2016 group report
Group Members:
Sze Man Nga(21222541)
Tsui Hei Yi(21232938)
Tam Kar Nam(21225133)

part 2:
Here is the er diagram of our design:

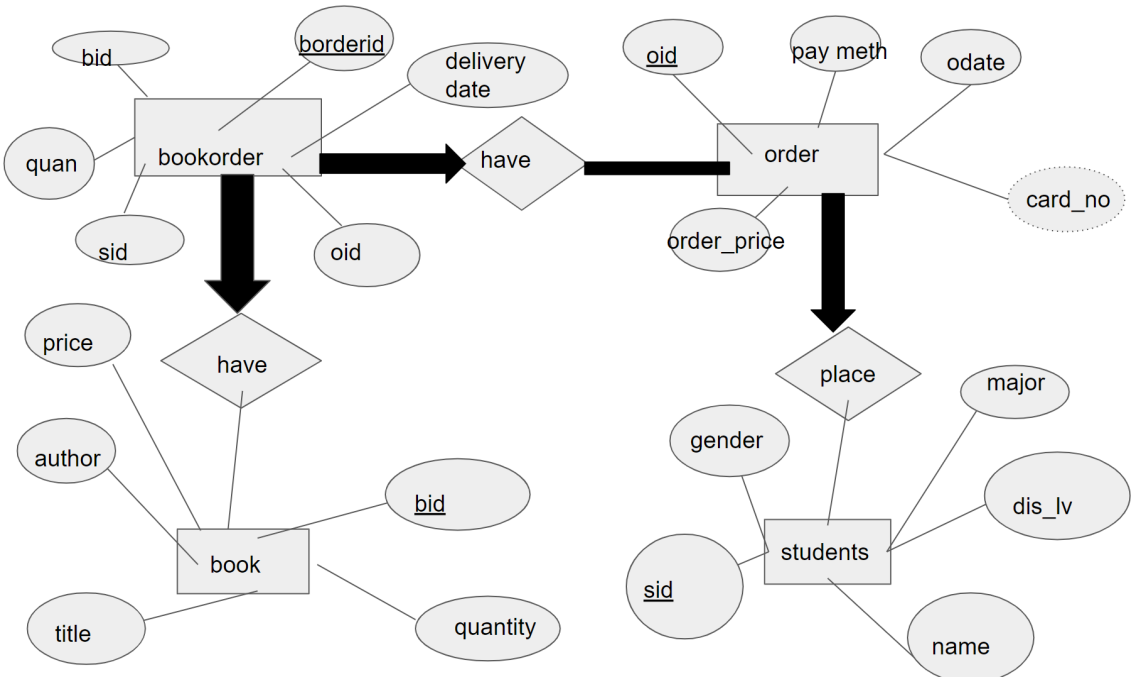

Table design:

STUDENTS(SID, GENDER , NAME ,MAJOR,DISCOUNT_LEVEL)
ORDER_PLACED(SID,order_price,paymethod ,oid,odate, card_no)
Books(bid , TITLE , quantity , author, price)
BookOrder_Hv(deliverydate,boid ,quantity ,oid , sid ,bid);
hv(bid ,boid);

Students				
sid	Gender	Name	Major	Discount_level

Book				
BID	TITLE	QUANTITY	AUTHOR	Price

Hv	
Boid	Bid

Bookorder_hv					
Delivery date	BOID	QUANTITY	OID	SID	BID

Order_placed					
sid	order_priece	paymethod	oid	odate	cardno

Part 3:

CREATE TABLE STUDENTS(SID INTEGER, GENDER CHAR(6), NAME CHAR(20),MAJOR CHAR(20),DISCOUNT_LEVEL INTEGER,Primary KEY(SID));
CREATE TABLE ORDER_PLACED(SID INTEGER NOT NULL,order_price Integer,paymethod char(20),oid integer,odate date, card_no integer,primary key(oid), Foreign key(sid) references students);

CREATE TABLE Books(bid INTEGER, TITLE char(20), quantity integer, author char(20), price integer,Primary key(bid));

CREATE TABLE BookOrder_Hv(deliverydate date,boid integer,quantity integer,oid integer not null, sid integer not null,bid integer not null, primary key(boid),foreign key(oid) references order_placed on delete cascade,foreign key(bid) references books);
create table hv(bid integer,boid integer not null,primary key(boid),foreign key(bid) references books, foreign key(boid) references bookorder_hv on delete cascade);

Above is the relational table of our design, we follow the requirement of er diagram to build the table.In the table since each order must have a customer so we set sid to not null to ensure every order has a customer. And the bookorder_hv because each book order should have an order. Should the oid should be not null in this relational table. And we set some tables in the delete cascade so when the customer deletes an order or cancels an order the order details are also canceled.

We create several of trigger to do the checking. Here is some explanation of how these trigger work

```
CREATE or replace TRIGGER CARD_CONSTRAINT

BEFORE INSERT OR UPDATE ON ORDER_PLACED

FOR EACH ROW

BEGIN

IF(:NEW.paymethod = 'creditcard' AND :NEW.card_no = NULL)

THEN

RAISE_APPLICATION_ERROR(-20010,  'INVALID CARD ');

END IF;

END;
```

In the above trigger, We want to check if the paymethod is a card, is there have a appropriate card number. If it is not an appropriate card number. It will raise a application error

```
CREATE OR REPLACE TRIGGER successinsert

    AFTER INSERT ON bookorder_hv

    FOR EACH ROW

    DECLARE

        c INTEGER;
```

```
        d integer;
    BEGIN
     SELECT SUM(order_price) into c FROM order_placed
        WHERE sid = :new.sid;
     select quantity into d from books where bid=:new.bid;
     UPDATE BOOKS SET QUANTITY=d-:new.quantity WHERE bid=:new.bid;
     IF (c>2000) THEN
         UPDATE Students SET DISCOUNT_LEVEL=20 WHERE sid = :new.sid;
             ELSIF (c>1000) THEN
         UPDATE Students SET DISCOUNT_LEVEL=10 WHERE sid = :new.sid;
     END IF;
    END;
```

In this trigger we will do the updates of discount level and quantity after an insertion is made. In this trigger we declare different variable. And store the sum of the customer current made in the bookshop. storing the current quantity of a book. If we find that there is a book made in the order we will update the quantity minus how many books that the customer placed. And We wll also see the total price is over 1000 and 2000 or not. If we find out that the customer have a total price more than 1000 or 2000. We will update the discount level of the customer.

```
CREATE OR REPLACE TRIGGER successdelete
    AFTER DELETE ON bookorder_hv
    FOR EACH ROW
    DECLARE
        c INTEGER;
        d integer;
e integer;
```

```
    BEGIN
select price into e from books where bid=:old.bid;
      SELECT SUM(order_price)-e*:old.quantity into c FROM order_placed
          WHERE sid = :old.sid;
      select quantity into d from books where bid=:old.bid;
      UPDATE BOOKS SET QUANTITY= d+:old.quantity WHERE bid=:old.bid;
      IF (c<1000) THEN
           UPDATE Students SET DISCOUNT_LEVEL=0 WHERE sid = :old.sid;
      ELSIF (c<2000) THEN
           UPDATE Students SET DISCOUNT_LEVEL=10 WHERE sid = :old.sid;
      END IF;
    END;
```

This trigger is similar to the above trigger successinsert. We will do the checking if we delete the book order. If we delete the bookorder. It will re-calculate the sum of the order and see whether the sum of the order is lower than 2000. If the sum of the order is lower than 2000 we will update the discount level of the student. Also we will update the quantity of the book.

And for the insertion trigger, there are several of trigger to check the insertion of a order

```
CREATE TRIGGER order_placed_trigger
BEFORE INSERT ON bookorder_hv
FOR EACH ROW
DECLARE
book_quantity INTEGER;
BEGIN
SELECT quantity
INTO book_quantity
```

```
FROM books

WHERE bid =:new.bid;

 IF book_quantity <= 0 or book_quantity<:new.quantity THEN

    RAISE_APPLICATION_ERROR(-20001, 'One or more books in the order are out of
stock.');

END IF;

end;
```

In this trigger we will do the checking of whether the students have placed a order which the bookshop cannot give him the required copies to him. In the trigger, we will first declare a variable to store the stock of the books that books current have and we will do the checking if the customer place a quantity of book is more than the stock or the stock is 0 but the customer still place a order in it so it will raise a application error to let the custom know he made a wrong order.

```
create trigger out_standing_trigger

BEFORE INSERT ON order_placed

FOR EACH ROW

DECLARE

outstanding_orders INTEGER;

begin



SELECT COUNT(*)

INTO outstanding_orders

FROM bookorder_hv

WHERE sid = :NEW.sid

AND deliverydate IS not NULL;


IF outstanding_orders > 0 THEN
```

```
    RAISE_APPLICATION_ERROR(-20002, 'The student has outstanding orders.');
END IF;
end;
```

In this trigger, it will check is that have any outstanding order. So we will do the checking to check is there have any outstanding order. In the trigger, we will see the order in the bookorder is there have null value. If there is no value in the bookorder. We will consider there is a outstanding order in the order and it will raise application error.

```
CREATE TRIGGER totalsum_trigger
before INSERT ON bookorder_hv
FOR EACH ROW
DECLARE
    total_price INTEGER;
    sum1 INTEGER;
    dis INTEGER;
    pri INTEGER;
BEGIN
    SELECT discount_level INTO dis FROM students WHERE sid= :new.sid;
    SELECT price INTO pri FROM books WHERE bid=:new.bid;
    SELECT order_price INTO total_price FROM order_placed WHERE oid= :new.oid;
    sum1 := (:new.quantity * pri * (1 - (dis/ 100)));
    UPDATE order_placed SET order_price = total_price + sum1 WHERE oid = :new.oid;
END;
```

In this trigger, we will also do a calculation of the sum of the book order. As in a order, there can be many books order. So we have to sum it up in the book order and update the total price of the order. In the trigger, we will first assign different values to the variable such as

discount level and the order_price into different variables. And we will use a formula to calculate the sum and update the total price of the order_placed.

And for the deletion there are several trigger to check whether it can be delete:

```
CREATE OR REPLACE TRIGGER day7

BEFORE DELETE ON ORDER_PLACED

FOR EACH ROW

DECLARE

   delivered_books INT;

   order_age INT;

BEGIN

   SELECT (TRUNC(SYSDATE) - TRUNC(:OLD.odate)) INTO order_age

   FROM dual;

   IF order_age > 7 THEN

      RAISE_APPLICATION_ERROR (-20011, 'Cannot cancel order: Order was made more than 7 days ago.');

   END IF;

END;
```

In this trigger, we will do the checking of if the customers have already placed an order for more than 7 days. In the trigger, We compare the date between the deleted date and system date and store it into a variable to see whether the customer tries to delete an order which is more than 7 day. If it is more than 7 days it will raise an error message of cannot cancel the order.

```
CREATE OR REPLACE TRIGGER check_order_cancellation

BEFORE DELETE ON ORDER_PLACED

for each row
```

```
DECLARE
   delivered_books INT;
BEGIN
   SELECT COUNT(boid) INTO delivered_books
   FROM BookOrder_Hv
   WHERE oid = :OLD.oid AND deliverydate IS NULL;

   IF delivered_books > 0 THEN
      RAISE_APPLICATION_ERROR (-20012, 'Cannot cancel order: Some books have been
delivered.');
   END IF;
end;
```

And in this trigger, we will check whether there are some orders that have been delivered. We will use a variable call count to count the number of delivered books. If there is a book delivered the variable will plus one if the variable is more than zero so it will raise a error message that cannot be deleted.

And for the update function. I implement a pl/sql statement to help me to do the updates. And this is the following statement:

```
Statement stm = conn.createStatement();
                  Statement stm2 = conn.createStatement();
                  String sql123="select boid from bookorder_hv where
sid="+yy;
                  ResultSet rs = stm2.executeQuery(sql123);
           while(rs.next()) {
                  String sql ="DECLARE\r\n" +
                              "  n DATE;\r\n" +
                              "  v NUMBER;\r\n" +
                              "BEGIN\r\n" +
                              "  SELECT deliverydate INTO n FROM
bookorder_hv WHERE boid="+rs.getString(1)+" ;\r\n" +
                              "\r\n" +
                              "  SELECT SYSDATE - n INTO v FROM dual;\r\n" +
                              "\r\n" +
                              "  UPDATE bookorder_hv SET deliverydate = NULL
WHERE  boid="+rs.getString(1)+"  AND v > 0;\r\n" +
```

```
                                "END;"

                                ;
        stm.execute(sql); // please pay attention that we use

             }} catch (SQLException e1) {
                    e1.printStackTrace();
                    noException = false;
              }


}
```

In the function, I first try to find there are how many bookorder in a order and try to check each of the book order. If it is delivered the number should be larger than zero so we will update and consider it.

.