Yellow Part – Codes

Green Parts – PPT

Blue Parts – information on script/screencasting

Hello everyone and welcome to this session. I hope all of you are doing great. Today we will work around on creating a MySQL database as per given instructions. And once the database is created, we will perform queries on the same. The queries that we will run will be based on the problem statements that's shared with us. I will make sure to explain each and every code to all of you so that you find no difficulty in understanding both the code and why the same is being used. So let's start the journey, shall we ?

We will use MySQL workbench for our project today , we can download and install workbench from this site –

<https://dev.mysql.com/downloads/workbench/>

installation steps to be shown –

Now that we have workbench installed , let’s note down our password because we need to remember the same. Our user is ‘root’ here , from top left corner open sql scripts , open a new script and let’s start coding.

The database that we will create will handle a grocery store data. The structure will be as given below -

DataBase - Grocery store

3 Tables -

Items -

- Product Name

- Price/unit

- In Stock

- Item\_id - Primary Key

The SQL code for this table is as given , we will discuss the entire code as well here ,

CREATE TABLE `groccery\_store`.`items` (

`product\_name` VARCHAR(45) NOT NULL,

`price\_per\_unit` INT NOT NULL,

`in\_stock` INT NOT NULL,

`product\_id` INT NOT NULL,

PRIMARY KEY (`product\_id`));

Let’s go one by one , at first we are just stating that the items table is to be created within the grocery store database . Then we are putting up columns sequentially, first column product name denotes that it will contain varchar or variable characters and the maximum length of the same would be 45. By not null we simply mean that it should not contain any null or missing values. Next where we see INT as a characteristic of the column it means that the records to be inserted here should be of integer type , Then let’s come to the Primary key which in here we stated as ‘product\_id’. This denotes that product\_id here is an unique column which will assist us to join this table with others in the database.

For PPT –

Primary Key –

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

Customers -

- Cust\_name

- Phone\_num

- Purchased\_item\_id

- Purchase Qty

- Cust\_id - Primary Key

The SQL code for this table is as given , we will discuss the entire code as well here ,

CREATE TABLE `groccery\_store`.`customers` (

`Cust\_Name` VARCHAR(45) NULL,

`Phone\_Num` BIGINT(15) NOT NULL,

`Purchased\_Item\_id` INT NOT NULL,

`Transaction\_id` INT NOT NULL,

`Purchase\_Qty` INT NOT NULL,

`Cust\_id` INT NOT NULL,

PRIMARY KEY (`Transaction\_id`),

UNIQUE INDEX `Transaction\_id\_UNIQUE` (`Transaction\_id` ASC) VISIBLE);

The code details are pretty much the same as above , the significant difference here is the ‘BIGINT’ which denotes a bigger integer number to be a constraint in the column itself –

For PPT –

Constraints –

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Prevents actions that would destroy links between tables

CHECK - Ensures that the values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column if no value is specified

CREATE INDEX - Used to create and retrieve data from the database very quickly

For PPT –

Transactions -

- Cust\_id - Primary Key

- Transaction\_id

- Purchase\_date

- Transactions\_value

- Product Name

The SQL code for this table is as given , we will discuss the entire code as well here ,

CREATE TABLE `groccery\_store`.`transaction` (

`transactions\_id` INT NOT NULL,

`cust\_id` INT NOT NULL,

`Purchase Date` DATE NOT NULL,

`product\_id` INT NOT NULL,

PRIMARY KEY (`transactions\_id`),

UNIQUE INDEX `transactions\_id\_UNIQUE` (`transactions\_id` ASC) VISIBLE);

Again the majority of the codes here are self-explanatory as stated above , the difference here as you can correctly guess here is the Date type column that’s written , it’s simply stating that that particular column will hold only date type of data in it.

Now that the tables are created , let us just have a quick refresher of the table details and it’s constraints , INT , BIGINT simply refers to numbers , BIGINT also takes a limit of numerical

characters as a argument, NOT NULL stands for the constraint that states , in this particular column , no null values can be present , primary key refers to a particular column that

holds unique value in respective to that table and will be used to connect/join the table with other ones in the database.

Let's now add some values in our created tables.

INSERT INTO `groccery\_store`.`customers` (`Cust\_Name`, `Phone\_Num`, `Purchased\_Item\_id`, `Transaction\_id`, `Purchase\_Qty`, `Cust\_id`)

VALUES ('Ron', '9836234356', '1345', '10', '2', '001');

INSERT INTO `groccery\_store`.`customers` (`Cust\_Name`, `Phone\_Num`, `Purchased\_Item\_id`, `Transaction\_id`, `Purchase\_Qty`, `Cust\_id`)

VALUES ('Harry', '9836234359', '1346', '11', '1', '002');

INSERT INTO `groccery\_store`.`customers` (`Cust\_Name`, `Phone\_Num`, `Purchased\_Item\_id`, `Transaction\_id`, `Purchase\_Qty`, `Cust\_id`)

VALUES ('Neville', '9836234351', '1347', '12', '3', '003');

INSERT INTO `groccery\_store`.`customers` (`Cust\_Name`, `Phone\_Num`, `Purchased\_Item\_id`, `Transaction\_id`, `Purchase\_Qty`, `Cust\_id`)

VALUES ('Neville', '9836234351', '1348', '13', '1', '003');

INSERT INTO `groccery\_store`.`customers` (`Cust\_Name`, `Phone\_Num`, `Purchased\_Item\_id`, `Transaction\_id`, `Purchase\_Qty`, `Cust\_id`)

VALUES ('Harry', '9836234171', '1346', '14', '1', '004');

As you can see , we have added values in the customers table , now there are something that needs to be pointed out here.. you can see the name harry coming twice in the data , but make sure to identify their cust\_id which is differenct and hence it shows that these records are for two different individuals. Also you can see that neville is also present twice in the data but this time their customer id is the same , which goes to show that they are same people , but the transaction id

and the item id is different , which goes to show that Neville bought two different items in two transactions.

Now let's go ahead and put some data in our items table ,

INSERT INTO `groccery\_store`.`items` (`product\_name`, `price\_per\_unit`, `in\_stock`, `product\_id`) VALUES ('Hair Oil', '130', '50', '1345');

INSERT INTO `groccery\_store`.`items` (`product\_name`, `price\_per\_unit`, `in\_stock`, `product\_id`) VALUES ('Shampoo', '85', '120', '1346');

INSERT INTO `groccery\_store`.`items` (`product\_name`, `price\_per\_unit`, `in\_stock`, `product\_id`) VALUES ('Rice', '25', '100', '1347');

INSERT INTO `groccery\_store`.`items` (`product\_name`, `price\_per\_unit`, `in\_stock`, `product\_id`) VALUES ('Icecream', '90', '35', '1348');

here value addition is pretty straight forward , different items having different Item\_id and the stock here referes to their inventory and unit in numbers.

It's time for the transactions table , we will add values in there as well.

INSERT INTO `groccery\_store`.`transaction` (`transactions\_id`, `cust\_id`, `Purchase Date`, `product\_id`) VALUES ('10', '1', '2023-01-11', '1345');

INSERT INTO `groccery\_store`.`transaction` (`transactions\_id`, `cust\_id`, `Purchase Date`, `product\_id`) VALUES ('11', '2', '2022-12-10', '1346');

INSERT INTO `groccery\_store`.`transaction` (`transactions\_id`, `cust\_id`, `Purchase Date`, `product\_id`) VALUES ('12', '3', '2023-01-05', '1347');

INSERT INTO `groccery\_store`.`transaction` (`transactions\_id`, `cust\_id`, `Purchase Date`, `product\_id`) VALUES ('13', '3', '2023-01-08', '1348');

INSERT INTO `groccery\_store`.`transaction` (`transactions\_id`, `cust\_id`, `Purchase Date`, `product\_id`) VALUES ('14', '4', '2022-12-21', '1346');

As we can see here all the values are submitted and as discussed prior , transactions\_id is considered

as the primary key here.

Now that the tables are created we will try to solve some basic problem statements that can be generated from this data with the help of SQL query writing ,

We can see that in the transactions table purchase date column is written with a space. let's fix this and change the column name with an \_ ,

Here is the SQL code for the same ,

ALTER TABLE `groccery\_store`.`transaction`

CHANGE COLUMN `Purchase Date` `purchase\_date` DATE NOT NULL ;

The reason why we are doing this is because date is a valid keyword in mysql so if you provide space before date , it will not be considered as an individual column in our query rather will be treated as an keyword , hence this fix is required. As you can see we have changed the column with the help of Alter syntax.

Lets write a query to generate a column containing only years of purchase in the transactions table.

Because when performing analysis based on only the years stated in our table , it is helpful to extract only the year. Same way we can extract month and day as well.

select cust\_id , year(purchase\_date) from transaction;

select cust\_id , month(purchase\_date) from transaction;

select cust\_id , day(purchase\_date) from transaction;

In the below code you can easily see that how we can extract specific year month and day from a date type column.

Now lets Write a query to showcase only the customer name and the product they purchased as per the database ,

select c.cust\_name , i.product\_name

from customers c

join items i

on c.Purchased\_Item\_id = i.product\_id;

We will perform a join here based on the product id , as it's present in both of the tables . One point to note here is that in both tables they are written differently

, in one table its written as product\_id and in another its Purchased\_Item\_id so we will need to make sure to provide correct name while joining them.

Now lets write a query to showcase the customer name , the product they purchased and total purchase value in the result

select c.cust\_name , i.product\_name, (c.purchase\_qty\*i.price\_per\_unit) as purchase\_value

from customers c

join items i

on c.Purchased\_Item\_id = i.product\_id;

We will perform the same join as above to connect items and customers table but while providing the select clause we will use a multiplication of purchase quantity and item price. Be very careful while performing the calculation here and accessing the data from the right column or else we will end up getting wrong results.

Now lets Write a query to display the customer name who have made more than one purchase

select distinct(cust\_name) from customers where cust\_id in(

select cust\_id

from transaction

group by cust\_id

having count(cust\_id)>1);

We will use the count method here and will chcek that if there is any two transactions where the cust\_id is same or in the other words , if same cust\_id gets repeated in

two transactions. And finding out the cust id will be done here using a subquery.

For ppt –

Subquery –

A subquery in MySQL is a query, which is nested into another SQL query and embedded with SELECT, INSERT, UPDATE or DELETE statement along with the various operators. We can also nest the subquery with another subquery. A subquery is known as the inner query, and the query that contains subquery is known as the outer query. The inner query executed first gives the result to the outer query, and then the main/outer query will be performed. MySQL allows us to use subquery anywhere, but it must be closed within parenthesis. All subquery forms and operations supported by the SQL standard will be supported in MySQL also.

The following are the rules to use subqueries:

Subqueries should always use in parentheses.

If the main query does not have multiple columns for subquery, then a subquery can have only one column in the SELECT command.

We can use various comparison operators with the subquery, such as >, <, =, IN, ANY, SOME, and ALL. A multiple-row operator is very useful when the subquery returns more than one row.

We cannot use the ORDER BY clause in a subquery, although it can be used inside the main query.

If we use a subquery in a set function, it cannot be immediately enclosed in a set function.

The following are the advantages of using subqueries:

The subqueries make the queries in a structured form that allows us to isolate each part of a statement.

The subqueries provide alternative ways to query the data from the table; otherwise, we need to use complex joins and unions.

The subqueries are more readable than complex join or union statements.

Now lets Write a query to display the amount spent on each purchase by the customers

select c.cust\_name , (c.purchase\_qty\*i.price\_per\_unit) as purchase\_value , i.product\_name

from customers c

join items i

on c.Purchased\_Item\_id = i.product\_id;

We will perform a join here on the items table and the customers table , from the items table we will be able to get the item price per unit and from the customers table we

will be able to extract the purchase quantity , multiplying these two , we will be getting our results in hand.

Now let's try to find out the products which are priced more than 100 alongside the product name.

select product\_name from items

where price\_per\_unit>100;

We can simply perform a where clause here to set up the logic based on which the filteration will take place. The clause will state to display all the products priced accordingly.

Now let's extend our query as stated above. Let's try to find out customers purchasing the product for which price is more than 100.

select c.cust\_name , i.product\_name from items i join customers c

on i.product\_id = c.Purchased\_Item\_id

where price\_per\_unit>100;

With the above written query we will simply add on a join statement and the join this time will take place with the product\_id/Purchased\_Item\_id.

Now lets try something different , for a fact we know that all columns are not present in each table in our given database. So let's now try see if we can fit different column data

values in a single column in our output.

We will use the union statement here , it will provide us all the data requested in a single column irrespective of the fact of them containing similar type of data or values. to

explain it further , date type data , integer , bigint type data everything can be present at once in a single column , we will take into consideration the product\_name , cust\_name

purchase\_date here

select product\_name from items

union all

select cust\_name from customers

union all

select purchase\_date from transaction;

Now let's go ahead and create view for extracting a specific set of data. Suppose we want to create a view which will only showcase the product details where only 1 product

was purchased.

create view

productone as

select \* from customers

where Purchase\_Qty = 1;

Now on your left hand tab , you would be able to see view named as 'product one' under view section. And you can access the view seprately , it will act as an individual table to

a certain extent.

For PPT -

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or

more real tables in the database. You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

let's now , how we can incorporate other important keywords in MySQL for our database exploration , let's talk about , And , or and not operators in MySQL.

For PPT -

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

The AND operator displays a record if all the conditions separated by AND are TRUE.

The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

SELECT \* FROM Customers

WHERE Purchased\_item\_id = 1346 AND Purchase\_Qty = 1;

This will display all the results for the product\_id 1346 marked item and the same sold of only one unit.

We will see a different result if we use 'or' inplace of 'and' as in that case any of the given condition if turns out to be true will display the results accordingly.

SELECT \* FROM Customers

WHERE Purchased\_item\_id = 1346 or Purchase\_Qty = 1;

The not keywords help us in negating the given statement , which simply means that the output we will get , simply must not include any item containing inside the not argument.

Suppose we write a query like below ,

SELECT \* FROM Customers

WHERE NOT Purchased\_item\_id = 1346;

here we will get all the details of customers table except the ones where 1346 stands to be a product id.

Let’s talk about order by statement , This statement helps us to order or sort our data on a specific parameter that’s given to us. Suppose we want to order by our data based on the purchased quantity we can write a query as like

select \* from customers

order by Purchase\_Qty ;

As it’s seen the purchase quantity column is sorted based on lowest to highest order. if we want this order to be reversed, we can do that by simply adding a descending argument in our query as below.

select \* from customers

order by Purchase\_Qty desc;

Now as you can see the purchase quantity given is ranked from highest to lowest in the output present.

We can also use multiple columns into our data for ordering by as per requirement, suppose we want to get all the data from the customer table where purchase quantity will be sorted in descending order and customer id will be sorted in ascending order. We can write a query as below ,

select \* from customers

order by Purchase\_Qty desc, cust\_id asc;

We can use Limit statement to control the number of records to be displayed in the output statement that we will get , a query like below will end up fetching only 2 records from all the records available under the logics provided in the query ,

select \* from customers

order by Purchase\_Qty desc, cust\_id asc limit 2;

While the query logic remains the same , with the help of the limit statement we will be able to get the output of only 2 records among all the records present in the table.

We often need to change the content of our existing table based on some given condition, to achieve the same we need the update statement to be used , it takes into consideration based on which it will filter the data and then will perform the updating in the data itself. Suppose I want to change the name of the customer to “peter” wherever customer id is 2 , we can run a query as below ,

UPDATE Customers

SET Cust\_name = 'peter'

WHERE cust\_id = 2;

As it can be seen here , the set statement goes hand in hand with the update statement for execution.

I Hope this session was helpful for all of you , we developed a database and ran few problem statement queries in it. Stay tuned , we will come back with more such projects .