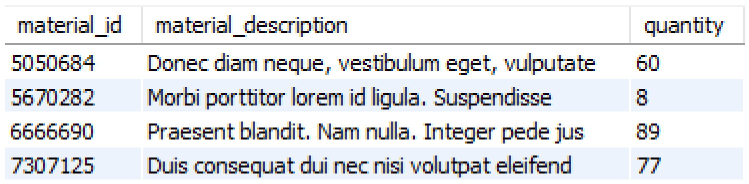
COMMENTS

This shows the inventory for our products where the product and their corresponding materials inventories are both less than 100. This will be our indication to put in an order for more materials to keep up our inventory.

QUERY 1

```
SELECT  mi.product_item_id AS material_id, pbom.product_description AS material_description,
mi.quantity
FROM ProjectGroup5.manufacturer_inventory mi
        INNER JOIN ProjectGroup5.product_bom pbom
                ON mi.product_item_id = pbom.product_item_id
WHERE mi.quantity <= 100
        AND pbom.quantity <=100;
```

RESULTS



material_id	material_description	quantity
5050684	Donec diam neque, vestibulum eget, vulputate	60
5670282	Morbi porttitor lorem id ligula. Suspendisse	8
6666690	Praesent blandit. Nam nulla. Integer pede jus	89
7307125	Duis consequat dui nec nisi volutpat eleifend	77

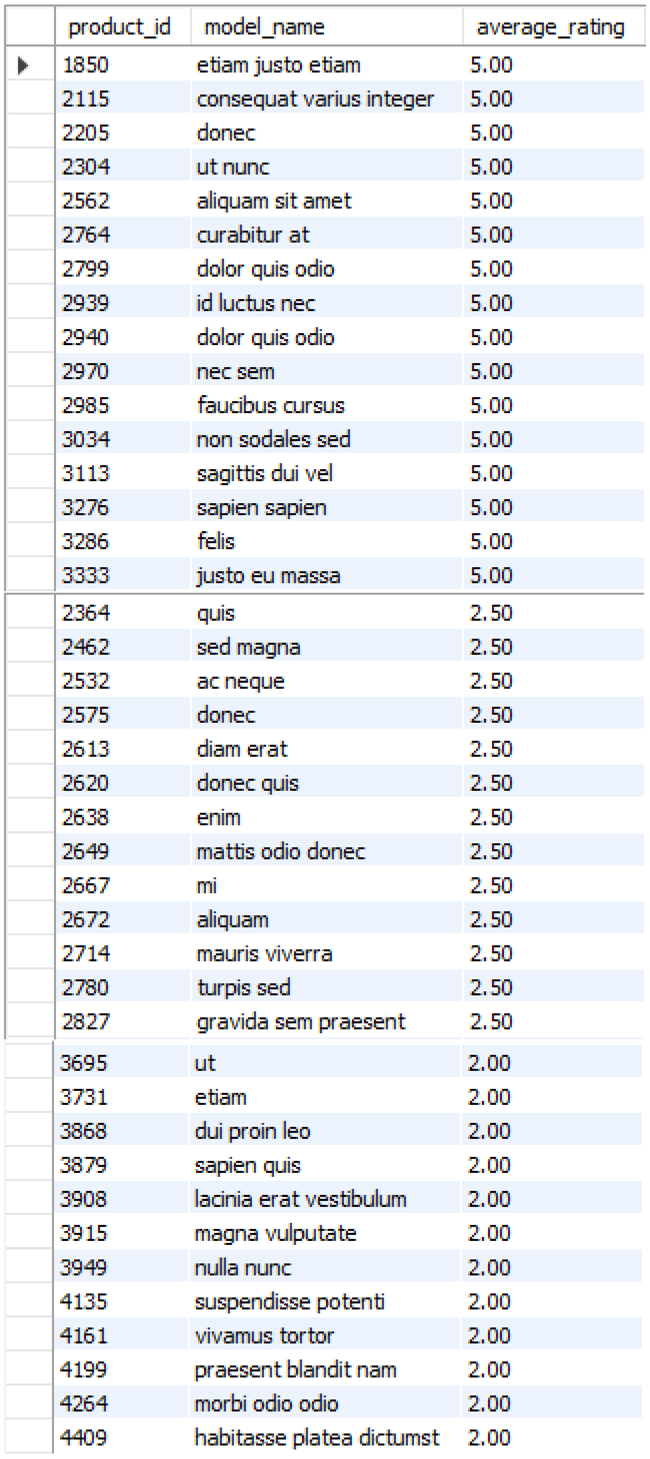
COMMENTS

This query shows the products with lowest ratings (1-2.5) and with highest ratings (5) for the last 30 days. This will aide in determining which products are our most popular and which are our least, so we can invest in the right materials.

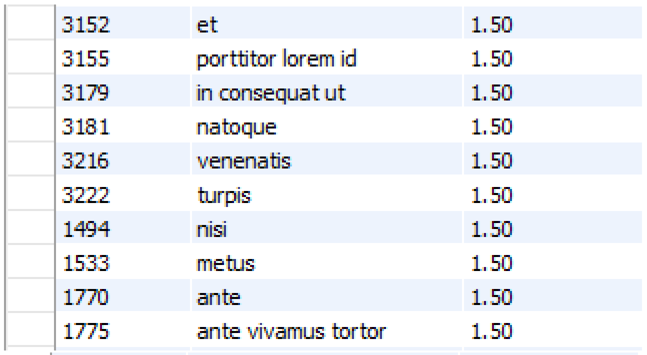
QUERY 2

```
SELECT  cr.product_id,
        p.model_name,
        ROUND (AVG(cr.rating_score),2) AS average_rating
FROM customer_ratings cr
        RIGHT JOIN products p
                ON cr.product_id = p.product_id
WHERE cr.review_date between (curdate() - interval 30 day) AND curdate()
GROUP BY 1
HAVING average_rating Between (1) AND  (2.9)
        OR average_rating = 5
ORDER BY average_rating desc;
```

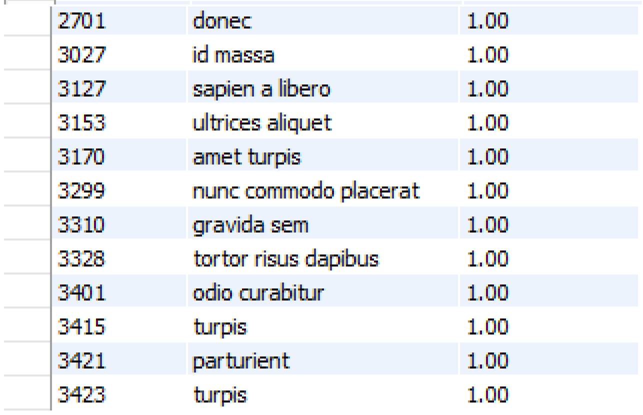
RESULTS (269 ROWS RETURNED, SNIPPITS SHOWN BELOW)



product_id	model_name	average_rating
1850	etiam justo etiam	5.00
2115	consequat varius integer	5.00
2205	donec	5.00
2304	ut nunc	5.00
2562	aliquam sit amet	5.00
2764	curabitur at	5.00
2799	dolor quis odio	5.00
2939	id luctus nec	5.00
2940	dolor quis odio	5.00
2970	nec sem	5.00
2985	faucibus cursus	5.00
3034	non sodales sed	5.00
3113	sagittis dui vel	5.00
3276	sapien sapien	5.00
3286	felis	5.00
3333	justo eu massa	5.00
2364	quis	2.50
2462	sed magna	2.50
2532	ac neque	2.50
2575	donec	2.50
2613	diam erat	2.50
2620	donec quis	2.50
2638	enim	2.50
2649	mattis odio donec	2.50
2667	mi	2.50
2672	aliquam	2.50
2714	mauris viverra	2.50
2780	turpis sed	2.50
2827	gravida sem praesent	2.50
3695	ut	2.00
3731	etiam	2.00
3868	dui proin leo	2.00
3879	sapien quis	2.00
3908	lacinia erat vestibulum	2.00
3915	magna vulputate	2.00
3949	nulla nunc	2.00
4135	suspendisse potenti	2.00
4161	vivamus tortor	2.00
4199	praesent blandit nam	2.00
4264	morbi odio odio	2.00
4409	habitasse platea dictumst	2.00



3152	et	1.50
3155	porttitor lorem id	1.50
3179	in consequat ut	1.50
3181	natoque	1.50
3216	venenatis	1.50
3222	turpis	1.50
1494	nisi	1.50
1533	metus	1.50
1770	ante	1.50
1775	ante vivamus tortor	1.50



2701	donec	1.00
3027	id massa	1.00
3127	sapien a libero	1.00
3153	ultrices aliquet	1.00
3170	amet turpis	1.00
3299	nunc commodo placerat	1.00
3310	gravida sem	1.00
3328	tortor risus dapibus	1.00
3401	odio curabitur	1.00
3415	turpis	1.00
3421	parturient	1.00
3423	turpis	1.00

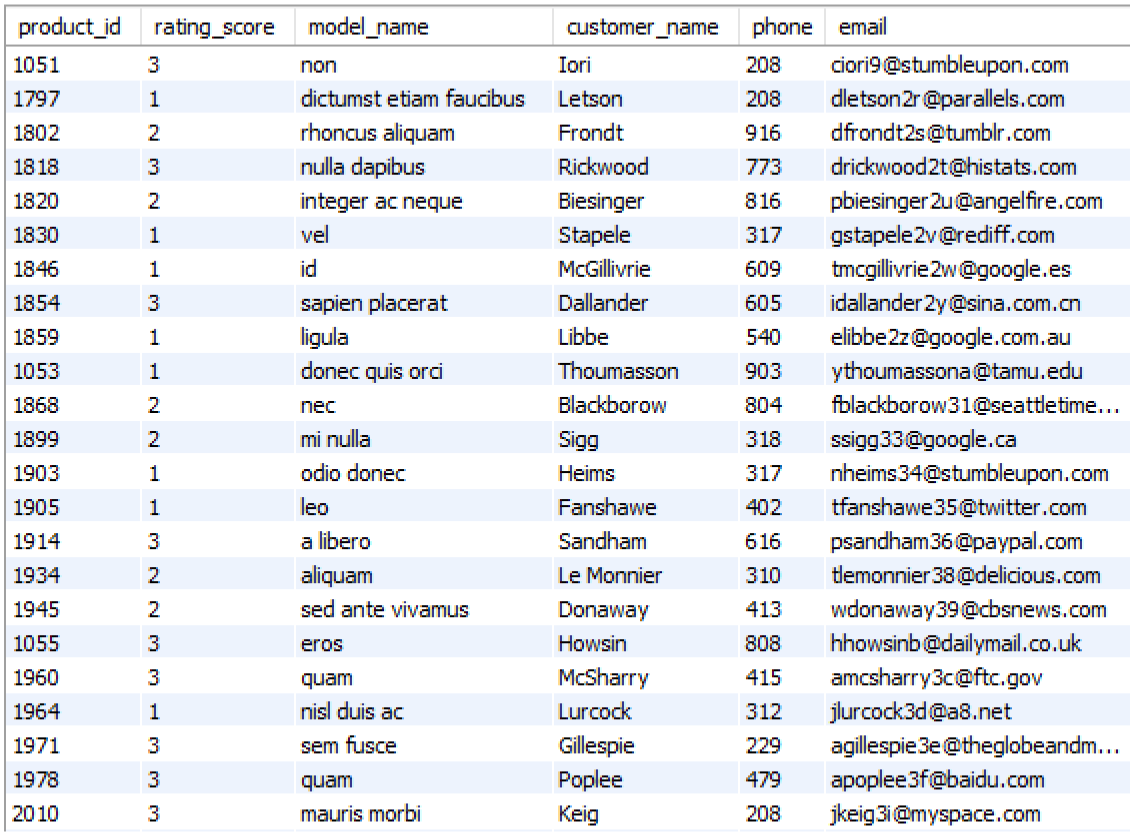
COMMENTS

This query will pull the ratings per customer per product for feedback (VOC). Customer data is pulled to initiate a contact with the customer to dive into how we can improve our products further as a part of our company customer outreach program

QUERY 3

```
SELECT cr.product_id, cr.rating_score, p.model_name, c.customer_name, c.phone, c.email
FROM ProjectGroup5.customer_ratings cr
        JOIN ProjectGroup5.customers c
                ON cr.customer_id = c.customer_id
        RIGHT JOIN ProjectGroup5.products p
                ON cr.product_id = p.product_id
WHERE cr.rating_score <= 3;
```

RESULTS (594 ROWS RETURNED, SNIPPIT SHOWN)



product_id	rating_score	model_name	customer_name	phone	email
1051	3	non	Iori	208	ciori9@stumbleupon.com
1797	1	dictumst etiam faucibus	Letson	208	dletson2r@parallels.com
1802	2	rhoncus aliquam	Frondt	916	dfrondt2s@tumblr.com
1818	3	nulla dapibus	Rickwood	773	drickwood2t@histats.com
1820	2	integer ac neque	Biesinger	816	pbiesinger2u@angelfire.com
1830	1	vel	Stapele	317	gstapele2v@rediff.com
1846	1	id	McGillivrie	609	tmcgillivrie2w@google.es
1854	3	sapien placerat	Dallander	605	idallander2y@sina.com.cn
1859	1	ligula	Libbe	540	elibbe2z@google.com.au
1053	1	donec quis orci	Thoumasson	903	ythoumassona@tamu.edu
1868	2	nec	Blackborow	804	fblackborow31@seattletime...
1899	2	mi nulla	Sigg	318	ssigg33@google.ca
1903	1	odio donec	Heims	317	nheims34@stumbleupon.com
1905	1	leo	Fanshawe	402	tfanshawe35@twitter.com
1914	3	a libero	Sandham	616	psandham36@paypal.com
1934	2	aliquam	Le Monnier	310	tlemonnier38@delicious.com
1945	2	sed ante vivamus	Donaway	413	wdonaway39@cbsnews.com
1055	3	eros	Howsin	808	hhowsinb@dailymail.co.uk
1960	3	quam	McSharry	415	amcsharry3c@ftc.gov
1964	1	nisl duis ac	Lurcock	312	jlurcock3d@a8.net
1971	3	sem fusce	Gillespie	229	agillespie3e@theglobeandm...
1978	3	quam	Poplee	479	apoplee3f@baidu.com
2010	3	mauris morbi	Keig	208	jkeig3i@myspace.com

COMMENTS

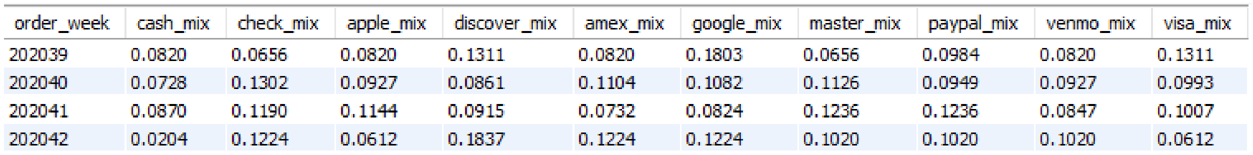
This query will pull the payment methods by week to determine which method is most used. This is insightful for us as we can invest more resources into making faster check out times for higher used payment methods and also look into how much each method is cost our business.

QUERY 4

```
with totals as (
SELECT yearweek(co.order_date_local, 4) as order_week
    , sum(case when pm.payment_method = 'Cash' then 1 else 0 end) as cash_count
    , sum(case when pm.payment_method = 'Check' then 1 else 0 end) as check_count
    , sum(case when pm.payment_method = 'Apple Pay' then 1 else 0 end) as apple_count
    , sum(case when pm.payment_method = 'Discover' then 1 else 0 end) as discover_count
    , sum(case when pm.payment_method = 'American Express' then 1 else 0 end) as amex_count
    , sum(case when pm.payment_method = 'Google Pay' then 1 else 0 end) as google_count
    , sum(case when pm.payment_method = 'Master Card' then 1 else 0 end) as master_count
    , sum(case when pm.payment_method = 'Paypal' then 1 else 0 end) as paypal_count
    , sum(case when pm.payment_method = 'Venmo' then 1 else 0 end) as venmo_count
    , sum(case when pm.payment_method = 'Visa' then 1 else 0 end) as visa_count
    , count(*) as total_count
FROM customer_orders co
            inner join payment_methods pm on co.payment_id = pm.payment_id
GROUP BY 1
)
```

```
SELECT order_week
    , cash_count / total_count as cash_mix
    , check_count / total_count as check_mix
    , apple_count / total_count as apple_mix
    , discover_count / total_count as discover_mix
    , amex_count / total_count as amex_mix
    , google_count / total_count as google_mix
    , master_count / total_count as master_mix
    , paypal_count / total_count as paypal_mix
    , venmo_count / total_count as venmo_mix
    , visa_count / total_count as visa_mix
FROM totals
ORDER BY 1;
```

RESULTS



order_week	cash_mix	check_mix	apple_mix	discover_mix	amex_mix	google_mix	master_mix	paypal_mix	venmo_mix	visa_mix
202039	0.0820	0.0656	0.0820	0.1311	0.0820	0.1803	0.0656	0.0984	0.0820	0.1311
202040	0.0728	0.1302	0.0927	0.0861	0.1104	0.1082	0.1126	0.0949	0.0927	0.0993
202041	0.0870	0.1190	0.1144	0.0915	0.0732	0.0824	0.1236	0.1236	0.0847	0.1007
202042	0.0204	0.1224	0.0612	0.1837	0.1224	0.1224	0.1020	0.1020	0.1020	0.0612

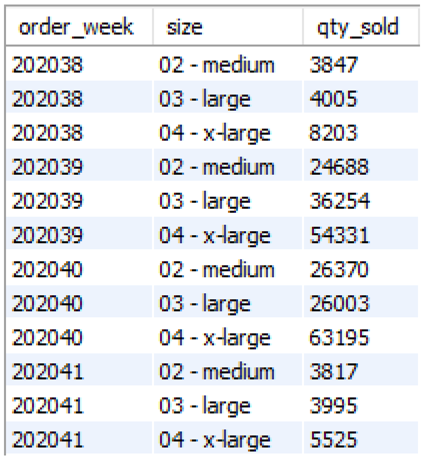
COMMENTS

This query shows which backpack sizes are our most popular. This type of data is insightful for determining which products we want to add to our store's selection and what sizes to offer for those products. We can also track trends on existing products to determine resource management.

QUERY 5

```
SELECT yearweek(co.order_date_local) as order_week
          , case when p.capacity_liters < 10.0 then '01 - small'
                  when p.capacity_liters < 40 then '02 - medium'
                  when p.capacity_liters < 75 then '03 - large'
                  else '04 - x-large' end as size
          , sum(cod.quantity) as qty_sold
FROM customer_orders co
          inner join customer_order_details cod on co.order_id = cod.order_id
          inner join products p on cod.product_id = p.product_id
GROUP BY 1,2
ORDER BY 1,2;
```

RESULTS



order_week	size	qty_sold
202038	02 - medium	3847
202038	03 - large	4005
202038	04 - x-large	8203
202039	02 - medium	24688
202039	03 - large	36254
202039	04 - x-large	54331
202040	02 - medium	26370
202040	03 - large	26003
202040	04 - x-large	63195
202041	02 - medium	3817
202041	03 - large	3995
202041	04 - x-large	5525

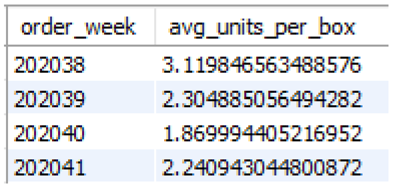
COMMENTS

This query will pull the average items per box by week. This will give us insights into products per customer order and the efficiency of our shipment processes.

QUERY 6

```
SELECT yearweek(co.order_date_local) as order_week
        , avg(cast(cod.quantity as double) / co.box_count) as avg_units_per_box
FROM customer_orders co
        inner join customer_order_details cod on co.order_id = cod.order_id
GROUP BY 1
ORDER BY 1;
```

RESULTS



order_week	avg_units_per_box
202038	3.119846563488576
202039	2.304885056494282
202040	1.869994405216952
202041	2.240943044800872

COMMENTS

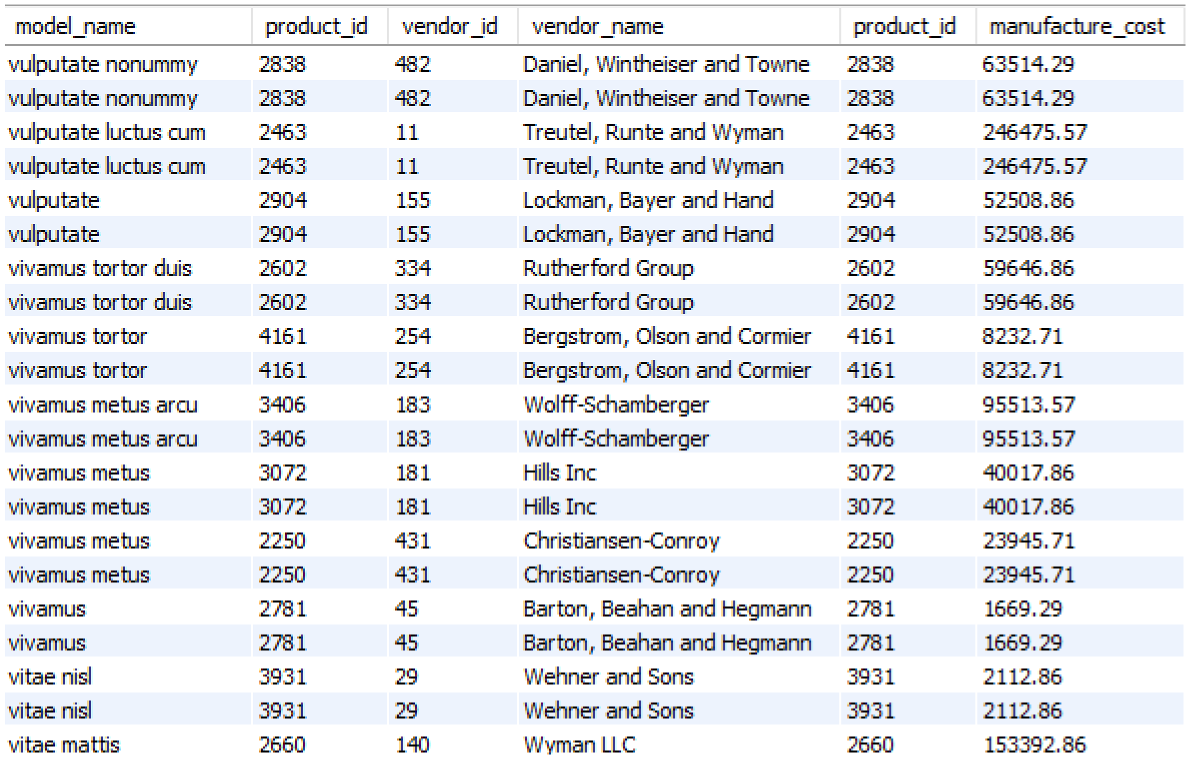
This query will show our most expensive vendor we are purchasing products. This is key to understanding which vendors we want to maintain a business relationship with.
The manufacture costs are treated as CNY and converted to USD, otherwise, we're not doing too good

QUERY 7

```
SELECT model_name,
        cod.product_id,
        vendor_id,
        vendor_name,
        cod.product_id,
        ROUND(pb.quantity * pod.purchase_price/7 ,2) as manufacture_cost
FROM customer_order_details as cod, product_bom as pb, products as p, purchase_order_details as
```

```
pod, vendors as v
WHERE (cod.product_id = pb.product_id
            AND cod.product_id = p.product_id
            AND pb.product_item_id = pod.product_item_id
            AND pb.product_item_id = v.product_item_id)
-- rank by cost
ORDER BY pb.quantity * pod.purchase_price/7 desc;
```

RESULTS (995 ROWS RETURNED, SNIPPIT SHOWN)



model_name	product_id	vendor_id	vendor_name	product_id	manufacture_cost
vulputate nonummy	2838	482	Daniel, Wintheiser and Towne	2838	63514.29
vulputate nonummy	2838	482	Daniel, Wintheiser and Towne	2838	63514.29
vulputate luctus cum	2463	11	Treutel, Runte and Wyman	2463	246475.57
vulputate luctus cum	2463	11	Treutel, Runte and Wyman	2463	246475.57
vulputate	2904	155	Lockman, Bayer and Hand	2904	52508.86
vulputate	2904	155	Lockman, Bayer and Hand	2904	52508.86
vivamus tortor duis	2602	334	Rutherford Group	2602	59646.86
vivamus tortor duis	2602	334	Rutherford Group	2602	59646.86
vivamus tortor	4161	254	Bergstrom, Olson and Cormier	4161	8232.71
vivamus tortor	4161	254	Bergstrom, Olson and Cormier	4161	8232.71
vivamus metus arcu	3406	183	Wolff-Schamberger	3406	95513.57
vivamus metus arcu	3406	183	Wolff-Schamberger	3406	95513.57
vivamus metus	3072	181	Hills Inc	3072	40017.86
vivamus metus	3072	181	Hills Inc	3072	40017.86
vivamus metus	2250	431	Christiansen-Conroy	2250	23945.71
vivamus metus	2250	431	Christiansen-Conroy	2250	23945.71
vivamus	2781	45	Barton, Beahan and Hegmann	2781	1669.29
vivamus	2781	45	Barton, Beahan and Hegmann	2781	1669.29
vitae nisl	3931	29	Wehner and Sons	3931	2112.86
vitae nisl	3931	29	Wehner and Sons	3931	2112.86
vitae mattis	2660	140	Wyman LLC	2660	153392.86

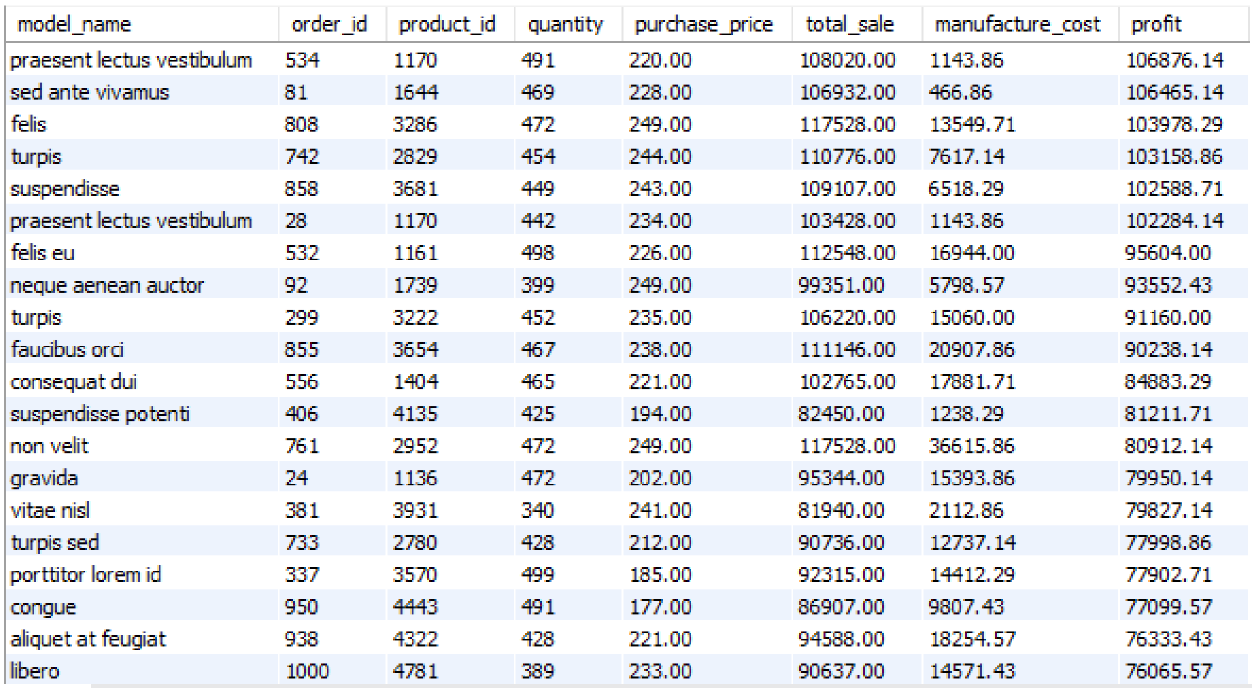
COMMENTS

This query pulls the products that we sell and shows which are most profitable., This is necessary for our business so that we can continue to invest our money and resources into the products that are performing the best at any given time.

QUERY 8

```
SELECT model_name,
          cod.order_id,
          cod.product_id,
          cod.quantity,
          cod.purchase_price,
          cod.quantity * cod.purchase_price  as total_sale,
          pb.quantity * pod.purchase_price/7  as manufacture_cost,
          cod.quantity * cod.purchase_price - pb.quantity * pod.purchase_price /7 as profit
FROM customer_order_details as cod, product_bom as pb, products as p, purchase_order_details as
pod
WHERE (cod.product_id = pb.product_id
          AND cod.product_id = p.product_id
          AND pb.product_item_id = pod.product_item_id
          AND (cod.quantity * cod.purchase_price) - (pb.quantity * pod.purchase_price /7) > 0)
ORDER BY (cod.quantity * cod.purchase_price) - (pb.quantity * pod.purchase_price /7) desc;
```

RESULTS (316 ROWS RETURNED, SNIPPIT SHOWN)



model_name	order_id	product_id	quantity	purchase_price	total_sale	manufacture_cost	profit
praesent lectus vestibulum	534	1170	491	220.00	108020.00	1143.86	106876.14
sed ante vivamus	81	1644	469	228.00	106932.00	466.86	106465.14
felis	808	3286	472	249.00	117528.00	13549.71	103978.29
turpis	742	2829	454	244.00	110776.00	7617.14	103158.86
suspendisse	858	3681	449	243.00	109107.00	6518.29	102588.71
praesent lectus vestibulum	28	1170	442	234.00	103428.00	1143.86	102284.14
felis eu	532	1161	498	226.00	112548.00	16944.00	95604.00
neque aenean auctor	92	1739	399	249.00	99351.00	5798.57	93552.43
turpis	299	3222	452	235.00	106220.00	15060.00	91160.00
faucibus orci	855	3654	467	238.00	111146.00	20907.86	90238.14
consequat dui	556	1404	465	221.00	102765.00	17881.71	84883.29
suspendisse potenti	406	4135	425	194.00	82450.00	1238.29	81211.71
non velit	761	2952	472	249.00	117528.00	36615.86	80912.14
gravida	24	1136	472	202.00	95344.00	15393.86	79950.14
vitae nisl	381	3931	340	241.00	81940.00	2112.86	79827.14
turpis sed	733	2780	428	212.00	90736.00	12737.14	77998.86
porttitor lorem id	337	3570	499	185.00	92315.00	14412.29	77902.71
congue	950	4443	491	177.00	86907.00	9807.43	77099.57
aliquet at feugiat	938	4322	428	221.00	94588.00	18254.57	76333.43
libero	1000	4781	389	233.00	90637.00	14571.43	76065.57

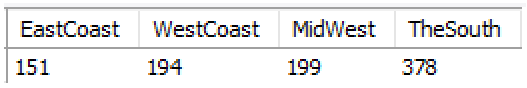
COMMENTS

This query gets our customer data across the country by region (EastCoast, WestCoast, MidWest) to see how our general customers are spread out. This also helps us determine where to set up new offices or where to prioritize our growth.

QUERY 9

```
SELECT count(EastCoast) as EastCoast,
        count(WestCoast) as WestCoast,
        count(MidWest) as MidWest,
        count(TheSouth) as TheSouth
FROM ( SELECT
                case when state in ('Rh','Ma','Co','No','So','Vi','De') then 1 end EastCoast,
                case when state in ('Or', 'Wa','Ca','Ha','Id','Wy','Ut','Al') then 1 end WestCoast,
                case when state in ('Pe', 'Il','In','Ka','Wi', 'Ne','Ar','Io') then 1 end MidWest,
                case when state in ('Ge', 'Fl','Ke','Ok','Mi','Ma','Te') then 1 end TheSouth
        FROM customers
) customers;
```

RESULTS



EastCoast	WestCoast	MidWest	TheSouth
151	194	199	378

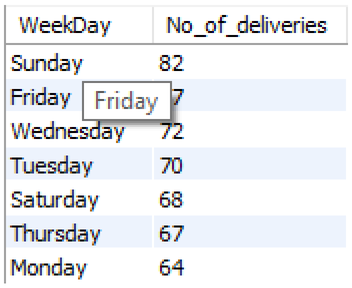
COMMENTS

This query gets the number of expected arrivals per delivery day and subsequently find our busiest day. We found the busiest day to be Sunday. This will enable us to make sure there is adequate support for our customers on that day, in case they have trouble with the shipment

QUERY 10

```
SELECT  DAYNAME(arrival_date_local) as WeekDay, COUNT(*) as No_of_deliveries
FROM purchase_orders
GROUP BY DAYNAME(arrival_date_local)
ORDER BY COUNT(*) DESC;
```

RESULTS



WeekDay	No_of_deliveries
Sunday	82
Friday	7
Wednesday	72
Tuesday	70
Saturday	68
Thursday	67
Monday	64

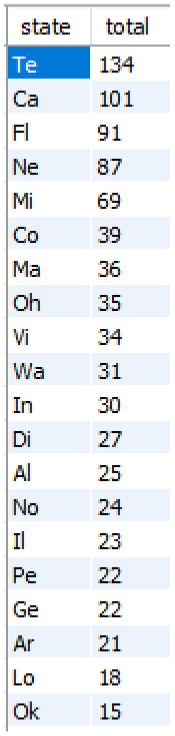
COMMENTS

This query will pull our customer distribution by state. This will show us where we need to invest more advertising as well as show us where to invest more resources to manage coverage.

STORED PROCEDURE QUERY

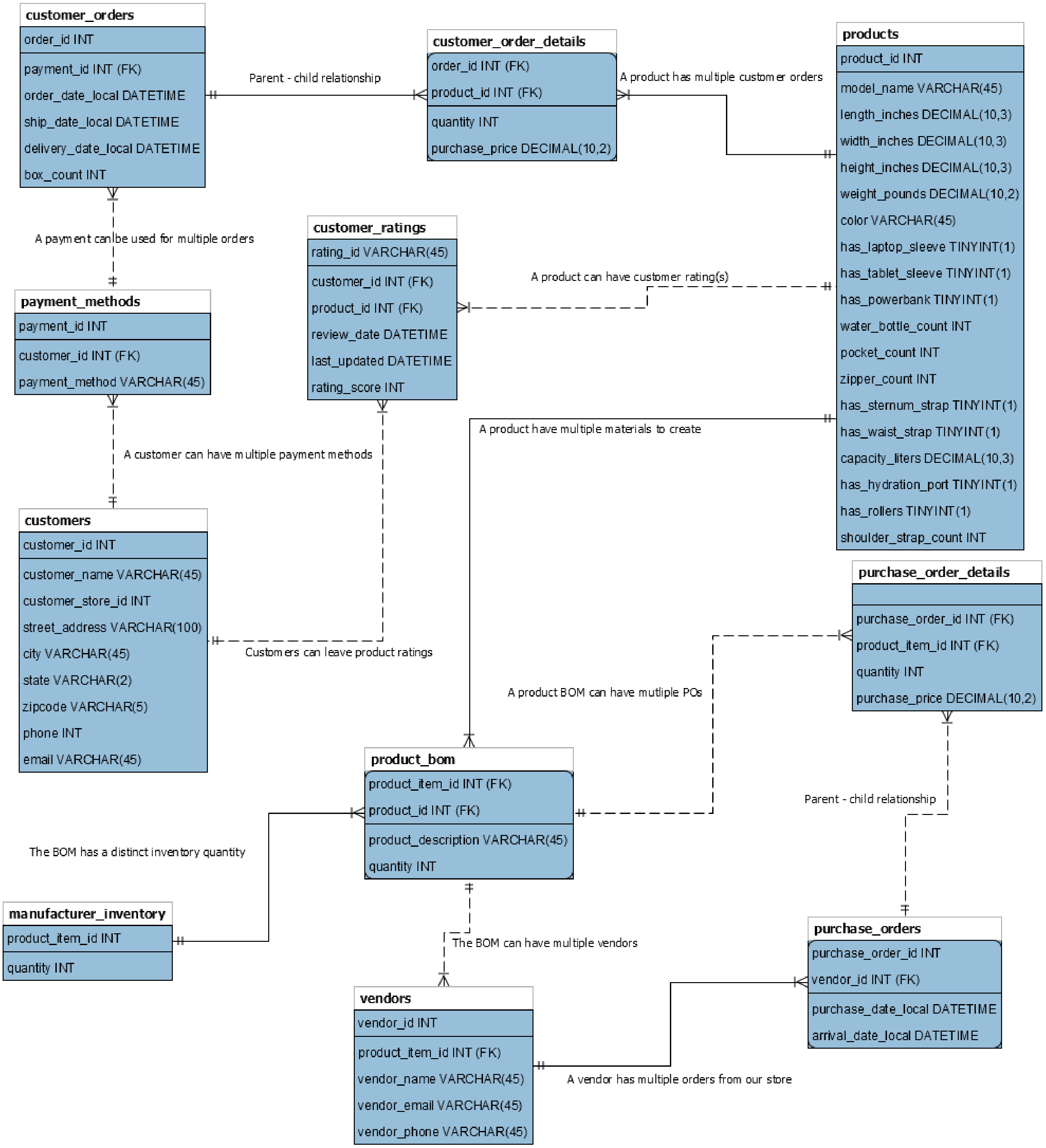
```
DELIMITER //
DROP PROCEDURE customerbystate;
CREATE PROCEDURE customerbystate
()
BEGIN
SELECT state, count(*) as total
FROM customers
GROUP BY state
ORDER BY total desc;
END //

DELIMITER ;

CALL customerbystate();
```

RESULTS (34 ROW RETURNED, SNIPPIT SHOWN)



state	total
Te	134
Ca	101
Fl	91
Ne	87
Mi	69
Co	39
Ma	36
Oh	35
Vi	34
Wa	31
In	30
Di	27
Al	25
No	24
Il	23
Pe	22
Ge	22
Ar	21
Lo	18
Ok	15

ERD Model


customer_orders
order_id INT
payment_id INT (FK)
order_date_local DATETIME
ship_date_local DATETIME
delivery_date_local DATETIME
box_count INT
Parent - child relationship
customer_order_details
order_id INT (FK)
product_id INT (FK)
quantity INT
purchase_price DECIMAL(10,2)
A product has multiple customer orders
products
product_id INT
model_name VARCHAR(45)
length_inches DECIMAL(10,3)
width_inches DECIMAL(10,3)
height_inches DECIMAL(10,3)
weight_pounds DECIMAL(10,2)
color VARCHAR(45)
has_laptop_sleeve TINYINT(1)
has_tablet_sleeve TINYINT(1)
has_powerbank TINYINT(1)
water_bottle_count INT
pocket_count INT
zipper_count INT
has_sternum_strap TINYINT(1)
has_waist_strap TINYINT(1)
capacity_liters DECIMAL(10,3)
has_hydration_port TINYINT(1)
has_rollers TINYINT(1)
shoulder_strap_count INT
A payment can be used for multiple orders
customer_ratings
rating_id VARCHAR(45)
customer_id INT (FK)
product_id INT (FK)
review_date DATETIME
last_updated DATETIME
rating_score INT
A product can have customer rating(s)
payment_methods
payment_id INT
customer_id INT (FK)
payment_method VARCHAR(45)
A product have multiple materials to create
A customer can have multiple payment methods
customers
customer_id INT
customer_name VARCHAR(45)
customer_store_id INT
street_address VARCHAR(100)
city VARCHAR(45)
state VARCHAR(2)
zipcode VARCHAR(5)
phone INT
email VARCHAR(45)
Customers can leave product ratings
purchase_order_details
purchase_order_id INT (FK)
product_item_id INT (FK)
quantity INT
purchase_price DECIMAL(10,2)
A product BOM can have mutliple POs
product_bom
product_item_id INT (FK)
product_id INT (FK)
product_description VARCHAR(45)
quantity INT
Parent - child relationship
The BOM has a distinct inventory quantity
manufacturer_inventory
product_item_id INT
quantity INT
The BOM can have multiple vendors
purchase_orders
purchase_order_id INT
vendor_id INT (FK)
purchase_date_local DATETIME
arrival_date_local DATETIME
vendors
vendor_id INT
product_item_id INT (FK)
vendor_name VARCHAR(45)
vendor_email VARCHAR(45)
vendor_phone VARCHAR(45)
A vendor has multiple orders from our store