PROJECT REPORT

ONLINE RETAIL APPLICATION DATABASE

BUAN 6320

Group 4:

Group Members

AKSHAY KAMOJI

CATHERINE HARPER

CONNOR SAVARD

NEHA VIJAYANANDA

Team Member Contribution

Project Category	Team Member(s)
Project Write Up	Connor, Neha, Catherine
Database Design & ER Diagram	Catherine, Akshay
.SQL File Creation	Akshay
Standard Reports	Connor, Catherine, Akshay
Stored Procedures	Neha, Catherine
Stored Functions	Catherine, Akshay
Views	Catherine, Neha
Triggers	Connor, Catherine
Report Consolidation	Connor, Catherine

Online Retail Environment

Online retail stores are virtual businesses that engage in e-commerce transactions to sell a range of goods to consumers or other businesses. They enable an owner to reach a larger and more geographically varied group of potential customers. By removing the need for a brick-and-mortar storefront site, online retailers can cut operating expenses by enabling an owner to reach a larger and more geographically varied group of potential customers. E-Commerce, since its inception, has become extremely popular and a preferred way of shopping for many people these days. This popularity is because of the advantages over typical shopping methods such as commuting, pricing, and more variety in terms of products. The chances for successful online firms will increase as the popularity of e-commerce rises. More than $1 in every $5 spent on retail purchases came from online orders in the first quarter of 2022. As sales revenues grow, so will the amount of data that retailers will collect and have their disposal to base future growth strategies on.

Industry Challenges

The online retail industry has its fair share of challenges when it comes to storing vast amounts of data and using it effectively to make actionable business decisions. The sheer scale of big data makes it necessary to use a nimble and well-organized database solution. Having a well-sorted database allows an online retailer to address today the major challenges they face, such as the following:

- **Gaining insight into customer behavior:** Customer behavior analysis provides insight into the different variables influencing a target audience. It provides information on the motives, priorities, and decision-making methods customers consider during their journey.
- **Setting prices more effectively:** The objective of online retailers is to maximize profits, and data analytics can help determine how to set prices more effectively by analyzing consumer trends and adjusting pricing accordingly.

- **Supply chain and inventory management:** One of the fundamental things for any business is to track sales, monitor stock levels, and automate the supply chain as much as possible.
- **Customer retention:** While it is important to attract loyal customers, it is just as important to retain them and make repeat customers. Behavior data analysis can help identify good and bad customer traits and reduce customer turn.
- **Demand forecasting:** Online retailers can use past data to anticipate the needed stock to meet demand and reduce unpurchased inventory and out-of-stock items.

Why Our Product

Our system is dependable, easy to use, organized, efficient, and gives end-users the tools and information they need to track and improve their retail organization's performance results. These system enhancements support necessary decision-making processes for the end-user while strategically enhancing customer services. Lastly, our system allows for increased capacity to manage product inventory, investigate trends, and track all sales transactions and activities – which have historically been labor-intensive operations. Through these unique product features, your retail organization will thrive in an increasingly competitive market while minimizing manual costs and efforts.

Scope

The project has these following capabilities:

- Inventory management
- Price information and management
- Product margin tracking
- Vendor information
- Customer information, including loyalty program members
- Customer analytics
- Order history and tracking

End-User Features

- Various organization departments shall mainly use the online retail application database to analyze product sales and other day-to-day operations.
- Built-in Triggers in the database which automatically react to specific actions or events
- Data entry automation
- Generate information related to various Business metrics such as:
    - Average time a customer spends on the app
    - Most profitable products
    - Ten highest and least selling products

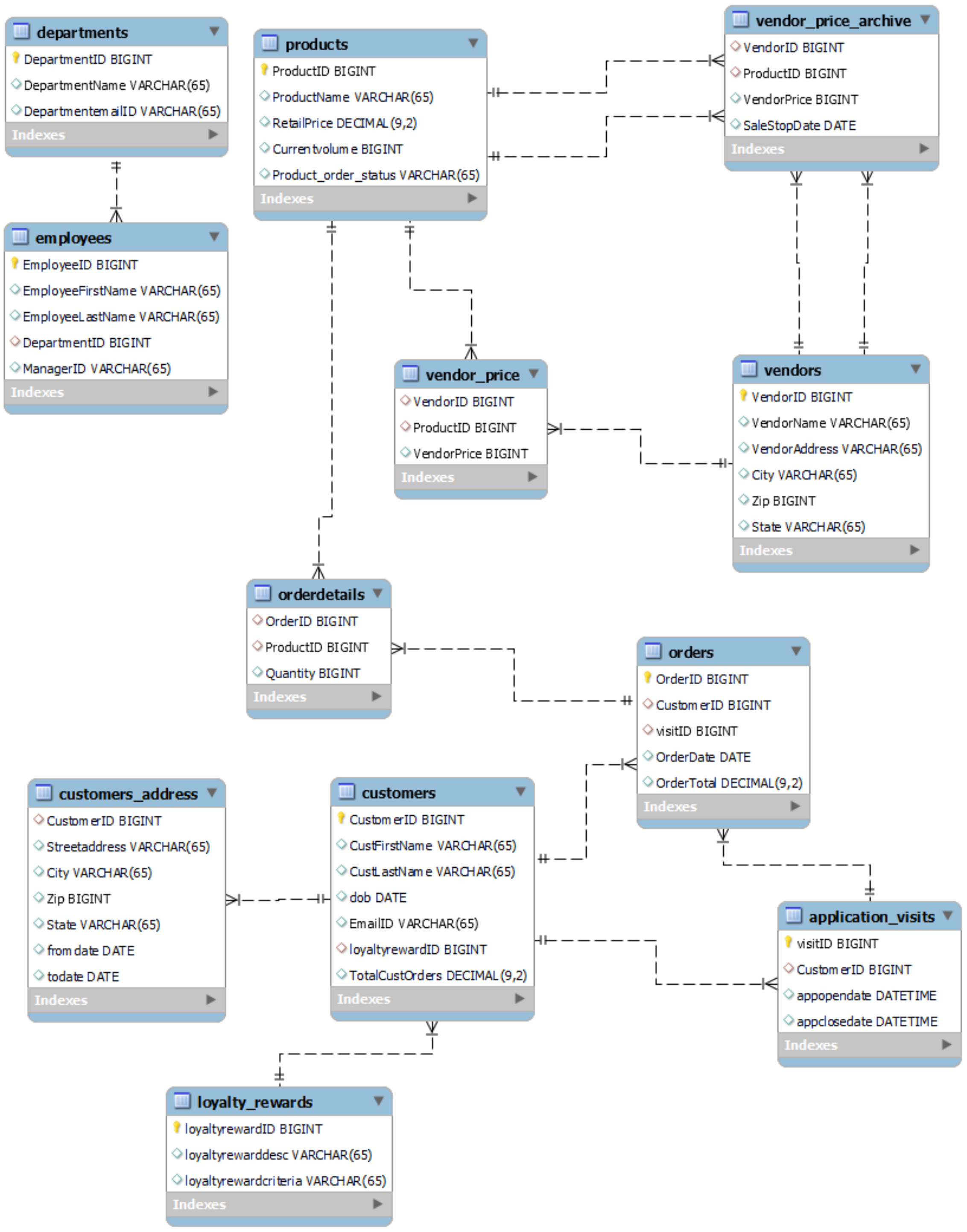
Benefits

1. Quick and easy access to data whenever required, with all the concerned departments of the organization being connected to a database that acts as a hub
2. Triggers can alert the concerned departments for impending actions or events that may cause bottlenecking, such as a shortage of a particular product in an inventory
3. Save time and reduce errors in data due to automated data entry
4. More precise tracking of various business metrics and other operations since different functions and stored procedures provide the ability to pull the desired information quickly
5. Provide business with actionable insights on revenue and expense performance and areas of opportunity to improve overall margin

Database Design

Table Name	Primary Key	Foreign Key	Non-Key Attributes	# of Rows
Vendor_Price		ProductID	VendorPrice	20
		VendorID		
Vendor_Price_Archive		ProductID	VendorPrice	0
		VendorID	SaleStopDate	
Products	ProductID		ProductName	15
			RetailPrice	
			CurrentVolume	
			Product_Order_Status	
Vendors	VendorID		VendorName	10
			VendorAddress	
			City	
			Zip	
			State	
Orders	OrderID	CustomerID	OrderDate	20
		VisitID	OrderTotal	
Orderdetails		OrderID	Quantity	35
		ProductID		
Customers	CustomerID	LoyaltyRewardID	CustFirstName	10
			CustLastName	
			DOB	
			EmailID	
			TotalCustOrders	
Customers_Address		CustomerID	StreetAddress	15
			City	
			Zip	
			State	
			FromDate	
			ToDate	
LoyaltyRewards	LoyaltyRewardID		LoyaltyRewardDesc	3
			LoyaltyRewardCriteria	
Employees	EmployeeID	DepartmentID	EmployeeFirstName	12
			EmployeeLastName	
			ManagerID	
Departments	DepartmentID		DepartmentName	3
			DepartmentEmailID	
Application_Visits	VisitID	CustomerID	AppOpenDate	50
			AppCloseDate	

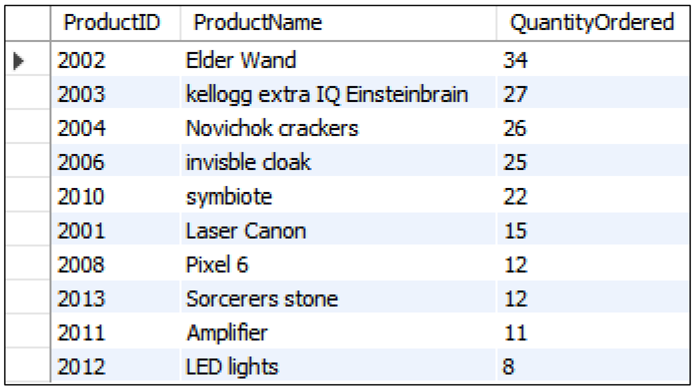
ER Diagram


departments
DepartmentID BIGINT
DepartmentName VARCHAR(65)
DepartmentemailID VARCHAR(65)
Indexes
employees
EmployeeID BIGINT
EmployeeFirstName VARCHAR(65)
EmployeeLastName VARCHAR(65)
DepartmentID BIGINT
ManagerID VARCHAR(65)
Indexes
products
ProductID BIGINT
ProductName VARCHAR(65)
RetailPrice DECIMAL(9,2)
Currentvolume BIGINT
Product_order_status VARCHAR(65)
Indexes
vendor_price_archive
VendorID BIGINT
ProductID BIGINT
VendorPrice BIGINT
SaleStopDate DATE
Indexes
vendor_price
VendorID BIGINT
ProductID BIGINT
VendorPrice BIGINT
Indexes
vendors
VendorID BIGINT
VendorName VARCHAR(65)
VendorAddress VARCHAR(65)
City VARCHAR(65)
Zip BIGINT
State VARCHAR(65)
Indexes
orderdetails
OrderID BIGINT
ProductID BIGINT
Quantity BIGINT
Indexes
orders
OrderID BIGINT
CustomerID BIGINT
visitID BIGINT
OrderDate DATE
OrderTotal DECIMAL(9,2)
Indexes
customers_address
CustomerID BIGINT
Streetaddress VARCHAR(65)
City VARCHAR(65)
Zip BIGINT
State VARCHAR(65)
fromdate DATE
todate DATE
Indexes
customers
CustomerID BIGINT
CustFirstName VARCHAR(65)
CustLastName VARCHAR(65)
dob DATE
EmailID VARCHAR(65)
loyaltyrewardID BIGINT
TotalCustOrders DECIMAL(9,2)
Indexes
application_visits
visitID BIGINT
CustomerID BIGINT
appopendate DATETIME
appclosedate DATETIME
Indexes
loyalty_rewards
loyaltyrewardID BIGINT
loyaltyrewarddesc VARCHAR(65)
loyaltyrewardcriteria VARCHAR(65)
Indexes

Standard Reports

1. **Name:** Top 10 Selling Products
   - **Team Member(s):** Connor
   - **Business Case:** Quickly displays the top 10 selling products. This will assist employees in knowing which products are demanded most by customers.

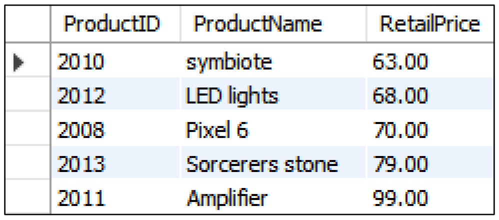
```
SELECT ProductID, ProductName, SUM(Quantity) AS 'QuantityOrdered'
FROM products
INNER JOIN OrderDetails USING (ProductID)
GROUP BY 1
ORDER BY QuantityOrdered DESC
LIMIT 10;
```



ProductID	ProductName	QuantityOrdered
2002	Elder Wand	34
2003	kellogg extra IQ Einsteinbrain	27
2004	Novichok crackers	26
2006	invisble cloak	25
2010	symbiote	22
2001	Laser Canon	15
2008	Pixel 6	12
2013	Sorcerers stone	12
2011	Amplifier	11
2012	LED lights	8

2. **Name:** 5 Cheapest Products
   - **Team Member(s):** Connor
   - **Business Case:** Displays the 5 products priced the lowest. This will assist employees in knowing which products are the most affordable.

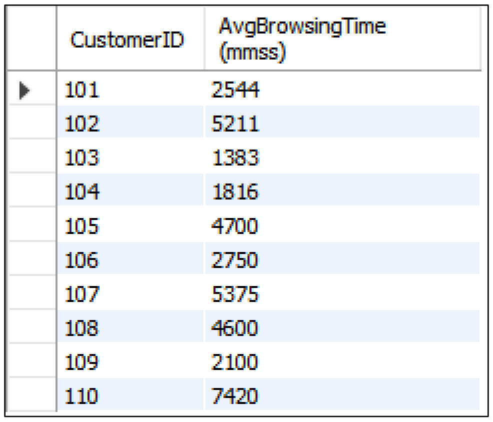
```
SELECT ProductID, ProductName, RetailPrice
FROM Products
ORDER BY RetailPrice ASC
LIMIT 5;
```



ProductID	ProductName	RetailPrice
2010	symbiote	63.00
2012	LED lights	68.00
2008	Pixel 6	70.00
2013	Sorcerers stone	79.00
2011	Amplifier	99.00

3. **Name:** Average Customer Browsing Time
    - **Team Member(s):** Connor
    - **Business Case:** Displays the average app browsing time by customer. This will assist employees in optimizing the app experience to be as user friendly as possible for customers.

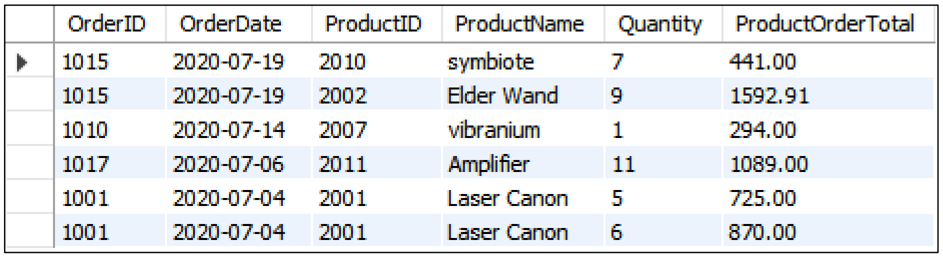
```
SELECT CustomerID, CAST(AVG(BrowsingTime) AS CHAR (4)) AS 'AvgBrowsingTime
(mmss)'
FROM
(SELECT CustomerID, TIMEDIFF(AppCloseDate,AppOpenDate) AS 'BrowsingTime' FROM
Application_Visits) sub
GROUP BY CustomerID;
```



CustomerID	AvgBrowsingTime (mmss)
101	2544
102	5211
103	1383
104	1816
105	4700
106	2750
107	5375
108	4600
109	2100
110	7420

4. **Name:** All Products Ordered by a Customer
    - **Team Member(s):** Connor
    - **Business Case:** Displays information on the products ordered by a particular customer. The customer of interest can be adjusted by changing the customer ID is the query. This will assist employees in knowing what products a repeat customer is usually interested in, which can be helpful to understand when developing promotions.

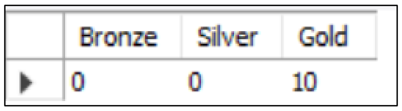
```
SELECT OrderID, OrderDate, ProductID, ProductName, Quantity, RetailPrice*Quantity AS
'ProductOrderTotal'
FROM Orders
INNER JOIN OrderDetails USING(OrderID)
INNER JOIN Products USING(ProductID)
WHERE CustomerID = 101
ORDER BY OrderDate DESC;
```



	OrderID	OrderDate	ProductID	ProductName	Quantity	ProductOrderTotal
▶	1015	2020-07-19	2010	symbiote	7	441.00
	1015	2020-07-19	2002	Elder Wand	9	1592.91
	1010	2020-07-14	2007	vibranium	1	294.00
	1017	2020-07-06	2011	Amplifier	11	1089.00
	1001	2020-07-04	2001	Laser Canon	5	725.00
	1001	2020-07-04	2001	Laser Canon	6	870.00

5. **Name:** Loyalty Status Count
    - **Team Member(s):** Connor
    - **Business Case:** Displays the count of status members by tier. Useful for employees when making updates/changes to the loyalty program.

```
SELECT
(SELECT COUNT(LoyaltyRewardID) FROM Customers WHERE LoyaltyRewardID = 11) AS
'Bronze',
(SELECT COUNT(LoyaltyRewardID) FROM Customers WHERE LoyaltyRewardID = 12) AS
'Silver',
(SELECT COUNT(LoyaltyRewardID) FROM Customers WHERE LoyaltyRewardID = 13) AS
'Gold';
```

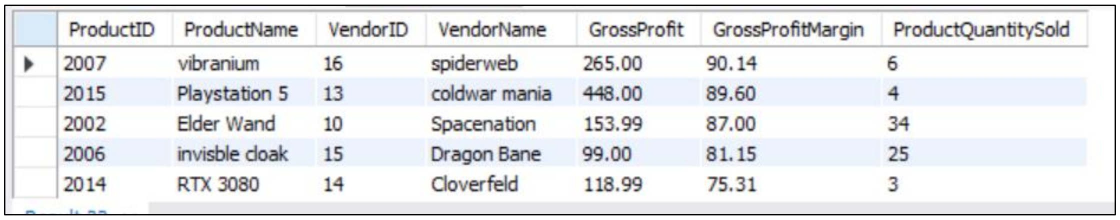


	Bronze	Silver	Gold
▶	0	0	10

6. **Name:** Top 5 Products by Margin vs Quantity Sold
    - **Team Member(s):** Catherine
    - **Business Case:** Pull the top 5 products by margin to compare to quantity sold. This can be also pulled for all products, bottom 5 products, etc. By grouping by ProductID, only the vendor providing the current largest margin/profit for that product is selected, as this is likely the vendor that management will want to order from. This view provides insight into pricing and customer purchase trends. For example, if a product has a very large margin but a very low volume of products sold, or vice versa, management may want to reconsider pricing strategies for that product.

This query would reflect ProductQuantitySold all-time:

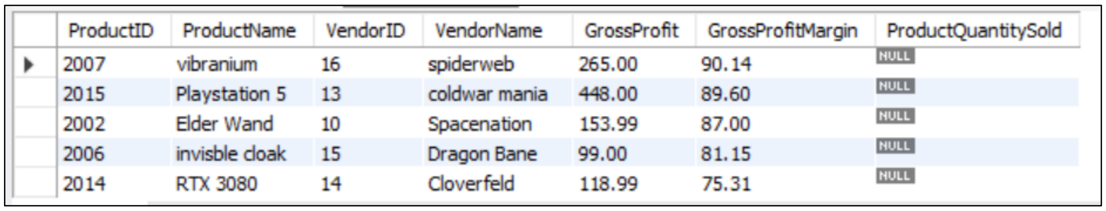
```
SELECT DISTINCT M.ProductID, M.ProductName, M.VendorID, M.VendorName,
M.GrossProfit, M.GrossProfitMargin,
(SELECT SUM(D.Quantity) FROM OrderDetails D INNER JOIN Orders O ON O.OrderID =
D.OrderID WHERE D.ProductID = M.ProductID) AS 'ProductQuantitySold'
FROM Current_Vendor_Margins M
GROUP BY ProductID
ORDER BY GrossProfitMargin desc
LIMIT 5;
```



ProductID	ProductName	VendorID	VendorName	GrossProfit	GrossProfitMargin	ProductQuantitySold
2007	vibranium	16	spiderweb	265.00	90.14	6
2015	Playstation 5	13	coldwar mania	448.00	89.60	4
2002	Elder Wand	10	Spacenation	153.99	87.00	34
2006	invisble cloak	15	Dragon Bane	99.00	81.15	25
2014	RTX 3080	14	Cloverfeld	118.99	75.31	3

This query would reflect ProductQuantitySold within the past 30 days (timeframe is adjustable):

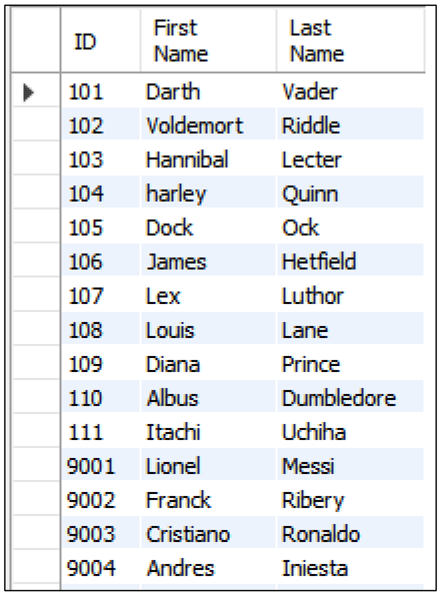
```
SELECT DISTINCT M.ProductID, M.ProductName, M.VendorID, M.VendorName,
M.GrossProfit, M.GrossProfitMargin,
(SELECT SUM(D.Quantity) FROM OrderDetails D INNER JOIN Orders O ON O.OrderID =
D.OrderID WHERE D.ProductID = M.ProductID AND O.OrderDate >= current_date()-30)
AS 'ProductQuantitySold'
FROM Current_Vendor_Margins M
GROUP BY ProductID
ORDER BY GrossProfitMargin desc
LIMIT 5;
```



ProductID	ProductName	VendorID	VendorName	GrossProfit	GrossProfitMargin	ProductQuantitySold
2007	vibranium	16	spiderweb	265.00	90.14	NULL
2015	Playstation 5	13	coldwar mania	448.00	89.60	NULL
2002	Elder Wand	10	Spacenation	153.99	87.00	NULL
2006	invisble cloak	15	Dragon Bane	99.00	81.15	NULL
2014	RTX 3080	14	Cloverfeld	118.99	75.31	NULL

7. **Name:** Customer and Employee List
   - **Team Member(s):** Connor
   - **Business Case:** Displays the ID, first name, and last name of all customers and employees. Useful as a quick reference for management on what the current customer and employee IDs are.

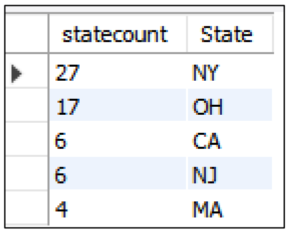
```
SELECT CustomerID AS 'ID', CustFirstName AS 'First Name', CustLastName AS 'Last
Name'
FROM Customers
UNION
SELECT EmployeeID, EmployeeFirstName, EmployeeLastName
FROM Employees;
```



ID	First Name	Last Name
101	Darth	Vader
102	Voldemort	Riddle
103	Hannibal	Lecter
104	harley	Quinn
105	Dock	Ock
106	James	Hetfield
107	Lex	Luthor
108	Louis	Lane
109	Diana	Prince
110	Albus	Dumbledore
111	Itachi	Uchiha
9001	Lionel	Messi
9002	Franck	Ribery
9003	Cristiano	Ronaldo
9004	Andres	Iniesta

8. **Name:** States by visitors
   - **Team Member(s):** Akshay
   - **Business Case:** This query gives the number of visitors by each state

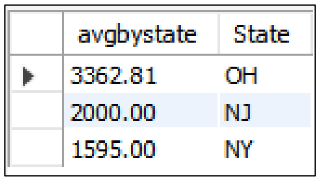
```
select count(a.CustomerID) as statecount, c.State from
(application_visits a inner join customers_address c on a.CustomerID = c.CustomerID)
group by c.State
order by
statecount desc;
```



statecount	State
27	NY
17	OH
6	CA
6	NJ
4	MA

9. **Name:** States whose total order amount is greater than the average order amount.
    - **Team Member(s):** Akshay
    - **Business Case:** This query gives the list of states whose total order amount is greater than the average order amount

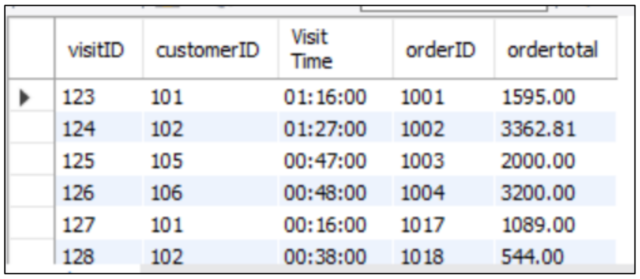
```
select o.OrderTotal as avgbystate, c.State from
(orders o inner join customers_address c on o.CustomerID = c.CustomerID)
where OrderTotal > (select avg(OrderTotal) from orders)
group by State
order by
        avgbystate desc;
```



avgbystate	State
3362.81	OH
2000.00	NJ
1595.00	NY

10. **Name:** Application Visits vs Orders
    - **Team Member(s):** Catherine
    - **Business Case:** This query shows all visits. If an order was placed during that visit, that order detail will be included. Analysis can be done to compare things such as time spent during that visit and if an order was placed. For example, do customers who browse longer end up making a purchase?

```
select v.visitID, v.customerID, TIMEDIFF(appclosedate,appopendate) as 'Visit Time',
o.orderID, o.ordertotal
from application_visits v
left outer join orders o on v.visitid = o.visitid
order by 'Visit Time' desc;
```

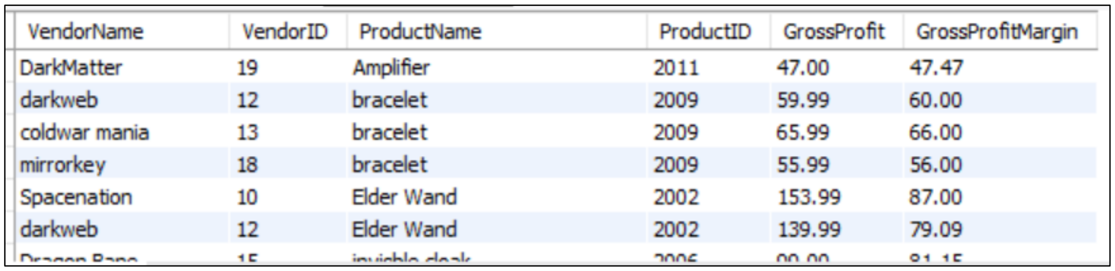


visitID	customerID	Visit Time	orderID	ordertotal
123	101	01:16:00	1001	1595.00
124	102	01:27:00	1002	3362.81
125	105	00:47:00	1003	2000.00
126	106	00:48:00	1004	3200.00
127	101	00:16:00	1017	1089.00
128	102	00:38:00	1018	544.00

Views

1. **Name:** Current_Vendor_Margins
   - **Team Member(s):** Catherine
   - **Business Case:** Provide an easy-access view for individuals responsible for re-ordering stock from vendors or setting Retail Price. For example, once Product_Order_Status = ORDER, this view will assist employees in reviewing from which vendor they should re-order low stock products from, based on reviewing product margin.

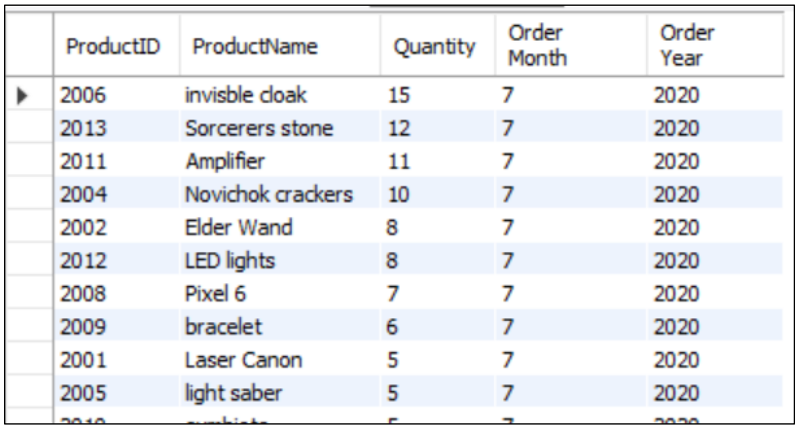
```
CREATE VIEW Current_Vendor_Margins AS
SELECT V.VendorName, V.VendorID, P.ProductName, P.ProductID, (P.RetailPrice -
R.VendorPrice) AS 'GrossProfit', Round((((P.RetailPrice - R.VendorPrice)/P.RetailPrice)*100),2)
AS 'GrossProfitMargin'
FROM Vendors V
INNER JOIN Vendor_Price R ON V.VendorID = R.VendorID
INNER JOIN Products P ON P.ProductID = R.ProductID
ORDER BY PRODUCTNAME ASC, 'GrossProfit' DESC;
```



VendorName	VendorID	ProductName	ProductID	GrossProfit	GrossProfitMargin
DarkMatter	19	Amplifier	2011	47.00	47.47
darkweb	12	bracelet	2009	59.99	60.00
coldwar mania	13	bracelet	2009	65.99	66.00
mirrorkey	18	bracelet	2009	55.99	56.00
Spacenation	10	Elder Wand	2002	153.99	87.00
darkweb	12	Elder Wand	2002	139.99	79.09
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

2. **Name:** Products_Sold_Trending
   - **Team Member(s):** Catherine
   - **Business Case:** Analyze trending of quantity of products sold each month/year. Easily see if a product has recently increased or decreased in popularity.

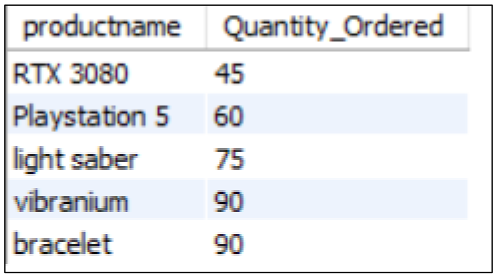
```
CREATE VIEW Products_Sold_Trending AS
SELECT D.ProductID, P.ProductName, D.Quantity, Month(O.OrderDate) as "Order
Month", Year(O.OrderDate) as "Order Year"
FROM Products P, Orderdetails D, Orders O
WHERE D.ProductID = P.ProductID
AND D.OrderID = O.OrderID
GROUP BY D.ProductID, P.ProductName, Month(O.OrderDate), Year(O.OrderDate)
ORDER BY Quantity DESC;
```



ProductID	ProductName	Quantity	Order Month	Order Year
2006	invisble cloak	15	7	2020
2013	Sorcerers stone	12	7	2020
2011	Amplifier	11	7	2020
2004	Novichok crackers	10	7	2020
2002	Elder Wand	8	7	2020
2012	LED lights	8	7	2020
2008	Pixel 6	7	7	2020
2009	bracelet	6	7	2020
2001	Laser Canon	5	7	2020
2005	light saber	5	7	2020

3. **Name:** Least Bought Products
   - **Team Member(s):** Neha
   - **Business Case:** To know the least bought products by customers

```
CREATE VIEW Least_Bought AS
select p.productname, sum(quantity) as Quantity_Ordered
from orderdetails as od,products as p inner join products
where p.productid = od.productid
group by productname
order by sum(quantity) limit 5 ;
```



productname	Quantity_Ordered
RTX 3080	45
Playstation 5	60
light saber	75
vibranium	90
bracelet	90

Triggers

1. **Name:** Total_Customer_Orders
   - **Team Member(s):** Connor, Catherine
   - **Business Case:** Our database supports tracking a loyalty rewards program. After a customer makes a new order, the Customers table will track the customer's total order $ volume.

```
DELIMITER $$
CREATE TRIGGER total_customer_orders
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
  UPDATE customers c
  SET c.totalcustorders = (select sum(o.ordertotal)
  FROM orders o
  WHERE c.customerid = o.customerid);
END$$
DELIMITER ;
```

2. **Name:** Loyalty_Status_Update
   - **Team Member(s):** Connor, Catherine
   - **Business Case:** As the customer orders and their TotalCustOrder value increases, their loyalty status will be automatically updated.

```
CREATE TRIGGER LOYALTY_STATUS_UPDATE
AFTER INSERT on ORDERS
FOR EACH ROW
UPDATE CUSTOMERS
SET LOYALTYREWARDID =
(SELECT
  CASE
    WHEN TOTALCUSTORDERS BETWEEN 250.00 AND 499.99 THEN '11'
    WHEN TOTALCUSTORDERS BETWEEN 500.00 AND 999.99 THEN '12'
    WHEN TOTALCUSTORDERS >= 1000 THEN '13'
    ELSE null END);
```

3. **Name:** Archive _Vendor_Price
   - **Team Member(s):** Connor
   - **Business Case:** If a vendor stops selling a product, archive. It is no longer a valid vendor to reorder from.

```
CREATE TRIGGER ARCHIVE_VENDOR_PRICE
BEFORE DELETE ON Vendor_Price
FOR EACH ROW
INSERT INTO Vendor_Price_Archive
Set ProductID = Old.ProductID,
VendorID = Old.VendorID,
VendorPrice = Old.VendorPrice,
SaleStopDate = now();
```

4. **Name:** Archive _Vendor_PriceChange
    - **Team Member(s):** Connor
    - **Business Case:** If a vendor stops selling a product at a specific price, archive.

```
CREATE TRIGGER ARCHIVE_VENDOR_PRICECHANGE
BEFORE UPDATE ON Vendor_Price
FOR EACH ROW
INSERT INTO Vendor_Price_Archive
Set ProductID = Old.ProductID,
VendorID = Old.VendorID,
VendorPrice = Old.VendorPrice,
SaleStopDate = now();
```

5. **Name:** Low Stock
    - **Team Member(s):** Connor
    - **Business Case:** If current product quantity drops below specified threshold, update product_order_status accordingly. Department responsible for ordering will review products needing reordering.

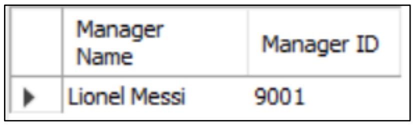
```
CREATE TRIGGER LOW_STOCK
AFTER UPDATE ON PRODUCTS
FOR EACH ROW
UPDATE PRODUCTS
SET PRODUCT_ORDER_STATUS =
   CASE
     WHEN CURRENTVOLUME < 15 THEN 'ORDER'
     WHEN CURRENTVOLUME BETWEEN 16 AND 30 THEN 'REVIEW'
     ELSE 'NO ORDER NEEDED' END;
```

Stored Procedures

1. **Name:** Get_Manager_Name
   - **Team Member(s):** Catherine
   - **Business Case:** Easily retrieve manager name for the employee ID in question

```
DELIMITER //
CREATE PROCEDURE Manager_Name
(employee_id_param INT)
BEGIN
select CONCAT(M.employeefirstname,' ',M.employeelastname) as "Manager Name",
m.employeeid as "Manager ID"
from Employees m
RIGHT OUTER JOIN Employees e
ON e.managerid = m.employeeid
WHERE e.employeeID = employee_id_param;
END//
```

Example: call manager_name(9003);

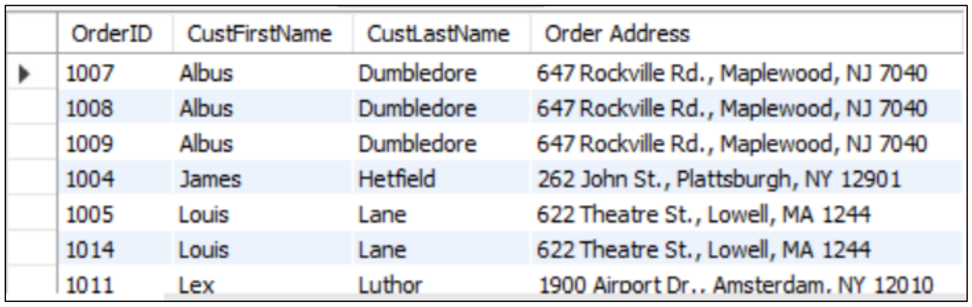


Manager Name	Manager ID
Lionel Messi	9001

2. **Name:** Order Address
   - **Team Member(s):** Catherine
   - **Business Case:** If a customer has multiple addresses over time, this can easily pull history of customer address at time of ordering.

```
DELIMITER //
CREATE PROCEDURE order_address ()
BEGIN
  SELECT OrderID, CustFirstName, CustLastName, concat(streetaddress,', ',city,', ',state,'
',zip) as 'Order Address'
  from customers_address c
  inner join orders o
  on o.customerid = c.customerid
  inner join customers t
  on t.customerid = o.customerid
  where fromdate <= orderdate
  and todate >= orderdate
  order by custlastname;
END //
```

Example: Call order_address ();

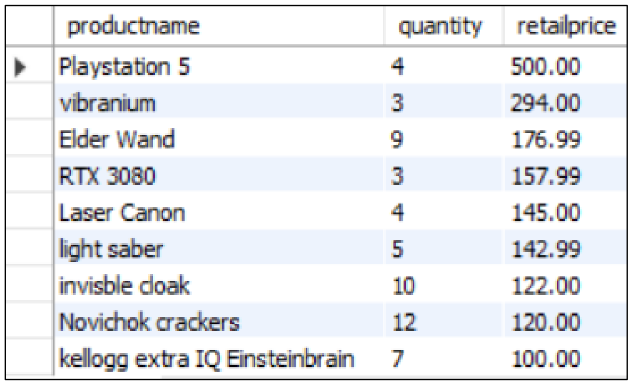


	OrderID	CustFirstName	CustLastName	Order Address
▶	1007	Albus	Dumbledore	647 Rockville Rd., Maplewood, NJ 7040
	1008	Albus	Dumbledore	647 Rockville Rd., Maplewood, NJ 7040
	1009	Albus	Dumbledore	647 Rockville Rd., Maplewood, NJ 7040
	1004	James	Hetfield	262 John St., Plattsburgh, NY 12901
	1005	Louis	Lane	622 Theatre St., Lowell, MA 1244
	1014	Louis	Lane	622 Theatre St., Lowell, MA 1244
	1011	Lex	Luthor	1900 Airport Dr., Amsterdam, NY 12010

3. **Name:** Top ten products by price
   - **Team Member(s):** Neha
   - **Business Case:** If a customer wants to see the top highest priced products and the number of quantities bought.

DELIMITER //
CREATE PROCEDURE `topten_price`()
BEGIN
select p.productname,quantity,p.retailprice from orderdetails as od,products as p inner join products
where p.productid = od.productid group by retailprice
order by retailprice desc limit 10 ;
END //

Example: call topten_price();



	productname	quantity	retailprice
▶	Playstation 5	4	500.00
	vibranium	3	294.00
	Elder Wand	9	176.99
	RTX 3080	3	157.99
	Laser Canon	4	145.00
	light saber	5	142.99
	invisble cloak	10	122.00
	Novichok crackers	12	120.00
	kellogg extra IQ Einsteinbrain	7	100.00

4. **Name:** Updating Product Price
   - **Team Member(s):** Neha
   - **Business Case:** Updating price of a product

```
DELIMITER //
CREATE PROCEDURE `product_update`(
  in p_id bigint,in p_price decimal(9,2))
BEGIN
        UPDATE products SET retailprice = p_price  WHERE productid = p_id;
END //
```

Example: call product_update (2001,200);

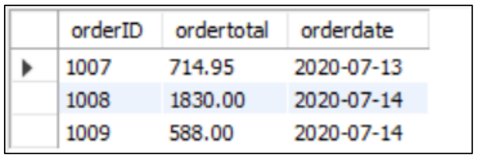
Stored Functions

1. **Name:** Get_Customer_ID
   - **Team Member(s):** Catherine
   - **Business Case:** Should a customer reach out via e-mail but not provide additional information, the customer's information, order history, etc. can be easily accessed.

```
DELIMITER //
CREATE FUNCTION get_customer_ID
(emailaddress_param VARCHAR(65))
RETURNS INT
DETERMINISTIC
BEGIN
DECLARE customer_id_Var INT;
SELECT customerID INTO customer_id_var FROM Customers
WHERE emailaddress_param = emailid;
RETURN(Customer_id_var);
END//
DELIMITER ;
```

Example Query and Results; Pull Order history for albusthewise@gmail.com

```
SELECT orderID, ordertotal, orderdate
FROM orders
WHERE customerid = get_customer_id('albusthewise@gmail.com');
```



orderID	ordertotal	orderdate
1007	714.95	2020-07-13
1008	1830.00	2020-07-14
1009	588.00	2020-07-14

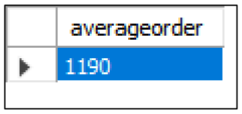
2. **Name:** avgorders

- **Team Member(s): Akshay**
- **Business Case:** A function that returns the average order amount of a customer given a customerID

```
DELIMITER //
CREATE FUNCTION avgorders
(
custid int
)
returns decimal
READS SQL DATA
not DETERMINISTIC
begin
declare avgorders decimal;
select avg(OrderTotal) into avgorders from orders where CustomerID = custid;
return avgorders;
END //
```

Example Query and Results; Pull average total for customer whose ID is 104

```
select avgorders(104) as averageorder from orders
group by
averageorder;
```



```
averageorder
1190
```

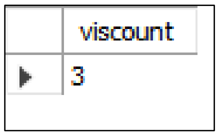
3. **Name:** viscount

- **Team Member(s): Akshay**
- **Business Case:** A function that returns the numbers of visitors for a given date

```
DELIMITER //
CREATE FUNCTION viscount
(
datevalue date
)
returns int
READS SQL DATA
DETERMINISTIC
begin
declare days int;
select count(CustomerID)  into days  from application_visits
where date(appopendate) = datevalue;
return days;
END //
```

Example Query and Results; Pull Number of customers who visited on 2020-07-06

```
select viscount('2020-07-06') as viscount from orders
group by
viscount;
```



	viscount
▶	3