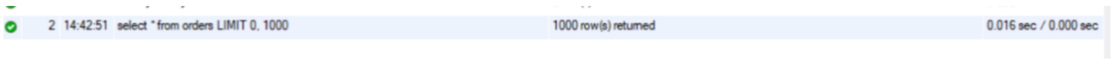
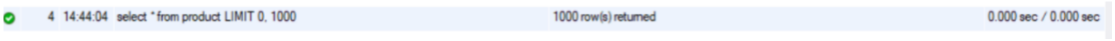
Pt1 Stage 3
Group: NGGYU

In this stage, we implemented more than four tables of the previous design: the order table, the product table, the child tables of products (the cellphone table, the computer table, and the Game table), the customer table, the shop table, and relations tables includes order_shop, order_customer, product_order.
Among these tables, the order table, the product table, the computer table, and the cellphone table have more than 1000 rows.



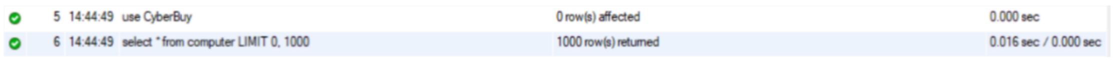
```
2  14:42:51  select * from orders LIMIT 0, 1000      1000 row(s) returned      0.016 sec / 0.000 sec
```

Figure 1 check the number of tuples in the 'Orders' table



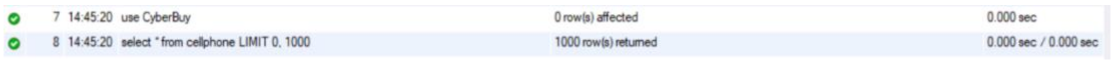
```
4  14:44:04  select * from product LIMIT 0, 1000     1000 row(s) returned      0.000 sec / 0.000 sec
```

Figure 2 check the number of tuples in the 'Product' table



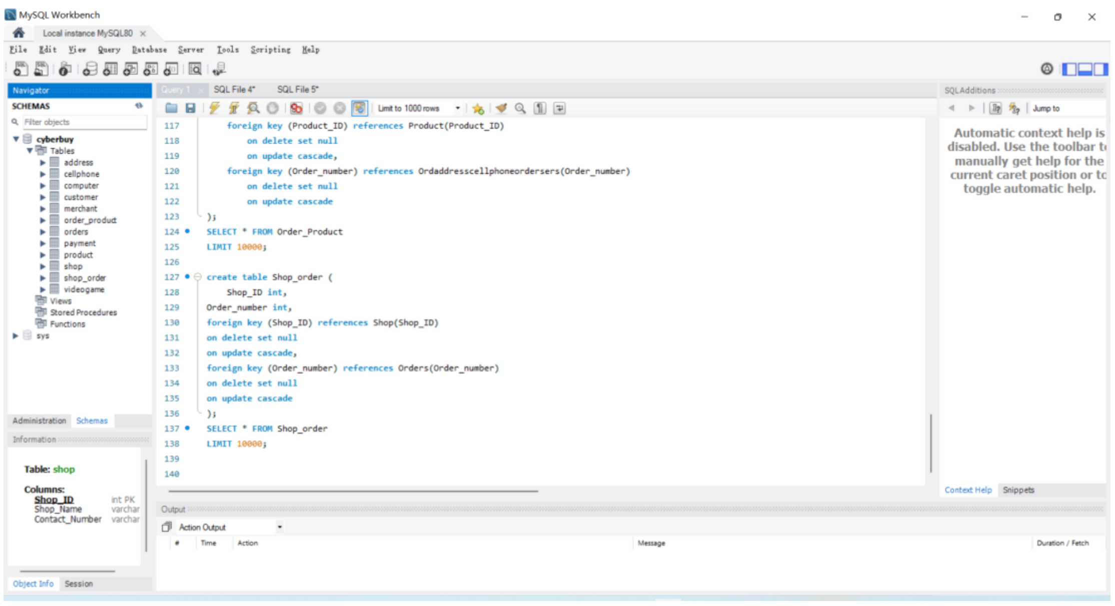
```
5  14:44:49  use CyberBuy                            0 row(s) affected         0.000 sec
6  14:44:49  select * from computer LIMIT 0, 1000    1000 row(s) returned      0.016 sec / 0.000 sec
```

Figure 3 check the number of tuples in the 'Computer' table



```
7  14:45:20  use CyberBuy                            0 row(s) affected         0.000 sec
8  14:45:20  select * from cellphone LIMIT 0, 1000   1000 row(s) returned      0.000 sec / 0.000 sec
```

Figure 4 check the number of tuples in the 'Cellphone' table



```
    foreign key (Product_ID) references Product(Product_ID)
        on delete set null
        on update cascade,
    foreign key (Order_number) references Ordaddresscellphoneordersers(Order_number)
        on delete set null
        on update cascade
);
SELECT * FROM Order_Product
LIMIT 10000;

create table Shop_order (
    Shop_ID int,
Order_number int,
foreign key (Shop_ID) references Shop(Shop_ID)
on delete set null
on update cascade,
foreign key (Order_number) references Orders(Order_number)
on delete set null
on update cascade
);
SELECT * FROM Shop_order
LIMIT 10000;
```

Table: shop
Columns:
Shop_ID int PK
Shop_Name varchar
Contact_Number varchar

Figure 5 terminal information

DDL COMMAND

```
create table Payment (
	Payment_ID int primary key,
	Payment_amount real,
	Payment_type varchar(20),
	Card_number int
);

create table Shop (
	Shop_ID int primary key,
	Shop_Name varchar(20),
	Contact_Number varchar(20)
);

create table Address (
	Address_ID int primary key,
	State varchar(15),
	Street_line varchar(255),
	Zip_code int
);

create table Customer (
	Customer_ID int primary key,
	Email varchar(255),
	Contact_number varchar(20),
	Last_name varchar(20),
	First_name varchar(20),
	Password varchar(20),
	Address_ID int,
	foreign key (Address_ID) references Address(Address_ID)
		on delete set null
		on update cascade
);

create table Merchant (
	Merchant_ID int primary key,
	Email varchar(20),
	Password varchar(50),
	Shop_ID int,
	foreign key (Shop_ID) references Shop(Shop_ID)
		on delete set null
		on update cascade
);

create table Product(
	Product_ID int primary key,
	Title varchar(255),
	Stock int,
	Price real,
	Product_Type varchar(20),
	Shop_ID int,
	foreign key (Shop_ID) references Shop(Shop_ID)
			on delete set null
			on update cascade
	);


create table Computer (
	Product_ID int,
	RAM int,
	Storages int,
	Size real,
	foreign key (Product_ID) references Product(Product_ID)
		on delete set null
		on update cascade
	);

create table Game(
	Product_ID int,
	Game_Name varchar(30),
	foreign key (Product_ID) references Product(Product_ID)
		on delete set null
		on update cascade
	);

create table Cellphone(
	Product_ID int,
	Storages int,
	Color varchar(20),
	foreign key (Product_ID) references Product(Product_ID)
		on delete set null
		on update cascade
	);

create table Orders (
	   Order_number int primary key,
	   Order_status varchar(20),
	   Order_date varchar(20),
	   Customer_ID int,
	   Payment_ID int,
            Shop_ID int,
            Product_ID int,
            foreign key (Customer_ID) references Customer(Customer_ID)
                        on delete set null
                        on update cascade,
            foreign key (Payment_ID) references Payment(Payment_ID)
                        on delete set null
                        on update cascade,
            foreign key (Shop_ID) references Shop(Shop_ID)
                        on delete set null
                        on update cascade,
            foreign key (Product_ID) references Product(Product_ID)
                        on delete set null
                        on update cascade
);


create table Shop_order (
       Shop_ID int,
       Order_number int,
       foreign key (Shop_ID) references Shop(Shop_ID)
              on delete set null
              on update cascade,
       foreign key (Order_number) references Orders(Order_number)
              on delete set null
              on update cascade
);

create table Order_Product(
       Product_ID int,
       Order_number int,
       Quantity int,
       foreign key (Product_ID) references Product(Product_ID)
                     on delete set null
                     on update cascade,
       foreign key (Order_number) references Orders(Order_number)
                     on delete set null
                     on update cascade
);
```

Advanced Queries

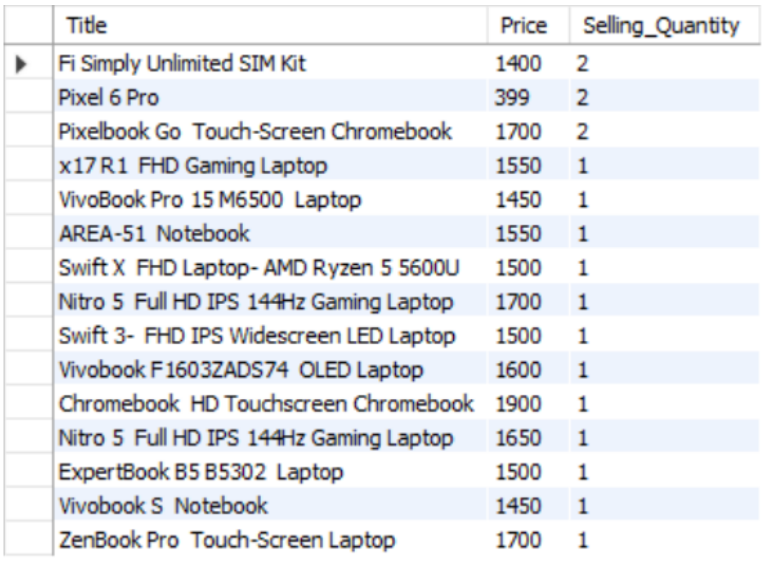
1. **Get the order number of each product in a shop.**

   The first advanced SQL query is to show the sales of every product in a shop, and this query would be used to display the sales report of the shop for merchants.

a) **SQL query:**

```
SELECT product.Title, product.Price,getid.Selling_Quantity
FROM product left join
    (SELECT Product_ID, COUNT(order_product.Product_ID) as Selling_Quantity
    FROM shop natural join shop_order
    left join order_product on shop_order.Order_number = order_product.Order_number
    WHERE Shop_name = 'Google'
    GROUP BY order_product.Product_ID) AS getid ON product.Product_ID = getid.Product_ID
ORDER BY getid.Selling_Quantity DESC
LIMIT 15;
```

b) **Results:**



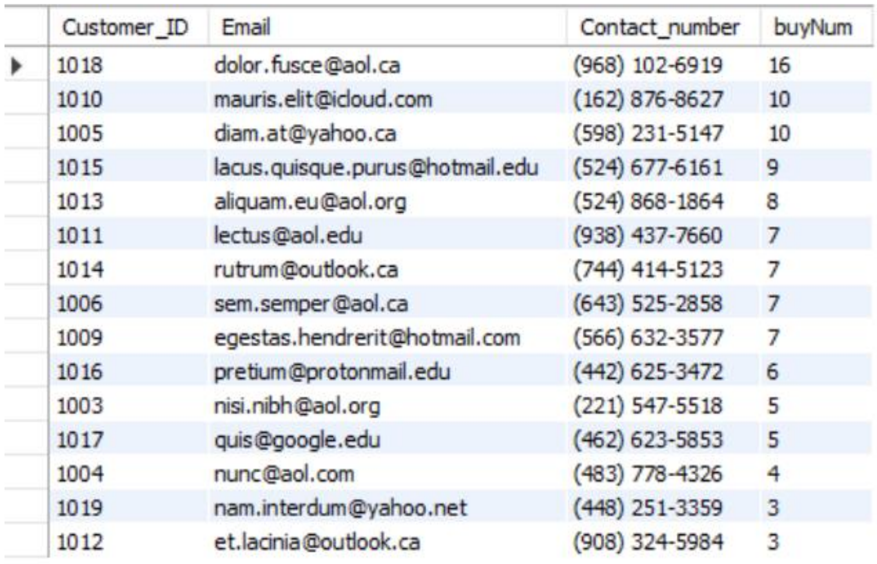
Title	Price	Selling_Quantity
Fi Simply Unlimited SIM Kit	1400	2
Pixel 6 Pro	399	2
Pixelbook Go Touch-Screen Chromebook	1700	2
x17 R1 FHD Gaming Laptop	1550	1
VivoBook Pro 15 M6500 Laptop	1450	1
AREA-51 Notebook	1550	1
Swift X FHD Laptop- AMD Ryzen 5 5600U	1500	1
Nitro 5 Full HD IPS 144Hz Gaming Laptop	1700	1
Swift 3- FHD IPS Widescreen LED Laptop	1500	1
Vivobook F1603ZADS74 OLED Laptop	1600	1
Chromebook HD Touchscreen Chromebook	1900	1
Nitro 5 Full HD IPS 144Hz Gaming Laptop	1650	1
ExpertBook B5 B5302 Laptop	1500	1
Vivobook S Notebook	1450	1
ZenBook Pro Touch-Screen Laptop	1700	1

2. **Get the customer who places the most orders in a shop.**

   The second advanced SQL query is to show the number of orders customers have placed and sort the results in descending order based on the number of orders to see the regular customers and send advertisements to them.

a) **SQL query:**

```
SELECT customer.Customer_ID, customer.Email, customer.Contact_number, getBuyNum.buyNum
FROM customer natural join
    (SELECT orders.Customer_ID, COUNT(orders.Customer_ID) AS buyNum
    FROM shop natural join shop_order left join orders on shop_order.Order_number = orders.Order_number
    WHERE shop.Shop_Name = 'Acer'
    group by orders.Customer_ID) AS getBuyNum
ORDER BY getBuyNum.buyNum DESC
LIMIT 15;
```



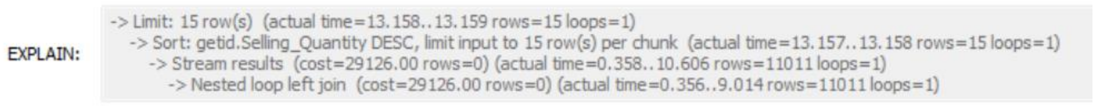
Customer_ID	Email	Contact_number	buyNum
1018	dolor.fusce@aol.ca	(968) 102-6919	16
1010	mauris.elit@icloud.com	(162) 876-8627	10
1005	diam.at@yahoo.ca	(598) 231-5147	10
1015	lacus.quisque.purus@hotmail.edu	(524) 677-6161	9
1013	aliquam.eu@aol.org	(524) 868-1864	8
1011	lectus@aol.edu	(938) 437-7660	7
1014	rutrum@outlook.ca	(744) 414-5123	7
1006	sem.semper@aol.ca	(643) 525-2858	7
1009	egestas.hendrerit@hotmail.com	(566) 632-3577	7
1016	pretium@protonmail.edu	(442) 625-3472	6
1003	nisi.nibh@aol.org	(221) 547-5518	5
1017	quis@google.edu	(462) 623-5853	5
1004	nunc@aol.com	(483) 778-4326	4
1019	nam.interdum@yahoo.net	(448) 251-3359	3
1012	et.lacinia@outlook.ca	(908) 324-5984	3

Index analysis

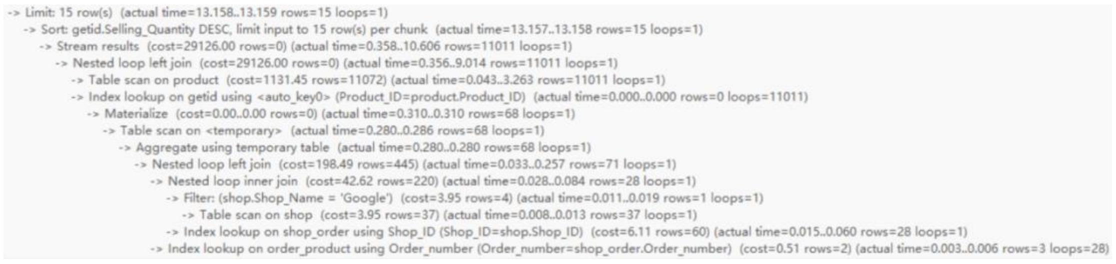
A. First Query

For the first query, the default index is the primary key (product_ID, Order_number), so we tried to add an index on the shop name, product title, and product price.

1. before adding any index



```
EXPLAIN:
-> Limit: 15 row(s)  (actual time=13.158..13.159 rows=15 loops=1)
    -> Sort: getid.Selling_Quantity DESC, limit input to 15 row(s) per chunk  (actual time=13.157..13.158 rows=15 loops=1)
        -> Stream results  (cost=29126.00 rows=0) (actual time=0.358..10.606 rows=11011 loops=1)
            -> Nested loop left join  (cost=29126.00 rows=0) (actual time=0.356..9.014 rows=11011 loops=1)
```

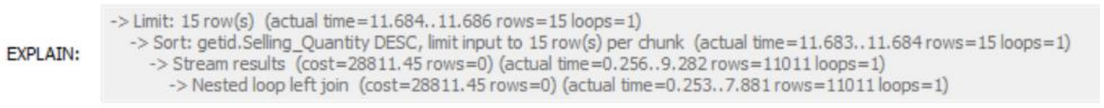


```
-> Limit: 15 row(s)  (actual time=13.158..13.159 rows=15 loops=1)
    -> Sort: getid.Selling_Quantity DESC, limit input to 15 row(s) per chunk  (actual time=13.157..13.158 rows=15 loops=1)
        -> Stream results  (cost=29126.00 rows=0) (actual time=0.358..10.606 rows=11011 loops=1)
            -> Nested loop left join  (cost=29126.00 rows=0) (actual time=0.356..9.014 rows=11011 loops=1)
                -> Table scan on product  (cost=1131.45 rows=11072) (actual time=0.043..3.263 rows=11011 loops=1)
                -> Index lookup on getid using <auto_key0> (Product_ID=product.Product_ID)  (actual time=0.000..0.000 rows=0 loops=11011)
                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=0.310..0.310 rows=68 loops=1)
                        -> Table scan on <temporary>  (actual time=0.280..0.286 rows=68 loops=1)
                            -> Aggregate using temporary table  (actual time=0.280..0.280 rows=68 loops=1)
                                -> Nested loop left join  (cost=198.49 rows=445) (actual time=0.033..0.257 rows=71 loops=1)
                                    -> Nested loop inner join  (cost=42.62 rows=220) (actual time=0.028..0.084 rows=28 loops=1)
                                        -> Filter: (shop.Shop_Name = 'Google')  (cost=3.95 rows=4) (actual time=0.011..0.019 rows=1 loops=1)
                                            -> Table scan on shop  (cost=3.95 rows=37) (actual time=0.008..0.013 rows=37 loops=1)
                                        -> Index lookup on shop_order using Shop_ID (Shop_ID=shop.Shop_ID)  (cost=6.11 rows=60) (actual time=0.015..0.060 rows=28 loops=1)
                                    -> Index lookup on order_product using Order_number (Order_number=shop_order.Order_number)  (cost=0.51 rows=2) (actual time=0.003..0.006 rows=3 loops=28)
```

- The time to return to the first row:13.158
- The time to return all rows: 13.159

2. Create an index for the shop name:

   CREATE INDEX index_Shop_Name ON shop (Shop_Name);

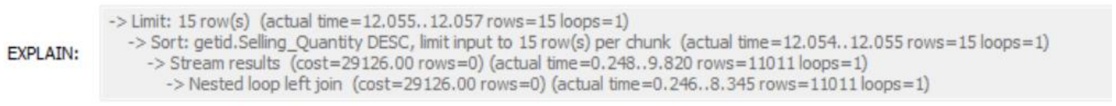


```
EXPLAIN:
-> Limit: 15 row(s)  (actual time=11.684..11.686 rows=15 loops=1)
    -> Sort: getid.Selling_Quantity DESC, limit input to 15 row(s) per chunk  (actual time=11.683..11.684 rows=15 loops=1)
        -> Stream results  (cost=28811.45 rows=0) (actual time=0.256..9.282 rows=11011 loops=1)
            -> Nested loop left join  (cost=28811.45 rows=0) (actual time=0.253..7.881 rows=11011 loops=1)
```

- The time to return to the first row:11.684
- The time to return all rows: 11.686

3. Create an index for product title

   CREATE INDEX index_product_title ON product (Title);

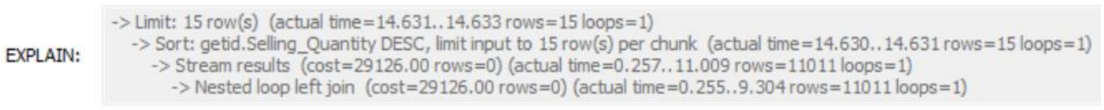


```
EXPLAIN:
-> Limit: 15 row(s)  (actual time=12.055..12.057 rows=15 loops=1)
    -> Sort: getid.Selling_Quantity DESC, limit input to 15 row(s) per chunk  (actual time=12.054..12.055 rows=15 loops=1)
        -> Stream results  (cost=29126.00 rows=0) (actual time=0.248..9.820 rows=11011 loops=1)
            -> Nested loop left join  (cost=29126.00 rows=0) (actual time=0.246..8.345 rows=11011 loops=1)
```

- The time to return to the first row:12.055
- The time to return all rows: 12.057

4. Create an index for the product price

CREATE INDEX index_product_price ON product (Price);



```
EXPLAIN:  -> Limit: 15 row(s)  (actual time=14.631..14.633 rows=15 loops=1)
              -> Sort: getid.Selling_Quantity DESC, limit input to 15 row(s) per chunk  (actual time=14.630..14.631 rows=15 loops=1)
                  -> Stream results  (cost=29126.00 rows=0) (actual time=0.257..11.009 rows=11011 loops=1)
                      -> Nested loop left join  (cost=29126.00 rows=0) (actual time=0.255..9.304 rows=11011 loops=1)
```

- The time to return to the first row:14.631
- The time to return all rows: 14.633

Conclusion:

Since the shop name is used to search for searching records related to specific shops when merchants want to see the sales reports of their shop, we tried to use it as an index. The indexes product title and product price have not been used in this query but may be useful when searching for a product on the web page or filtering products with the price range.

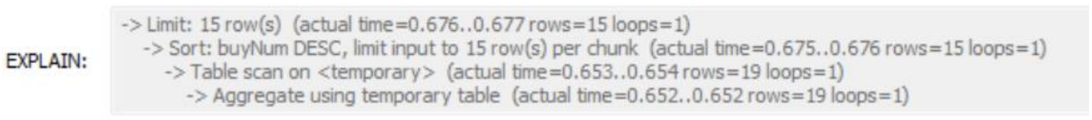
In the process, a materialized subquery "getid" is created by first operating a table scan on the product table, aggregate function is used to calculate the number of orders for each product, then a filter is applied for finding the shop "Google".

From the above running results, we can tell that using shop_name as an index is a good choice, even though the running time only has a little reduction, it could be more useful when multiple merchants click to see their shops' sales report.

B. second query.

For the second query, the default index is the primary key (Customer_ID, Order_number, Shop_ID), so we tried adding an index on the shop name, customer email, and customer contact number.

1. Before adding any index



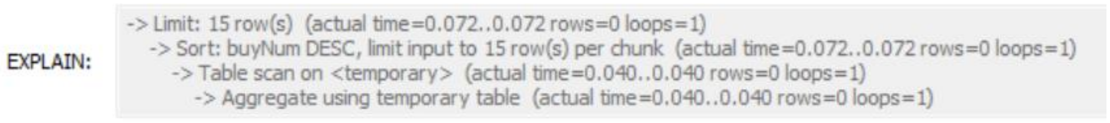
```
EXPLAIN:  -> Limit: 15 row(s)  (actual time=0.676..0.677 rows=15 loops=1)
              -> Sort: buyNum DESC, limit input to 15 row(s) per chunk  (actual time=0.675..0.676 rows=15 loops=1)
                  -> Table scan on <temporary>  (actual time=0.653..0.654 rows=19 loops=1)
                      -> Aggregate using temporary table  (actual time=0.652..0.652 rows=19 loops=1)
```

- The time to return the first row:0.676
- The time to return all rows: 0.677

- In the explained analysis results, the Index lookup on shop_order using Shop_ID cost 6.11 for an expected 118 rows (per loop).
- The filter cost 3.95 for an expected 4(per loop).

2. Add an index on the shop name

   CREATE INDEX index_Shop_Name ON shop (Shop_Name);

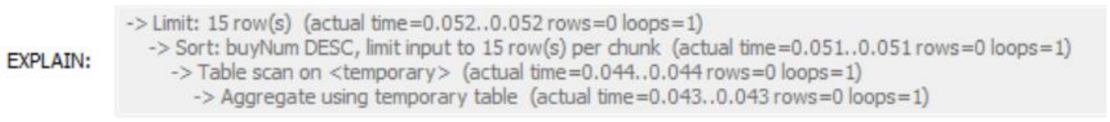


```
EXPLAIN:  -> Limit: 15 row(s)  (actual time=0.072..0.072 rows=0 loops=1)
            -> Sort: buyNum DESC, limit input to 15 row(s) per chunk  (actual time=0.072..0.072 rows=0 loops=1)
              -> Table scan on <temporary>  (actual time=0.040..0.040 rows=0 loops=1)
                -> Aggregate using temporary table  (actual time=0.040..0.040 rows=0 loops=1)
```

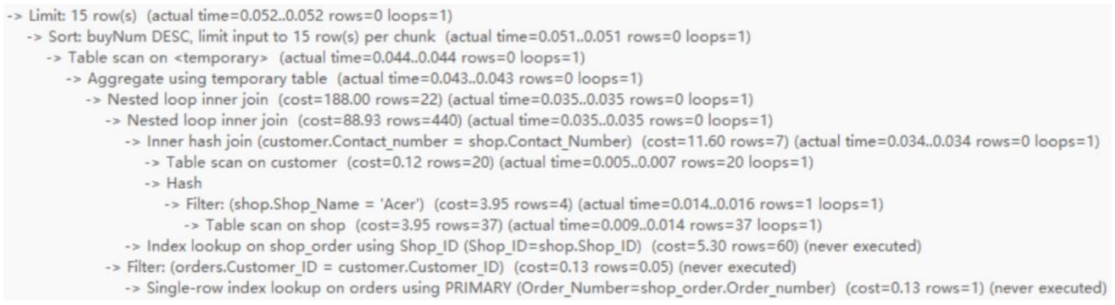
- The time to return the first row:0.072
- The time to return all rows: 0.072
- Covering index lookup on shop using index_Shop_Name costs 0.35 for an expected 1 row (per loop)

3. Add index on customer email

   CREATE INDEX index_customer_email ON customer (Email);



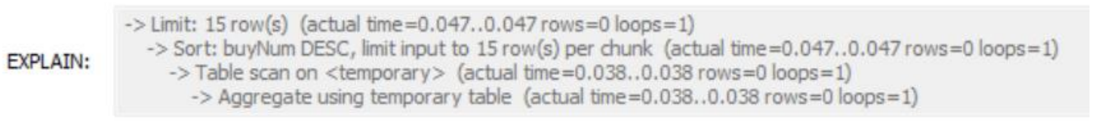
```
EXPLAIN:  -> Limit: 15 row(s)  (actual time=0.052..0.052 rows=0 loops=1)
            -> Sort: buyNum DESC, limit input to 15 row(s) per chunk  (actual time=0.051..0.051 rows=0 loops=1)
              -> Table scan on <temporary>  (actual time=0.044..0.044 rows=0 loops=1)
                -> Aggregate using temporary table  (actual time=0.043..0.043 rows=0 loops=1)
```



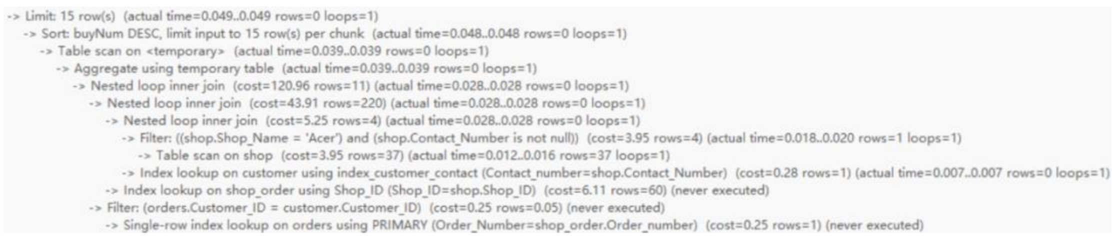
```
-> Limit: 15 row(s)  (actual time=0.052..0.052 rows=0 loops=1)
  -> Sort: buyNum DESC, limit input to 15 row(s) per chunk  (actual time=0.051..0.051 rows=0 loops=1)
    -> Table scan on <temporary>  (actual time=0.044..0.044 rows=0 loops=1)
      -> Aggregate using temporary table  (actual time=0.043..0.043 rows=0 loops=1)
        -> Nested loop inner join  (cost=188.00 rows=22) (actual time=0.035..0.035 rows=0 loops=1)
          -> Nested loop inner join  (cost=88.93 rows=440) (actual time=0.035..0.035 rows=0 loops=1)
            -> Inner hash join (customer.Contact_number = shop.Contact_Number)  (cost=11.60 rows=7) (actual time=0.034..0.034 rows=0 loops=1)
              -> Table scan on customer  (cost=0.12 rows=20) (actual time=0.005..0.007 rows=20 loops=1)
              -> Hash
                -> Filter: (shop.Shop_Name = 'Acer')  (cost=3.95 rows=4) (actual time=0.014..0.016 rows=1 loops=1)
                  -> Table scan on shop  (cost=3.95 rows=37) (actual time=0.009..0.014 rows=37 loops=1)
            -> Index lookup on shop_order using Shop_ID (Shop_ID=shop.Shop_ID)  (cost=5.30 rows=60) (never executed)
          -> Filter: (orders.Customer_ID = customer.Customer_ID)  (cost=0.13 rows=0.05) (never executed)
            -> Single-row index lookup on orders using PRIMARY (Order_Number=shop_order.Order_number)  (cost=0.13 rows=1) (never executed)
```

- The time to return the first row:0.052
- The time to return all rows: 0.052

4. Add an index on the customer contact number



```
EXPLAIN:  -> Limit: 15 row(s)  (actual time=0.047..0.047 rows=0 loops=1)
            -> Sort: buyNum DESC, limit input to 15 row(s) per chunk  (actual time=0.047..0.047 rows=0 loops=1)
              -> Table scan on <temporary>  (actual time=0.038..0.038 rows=0 loops=1)
                -> Aggregate using temporary table  (actual time=0.038..0.038 rows=0 loops=1)
```



```
-> Limit: 15 row(s)  (actual time=0.049..0.049 rows=0 loops=1)
    -> Sort: buyNum DESC, limit input to 15 row(s) per chunk  (actual time=0.048..0.048 rows=0 loops=1)
        -> Table scan on <temporary>  (actual time=0.039..0.039 rows=0 loops=1)
            -> Aggregate using temporary table  (actual time=0.039..0.039 rows=0 loops=1)
                -> Nested loop inner join  (cost=120.96 rows=11) (actual time=0.028..0.028 rows=0 loops=1)
                    -> Nested loop inner join  (cost=43.91 rows=220) (actual time=0.028..0.028 rows=0 loops=1)
                        -> Nested loop inner join  (cost=5.25 rows=4) (actual time=0.028..0.028 rows=0 loops=1)
                            -> Filter: ((shop.Shop_Name = 'Acer') and (shop.Contact_Number is not null))  (cost=3.95 rows=4) (actual time=0.018..0.020 rows=1 loops=1)
                                -> Table scan on shop  (cost=3.95 rows=37) (actual time=0.012..0.016 rows=37 loops=1)
                            -> Index lookup on customer using index_customer_contact (Contact_number=shop.Contact_Number)  (cost=0.28 rows=1) (actual time=0.007..0.007 rows=0 loops=1)
                        -> Index lookup on shop_order using Shop_ID (Shop_ID=shop.Shop_ID)  (cost=6.11 rows=60) (never executed)
                    -> Filter: (orders.Customer_ID = customer.Customer_ID)  (cost=0.25 rows=0.05) (never executed)
                        -> Single-row index lookup on orders using PRIMARY (Order_Number=shop_order.Order_number)  (cost=0.25 rows=1) (never executed)
```

- The time to return the first row:0.047
- The time to return all rows: 0.047

Conclusion

In the second query, we tried the shop name, customer email, and contact number as an index. From the above running results, the index shop name has been used and the running time has a little reduction. The contact number and email of customers have not been used in this query, but could be helpful in after-sales service. So the shop name may be a good choice in this query.

In the process, a temporary table is created and the aggregate function is applied to calculate the number of order group by Customer_ID. And a filter is applied to find shop whose name is “Acer”.