DBMS.2

END TERM PROJECT

PROJECT NAME: ONLINE BOOK SHOP

PRESENTED TO: AZAMAT SEREK

PRESENTED BY:	KANGILOV SULAIMON	210103464
	AZHARBAYEV RAIYMBEK	210103300
	ASSIMOV ILKHAM	210103427
	GARIFULLA AMIR	210103024

PROJECT DESCRIPTION:

- This project entails the development of a database management system (DBMS) to support data manipulation and storage in a bookstore context. The objective is to facilitate online sales, provide customers with accurate and timely information on books, effectively manage data on customers and their transactions, as well as comprehensively monitor the inventory of books, sales transactions, and financial transfers.

- The proposed system will enable the bookstore to efficiently store and manage data on the inventory of books, including the quantity in stock, availability, and pricing information. Additionally, the system will allow for the integration of customer data, including personal information, purchase history, and credit card details, to support efficient sales transactions. The system will also capture detailed data on sales, including revenue, cost of goods sold, and profits, and generate reports to provide insight into the performance of the bookstore.

- Furthermore, the system will encompass features for managing online transactions, tracking the delivery of books to buyers, and maintaining records on the status of payments and deliveries. These functionalities will ensure that customers receive timely updates on their orders, and that the bookstore maintains accurate records of all transactions.

- The proposed DBMS is specifically designed to cater to the needs of small and medium-sized businesses, providing them with a comprehensive solution for data storage and manipulation, transaction processing, and financial management. By adopting this system, bookstores can streamline their operations, reduce errors, and enhance the overall customer experience.

PROJECT STRUCTURE:

TABLES: BOOK, BASKET, SUPPLIER, DELIVERY, TRANSACTION, CUSTOMER, CARD, ORDERS, ORDER_DETAIL

BOOK:

```
  CREATE TABLE "BOOK"
   ( "BOOK_ID" NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY
1 START WITH 1 CACHE 20 NOORDER NOCYCLE NOKEEP NOSCALE NOT
NULL ENABLE,
"AMOUNT" NUMBER NOT NULL ENABLE,
"SUPPLIER_ID" NUMBER NOT NULL ENABLE,
"ISBN" VARCHAR2(20 CHAR) NOT NULL ENABLE,
"AUTHOR" VARCHAR2(50 CHAR) NOT NULL ENABLE,
"NAME" VARCHAR2(50 CHAR) NOT NULL ENABLE,
"DESCRIPTION" VARCHAR2(4000 CHAR) NOT NULL ENABLE,
"PRICE" NUMBER NOT NULL ENABLE,
 CONSTRAINT "BOOK_PK" PRIMARY KEY ("BOOK_ID")
  USING INDEX ENABLE
   ) ;

  ALTER TABLE "BOOK" ADD CONSTRAINT "BOOK_FK" FOREIGN KEY
("SUPPLIER_ID")
  REFERENCES "SUPPLIER" ("SUPPLIER_ID") ENABLE;

  CREATE OR REPLACE EDITIONABLE TRIGGER "BOOK_SHOW_ROWS"
before insert on book
for each row
declare
num number;
begin
select count(*) into num from book;
dbms_output.put_line('Before insertion there are '||num||'
rows');
end;
/
ALTER TRIGGER "BOOK_SHOW_ROWS" ENABLE;
```

BASKET:

```
    CREATE TABLE "BASKET"
   ( "CUSTOMER_ID" NUMBER NOT NULL ENABLE,
"BOOK_ID" NUMBER NOT NULL ENABLE,
"QUANTITY" NUMBER NOT NULL ENABLE
   ) ;
```

```
  ALTER TABLE "BASKET" ADD CONSTRAINT "BASKET_FK2" FOREIGN KEY
("BOOK_ID")
  REFERENCES "BOOK" ("BOOK_ID") ENABLE;
  ALTER TABLE "BASKET" ADD FOREIGN KEY ("CUSTOMER_ID")
  REFERENCES  "CUSTOMER"  ("CUSTOMER_ID")  ON  DELETE  CASCADE
ENABLE;

  CREATE OR REPLACE EDITIONABLE TRIGGER "BASKET_SHOW_ROWS"
before insert on basket
for each row
declare
num number;
begin
select count(*) into num from basket;
dbms_output.put_line('Before  insertion  there  are  '||num||'
rows');
end;
/
ALTER TRIGGER "BASKET_SHOW_ROWS" ENABLE;
```

SUPPLIER:

```
  CREATE TABLE "SUPPLIER" 
   ( "SUPPLIER_ID" NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER  NOCYCLE  NOKEEP  NOSCALE  NOT NULL ENABLE, 
"NAME" VARCHAR2(50 CHAR) NOT NULL ENABLE, 
"PHONE_NO" VARCHAR2(20 CHAR) NOT NULL ENABLE, 
"EMAIL" VARCHAR2(50 CHAR) NOT NULL ENABLE, 
 CONSTRAINT "SUPPLIER_PK" PRIMARY KEY ("SUPPLIER_ID")
  USING INDEX  ENABLE
   ) ;
```

DELIVERY:

```
  CREATE TABLE "DELIVERY" 
   ( "DELIVERY_ID" NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER  NOCYCLE  NOKEEP  NOSCALE  NOT NULL ENABLE, 
"ORDER_ID" NUMBER NOT NULL ENABLE, 
"STATUS" VARCHAR2(50) NOT NULL ENABLE, 
 CONSTRAINT "DELIVERY_PK" PRIMARY KEY ("DELIVERY_ID")
  USING INDEX  ENABLE
   ) ;

  ALTER TABLE "DELIVERY" ADD CONSTRAINT "DELIVERY_FK" FOREIGN KEY ("ORDER_ID")
  REFERENCES "ORDER" ("ORDER_ID") ENABLE;

  CREATE OR REPLACE EDITIONABLE TRIGGER "DELIVERY_SHOW_ROWS" 
before insert on delivery
for each row
declare
num number;
begin
select count(*) into num from delivery;
dbms_output.put_line('Before insertion there are '||num||' rows');
end;
/
ALTER TRIGGER "DELIVERY_SHOW_ROWS" ENABLE;
```

TRANSACTION:

```
  CREATE TABLE "TRANSACTION"
   ( "TRANSACTION_ID" NUMBER GENERATED BY DEFAULT ON NULL AS
IDENTITY MINVALUE 1 MAXVALUE 9999999999999999999999999999
INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER  NOCYCLE  NOKEEP
NOSCALE  NOT NULL ENABLE,
"CUSTOMER_ID" NUMBER NOT NULL ENABLE,
"TRANSACTION_DATE" DATE NOT NULL ENABLE,
"STATUS" VARCHAR2(50) NOT NULL ENABLE,
"TOTAL_SUM" NUMBER NOT NULL ENABLE,
 CONSTRAINT "TRANSACTION_PK" PRIMARY KEY ("TRANSACTION_ID")
  USING INDEX  ENABLE
   ) ;

  ALTER TABLE "TRANSACTION" ADD CONSTRAINT "TRANSACTION_FK2"
FOREIGN KEY ("CUSTOMER_ID")
  REFERENCES "CUSTOMER" ("CUSTOMER_ID") ENABLE;
```

CUSTOMER:

```
  CREATE TABLE "CUSTOMER"
   ( "CUSTOMER_ID" NUMBER GENERATED BY DEFAULT ON NULL AS
IDENTITY MINVALUE 1 MAXVALUE 9999999999999999999999999999
INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER  NOCYCLE  NOKEEP
NOSCALE  NOT NULL ENABLE,
"FIRST_NAME" VARCHAR2(50 CHAR) NOT NULL ENABLE,
"LAST_NAME" VARCHAR2(50 CHAR) NOT NULL ENABLE,
"ADDRESS" VARCHAR2(50 CHAR) NOT NULL ENABLE,
"PHONE_NUMBER" VARCHAR2(50 CHAR) NOT NULL ENABLE,
 CONSTRAINT "CUSTOMER_PK" PRIMARY KEY ("CUSTOMER_ID")
  USING INDEX  ENABLE
   ) ;

  CREATE OR REPLACE EDITIONABLE TRIGGER "CUSTOMER_SHOW_ROWS"
before insert on customer
for each row
declare
num number;
begin
select count(*) into num from customer;
dbms_output.put_line('Before insertion there are '||num||'
rows');
end;
/
ALTER TRIGGER "CUSTOMER_SHOW_ROWS" ENABLE;
```

CARD:

```
  CREATE TABLE "CARD"
   ( "CUSTOMER_ID" NUMBER NOT NULL ENABLE,
"CARD_NUM" NUMBER NOT NULL ENABLE,
"CVV" NUMBER NOT NULL ENABLE,
"EXP_DATE" VARCHAR2(15) NOT NULL ENABLE,
"BALANCE" NUMBER NOT NULL ENABLE,
 UNIQUE ("CUSTOMER_ID")
  USING INDEX  ENABLE
   ) ;

  ALTER TABLE "CARD" ADD CONSTRAINT "CARD_FK" FOREIGN KEY
("CUSTOMER_ID")
  REFERENCES "CUSTOMER" ("CUSTOMER_ID") ENABLE;

  CREATE OR REPLACE EDITIONABLE TRIGGER "CARD_SHOW_ROWS"
before insert on card
for each row
declare
num number;
begin
select count(*) into num from card;
dbms_output.put_line('Before insertion there are '||num||'
rows');
end;
/
ALTER TRIGGER "CARD_SHOW_ROWS" ENABLE;
```

ORDER:

```
  CREATE TABLE "ORDER"
   ( "ORDER_ID" NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1
START WITH 1 CACHE 20 NOORDER  NOCYCLE  NOKEEP  NOSCALE  NOT
NULL ENABLE,
"CUSTOMER_ID" NUMBER NOT NULL ENABLE,
"TRANSACTION_ID" NUMBER NOT NULL ENABLE,
 CONSTRAINT "ORDER_PK" PRIMARY KEY ("ORDER_ID")
  USING INDEX  ENABLE
   ) ;

  ALTER TABLE "ORDER" ADD CONSTRAINT "ORDER_FK" FOREIGN KEY
("CUSTOMER_ID")
  REFERENCES "CUSTOMER" ("CUSTOMER_ID") ENABLE;
  ALTER TABLE "ORDER" ADD CONSTRAINT "TRANSACTION_FK" FOREIGN
KEY ("TRANSACTION_ID")
```

```
  REFERENCES "TRANSACTION" ("TRANSACTION_ID") ON DELETE CASCADE
ENABLE;
```

ORDER_DETAIL:

```
  CREATE TABLE "ORDER_DETAIL"
   ( "ORDER_ID" NUMBER NOT NULL ENABLE,
"BOOK_ID" NUMBER NOT NULL ENABLE,
"QUANTITY" NUMBER NOT NULL ENABLE
   ) ;

  ALTER TABLE "ORDER_DETAIL" ADD CONSTRAINT "HISTORY_FK" FOREIGN
KEY ("ORDER_ID")
  REFERENCES "ORDER" ("ORDER_ID") ON DELETE CASCADE ENABLE;
  ALTER TABLE "ORDER_DETAIL" ADD CONSTRAINT "HISTORY_FK1"
FOREIGN KEY ("BOOK_ID")
  REFERENCES "BOOK" ("BOOK_ID") ON DELETE CASCADE ENABLE;
```

ENTITY RELATIONSHIP DIAGRAM (ERD):

HERE IS THE LINK TO (ERD)

PROCEDURES:

1)
```
create or replace procedure order_book

(customer_id in number)

is

basket_empty exception;

not_enough exception;

no_customer exception;

not_enough_book exception;

num_of_products number;

row_basket basket%rowtype;

total_sum number := 0;

price number;

amount number;

sum_of_card number;
```

```
order_id number;

transaction_id number;

num number;

begin

select count(*) into num from customer where CUSTOMER_ID =
customer_id;

if num = 0 then

raise no_customer;

end if;

select count(*) into num_of_products from basket where
CUSTOMER_ID=customer_id;

if num_of_products = 0 then

raise basket_empty;

end if;

for row_basket in (select * from basket where
CUSTOMER_ID=customer_id)


loop

select AMOUNT into amount from book where
BOOK_ID=row_basket.BOOK_ID;

if amount<row_basket.QUANTITY then

raise not_enough_book;

end if;

end loop;

select BALANCE into sum_of_card from card where
CUSTOMER_ID=customer_id;

for row_basket in (select * from basket where
CUSTOMER_ID=customer_id)

loop

select PRICE into price from book where
BOOK_ID=row_basket.BOOK_ID;

total_sum := total_sum + (price * row_basket.QUANTITY);

end loop;
```

```
if total_sum > sum_of_card then

insert into "TRANSACTION" (CUSTOMER_ID, TRANSACTION_DATE,
STATUS, TOTAL_SUM) values (customer_id, sysdate(), 'failed',
total_sum);

raise not_enough;

end if;

insert into "TRANSACTION" (CUSTOMER_ID, TRANSACTION_DATE,
STATUS, TOTAL_SUM) values (customer_id, sysdate(),
'successful', total_sum) returning TRANSACTION_ID into
transaction_id;

insert into "ORDER" (CUSTOMER_ID, TRANSACTION_ID) values
(customer_id, transaction_id) returning ORDER_ID into
order_id;

for row_basket in (select * from basket where
CUSTOMER_ID=customer_id)

loop

insert into order_detail(ORDER_ID, BOOK_ID, QUANTITY)
values(order_id, row_basket.BOOK_ID, row_basket.QUANTITY);

update book set AMOUNT = AMOUNT - row_basket.QUANTITY where
BOOK_ID = row_basket.BOOK_ID;

end loop;

update card set balance = sum_of_card - total_sum where
CUSTOMER_ID = customer_id;

delete from basket where CUSTOMER_ID = customer_id;

insert into delivery (ORDER_ID, STATUS) values (order_id, 'on
the way');

exception

when basket_empty then

dbms_output.put_line('Basket is empty');

when not_enough then

dbms_output.put_line('Not enough balance in card');

when no_customer then

dbms_output.put_line('There is no such customer');

when not_enough_book then
```

```
dbms_output.put_line('Unfortunately there is not enough amount
of books for you');

when others then

dbms_output.put_line('Customer doesn''t have card');

end;
```

2)
```
create or replace procedure confirm_delivery

(order_id in number)

is

begin

update delivery set STATUS = 'delivered' where ORDER_ID =
order_id;

delete from "ORDER" where ORDER_ID = order_id;

delete from order_detail where ORDER_ID = order_id;

exception

when others then

dbms_output.put_line('There is no such order');

end;
```

3)
```
create or replace PROCEDURE CUSTOMERS_BASKET(

    CUST_ID BASKET.CUSTOMER_ID%TYPE

) IS

    F_NAME CUSTOMER.FIRST_NAME%TYPE;

    L_NAME CUSTOMER.LAST_NAME%TYPE;

    CURSOR "C_BASKET" IS

        SELECT BOOK."NAME" AS TITLE, BASKET.QUANTITY AS
QUANTITY

        FROM BASKET JOIN BOOK USING (BOOK_ID)

        WHERE CUSTOMER_ID = CUST_ID;

BEGIN

    SELECT FIRST_NAME, LAST_NAME INTO F_NAME, L_NAME FROM
CUSTOMER WHERE CUSTOMER_ID = CUST_ID;

    DBMS_OUTPUT.PUT_LINE('Customer full name: ' || F_NAME || '
' || L_NAME);
```

```
    FOR "BASKET" IN "C_BASKET" LOOP

        DBMS_OUTPUT.PUT_LINE('Book name: ' || "BASKET"."TITLE"
|| ', Quantity: ' || "BASKET"."QUANTITY");

    END LOOP;

END;
```

4)
```
create or replace PROCEDURE SUPPLIER_BOOKS IS

    CURSOR "C_INFO" IS

        SELECT "SUPPLIER_ID", "SUPPLIER"."NAME" AS
"SUPPLIER_NAME", COUNT(*) AS "NUMBER_OF_BOOKS"

        FROM "BOOK" JOIN "SUPPLIER" USING ("SUPPLIER_ID")

        GROUP BY "SUPPLIER_ID", "SUPPLIER"."NAME"

        ORDER BY "SUPPLIER_ID";

BEGIN

    FOR "INFO" IN "C_INFO" LOOP

        DBMS_OUTPUT.PUT_LINE('Supplier name: ' ||
"INFO"."SUPPLIER_NAME");

        DBMS_OUTPUT.PUT_LINE('Number of books: ' ||
"INFO"."NUMBER_OF_BOOKS");

        DBMS_OUTPUT.PUT_LINE('---------------');

    END LOOP;

END;
```

FUNCTION:

```
create or replace function count_rows

(table_name in varchar)

return number

is

num number;

begin

execute immediate 'select count(*) from '||table_name into
num;

return num;
```

```
exception

when others then

if sqlcode = -942 then

dbms_output.put_line('No such table exists');

else

dbms_output.put_line('Something else happened');

end if;

end;
```

PACKAGE:

```
create or replace PACKAGE CARD_DATA AS
        PROCEDURE ADD_CARD(
        CRD_ID CARD.CUSTOMER_ID%TYPE,
        CRD_NUM CARD.CARD_NUM%TYPE,
        CRD_CVV CARD.CVV%TYPE,
        CRD_EXP CARD.EXP_DATE%TYPE,
        CRD_BALANCE CARD.BALANCE%TYPE
        );
END CARD_DATA;

create or replace PACKAGE BODY CARD_DATA AS
    PROCEDURE ADD_CARD(
        CRD_ID CARD.CUSTOMER_ID%TYPE,
        CRD_NUM CARD.CARD_NUM%TYPE,
        CRD_CVV CARD.CVV%TYPE,
        CRD_EXP CARD.EXP_DATE%TYPE,
        CRD_BALANCE CARD.BALANCE%TYPE
    ) IS
        lcount NUMBER;
    BEGIN
        SELECT COUNT(*) INTO lcount FROM CUSTOMER WHERE
CUSTOMER_ID IN (SELECT CUSTOMER_ID FROM CARD WHERE CUSTOMER_ID
= CRD_ID);
        IF lcount = 0 THEN
            INSERT INTO CARD
            (CUSTOMER_ID, CARD_NUM, CVV, EXP_DATE, BALANCE)
            VALUES
            (CRD_ID, CRD_NUM, CRD_CVV, CRD_EXP, CRD_BALANCE);
            DBMS_OUTPUT.PUT_LINE('CARD IS SUCCESSFULLY
CREATED!');
        ELSE
```

```
            DBMS_OUTPUT.PUT_LINE('THERE IS ALREADY EXIST CARD
WITH THIS ID, PLEASE WRITE ANOTHER');
        END IF;
    END ADD_CARD;
END CARD_DATA;
/


create or replace PACKAGE CUSTOMER_DATA AS
    PROCEDURE ADD_CUSTOMER(
        CUST_FNAME CUSTOMER.FIRST_NAME%TYPE,
        CUST_LNAME CUSTOMER.LAST_NAME%TYPE,
        CUST_ADDRESS CUSTOMER.ADDRESS%TYPE,
        CUST_PHONE CUSTOMER.PHONE_NUMBER%TYPE
        );
        PROCEDURE DELETE_CUSTOMER(
        CUST_ID CUSTOMER.CUSTOMER_ID%TYPE
        );
END CUSTOMER_DATA;
/
create or replace PACKAGE BODY CUSTOMER_DATA AS
    PROCEDURE ADD_CUSTOMER(
        CUST_FNAME CUSTOMER.FIRST_NAME%TYPE,
        CUST_LNAME CUSTOMER.LAST_NAME%TYPE,
        CUST_ADDRESS CUSTOMER.ADDRESS%TYPE,
        CUST_PHONE CUSTOMER.PHONE_NUMBER%TYPE
    ) IS
    c_id NUMBER;
    c_id1 NUMBER;
    BEGIN
        SELECT COUNT(*) INTO c_id FROM CUSTOMER;
        c_id1:= c_id+1;
        INSERT INTO CUSTOMER
        (CUSTOMER_ID, FIRST_NAME, LAST_NAME, ADDRESS,
PHONE_NUMBER)
        VALUES
        (c_id1, CUST_FNAME, CUST_LNAME, CUST_ADDRESS,
CUST_PHONE);
        DBMS_OUTPUT.PUT_LINE('CUSTOMER IS SUCCESSFULLY
CREATED!');

    END ADD_CUSTOMER;

        PROCEDURE DELETE_CUSTOMER(
        CUST_ID CUSTOMER.CUSTOMER_ID%TYPE
    ) IS
    BEGIN
```

```
        DELETE FROM CARD WHERE CUSTOMER_ID = CUST_ID;
        DELETE FROM CUSTOMER WHERE CUSTOMER_ID = CUST_ID;
        DBMS_OUTPUT.PUT_LINE('CUSTOMER AND CARD WAS
SUCCESSFULLY DELETED!');
    END DELETE_CUSTOMER;
END CUSTOMER_DATA;
```

FUNCTIONAL DEPENDENCY (FD):

Keys: CUSTOMER_ID, CARD_ID, BOOK_ID, SUPPLIER_ID, DELIVERY_ID, ORDER_ID, TRANSACTION_ID

The minimal cover of FDs:
CUSTOMER_ID => CARD_ID
CUSTOMER_ID => ORDER_ID
CUSTOMER_ID => BOOK_ID
CUSTOMER_ID => TRANSACTION_ID
BOOK_ID => SUPPLIER_ID
ORDER_ID => DELIVERY_ID

The super key: CUSTOMER_ID
I. CUSTOMER_ID => TRANSACTION_ID, BOOK_ID, ORDER_ID, CARD_ID
II. BOOK_ID => SUPPLIER_ID
III. ORDER_ID => DELIVERY_ID