book;
3. **Specify groups of users and their access rights:**
   1. Staffs can see information about readers but Readers can`t.
   2. Staffs can add and remove books, but Readers can`t.
4. **List potential questions from users to the database (queries):**
   1. SELECT \*FROM Tickets.
   2. SELECT \*FROM Jobs.
   3. INSERT \*INTO Genres.
   4. DELETE \*FROM Books
   5. ALTER TABLE authors DROP column nationality,

add nationality\_id int references Nationality;

* 1. ER diagram (part 2)



2 SQL SCRIPT FOR IMPLEMENTING THE DATABASE

2.1 Creating tables

CREATE TABLE GENRE(

genre\_id INT PRIMARY KEY,

genre\_name varchar (20)

);

CREATE TABLE ROOMS(

room\_id INT PRIMARY KEY,

room\_name varchar(20)

);

CREATE TABLE AUTHORS(

author\_id INT PRIMARY KEY,

author\_name varchar(20),

author\_surname varchar(20),

author\_middle\_name varchar(20)

);

CREATE TABLE TICKETS(

ticket\_id INT PRIMARY KEY,

ticket\_name varchar(35),

creation\_date date

);

CREATE TABLE READERS(

reader\_id INT PRIMARY KEY,

reader\_name varchar(11),

reader\_surname varchar(11),

reader\_brthd date,

address varchar(35),

tel\_num varchar(11),

jsn\_number varchar (12),

ticket\_id int references TICKETS

);

CREATE TABLE BOOKS(

book\_id INT PRIMARY KEY,

book\_name varchar(50),

book\_description varchar(200),

author\_id int references AUTHORS,

wrote\_date date,

genre\_id int references GENRE,

quantity\_page int,

price int,

quantity int

);

CREATE TABLE JOBS(

job\_id INT PRIMARY KEY,

job\_name varchar(20),

quantity\_workers int

);

CREATE TABLE STAFFS(

staff\_id INT PRIMARY KEY,

staff\_name varchar(11),

staff\_surname varchar(11),

address varchar(35),

tel\_num varchar(11),

jsn\_number varchar (12),

job\_id int references JOBS

);

CREATE TABLE DEBTORS(

debtor\_id INT PRIMARY KEY,

book\_id int references BOOKS,

ticket\_id int references TICKETS,

room\_id int references ROOMS,

order\_date date

);

CREATE TABLE BOOKRETURNS(

b\_return\_id INT PRIMARY KEY,

debtor\_id int references DEBTORS,

return\_date date

);

ALTER TABLE STAFFS

ADD staff\_brthd date;

ALTER TABLE BOOKS

DROP column book\_description;

Alter table Authors add nationality varchar(20);

CREATE TABLE NATIONALITY(

nationality\_id int primary key,

nationality\_name varchar(20)

);

ALTER TABLE authors

DROP column nationality;

ALTER TABLE authors

add nationality\_id int references Nationality;

2.2 Populating tables

insert into genre values (1, 'novel');

insert into genre values (2, 'historical');

insert into genre values (3, 'detective');

insert into genre values (4, 'fantasy');

insert into genre values (5, 'fairy tale');

insert into genre values (6, 'lyrical');

insert into genre values (7, 'adventure');

select \* from genre;

insert into nationality values (1, 'Kazakh');

insert into nationality values (2, 'Russian');

insert into nationality values (3, 'English');

insert into nationality values (4, 'French');

insert into nationality values (5, 'Chinese');

insert into nationality values (6, 'Greek');

insert into nationality values (7, 'British');

delete from nationality where nationality\_id=5;

select \* from nationality;

insert into AUTHORS values (1, 'Mukhtar', 'Auezov', 'Omarkhanuly', 1);

insert into AUTHORS values (2, 'Lev', 'Tolstoi', 'Nikolaevich', 2);

insert into AUTHORS values (3, 'William', 'Folkner', '', 3);

insert into AUTHORS values (4, 'Homer', '', '', 6);

insert into AUTHORS values (5, 'Artur', 'Konan', 'Doila', 7);

insert into AUTHORS values (6, 'Dosbol', 'Sheri', 'Sabituly', 1);

select \* from authors;

insert into Books values (1,'War and World', 1,date '1942-09-25', 1, 870, 2000, 3);

insert into Books values (2,'Shum and Yarost', 3, date'1863-02-24', 1, 1300, 3000, 2);

insert into Books values (3,'War and World', 2, date'1863-02-27', 1, 1274, 2000, 3);

insert into Books values (4,'Sherlok Holmes', 5,date '1891-02-27', 3, 1500, 3500, 3);

insert into Books values (5,'In The Dream', 6,date '2000-09-25', 4, 2020, 65000, 1);

insert into Books values(6,'Mahabbat, kyzyk mol zhyldar',6, date '2000-09-25',(select genre\_id from Genre where genre\_name = 'lyrical'),1200, 5000, 3);

select \*from Books;

UPDATE books SET book\_name='Abay Zholy' WHERE book\_id =1;

select \* from Books;

insert into rooms values(1, 'Grand Hall');

insert into rooms values(2, 'Small Hall');

insert into rooms values(3, 'individual room\_1');

insert into rooms values(4, 'individual room\_2');

insert into rooms values(5, 'individual room\_3');

insert into rooms values(6, 'Outside');

select \* from rooms;

insert into jobs values(1, 'Admin', 1);

insert into jobs values(2, 'Adviser', 15);

insert into jobs values(3, 'Vahter',3);

insert into jobs values(4, 'Tehnichka', 6);

insert into jobs values(5, 'Security',10);

UPDATE jobs SET job\_name='Cleaningman' WHERE job\_id =4;

select \* from jobs;

insert into staffs values (1, 'Berkut', '', 'Tauda', '87076674745', '121212121212',5,date'1981-01-01');

insert into staffs values (2, 'Kairat', 'Nurtas', 'Mamasynyn uiynde', '87476674745', '121212121213',2,date'1982-02-02');

insert into staffs values (3, 'Nurlan', 'Espanov', 'Kempirlerdin qasynda', '87086674745', '141212121212',4,date'1984-03-03');

insert into staffs values (4, 'Toregali', 'Toreali', 'kiyz uide', '87026674745', '151512121212',3,date'1985-04-04');

select \* from staffs;

insert into tickets values (1,'num:1', date'2020-03-18');

insert into tickets values (2,'num:2', date'2020-02-18');

insert into tickets values (3,'num:3', date'2020-03-15');

insert into tickets values (4,'neogranichennyi', date'2020-03-29');

select \*from tickets;

insert into readers values (1, 'qsqqsqs', 'dsfsf', date'1981-01-01', 'Jerdin bir jerinde','87076674745','00008888444', 1);

insert into readers values (2, 'Kairat', 'Nurtas', date'1982-02-02', 'Mamasynyn uiynde', '87476674745', '121212121213',2);

insert into readers values (3, 'Nurlan', 'Espanov',date'1983-03-03', 'Kempirlerdin qasynda', '87086674745', '141212121212',4);

insert into readers values (4, 'Toregali', 'Toreali',date'1984-04-04', 'kiyz uide', '87026674745', '151512121212',3);

select \*from readers;

Alter table tickets add reader\_id int references readers;

UPDATE tickets SET reader\_id=1 WHERE ticket\_id =1;

UPDATE tickets SET reader\_id=2 WHERE ticket\_id =2;

UPDATE tickets SET reader\_id=4 WHERE ticket\_id =3;

UPDATE tickets SET reader\_id=3 WHERE ticket\_id =4;

select \*from tickets;

update readers set ticket\_id=1 where reader\_id=1;

insert into debtors values(1,2,3,4,date'2020-05-25');

insert into debtors values(2,4,1,3,date'2020-05-25');

insert into debtors values(3,2,2,4,date'2020-05-25');

insert into debtors values(4,2,4,6,date'2020-05-25');

select \*from debtors;

insert into bookreturns values(1,2,date'2020-09-25');

insert into bookreturns values(2,4,date'2020-09-25');

select \*from bookreturns;

2.3 Functions

It is a small function which entered number and output book\_name where id equal to this number;

create or replace function get\_title

(b\_id in books.book\_id%type)

return books.book\_name%type

is

b\_name books.book\_name%type;

begin

select book\_name into b\_name from books

where book\_id=b\_id;

RETURN b\_name;

EXCEPTION

WHEN no\_data\_found THEN

RETURN ('Book whith id=' ||b\_id ||' doesn`t exist' );

WHEN others THEN

RETURN ('Error!');

end get\_title;

declare

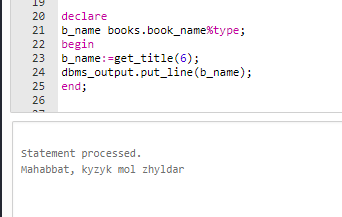
b\_name books.book\_name%type;

begin

b\_name:=get\_title(9);

dbms\_output.put\_line(b\_name);

end;



2) This function entered two numbesr id and count of books then output total price of books;

CREATE OR REPLACE FUNCTION book\_price(b\_id IN books.book\_id%TYPE, bcount int)

RETURN books.price%TYPE

is

all\_price books.price%TYPE;

bprice books.price%TYPE;

BEGIN

SELECT price INTO bprice FROM books WHERE book\_id=b\_id;

all\_price:=bprice \* bcount;

RETURN all\_price;

EXCEPTION

WHEN no\_data\_found THEN

return 0;

END book\_price;

DECLARE

all\_price books.price%type;

b\_id books.book\_id%type:=9;

BEGIN

all\_price:=book\_price(b\_id,5);

if all\_price = 0 then

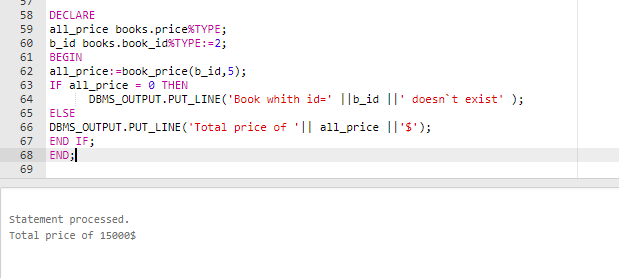
DBMS\_OUTPUT.PUT\_LINE('Book whith id=' ||b\_id ||' doesn`t exist' );

else

dbms\_output.put\_line('Total price of '|| all\_price ||'$');

end if;

END;



3) Fuction for finding books by authors

CREATE OR REPLACE FUNCTION get\_book(a\_surname authors.author\_surname%TYPE)

RETURN books.book\_name%type

IS

id\_a authors.author\_id%type;

a\_book books.book\_name%type;

BEGIN

SELECT author\_id into id\_a FROM authors where author\_surname=a\_surname;

for bb in (SELECT book\_name FROM books WHERE author\_id = id\_a)

loop

DBMS\_OUTPUT.PUT\_LINE(a\_surname || '`s book: ' || bb.book\_name);

end loop;

RETURN a\_book ;

END get\_book;

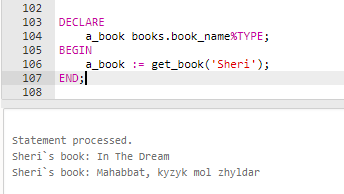
DECLARE

a\_book books.book\_name%TYPE;

BEGIN

a\_book := get\_book('Sheri');

END;



2.4 Procedures

This procedure output Author`s information about the book , When he wrote it and the title of book.

CREATE OR REPLACE PROCEDURE GET\_ALL\_INFO(

a\_id IN authors.author\_id%type

)IS

a\_name authors.author\_name%type;

BEGIN

SELECT author\_surname INTO a\_name FROM authors WHERE author\_id=a\_id;

FOR book IN (SELECT book\_name, wrote\_date FROM books WHERE author\_id = a\_id)

LOOP

DBMS\_OUTPUT.PUT\_LINE(a\_name||' wrote '|| book.book\_name || ' in ' || book.wrote\_date);

END LOOP;

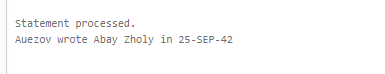
EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('There is no Author with id ' || a\_id);

END GET\_ALL\_INFO;

EXECUTE GET\_ALL\_INFO(1);



2) this procedure which insert into Authors which born in UK

CREATE OR REPLACE PROCEDURE ADD\_BRITISH\_WRITER(

a\_id authors.author\_id%type,

a\_name authors.author\_name%type,

a\_surname authors.author\_surname%type,

a\_mid authors.author\_middle\_name%type

)IS

BEGIN

INSERT INTO authors values (a\_id,a\_name,a\_surname,a\_mid,7);

EXCEPTION

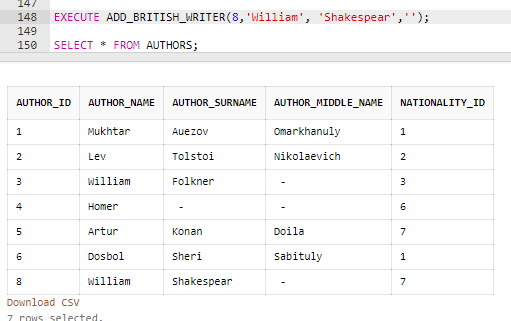
WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('this id already exist: ' || a\_id);

END ADD\_BRITISH\_WRITER;

EXECUTE ADD\_BRITISH\_WRITER(8,'William', 'Shakespear','');

SELECT \* FROM AUTHORS;



2.5 Triggers

This trigger come out when nationality which deleted has child

CREATE OR REPLACE TRIGGER before\_delete\_NAT

BEFORE DELETE ON nationality

FOR EACH ROW

BEGIN

FOR nat IN (SELECT nationality\_id FROM authors)

LOOP

IF :OLD.nationality\_id = nat.nationality\_id THEN

-- nn nationality.nationality\_name%type:=nationality.nationality\_name;

RAISE\_APPLICATION\_ERROR(-20500, 'this nationality has a child');

ELSE

DBMS\_OUTPUT.PUT\_LINE('SMELO BRAT!');

END IF;

END LOOP;

END;

DELETE FROM nationality Where nationality\_id = 2;

2) This trigger check price of books

CREATE OR REPLACE TRIGGER before\_update\_insert

BEFORE UPDATE OR INSERT ON books

FOR EACH ROW

DECLARE

BEGIN

IF UPDATING THEN

IF :NEW.price > 10000 THEN

RAISE\_APPLICATION\_ERROR(-20500, 'You entered very high price!’);

END IF;

IF :NEW.price = :OLD.price THEN

RAISE\_APPLICATION\_ERROR(-20500, 'You try to write same price');

END IF;

END IF;

IF INSERTING THEN

IF :NEW.price > 10000 THEN

RAISE\_APPLICATION\_ERROR(-20500, 'Price should be small than 10000tg. Ato kalta kotermidy');

END IF;

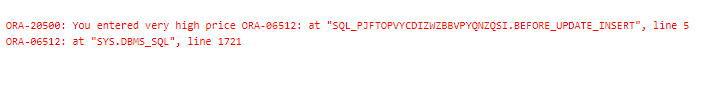
END IF;

END;

Update books SET price=11111 WHERE book\_id=3

select \* from books

select \* from nationality



3) This trigger output message after some activity of staffs

CREATE OR REPLACE TRIGGER trigger\_stf

BEFORE INSERT OR UPDATE OR DELETE ON staffs

FOR EACH ROW

DECLARE

BEGIN

IF DELETING THEN

DBMS\_OUTPUT.PUT\_LINE('The employee with id ' || :OLD.staff\_id || ' ' || 'is deleted');

END IF;

IF INSERTING THEN

DBMS\_OUTPUT.PUT\_LINE('The employee with id ' || :NEW.staff\_id || ' ' || 'is inserted');

END IF;

IF UPDATING THEN

DBMS\_OUTPUT.PUT\_LINE('The employee with id ' || :OLD.staff\_id || ' ' || 'is updated');

END IF;

END;

DELETE FROM staffs WHERE staff\_id = 3

UPDATE staffs

SET staff\_surname = 'Pushkin'

WHERE staff\_id = 2

INSERT INTO staffs VALUES (8, 'Cherepashki', 'Nindzia', 'lanalizacia', '87770009966', '151512121212',3,date'1985-04-04');

2.6 Packages

1) This package is designed to work with the table of nationalities .

CREATE OR REPLACE PACKAGE nat\_pkg IS

/\* A Procedure for .............\*/

PROCEDURE upd\_nat\_name(

name nationality.nationality\_name%type,

nat\_id nationality.nationality\_id%type );

END nat\_pkg;

CREATE OR REPLACE PACKAGE BODY nat\_pkg IS

FUNCTION valid\_nat\_id

(nat\_id IN nationality.nationality\_id%type)

RETURN BOOLEAN

IS

n NUMBER;

BEGIN

SELECT count(nationality\_id) INTO n FROM nationality WHERE nationality\_id = nat\_id;

IF (n = 1) THEN

RETURN TRUE;

ELSE RETURN FALSE;

END IF;

END valid\_nat\_id;

PROCEDURE upd\_nat\_name(

name nationality.nationality\_name%type,

nat\_id nationality.nationality\_id%type )

IS

invalid\_nat\_id EXCEPTION;

BEGIN

IF((valid\_nat\_id(nat\_id)) = TRUE) THEN

UPDATE nationality SET nationality\_name = name WHERE nationality\_id = nat\_id;

DBMS\_OUTPUT.PUT\_LINE('Nationalitys name with id='|| nat\_id ||' was updated to ' || name);

ELSE RAISE invalid\_nat\_id;

END IF;

EXCEPTION

WHEN invalid\_nat\_id THEN

DBMS\_OUTPUT.PUT\_LINE('Nationality with id='|| nat\_id ||' does not exist');

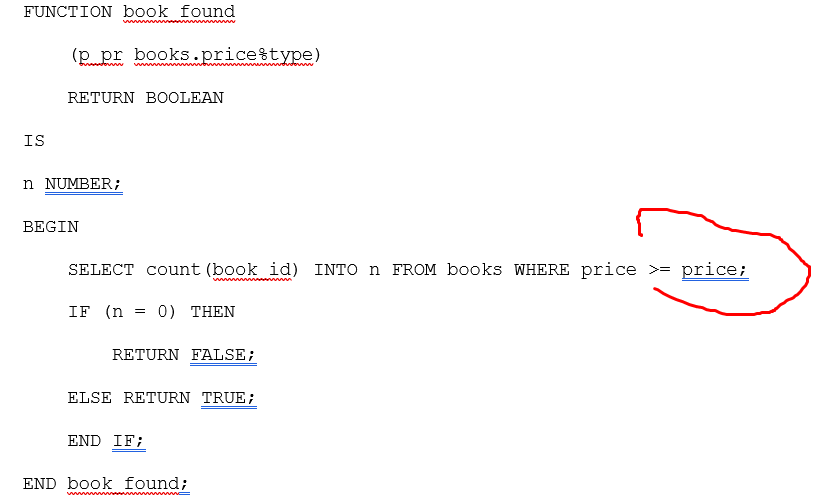
END upd\_nat\_name;

END nat\_pkg;

SELECT \* FROM nationality;

EXECUTE nat\_pkg.upd\_nat\_name('Nemec', 20);

2) This package is designed to work with the table of books .

(У меня здесь была ошибка при проверке экзепшна, тот который вы проверили. ошибся с переменной   
(человеческий фактор 😊)  
)

CREATE OR REPLACE PACKAGE book\_pkg IS

/\* A Procedure for .............\*/

PROCEDURE book\_price\_more\_than(p\_pr books.price%type);

function get\_title

(b\_id in books.book\_id%type)

return books.book\_name%type;

FUNCTION book\_price\_total

(b\_id IN books.book\_id%TYPE, bcount int)

RETURN books.price%TYPE;

END book\_pkg;

CREATE OR REPLACE PACKAGE BODY book\_pkg IS

FUNCTION book\_found

(p\_pr books.price%type)

RETURN BOOLEAN

IS

n NUMBER;

BEGIN

SELECT count(book\_id) INTO n FROM books WHERE price >= p\_pr;

IF (n = 0) THEN

RETURN FALSE;

ELSE RETURN TRUE;

END IF;

END book\_found;

PROCEDURE book\_price\_more\_than(p\_pr books.price%type)

IS

CURSOR pr\_cursor IS

SELECT book\_name, price

FROM books

WHERE price >= p\_pr;

not\_found EXCEPTION;

BEGIN

IF (book\_pkg.book\_found(p\_pr)=true) THEN

FOR i IN pr\_cursor

LOOP

DBMS\_OUTPUT.PUT\_LINE(i.book\_name|| ' ' || i.price);

END LOOP;

ELSE

RAISE not\_found;

END IF;

EXCEPTION

WHEN not\_found THEN

DBMS\_OUTPUT.PUT\_LINE('No BOOKS with such price');

END book\_price\_more\_than;

function get\_title

(b\_id in books.book\_id%type)

return books.book\_name%type

is

b\_name books.book\_name%type;

begin

select book\_name into b\_name from books

where book\_id=b\_id;

RETURN b\_name;

EXCEPTION

WHEN no\_data\_found THEN

RETURN ('Book whith id=' ||b\_id ||' doesn`t exist' );

WHEN others THEN

RETURN ('Error!');

end get\_title;

FUNCTION book\_price\_total(b\_id IN books.book\_id%TYPE, bcount int)

RETURN books.price%TYPE

is

all\_price books.price%TYPE;

bprice books.price%TYPE;

BEGIN

SELECT price INTO bprice FROM books WHERE book\_id=b\_id;

all\_price:=bprice \* bcount;

RETURN all\_price;

EXCEPTION

WHEN no\_data\_found THEN

return 0;

END book\_price\_total;

END book\_pkg;

EXECUTE book\_pkg.book\_price\_more\_than(90000);

declare

b\_name books.book\_name%type;

begin

b\_name:=book\_pkg.get\_title(6);

dbms\_output.put\_line(b\_name);

end;

DECLARE

all\_price books.price%type;

b\_id books.book\_id%type:=2;

BEGIN

all\_price:=book\_pkg.book\_price\_total(b\_id,6);

if all\_price = 0 then

DBMS\_OUTPUT.PUT\_LINE('Book whith id=' ||b\_id ||' doesn`t exist' );

else

dbms\_output.put\_line('Total price of '|| all\_price ||'$');

end if;

END;

SELECT \* FROM books;

3) This package is designed to work with the table of authors .

CREATE OR REPLACE PACKAGE auth\_pkg IS

PROCEDURE GET\_ALL\_INFO(a\_id IN authors.author\_id%type);

PROCEDURE ADD\_BRITISH\_WRITER(

a\_id authors.author\_id%type,

a\_name authors.author\_name%type,

a\_surname authors.author\_surname%type,

a\_mid authors.author\_middle\_name%type);

END auth\_pkg;

CREATE OR REPLACE PACKAGE BODY auth\_pkg IS

PROCEDURE GET\_ALL\_INFO(

a\_id IN authors.author\_id%type

)IS

a\_name authors.author\_name%type;

BEGIN

SELECT author\_surname INTO a\_name FROM authors WHERE author\_id=a\_id;

FOR book IN (SELECT book\_name, wrote\_date FROM books WHERE author\_id = a\_id)

LOOP

DBMS\_OUTPUT.PUT\_LINE(a\_name||' wrote '|| book.book\_name || ' in ' || book.wrote\_date);

END LOOP;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('There is no Author with id ' || a\_id);

END GET\_ALL\_INFO;

PROCEDURE ADD\_BRITISH\_WRITER(

a\_id authors.author\_id%type,

a\_name authors.author\_name%type,

a\_surname authors.author\_surname%type,

a\_mid authors.author\_middle\_name%type

)IS

BEGIN

INSERT INTO authors values (a\_id,a\_name,a\_surname,a\_mid,7);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('this id already exist: ' || a\_id);

END ADD\_BRITISH\_WRITER;

END auth\_pkg;

EXECUTE auth\_pkg.GET\_ALL\_INFO(3);

EXECUTE auth\_pkg.ADD\_BRITISH\_WRITER(9,'William', 'Shakespear','');

SELECT \* FROM AUTHORS;