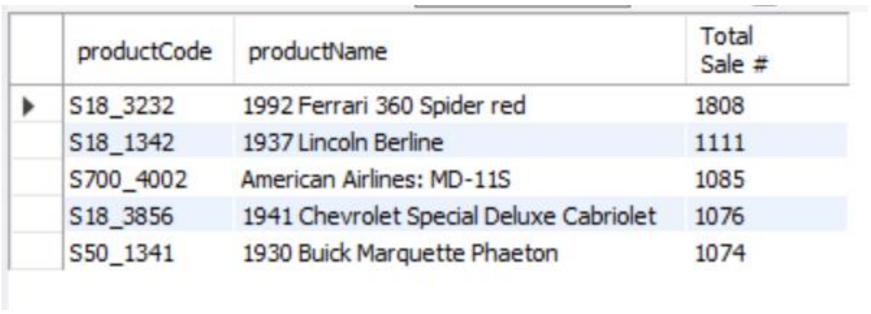
Birla Institute of Technology and Science Pilani, Pilani Campus
Department of Computer Science and Information System
2nd Semester 2020-21
Database System (CS F212)
Lab Quiz April-2021 (Solution)

Q:-1. List the top 5 selling products with name, product code, and total quantity sale. [2]

```
select orderdetails.productCode, products.productName,
sum(orderdetails.quantityOrdered) as 'Total Sale #'
from orderdetails natural join products
group by orderdetails.productCode
order by sum(orderdetails.quantityOrdered) DESC
limit 5;
```

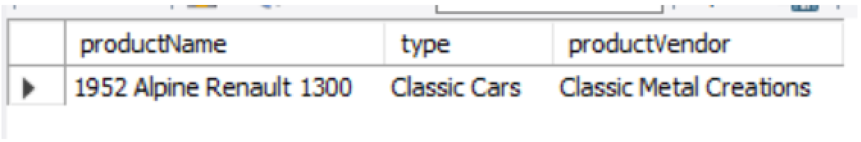


	productCode	productName	Total Sale #
▶	S18_3232	1992 Ferrari 360 Spider red	1808
	S18_1342	1937 Lincoln Berline	1111
	S700_4002	American Airlines: MD-11S	1085
	S18_3856	1941 Chevrolet Special Deluxe Cabriolet	1076
	S50_1341	1930 Buick Marquette Phaeton	1074

Q:-2. Write SQL queries to do the following task. [2+2+2+2+3]

a. Find the name, type, and vendor of the product that gives the seller the highest sale margin.

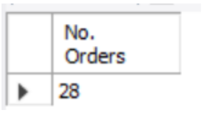
```
select  productName, type, productVendor from
products
where(msrp-buyPrice) = (select max(msrp-buyPrice)
from products);
```



	productName	type	productVendor
▶	1952 Alpine Renault 1300	Classic Cars	Classic Metal Creations

b. Count the number of orders for the product of the highest sale margin.

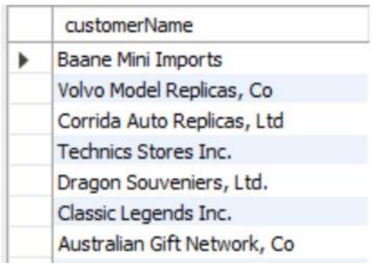
```
select count(orderID) as 'No. Orders' from products
natural join orderdetails natural join orders
where products.productCode = (
select products.productCode from products
where   (msrp-buyPrice) = (select max(msrp-buyPrice)
from products));
```



	No. Orders
▶	28

c. List the name of customers who brought the product that gives the highest sale margin to the seller.

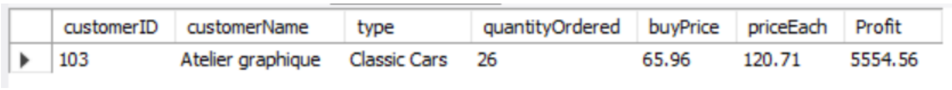
```
select customers.customerName from products natural
join orderdetails natural join orders natural join
customers
where products.productCode = (
select products.productCode from products
where   (msrp-buyPrice) = (select max(msrp-buyPrice)
from products));
```



	customerName
▶	Baane Mini Imports
	Volvo Model Replicas, Co
	Corrida Auto Replicas, Ltd
	Technics Stores Inc.
	Dragon Souveniers, Ltd.
	Classic Legends Inc.
	Australian Gift Network, Co

d. List the name of customer, customer ID, type of product, and quantity ordered; who gives the highest profit to the seller.

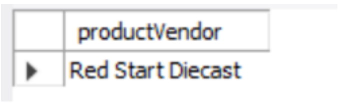
```
select customerID, customerName, type,
quantityOrdered, buyPrice, priceEach,
max((priceEach-buyPrice)*quantityOrdered) as
'Profit'
from products natural join orderdetails natural join
orders natural join customers;
```



	customerID	customerName	type	quantityOrdered	buyPrice	priceEach	Profit
▶	103	Atelier graphique	Classic Cars	26	65.96	120.71	5554.56

e. Find the name of the vendor who supplied all types of products to the seller.

```
select distinct s.productVendor from products s
where (
select count(distinct pds.type) from products pds
) = (
select count(distinct pd.Type) from products pd
where s.productVendor = pd.productVendor
);
```



	productVendor
▶	Red Start Diecast

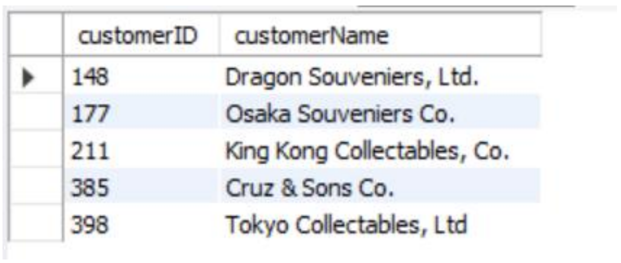
Q:-3. Write SQL queries to do the following task. [3+4]

a. Create a stored procedure, *'Get_Sale_repe_wise_Customers'* to print a sales representative-wise list of customers. The user inputs the sales representative's ID at the run time. Also, write the test query to call that procedure.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`Get_Sale_repe_wise_Customers`(IN empid INT)
READS SQL DATA
DETERMINISTIC
SQL SECURITY INVOKER
COMMENT 'customer reresentative'
```

```
BEGIN
select customers.customerID, customers.customerName
from customers
where customers.salesRepEmpID = empid;
END$$
DELIMITER ;

call Get_Sale_repe_wise_Customers(1621);
```

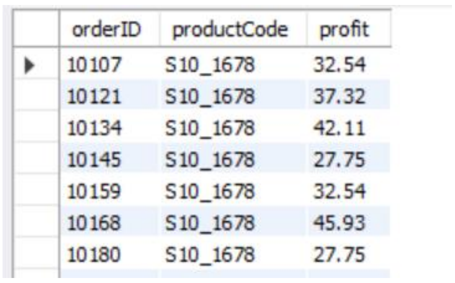


	customerID	customerName
▶	148	Dragon Souveniers, Ltd.
	177	Osaka Souveniers Co.
	211	King Kong Collectables, Co.
	385	Cruz & Sons Co.
	398	Tokyo Collectables, Ltd

b. Create a stored function, '*get_profit"* to show the seller's profit for each order, along with the order id and product code. Also, write the test query to call that function.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` FUNCTION
`get_profit`(order_id INT, pcode VARCHAR(15))
RETURNS decimal(5,2)
    READS SQL DATA
    DETERMINISTIC
BEGIN
  DECLARE profit DECIMAL(5,2);
  select (odr.priceEach - pds.buyPrice) into profit
  from orderdetails odr, products pds
  where odr.orderID = order_id
  and odr.productCode = pcode
  and odr.productCode = pds.productCode;
  RETURN profit;
END$$
DELIMITER ;

select  orderdetails.orderID,
orderdetails.productCode,
get_profit(orderdetails.orderID,
orderdetails.productCode) as profit from
orderdetails;
```



	orderID	productCode	profit
▶	10107	S10_1678	32.54
	10121	S10_1678	37.32
	10134	S10_1678	42.11
	10145	S10_1678	27.75
	10159	S10_1678	32.54
	10168	S10_1678	45.93
	10180	S10_1678	27.75

Q:-4. Write SQL queries to do the following task. [5+5]

a. Create a trigger, *'msrplog'* to maintain the logs of MSRP changes in a separate table, *MSRPlog<productCode, MSRP, updatetime>.* Write query for create table, create a trigger, and a test query to check the trigger's functionality.

```
CREATE TABLE `MSRPlog` (
  `productCode` varchar(15) NOT NULL,
  `MSRP` decimal(10,2) NOT NULL,
  `LastUpdate` timestamp NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`productCode`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

delimiter |
CREATE TRIGGER msrplog BEFORE update ON products
FOR EACH ROW
BEGIN
     INSERT into MSRPlog values (old.productCode,
     old.MSRP, NOW() );
END;
|
delimiter ;

UPDATE `indiamoters`.`products`
SET MSRP = 100
where products.productcode = 'S10_1678';
```

b. Write a stored procedure, *'fetch_type_orderdetails'* to fetch order details of specific product type and store these details in a separate table, *type_orderdetails<orderID, productCode, type, quantityOrdered, priceEach>*. The user passes product type as an input parameter. Write query for creating procedure, create table, and procedure call.

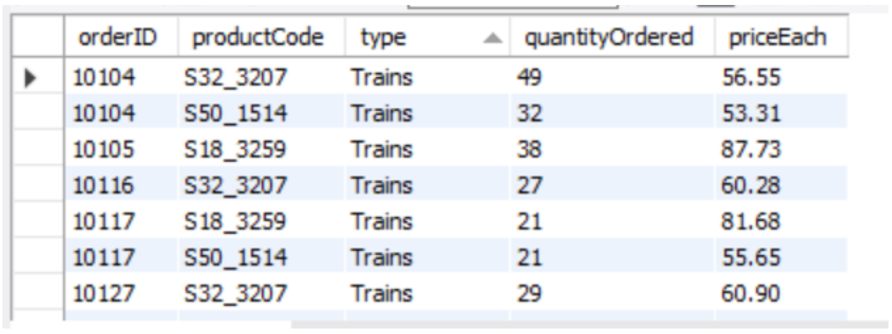
```
CREATE TABLE `type_orderdetails` (
  `orderID` int NOT NULL,
  `productCode` varchar(15) NOT NULL,
  `type` varchar(50) NOT NULL,
  `quantityOrdered` int NOT NULL,
  `priceEach` decimal(10,2) NOT NULL,
  PRIMARY KEY (`orderID`,`productCode`),
  KEY `productCode` (`productCode`),
  CONSTRAINT `type_orderdetails_ibfk_1` FOREIGN KEY
(`orderID`) REFERENCES `orders` (`orderID`),
  CONSTRAINT `type_orderdetails_ibfk_2` FOREIGN KEY
(`productCode`) REFERENCES `products`
(`productCode`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`fetch_type_orderdetails`(IN ptype VARCHAR(50))
READS SQL DATA
DETERMINISTIC
SQL SECURITY INVOKER
COMMENT 'fetch order details of specific product
type and store these details in separate table'
BEGIN
     DECLARE done INT DEFAULT FALSE;
     DECLARE b VARCHAR(15);
     DECLARE c VARCHAR(50);
     DECLARE a, d INT;
     DECLARE e DECIMAL(10,2);
     DECLARE cur1 CURSOR FOR select
     orderdetails.orderID, orderdetails.productCode,
     (select products.type from products where
     products.productCode =
     orderdetails.productCode)
     as ptype, orderdetails.quantityOrdered,
     orderdetails.priceEach
     from orderdetails;
     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done
= TRUE;
     OPEN cur1;
    read_loop: LOOP
          FETCH cur1 INTO a, b, c, d, e;
          IF done THEN
               LEAVE read_loop;
          END IF;
          IF c = ptype THEN
               INSERT INTO
               `indiamoters`.`type_orderdetails`
               (`orderID`, `productCode`, `type`,
               `quantityOrdered`, `priceEach`)
               VALUES (a, b, c, d, e);
          END IF;
          END LOOP;
     CLOSE cur1;
END$$
```

```
DELIMITER ;

CALL
`indiamoters`.`fetch_type_orderdetails`('Trains');

SELECT * FROM `indiamoters`.`type_orderdetails`;
```



orderID	productCode	type	quantityOrdered	priceEach
10104	S32_3207	Trains	49	56.55
10104	S50_1514	Trains	32	53.31
10105	S18_3259	Trains	38	87.73
10116	S32_3207	Trains	27	60.28
10117	S18_3259	Trains	21	81.68
10117	S50_1514	Trains	21	55.65
10127	S32_3207	Trains	29	60.90