A2Z

Online Retail Store

Aayush Ranjan (2021003)
Aishiki Bhattacharya (2021007)

<u>Scope</u>:
Today, due to the extremely fast-paced lives of people, it can be exhausting and time-consuming to visit different stores at different locations. Visiting a market and choosing the right product is cumbersome and may take several hours. Also, due to the recent COVID-19 pandemic, people prefer to get products delivered to their doorstep as quickly as possible.

Therefore, to cater to the needs of the public, we provide complete management solutions to manage information on customers, sellers, delivery agents and products efficiently.

A2Z gives the admins a forum to showcase their products of various categories to customers in a hassle-free and systematic way. Customers can conveniently browse products, manage items in the cart and buy as per their preferred payment method. Delivery agents are assigned orders by the admin, which are supposed to be delivered within the stipulated time to provide an excellent experience to customers. Each of the objects is identified by a unique attribute, i.e., their primary key - Customer_ID, Admin_ID, Delivery_Agent_ID, Order_ID, Product_ID and so on.
This application will enable the admin, delivery agent and customer to interact and coordinate efficiently with each other, resulting in a pleasant experience for all.

<u>TRIGGER QUERIES:</u>

1) **If any new product is added to any cart, the number of products of that specific cart increases by 1**

```
CREATE TRIGGER increase_numofprod
     AFTER INSERT
     ON contains FOR EACH ROW
     UPDATE cart SET NO_OF_PRODUCT =  NO_OF_PRODUCT + 1 where
     cart.Cart_Id = NEW.Cart_Id;
```

2) **If the customer has requested a return, the product has been collected and the status has been marked done, then the refund amount will be added to the customer's wallet.**

```
delimiter //
CREATE  TRIGGER updateCustWallet
            AFTER UPDATE
            ON `return` FOR EACH ROW
            BEGIN
              IF (NEW.status = 'done')
              THEN
                UPDATE customer set
                customer.wallet=customer.wallet+new.refund_amou
                nt where customer.customer_id=new.customer_id;
     END IF;
 END; //
```

3) **When an order is placed, the stock of all the products that are part of the order is reduced by 1.**

```
delimiter //
CREATE  TRIGGER updateStock
            AFTER INSERT
            ON `order` FOR EACH ROW
            BEGIN
            IF(new.cart_id is not null)
            THEN
update product set stock=stock-1 where product_id in (select
Product_Id from contains where cart_id = new.cart_id) and
stock>0;
                END IF;
 END; //
```

#Grouping Sets

1. This is a query made by implementing the logic of Grouping sets OLAP keyword using ROLLUP OLAP keyword. This query displays the average of stars(rating) grouped according to the customer_id and product_id one by one(Grouping Sets Logic)

```
->
drop view v1;
drop view v2;
create view v1 as select customer_id, product_id, avg(stars) as av
from review group by customer_id, product_id with ROLLUP;
create view v2 as select customer_id, product_id, avg(stars) as av
from review group by product_id, customer_id with ROLLUP;

drop view unio;
create view unio as select customer_id, product_id, avg(stars) as av
from review group by customer_id, product_id with ROLLUP
UNION
select customer_id, product_id, avg(stars) as av from review group by
product_id, customer_id with ROLLUP;

drop view inter;
create view inter as SELECT DISTINCT customer_id, product_id, av FROM
v1
INNER JOIN v2 USING(customer_id, product_id, av);

select DISTINCT unio.customer_id, unio.product_id, unio.av from unio
inner join inter where unio.customer_id is NULL and inter.customer_id
is not NULL or unio.product_id is null and inter.product_id is not
null;
```

#Cube

2. This is a query made by implementing the logic of Cube OLAP keyword using ROLLUP OLAP keyword. This query displays the average of stars(rating) grouped according to the customer_id and product_id using all permutations and combinations.(Cube Logic)

```
->
select customer_id, product_id, avg(stars) from review group by
customer_id, product_id with ROLLUP
UNION
```

```
select customer_id, product_id, avg(stars) from review group by
product_id, customer_id with ROLLUP;
```

#Rollup

3. To view the number of orders placed on the application grouped by year and month. It displays the number of orders placed in each year and in each month using ROLLUP.

->

```
select EXTRACT(YEAR FROM order_date) as year, EXTRACT(MONTH FROM
order_date) as month, count(order_id) from `order` group by
EXTRACT(YEAR FROM order_date), EXTRACT(MONTH FROM order_date) with
ROLLUP;
```

4. To view the revenue made by the application grouped according to the year, month and method of payment. It displays the revenue made by the application as the sum of all the payments done corresponding to each year, month and payment method.

->

```
select payment_method, EXTRACT(YEAR FROM payment_date) as year,
EXTRACT(MONTH FROM payment_date) as month, sum(price) from payment
group by  payment_method, EXTRACT(YEAR FROM payment_date),
EXTRACT(MONTH FROM payment_date) with ROLLUP;
```

5. To view the number of returns by a customer grouped by customer_Id and status of that return. It displays the number of returns by a customer since the start of the application corresponding to each customer_Id and status of the return.

->

```
select customer_id, status, count(order_id) from `return` group by
customer_id, status with ROLLUP;
```