PROJECT TARGET E-COMMERCE SQL

Hello

I am Vishalni Sivadasan. In this project, I have used all basic SQL queries to learn and apply them in this project.

01.

Basic SQL Syntax
SELECT Statement - Syntax: SELECT column_name FROM table_name;
WHERE Clause - Syntax: SELECT column_name FROM table_name WHERE condition;

02.

Data Definition Language (DDL)
Creating Tables - CREATE TABLE table_name (column1 datatype, column2 datatype, ...);
Altering Tables - ALTER TABLE table_name ADD column_name datatype;

03.

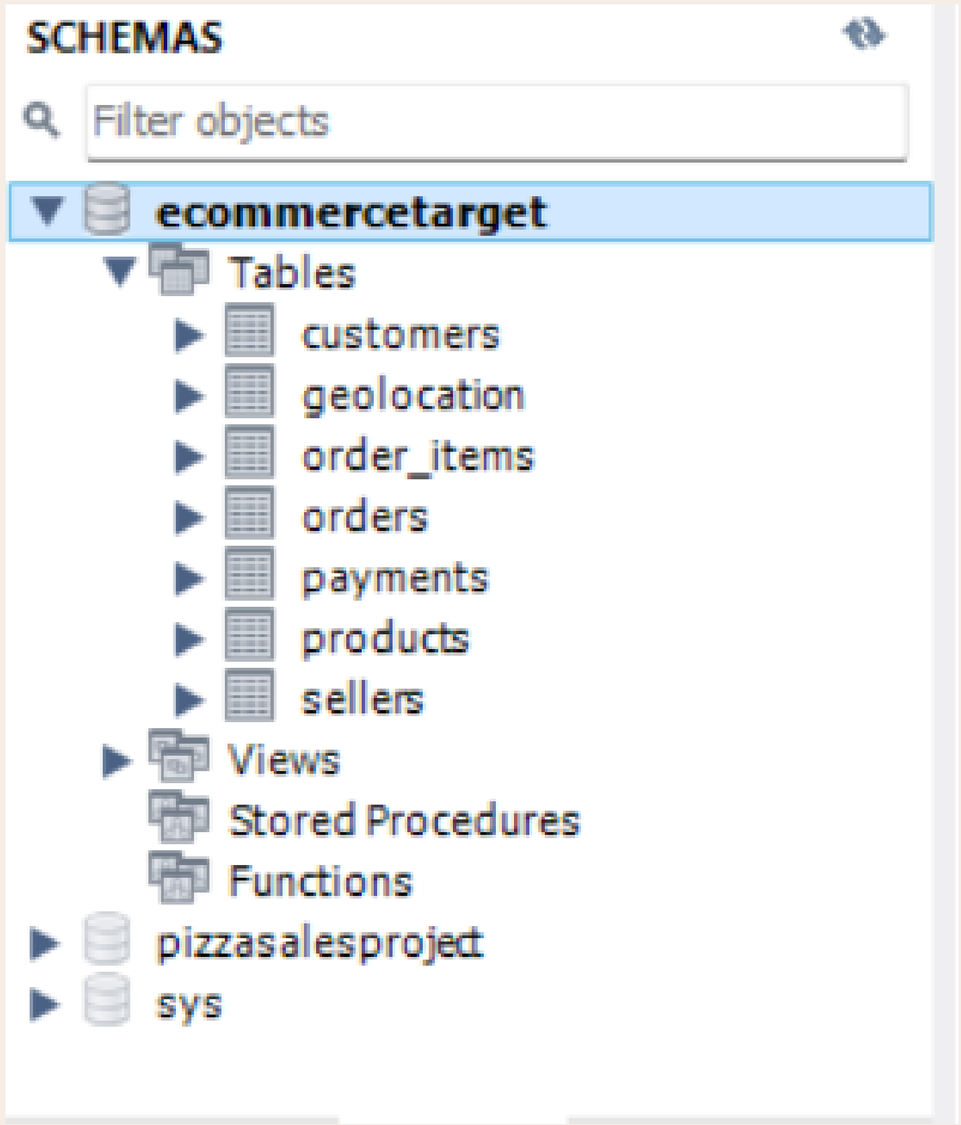
Data Manipulation Language (DML)
Inserting Data - INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);
Updating Data - UPDATE table_name SET column1 = value1 WHERE condition;
Deleting Data - DELETE FROM table_name WHERE condition;

04.

Data Query Language (DQL)
Selecting Data - `SELECT column_name(s) FROM table_name;
Filtering Results - `SELECT column_name(s) FROM table_name WHERE condition;

05.

Joins in SQL
Types of Joins - INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN
Aggregate Functions
Common Functions - `COUNT()`, `SUM()`, `AVG()`, `MAX()`, `MIN()`



01. This project is to learn all basic query in SQL

02. From this project, Everyone can know what is SQL.

03. Anyone can learn the basics of SQL from this project.

```sql
use ecommercetarget; -- use --> to highlight the particular DB in SCHEMAS

select * from orders; -- * --> everything

-- or --> to satisfy either condition
-- not --> to not satisfy the specified condition and shoulld be in ()
select * from customers where customer_state = "SP" or customer_state = "MG";
select * from customers where not(customer_state = "SP" or customer_state = "MG");

-- in --> to satisfy multiple condition at once
select * from customers where customer_state in ("SP","MG","SC","RJ");
-- not in --> to not satisfy the specified condition
select * from customers where customer_state not in ("SP","MG","SC","RJ");

-- and --> to satisfy all condition and can be used multiple condition to satisfy
SELECT * FROM payments where payment_type = "UPI" and payment_value >=500;
SELECT * FROM payments where payment_type = "credit_card"
and payment_value >=1000 and payment_installments = 1;

-- betwwen --> to get between specified numerical values
SELECT * FROM payments where payment_value between 1000 and 1500;
SELECT * FROM payments where payment_value >= 1000 and payment_value <= 1500; -- can also be written as
```

```sql
-- like --> to get pattern of the data
-- % --> used for unspecified letters of the remaining word
select * from products where product_category like "b%"; -- (a-z)% --> to get start of the specified alphabet
select * from products where product_category like "%h"; -- %(a-z) --> to get end of the specified alphabet
select * from products where product_category like "%b%"; -- %(a-z)% --> to get middle of the specified alphabet

-- order by --> to arrange by asc or desc

select * from order_items
order by price;

select * from order_items
order by price, freight_value desc; -- arrange price by asc and freight_value by desc according to price

select * from payments where payment_value = 0 order by payment_type desc;

-- limit --> gives the row what we want until
select * from payments limit 10; -- 1 parameter eg:10 --> gives the 10 rows
select * from payments limit 5,4; -- 2 parameter eg:5,4 --> skips 5 rows and gives the 4 rows
```

```sql
-- Aggregate functions

select sum(payment_value) from payments; -- sum --> to return total summed value(nou null)

select round(sum(payment_value)) from payments; -- round --> to round off the value
select round(sum(payment_value),2) from payments; -- number indicates to get specified value after decimal

select count(customer_city) from customers; -- count --> it returns no.of rows including null
select distinct customer_city from customers; -- distinct --> gives different values
select count(distinct customer_city) from customers; -- count(distinct) --> gives count of different values

select avg(payment_value) from payments; -- avg --> to return average value of expression

select max(payment_value) from payments; -- max --> to get the maximum value in the set
select min(payment_value) from payments; -- min --> to get the minimum value in the set
select count(payment_value) from payments where payment_value = 0; -- 9 payment_value is zero

select payment_value, ceil(payment_value), -- ceil --> gives upper round off value
floor(payment_value) from payments;        -- floor --> gives lower round off value
```

```sql
-- Text functions

select distinct geolocation_city,
length(geolocation_city), -- length --> gets the length of the alphabets of a word including space
length(trim(geolocation_city)) from geolocation; -- trim --> removes the white spaces

select geolocation_city, upper(geolocation_city) from geolocation; -- upper --> returns all aphabets in upper case
select geolocation_city, lower(geolocation_city) from geolocation; -- lower --> returns all aphabets in lower case

-- replace --> to replace letter or word
select geolocation_city, replace(geolocation_city, "a", "ã") from geolocation;
-- replace --> to replace letter or word after the word got replaced
select geolocation_city,
replace(replace(geolocation_city,  "a", "ã"), "pãulo", "paulo") as new_geolocation from geolocation;

-- concat --> to join two or more columns
select concat(geolocation_city, geolocation_state) from geolocation;
select concat(geolocation_city, " ", geolocation_state) from geolocation; -- can join with space
select concat(geolocation_city, " - ", geolocation_state) as city_state
from geolocation; -- can join with any characters
-- can join with other columns with new column
select *, concat(geolocation_city, "  ", geolocation_state) as city_state from geolocation;
```

```sql
-- Date functions
select order_delivered_customer_date,
day(order_delivered_customer_date),
dayname(order_delivered_customer_date),
month(order_delivered_customer_date),
monthname(order_delivered_customer_date),
year(order_delivered_customer_date) from orders;

select datediff(order_estimated_delivery_date,
order_delivered_customer_date) from orders; -- datediff --> to get difference between the dates

-- Time functions
select order_delivered_customer_date,
time(order_delivered_customer_date),
hour(order_delivered_customer_date),
minute(order_delivered_customer_date),
second(order_delivered_customer_date)
from orders;

select * from orders where order_delivered_customer_date = null; -- doesnt give the result
select * from orders where order_delivered_customer_date is null; -- gives the null values
-- doesn't give null value for order_approved_at for below queryS
select * from orders where order_approved_at and order_delivered_carrier_date is null;
select * from orders where order_approved_at is null and order_delivered_carrier_date is null;
```

```sql
-- Group by and order by

-- total order in each status
select order_status, count(order_status) total_order_count -- as is not required
from orders
group by order_status
order by total_order_count desc;

-- total product in each category
select product_category, count(product_category) total_product_category -- as is not required
from products
group by product_category
order by total_product_category;

-- total customer in each state
select customer_state, count(customer_state) total_customer_state -- as is not required
from customers
group by customer_state
order by total_customer_state desc;

select payment_type, round(sum(payment_value),2) from payments
group by payment_type;
```

```sql
-- WHERE CLAUSE
-- --> if you want to use seperate condition then WHERE should be used

select payment_type, round(avg(payment_value),2) from payments
where payment_installments <=5
group by payment_type;

-- HAVING CLAUSE
-- --> if you want to use condition on aggregate value then HAVING should be used

select payment_type, round(avg(payment_value),2) from payments
group by payment_type
having  avg(payment_value) <= 150;
```

```sql
-- JOINING --
-- inner joining
select orders.order_status, payments.payment_type
from orders join payments
on orders.order_id = payments.order_id
where order_status = "invoiced"
order by payment_type; -- select distinct order_status from orders;

select year(orders.order_purchase_timestamp) years,
round(sum(payments.payment_value),2)
from orders join payments
on orders.order_id = payments.order_id
group by years order by years;

select year(orders.order_purchase_timestamp) years,
round(sum(payments.payment_value),2)
from orders left join payments                          -- left join
on orders.order_id = payments.order_id
group by years order by years;

select year(orders.order_purchase_timestamp) years,
round(sum(payments.payment_value),2)
from orders right join payments                         -- Right join
on orders.order_id = payments.order_id
group by years order by years;

select t1.order_id, t2.order_id
from payments as t1, payments as t2                     -- self join
where t1.payment_type = t2.payment_type;
```

```sql
-- SUB QUERY

select category
from
(select products.product_category category, order_items.price price
from products join order_items
on products.product_id = order_items.product_id
order by price desc limit 5) as a ;

select distinct category
from
(select products.product_category category, order_items.price price
from products join order_items
on products.product_id = order_items.product_id
order by price desc ) as table1 limit 5;

select category
from
(select products.product_category category, sum(payments.payment_value) total_sales
from order_items join payments
on order_items.order_id = payments.order_id
join products
on order_items.product_id = products.product_id
group by category
order by total_sales desc limit 5) as table1;
```

```sql
-- COMMON TABLE EXPRESSION (CTE)]

-- act as a seperate table to fetch datas
with table1 as (select products.product_category category, sum(payments.payment_value) total_sales
from order_items join payments
on order_items.order_id = payments.order_id
join products
on order_items.product_id = products.product_id
group by category
order by total_sales desc)

select category from table1 ; -- select distinct category from table1

-- case operator

with table1 as (select products.product_category category, sum(payments.payment_value) total_sales
from order_items join payments
on order_items.order_id = payments.order_id
join products
on order_items.product_id = products.product_id
group by category
order by total_sales desc)

select *, case                                  -- select case
when total_sales <= 10000 then "low"            -- when total_sales <= 10000 then "low"
when total_sales <= 100000 then "high"          -- when total_sales <= 100000 then "high"
else "medium"                                   -- else "medium"
end as sales_type                               -- end as sales_type
from table1                                     -- from table1
order by sales_type
```

```sql
-- WINDOW FUNCTIONS

-- in total_sales_per_day given as cumilative series for result
select order_date, sales,
sum(sales) over(order by order_date) total_sales_per_day
from
(select date(orders.order_purchase_timestamp) order_date,
sum(payments.payment_value) sales
from orders join payments
on orders.order_id = payments.order_id
group by order_date) table1;

select order_date, sales,
round(sum(sales) over(order by order_date),2) total_sales_per_day
from
(select date(orders.order_purchase_timestamp) order_date,
sum(payments.payment_value) sales
from orders join payments
on orders.order_id = payments.order_id
group by order_date) table1;
```

```sql
-- RANKING FUNCTION

with table1 as (select products.product_category category, sum(payments.payment_value) total_sales
from order_items join payments
on order_items.order_id = payments.order_id
join products
on order_items.product_id = products.product_id
group by category), -- (",") --> comma is used for also getting table 2 if you use 1 table comma is not required

-- select category, total_sales, rank() over(order by total_sales) low_high_ranking
-- from table1 -- for ranking total_sales small to big value

-- select category, total_sales, rank() over(order by total_sales desc) high_low_ranking
-- from table1 -- for ranking total_sales big to small value

-- if have same value with multiple values then for ranking total_sales dense ranking should be used
-- select category, total_sales, dense_rank() over(order by total_sales) dense_ranking
-- from table1;

table2 as (select category, total_sales, rank() over(order by total_sales desc) high_low_ranking
from table1) -- with --> with also not needed for calling the second CTE table

select category, total_sales from table2
 where high_low_ranking <= 5
```

```sql
-- CREATE VIEW --> it will create a virtual table in schemas under views
-- it can also be treate as seperate table
-- it can be called as table name by product_category_sales for further analysis

create view product_category_sales as
select products.product_category category, sum(payments.payment_value) total_sales
from order_items join payments
on order_items.order_id = payments.order_id
join products
on order_items.product_id = products.product_id
group by category;
```

THANK YOU VERY MUCH!