

INFO 6210 Data Management and Database Design
Final Project 6

Database for Handling Online and in-store Sales at Liquor Shop

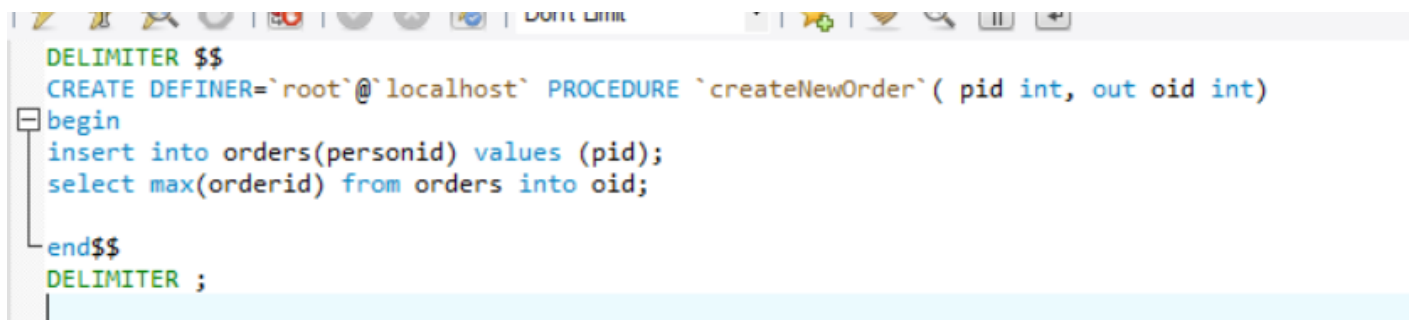
A liquor shop deals with a lot of products including different types of spirits and non-alcoholic commodities. A topic like this gives me the chance to explore lots of analytics. I am looking at helping customers buy liquor online as well and instore and I have written different stored procedures to place an order via the respective procurement medium.

To purchase alcohol in USA an individual must be 21 and above. Well, when this needs to be validated during an in-store sale, it shall be done physically. However, during the online procurement my stored procedure warns the Database Operator that this order should not be placed.

For me to place an order the DB operator must call three procedures. They are as follows;

1) createNewOrder(PersonID, @OrderID);

This function creates a new entry in the orders table and gives back the order id of this newly created table. Procedure Definition is as follows.



```
DELIMITER $$
CREATE DEFINER='root'@'localhost' PROCEDURE `createNewOrder` ( pid int, out oid int)
begin
insert into orders(personid) values (pid);
select max(orderid) from orders into oid;
end$$
DELIMITER ;
```

2) additem(ProductID, Quantity needed, @OrderID);

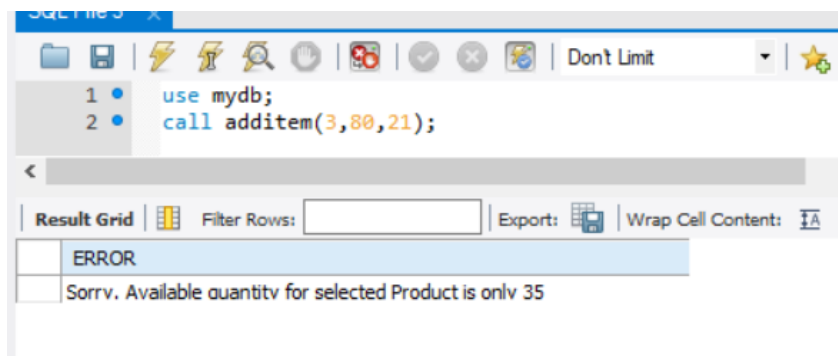
This procedure adds into the orderitem table with the orderID received from the previous procedure. Using this function:

- 1) We validate if inventory has available quantity
- 2) We can how many ever products we want to a single order.

```

DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `addOrderItem`(prodid int , quant int, ordid int)
Begin
if (quant> (select quantity from inventory where productid = prodid)) then
Select concat('Sorry. Available quantity for selected Product is only ',
(select quantity from inventory where productid = prodid)) as 'ERROR';
else
Insert into orderitems(productID,quantity,orderid, amount) values((select productid from product where productid = prodid),
quant, ordid, ((select price from product where productid = prodid) * quant));
End if;
end$$
DELIMITER ;

```



3) conclude order(@OrderID);

This procedure traverses through the orderItems table and calculates the sum of all items that belong to a certain Order.

```

DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `concludeorder`(oid int)
Begin
update orders
set Orderamount =(select sum(amount) from orderitems where orderid = oid) , Orderstatus ='Completed' where orderid=oid;
END$$
DELIMITER ;

```

4) createNewOnlineOrde

This procedure first checks the age of the customer and then alerts the DBO that he/she is underage or unverified. As we have online system lets imagine a scenario where I sign up for the service, send them a picture of my ID proving that I'm 21+ and then my status is verified. This age is verified using the checkage() UDF.

```

DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `createOnlineOrder`(persID int, out oid int)
begin
declare message varchar(100);
select checkage(persID) into message;
if message = 'Cannot buy'
then
Select 'Customer underage or unverified';
else
insert into onlineorders(personid) values (persid);
select max(OnlineOrderID) from onlineorders into oid;
end if;
end$$
DELIMITER ;

```

5) checkage()

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` FUNCTION `checkage`( pid int) RETURNS varchar(80) CHARSET utf8
begin
    declare message varchar(80);
    declare age float;
    declare veri varchar (20);
    set age:= (Select TIMESTAMPDIFF(YEAR, (select dateofbirth from person where PersonID=12), CURDATE()));
    set veri:= (select eligible from person where personid=pid);
    if (pid > (select max(personid) from person))
    then set message = 'Person doesnot exist';
    elseif ((age<21) || (veri = 'Unverified')) THEN
    set message = 'Cannot Buy';
    ELSE
    set message = 'Person is eligibile';
    end if;
    return message;
END$$
DELIMITER ;
```

Similarly, I have included other functions for online order for adding orderitems. Kindly find the concludeOnlineOrder() below.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `concludeonlineOrder`(in oid int)
Begin
update onlineorders
set amount =(select sum(costForProduct) from onlineorderitems where onlineorderid = oid),
orderstatus = 'completed', ShippingStatus='Ships next day of order' where onlineorderid=oid;
insert into shipment (OnlineOrderId, ShipmentStatus, LocationID, shipmentname)
values( oid , "Order Issued",
(select locationid from person where personid = (select personid from onlineorders where onlineorderid=oid)),
concat('Shipment - Order no-',oid));
END$$
DELIMITER ;
```

8) take_backup()

This procedure allows me to take a backup of all the data in the current DB.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `take_Backup`()
block1: BEGIN
    DECLARE tab_name char(50);
    DECLARE q varchar(1500);
    DECLARE done INTEGER DEFAULT 0;
    DECLARE cursorBackupTable CURSOR FOR SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES WHERE table_schema='mydb' and table_type='BASE TABLE';
    DROP DATABASE IF EXISTS mydb_backup;
    CREATE DATABASE mydb_backup;
    open cursorBackupTable;
    block2: begin
```

```

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
cur_loop:Loop
    FETCH cursorBackupTable into tab_name;
    IF done= 1 THEN LEAVE cur_loop;
    END IF;
    SET @q=CONCAT('DROP TABLE IF EXISTS mydb_backup.',tab_name);
    PREPARE stmt FROM @q;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
    SET @q=CONCAT('CREATE TABLE mydb_backup.',tab_name,' AS SELECT *
FROM mydb.',tab_name,' WHERE 1=1');
    PREPARE stmt FROM @q;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END LOOP cur_loop;
END block2;
close cursorBackupTable;
END block1$$
DELIMITER ;

```

Now that we are done with Stored procedures kindly find the views below:

1) allDrinks;

```

CREATE ALGORITHM=UNDEFINED DEFINER='root'@'localhost' SQL SECURITY DEFINER VIEW `mydb`.`alldrinks`
AS select `p`.`ProductName` AS `ProductName`,
(select `c`.`CategoryName` from `mydb`.`category` `c` where (`p`.`CategoryId` = `c`.`CategoryId`)) AS `Spirit`
,concat((select `m`.`ManufacturerName` from `mydb`.`manufacturer` `m` where (`p`.`ManufacturerId` = `m`.`Manufacturer`)),',',
(select `m`.`Country` from `mydb`.`manufacturer` `m` where (`p`.`ManufacturerId` = `m`.`Manufacturer`))) AS `Made by`,
concat(`p`.`AlcoholContent`,`%`) AS `Alcohol Content`,`i`.`Quantity` AS `Quantity Currently Availible`
from (`mydb`.`product` `p` join `mydb`.`inventory` `i` on((`p`.`ProductID` = `i`.`ProductID`)))
where (`p`.`ProductType` = 'Alcoholic');

```

Result Grid					
		Filter Rows:	Export:	Wrap Cell Content:	
	ProductName	Spirit	Made by	Alcohol Content	Quantity Currently Available
	Double Black Label	Whiskey	Jv Walker.Scotland	40.4%	16
	Samuel Adams	Beer	Samuel Adams.America	4.9%	46
	Blue Moon	Beer	Blue Moon Brewina Co..America	5.4%	46
	Harpoon IPA	Beer	Harpoon Brewerv.America	5.9%	50
	Radius Cabarnet	Red Wine	Perceot Wines.America	12%	34
	Jacob's Creek Shiraz	Red Wine	Jacob Creek Wines.Australia	13.9%	47
	Samara White	White Wine	Nashik Valleu Wine.India	14.5%	44
	Sula White Wine	White Wine	Sula Vinevards.India	14%	39
	Grev Goose	Vodka	Bacardi.France	39%	45
	Absolut	Vodka	The Absolut Companv.Sweden	41%	50
	Smirnoff	Vodka	Diaceo.England	44%	49
	Old Monk	Rum	Mohan Meakin .India	42.8%	46
	Bacardi White Rum	Rum	Bacardi.France	37.5%	47
	Captain Moran	Rum	Diaceo.England	41%	50
	Bombav Sapphire	Gin	Bombav Soirits Co.. Ltd.England	40%	40
	BeeFeater	Gin	James Burrough Ltd.England	40.5%	50

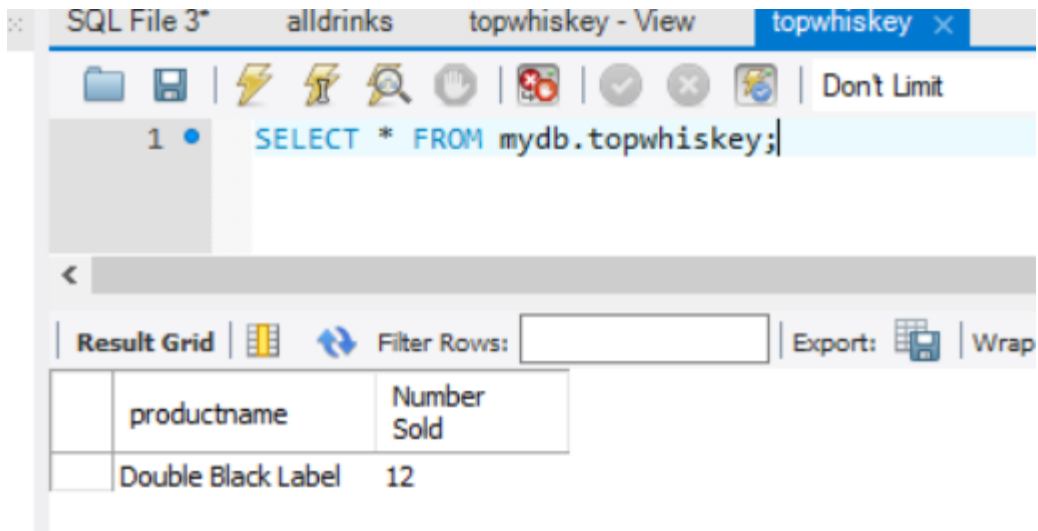
2) TopWhiskey

This Whiskey gives me the name of that whiskey that is the most sold joining three tables.

```

1 CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `root`@`localhost`
4     SQL SECURITY DEFINER
5     VIEW `mydb`.`topwhiskey` AS
6     SELECT
7         `p`.`ProductName` AS `productname`,
8         SUM(`oi`.`quantity`) AS `Number Sold`
9     FROM
10        ((`mydb`.`orderitems` `oi`
11        JOIN `mydb`.`product` `p` ON ((`p`.`ProductID` = `oi`.`productid`)))
12        JOIN `mydb`.`category` `c` ON ((`p`.`CategoryId` = `c`.`CategoryId`)))
13    WHERE
14        (`c`.`CategoryId` = 1)
15    GROUP BY `p`.`ProductID`
16    ORDER BY SUM(`oi`.`quantity`) DESC
17    LIMIT 1

```



We can now take a look at all the triggers created within the solution:

1) sale_ANSI: This trigger reduces the quantity once a sale is made. A similar trigger is written when an online sale is made.

2) Updateofavailibilty:

Once an order is canceled the quantity is updated in the inventory. A similar trigger is written when an online sale is made