Project: E-commerce Analytic System

Database Design

Entities and Relationships:

Products: Stores product details.

Customers: Stores customer information.

Orders: Stores order details, including order date and total amount.

OrderItems: Stores details of items in each order.

Reviews: Stores customer reviews for products.

Data:

Products:

|  |  |  |  |
| --- | --- | --- | --- |
| ProductName | Category | Price | Stock |
| 'Laptop' | 'Electronics' | 999.99 | 50 |
| 'Smartphone' | 'Electronics' | 599.99 | 100 |
| 'Desk Chair' | 'Furniture' | 89.99 | 200 |

Customers:

|  |  |  |  |
| --- | --- | --- | --- |
| FirstName | LastName | Email | DateOfRegistration |
| 'John' | 'Doe' | 'john.doe@example.com' | '2023-01-15' |
| 'Jane' | 'Smith' | 'jane.smith@example.com' | '2023-02-10' |

Orders:

|  |  |  |
| --- | --- | --- |
| CustomerID | OrderDate | TotalAmount |
| 1 | '2023-05-01' | 1099.98 |
| 2 | '2023-06-15' | 689.98 |

OrderItems:

|  |  |  |  |
| --- | --- | --- | --- |
| OrderID | ProductID | Quantity | Price |
| 1 | 1 | 1 | 999.99 |
| 1 | 3 | 1 | 99.99 |
| 2 | 2 | 1 | 599.99 |
| 2 | 3 | 1 | 89.99 |

Reviews:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ProductID | CustomerID | Rating | ReviewText | ReviewDate |
| 1 | 1 | 5 | 'Great laptop!', | '2023-05-10' |
| 2 | 2 | 4 | 'Good value for money.' | '2023-06-20' |

1. Create Tables

-- creating the products table

create table products (

productid int primary key auto\_increment,

productname varchar(100),

category varchar(50),

price decimal(10, 2),

stock int

);

-- creating the customers table

create table customers (

customerid int primary key auto\_increment,

firstname varchar(50),

lastname varchar(50),

email varchar(100),

dateofregistration date

);

-- creating the orders table

create table orders (

orderid int primary key auto\_increment,

customerid int,

orderdate date,

totalamount decimal(10, 2),

foreign key (customerid) references customers(customerid)

);

-- creating the orderitems table

create table orderitems (

orderitemid int primary key auto\_increment,

orderid int,

productid int,

quantity int,

price decimal(10, 2),

foreign key (orderid) references orders(orderid),

foreign key (productid) references products(productid)

);

-- creating the reviews table

create table reviews (

reviewid int primary key auto\_increment,

productid int,

customerid int,

rating int,

reviewtext text,

reviewdate date,

foreign key (productid) references products(productid),

foreign key (customerid) references customers(customerid)

);

2. Insert Data

-- inserting data into products table

insert into products (productname, category, price, stock) values ('laptop', 'electronics', 999.99, 50);

insert into products (productname, category, price, stock) values ('smartphone', 'electronics', 599.99, 100);

insert into products (productname, category, price, stock) values ('desk chair', 'furniture', 89.99, 200);

-- inserting data into customers table

insert into customers (firstname, lastname, email, dateofregistration) values ('john', 'doe', 'john.doe@example.com', '2023-01-15');

insert into customers (firstname, lastname, email, dateofregistration) values ('jane', 'smith', 'jane.smith@example.com', '2023-02-10');

-- inserting data into orders table

insert into orders (customerid, orderdate, totalamount) values (1, '2023-05-01', 1099.98);

insert into orders (customerid, orderdate, totalamount) values (2, '2023-06-15', 689.98);

-- inserting data into orderitems table

insert into orderitems (orderid, productid, quantity, price) values (1, 1, 1, 999.99);

insert into orderitems (orderid, productid, quantity, price) values (1, 3, 1, 99.99);

insert into orderitems (orderid, productid, quantity, price) values (2, 2, 1, 599.99);

insert into orderitems (orderid, productid, quantity, price) values (2, 3, 1, 89.99);

-- inserting data into reviews table

insert into reviews (productid, customerid, rating, reviewtext, reviewdate) values (1, 1, 5, 'great laptop!', '2023-05-10');

insert into reviews (productid, customerid, rating, reviewtext, reviewdate) values (2, 2, 4, 'good value for money.', '2023-06-20');

3. List all products along with their categories and prices:

select productname, category, price

from products;

4. Find all customers who registered in 2023:

select firstname, lastname, email

from customers

where dateofregistration between '2023-01-01' and '2023-12-31';

5. Get the total number of orders and total sales amount:

select count(orderid) as totalorders, sum(totalamount) as totalsales

from orders;

6. List all orders along with the customer name and total amount:

select orders.orderid, customers.firstname, customers.lastname, orders.totalamount

from orders

join customers on orders.customerid = customers.customerid;

7. Find all products that have received a rating of 4 or higher:

select products.productname, avg(reviews.rating) as averagerating

from reviews

join products on reviews.productid = products.productid

group by products.productname

having avg(reviews.rating) >= 4;

8. Calculate the total quantity of each product sold:

select products.productname, sum(orderitems.quantity) as totalquantitysold

from orderitems

join products on orderitems.productid = products.productid

group by products.productname;

9. Find customers who have placed more than one order:

select customers.firstname, customers.lastname, count(orders.orderid) as ordercount

from orders

join customers on orders.customerid = customers.customerid

group by customers.customerid, customers.firstname, customers.lastname

having count(orders.orderid) > 1;

10. Retrieve the details of the most expensive order:

select orders.orderid, customers.firstname, customers.lastname, orders.totalamount

from orders

join customers on orders.customerid = customers.customerid

order by orders.totalamount desc

limit 1;

11. Find the product with the highest total sales amount:

select products.productname, sum(orderitems.quantity \* orderitems.price) as totalsales

from orderitems

join products on orderitems.productid = products.productid

group by products.productname

order by totalsales desc

limit 1;

12. List customers along with the total amount they've spent:

select customers.firstname, customers.lastname, sum(orders.totalamount) as totalspent

from orders

join customers on orders.customerid = customers.customerid

group by customers.customerid, customers.firstname, customers.lastname;

13. Get the average rating of each product along with the number of reviews:

select products.productname, avg(reviews.rating) as averagerating, count(reviews.reviewid) as reviewcount

from reviews

join products on reviews.productid = products.productid

group by products.productname;

14. Add more Entities and Relationships to the database:

Categories: Separate table for product categories.

Suppliers: Stores supplier details.

Inventory: Manages product stock and supplier relations.

-- creating the categories table

create table categories (

categoryid int primary key auto\_increment,

categoryname varchar(50) not null

);

-- creating the suppliers table

create table suppliers (

supplierid int primary key auto\_increment,

suppliername varchar(100) not null,

contactname varchar(100),

phone varchar(15),

email varchar(100)

);

-- altering the products table

alter table products(

modify column productname varchar(100) not null;

drop column category,

add column categoryid int,

add column supplierid int,

modify column price decimal(10,2) not null;

modify column stock int not null;

add constraint add check(stock>=0)

add foreign key (categoryid) references categories(categoryid),

add foreign key (supplierid) references suppliers(supplierid)

);

-- altering the customers table

alter table customers(

modify column firstname varchar(50) not null;

modify column lastname varchar(50) not null;

modify column dateofregistration date not null;

add column email varchar(100) unique not null,

);

-- altering the orders table

alter table orders(

modify column customerid int not null;

modify column orderdate date not null;

modify totalamount decimal(10, 2) not null;

add foreign key (customerid) references customers(customerid)

);

-- altering the orderitems table

alter table orderitems (

modilfy orderid int not null,

modify productid int not null,

modify quantity int not null

add check (quantity > 0),

modify price decimal(10, 2) not null,

add foreign key (orderid) references orders(orderid),

foreign key (productid) references products(productid)

);

-- altering the reviews table

alter table reviews (

modify productid int not null,

modify customerid int not null,

add constarint check (rating between 1 and 5),

add foreign key (productid) references products(productid),

add foreign key (customerid) references customers(customerid)

);

-- creating the inventory table

create table inventory (

inventoryid int primary key auto\_increment,

productid int not null,

supplierid int not null,

stockadded int not null check (stockadded > 0),

dateadded date not null,

add foreign key (productid) references products(productid),

add foreign key (supplierid) references suppliers(supplierid)

);

15. Insert data into these tables.

-- inserting data into categories table

insert into categories (categoryname) values ('electronics');

insert into categories (categoryname) values ('furniture');

-- inserting data into suppliers table

insert into suppliers (suppliername, contactname, phone, email) values ('techsupplier inc.', 'alice johnson', '123-456-7890', 'alice@techsupplier.com');

insert into suppliers (suppliername, contactname, phone, email) values ('homecomforts', 'bob smith', '098-765-4321', 'bob@homecomforts.com');

-- inserting data into inventory table

insert into inventory (productid, supplierid, stockadded, dateadded) values (1, 1, 50, '2023-01-10');

insert into inventory (productid, supplierid, stockadded, dateadded) values (2, 1, 100, '2023-01-15');

insert into inventory (productid, supplierid, stockadded, dateadded) values (3, 2, 200, '2023-01-20');

16. Find the total sales for each product category:

select c.categoryname, sum(oi.quantity \* oi.price) as totalsales

from orderitems oi

join products p on oi.productid = p.productid

join categories c on p.categoryid = c.categoryid

group by c.categoryname;

17. Get the top 3 customers by total spending:

select c.firstname, c.lastname, sum(o.totalamount) as totalspent

from orders o

join customers c on o.customerid = c.customerid

group by c.customerid, c.firstname, c.lastname

order by totalspent desc

limit 3;

18. List products that have never been ordered:

select p.productname

from products p

left join orderitems oi on p.productid = oi.productid

where oi.orderitemid is null;

19. Find the average rating and number of reviews for each product:

select p.productname, avg(r.rating) as averagerating, count(r.reviewid) as reviewcount

from reviews r

join products p on r.productid = p.productid

group by p.productname;

20. Calculate the reorder level for products (if stock is less than a threshold, say 10 units):

select p.productname, p.stock

from products p

where p.stock < 10;

21. Add a new product:

delimiter $$

create procedure addproduct(

in pname varchar(100),

in pcategoryid int,

in psupplierid int,

in pprice decimal(10, 2),

in pstock int

)

begin

insert into products (productname, categoryid, supplierid, price, stock)

values (pname, pcategoryid, psupplierid, pprice, pstock);

end $$

delimiter ;

22. Update stock after an order:

delimiter $$

create procedure updatestockafterorder(

in porderid int

)

begin

declare done int default false;

declare pproductid int;

declare pquantity int;

declare cur cursor for

select productid, quantity

from orderitems

where orderid = porderid;

declare continue handler for not found set done = true;

open cur;

read\_loop: loop

fetch cur into pproductid, pquantity;

if done then

leave read\_loop;

end if;

update products

set stock = stock - pquantity

where productid = pproductid;

end loop;

close cur;

end $$

delimiter ;

23. Automatically update stock when a new order is placed:

delimiter $$

create trigger updatestock

after insert on orderitems

for each row

begin

update products

set stock = stock - new.quantity

where productid = new.productid;

end $$

delimiter ;

24. Log changes in product prices:

create table pricechanges (

changeid int primary key auto\_increment,

productid int,

oldprice decimal(10, 2),

newprice decimal(10, 2),

changedate timestamp default current\_timestamp,

foreign key (productid) references products(productid)

);

delimiter $$

create trigger logpricechange

before update on products

for each row

begin

if old.price <> new.price then

insert into pricechanges (productid, oldprice, newprice)

values (old.productid, old.price, new.price);

end if;

end $$

delimiter ;

25. Create Index on ProductName for faster search

create index idx\_product\_name on products(productname);

26. Create Composite index on OrderDate and CustomerID for optimizing order queries

create index idx\_order\_date\_customer on orders(orderdate, customerid);

27. Retrieve the Total Sales per Category and also analyze it to optimize the perfomance:

explain select c.categoryname, sum(oi.quantity \* oi.price) as totalsales

from orderitems oi

join products p on oi.productid = p.productid

join categories c on p.categoryid = c.categoryid

group by c.categoryname;

28. Ensure email uniqueness

alter table customers add constraint unique\_email unique (email);

29. Ensure non-negative prices

alter table products add constraint check\_price check (price >= 0);

30. Create a read-only user

create user 'readonly'@'localhost' identified by 'password';

grant select on your\_database.\* to 'readonly'@'localhost';

31. Create an admin user with full permissions

create user 'admin'@'localhost' identified by 'adminpassword';

grant all privileges on your\_database.\* to 'admin'@'localhost';

32. Partition the Orders table by OrderDate to improve performance.

create table orders\_partitioned (

orderid int,

customerid int,

orderdate date,

totalamount decimal(10, 2),

primary key (orderid, orderdate)

)

partition by range (year(orderdate)) (

partition p2023 values less than (2024),

partition p2024 values less than (2025),

partition pmax values less than maxvalue

);

33. Create a view to simplify customer order data access.

create view customerorders as

select c.customerid, c.firstname, c.lastname, o.orderid, o.orderdate, o.totalamount

from customers c

join orders o on c.customerid = o.customerid;

34. Create a materialized view for top-selling products.

create materialized view topsellingproducts as

select p.productid, p.productname, sum(oi.quantity) as totalquantity

from orderitems oi

join products p on oi.productid = p.productid

group by p.productid, p.productname;

35. Add full-text search capability for product names and reviews.

alter table products add fulltext (productname);

alter table reviews add fulltext (reviewtext);

select \* from products where match(productname) against ('laptop');

36. Set up replication for high availability (this requires server configuration outside SQL commands).

-- on the primary server:

show master status;

-- on the replica server:

change master to master\_host='primary\_server', master\_user='replica\_user', master\_password='password', master\_log\_file='mysql-bin.000001', master\_log\_pos= 123;

start slave;

37. Add cascading deletes to maintain referential integrity.

alter table orderitems

add constraint fk\_order

foreign key (orderid) references orders(orderid)

on delete cascade;

38. Implement auditing to track changes to the Products table.

create table productaudit (

auditid int primary key auto\_increment,

productid int,

oldprice decimal(10, 2),

newprice decimal(10, 2),

changedate timestamp default current\_timestamp

);

delimiter $$

create trigger productpriceaudit

after update on products

for each row

begin

if old.price <> new.price then

insert into productaudit (productid, oldprice, newprice)

values (old.productid, old.price, new.price);

end if;

end $$

delimiter ;

39. automate backups using scheduled events.

create event dailybackup

on schedule every 1 day

do

call backupprocedure();

delimiter $$

create procedure backupprocedure()

begin

-- backup logic here

end $$

delimiter ;

40. Use covering indexes to improve query performance.

CREATE INDEX idx\_covering\_order ON OrderItems (OrderID, ProductID, Quantity);

41. Optimize queries using hints.

select /\*+ index(p idx\_product\_name) \*/ productname

from products p

where productname = 'laptop';

42. Use window functions for advanced analytics.

select customerid, orderdate, totalamount,

sum(totalamount) over (partition by customerid order by orderdate) as runningtotal

from orders;

43. Use CTEs for better query organization.

with recentorders as (

select orderid, customerid, orderdate

from orders

where orderdate > '2024-01-01'

)

select o.orderid, c.firstname, c.lastname

from recentorders o

join customers c on o.customerid = c.customerid;

44. Encrypt sensitive data.

alter table customers add column encryptedemail varbinary(256);

insert into customers (firstname, lastname, encryptedemail)

values ('john', 'doe', aes\_encrypt('john.doe@example.com', 'encryption\_key'));

45. Calculate the total revenue generated by each customer over their lifetime [Customer Lifetime Value (CLV)].

select c.customerid, c.firstname, c.lastname, sum(o.totalamount) as lifetimevalue

from customers c

join orders\_partitioned o on c.customerid = o.customerid

group by c.customerid, c.firstname, c.lastname

order by lifetimevalue desc;

46. Analyze the monthly sales trend over the last year.

select date\_format(orderdate, '%y-%m') as month, sum(totalamount) as monthlysales

from orders\_partitioned

where orderdate >= date\_sub(curdate(), interval 1 year)

group by date\_format(orderdate, '%y-%m')

order by month;

47. Segment customers based on how frequently they place orders.

select c.customerid, c.firstname, c.lastname, count(o.orderid) as orderfrequency

from customers c

join orders\_partitioned o on c.customerid = o.customerid

group by c.customerid, c.firstname, c.lastname

order by orderfrequency desc;

48. Calculate the average order value per customer.

select c.customerid, c.firstname, c.lastname, avg(o.totalamount) as averageordervalue

from customers c

join orders\_partitioned o on c.customerid = o.customerid

group by c.customerid, c.firstname, c.lastname

order by averageordervalue desc;

49. Identify products that are frequently bought together.

select oi1.productid as product1, oi2.productid as product2, count(\*) as frequency

from orderitems oi1

join orderitems oi2 on oi1.orderid = oi2.orderid and oi1.productid <> oi2.productid

group by oi1.productid, oi2.productid

order by frequency desc

limit 10;

50. Find the top 5 products generating the highest revenue.

select p.productid, p.productname, sum(oi.quantity \* oi.price) as totalrevenue

from orderitems oi

join products p on oi.productid = p.productid

group by p.productid, p.productname

order by totalrevenue desc

limit 5;

51. Calculate the average time taken to fulfill orders.

select o.orderid, o.orderdate, min(oi.shipmentdate) as firstshipmentdate,

datediff(min(oi.shipmentdate), o.orderdate) as fulfillmenttime

from orders\_partitioned o

join orderitems oi on o.orderid = oi.orderid

group by o.orderid, o.orderdate;

52. Calculate the churn rate of customers (customers who have not placed an order in the last 6 months).

select count(\*) as churnedcustomers

from customers c

left join orders\_partitioned o on c.customerid = o.customerid and o.orderdate >= date\_sub(curdate(), interval 6 month)

where o.orderid is null;

53. Calculate the average product rating for each category.

select cat.categoryname, avg(r.rating) as averagerating

from reviews r

join products p on r.productid = p.productid

join categories cat on p.categoryid = cat.categoryid

group by cat.categoryname;

54. Measure the inventory turnover ratio, which indicates how often inventory is sold and replaced.

select p.productid, p.productname,

sum(oi.quantity) / avg(p.stock) as inventoryturnoverratio

from orderitems oi

join products p on oi.productid = p.productid

group by p.productid, p.productname

order by inventoryturnoverratio desc;

55. Analyze how product sales are distributed across different regions.

select r.regionname, p.productname, sum(oi.quantity \* oi.price) as totalsales

from orderitems oi

join orders\_partitioned o on oi.orderid = o.orderid

join products p on oi.productid = p.productid

join customers c on o.customerid = c.customerid

join regions r on c.regionid = r.regionid

group by r.regionname, p.productname

order by totalsales desc;

56. Calculate the percentage of customers who made repeat purchases.

with customerorders as (

select customerid, count(orderid) as ordercount

from orders\_partitioned

group by customerid

)

select (sum(case when ordercount > 1 then 1 else 0 end) / count(\*)) \* 100 as retentionrate

from customerorders;

57. Calculate the average time between orders for customers who have placed more than one order.

with orderintervals as (

select customerid, orderdate,

lag(orderdate) over (partition by customerid order by orderdate) as previousorderdate

from orders\_partitioned

)

select customerid, avg(datediff(orderdate, previousorderdate)) as avgdaysbetweenorders

from orderintervals

where previousorderdate is not null

group by customerid;

58. Track the growth in popularity of products over time.

select p.productname, date\_format(o.orderdate, '%y-%m') as month, count(oi.productid) as orderscount

from orderitems oi

join orders\_partitioned o on oi.orderid = o.orderid

join products p on oi.productid = p.productid

group by p.productname, date\_format(o.orderdate, '%y-%m')

order by p.productname, month;

59. Identify the most active customers based on the number of orders they have placed.

select c.customerid, c.firstname, c.lastname, count(o.orderid) as ordercount

from customers c

join orders\_partitioned o on c.customerid = o.customerid

group by c.customerid, c.firstname, c.lastname

order by ordercount desc

limit 10;

60. Calculate the average discount given per order, assuming there is a Discount column in the OrderItems table.

select o.orderid, avg(oi.discount) as avgdiscount

from orderitems oi

join orders\_partitioned o on oi.orderid = o.orderid

group by o.orderid

order by avgdiscount desc;