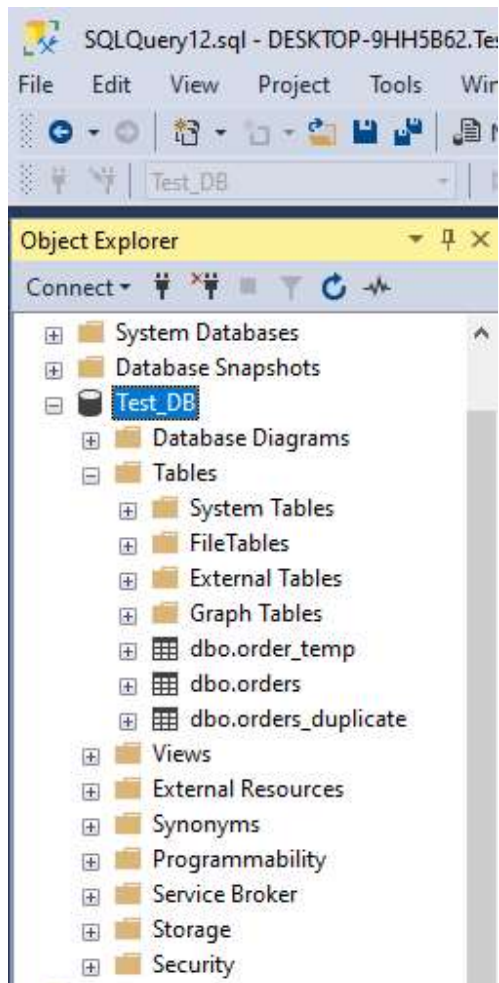


Name: Nitin

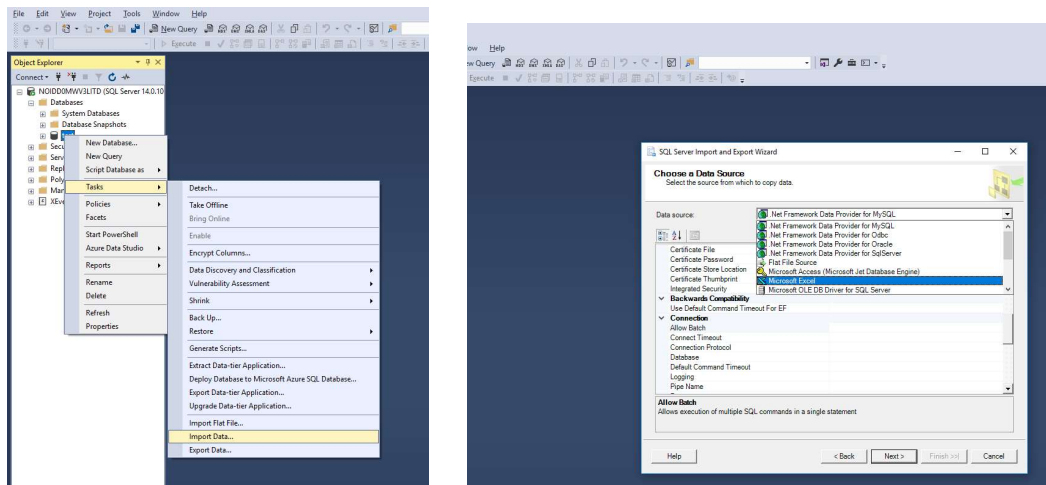
Email Id: nknitinyadav5@gmail.com

### Assessment 1

Create New Database "Test\_DB"

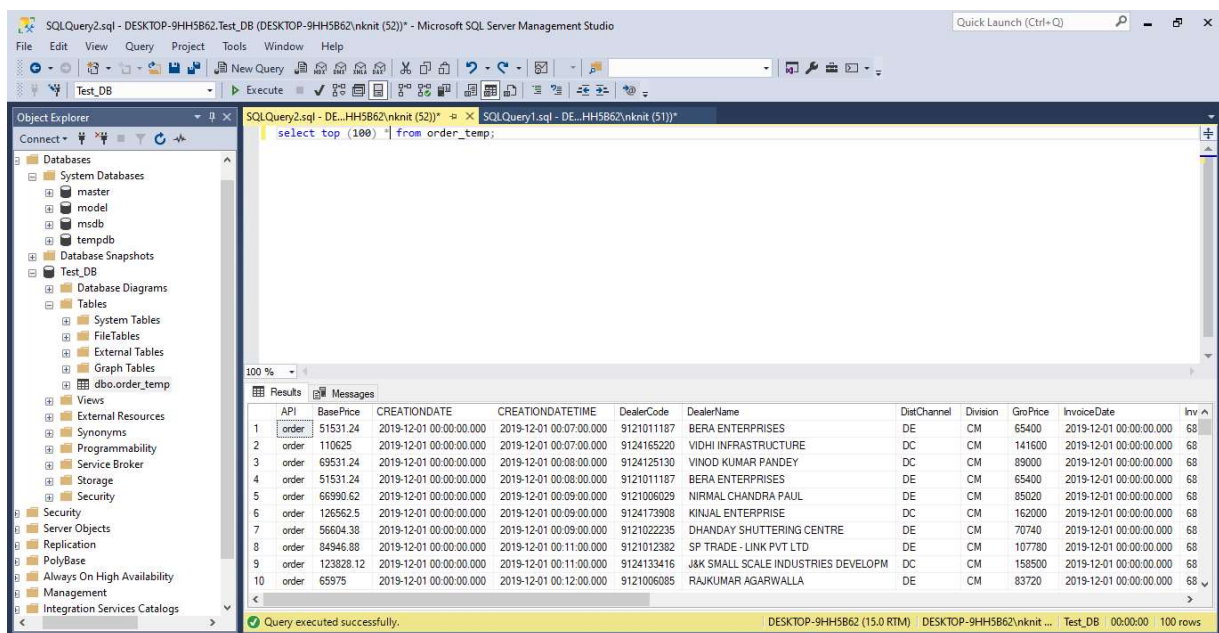


Importing Data From excel sheet in a temporary table with name “order\_temp” :



To check that table and data in that is importind properly I checked it with the following query:

Select top 100 from order\_temp;



Create two tables with name “orders” and “orders\_duplicate” table with following query:

**Orders**

```
CREATE TABLE orders (  
  API      nvarchar(255),  
  BasePrice float,  
  CREATIONDATE      datetime,  
  CREATIONDATETIME      datetime,  
  DealerCode      float,  
  DealerName      nvarchar(255),  
  DistChannel      nvarchar(255),  
  Division  nvarchar(255),  
  GroPrice  float,  
  InvoiceDate      datetime,  
  InvoiceNor      float,  
  MRP      float,  
  MaterialDescription      nvarchar(255),  
  MaterialGrp2      nvarchar(255),  
  MaterialNumber  float,  
  Month  nvarchar(255),  
  NSP      float,  
  OfficerName      nvarchar(255),  
  Officercode      nvarchar(255),  
  OrderCreationdate datetime,  
  QTY      float,  
  RSOCode nvarchar(255),  
  Region  nvarchar(255),  
  SalesOfficeCode  nvarchar(255),  
  SuppPLname  nvarchar(255),  
  SuppPlant   nvarchar(255),  
  Tax      nvarchar(255),  
  Year      nvarchar(255),  
);
```

**orders\_duplicate**

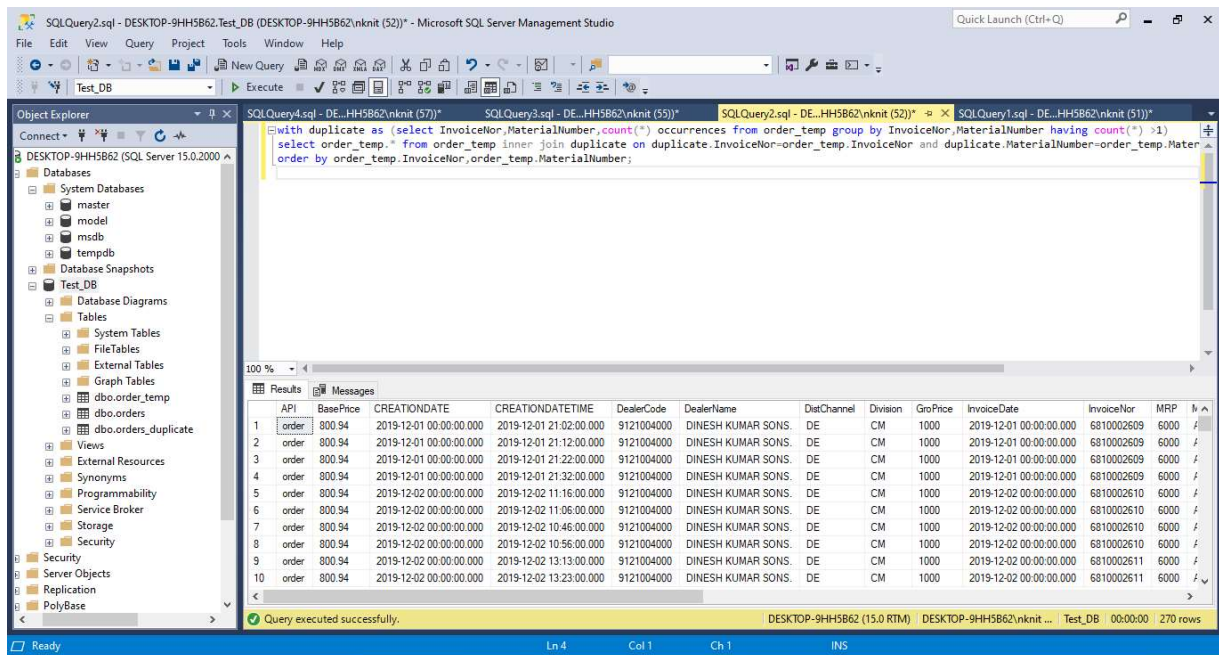
```
CREATE TABLE orders_duplicate (  
  API      nvarchar(255),  
  BasePrice float,  
  CREATIONDATE      datetime,  
  CREATIONDATETIME      datetime,  
  DealerCode      float,  
  DealerName      nvarchar(255),  
  DistChannel      nvarchar(255),  
  Division  nvarchar(255),  
  GroPrice  float,  
  InvoiceDate      datetime,  
  InvoiceNor      float,  
  MRP      float,  
  MaterialDescription      nvarchar(255),  
  MaterialGrp2      nvarchar(255),  
  MaterialNumber  float,  
  Month  nvarchar(255),  
  NSP      float,
```

```
OfficerName      nvarchar(255),
Officercode      nvarchar(255),
OrderCreationdate datetime,
QTY              float,
RSOCode nvarchar(255),
Region  nvarchar(255),
SalesOfficeCode  nvarchar(255),
SuppPLname      nvarchar(255),
SuppPlant       nvarchar(255),
Tax             nvarchar(255),
Year            nvarchar(255),
);
```

We have a table named “orders” which is empty now. In this table I have to insert only the unique data from “order\_temp”

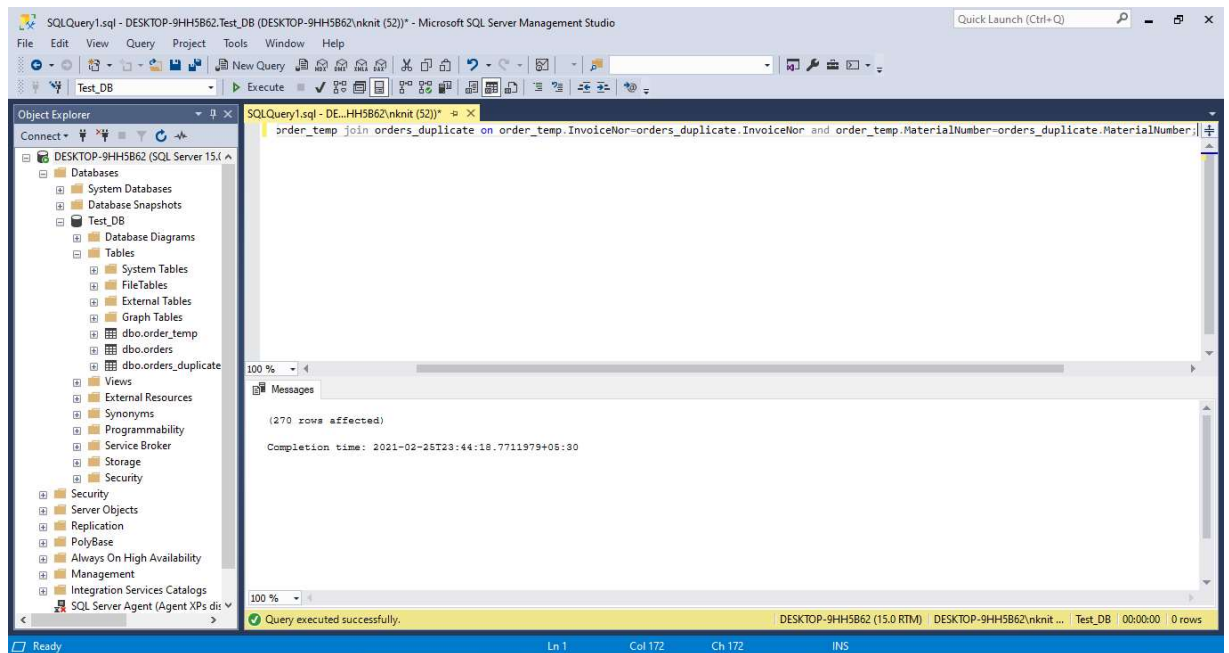
1. UPLOAD DATA IN ORDER TABLE WITH UNIQUE KEY ' InvoiceNor, MaterialNumber'.

```
Query: with duplicate as (select InvoiceNor,MaterialNumber,count(*) occurrences from
order_temp group by InvoiceNor,MaterialNumber having count(*) >1)
select order_temp.* from order_temp inner join duplicate on
duplicate.InvoiceNor=order_temp.InvoiceNor and
duplicate.MaterialNumber=order_temp.MaterialNumber
order by order_temp.InvoiceNor,order_temp.MaterialNumber;
```



1. Selecting all these entries, copied them and pasted in “orders\_duplicate” table now this table has 270 rows, now all these entries have duplicate rows which are not required to be inserted in “orders” table. So after keeping them safe in a temporary table I have deleted these entry from “order\_temp” using following query.

```
delete order_temp from order_temp JOIN orders_duplicate on
order_temp.InvoiceNor = orders_duplicate.InvoiceNor AND
order_temp.MaterialNumber = orders_duplicate.MaterialNumber
```

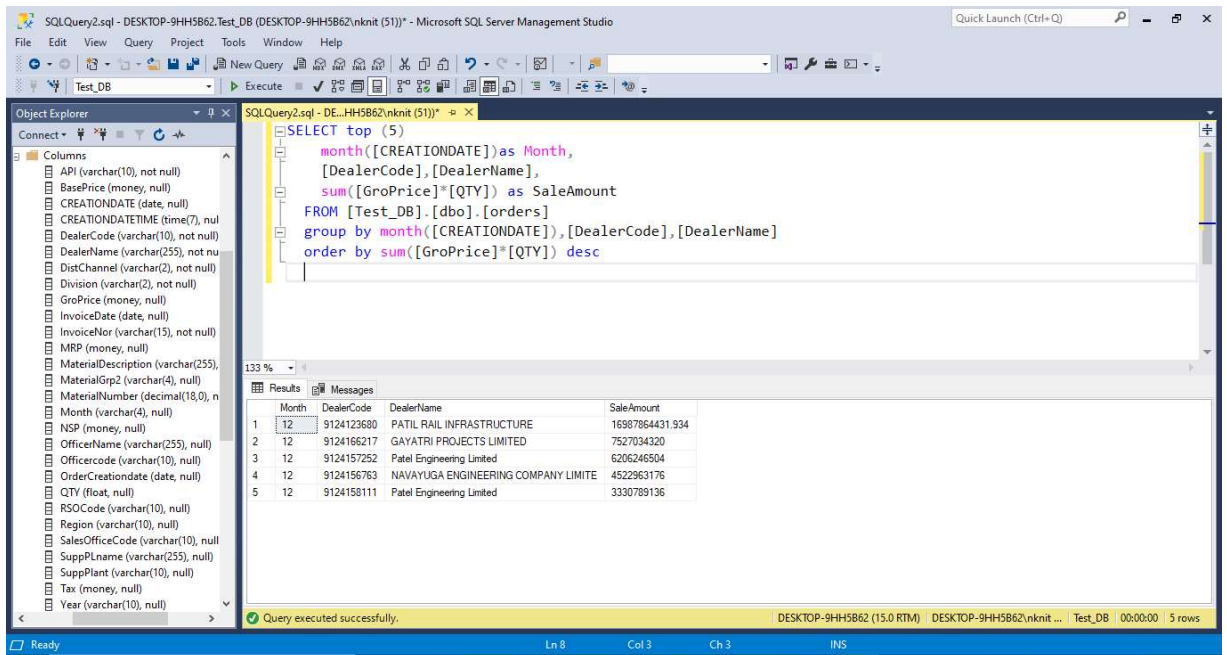


2. Now 270 rows are deleted and we have distinct rows and now inserting these rows to our “orders” table, using following query:  

```
Insert into orders  
select * from order_temp
```
3. 78715 rows are inserted.

Query 2 : FIND TOP 5 DEALER IN MONTH ON BASIS OF AMOUNT

```
SELECT top (5)
    month([CREATIONDATE]) as Month,
    [DealerCode],[DealerName],
    sum([GroPrice]*[QTY]) as SaleAmount
FROM [Test_DB].[dbo].[orders]
group by month([CREATIONDATE]),[DealerCode],[DealerName]
order by sum([GroPrice]*[QTY]) desc
```



The screenshot shows the Microsoft SQL Server Management Studio interface. The SQL Query window contains the following query:

```
SELECT top (5)
    month([CREATIONDATE]) as Month,
    [DealerCode],[DealerName],
    sum([GroPrice]*[QTY]) as SaleAmount
FROM [Test_DB].[dbo].[orders]
group by month([CREATIONDATE]),[DealerCode],[DealerName]
order by sum([GroPrice]*[QTY]) desc
```

The Results pane displays the following data:

	Month	DealerCode	DealerName	SaleAmount
1	12	9124123680	PATIL RAIL INFRASTRUCTURE	16987864431.934
2	12	9124166217	GAYATRI PROJECTS LIMITED	7527034320
3	12	9124157252	Patel Engineering Limited	6206246504
4	12	9124156763	NAVAYUGA ENGINEERING COMPANY LIMITE	4522963176
5	12	9124158111	Patel Engineering Limited	3330789136

The status bar at the bottom indicates "Query executed successfully." and "5 rows".

Query 3 : FIND LAST ORDER OF EACH DEALER

```
SELECT * FROM (
SELECT [DealerCode],[DealerName],[InvoiceNor],[CREATIONDATE], [CREATIONDATETIME],Rank()
    over (Partition BY DealerCode
        ORDER BY DealerCode,CREATIONDATE desc,CREATIONDATETIME desc) AS Rank
FROM [Test_DB].[dbo].[orders]
) rs where rank = 1
```



SQLQuery4.sql - DESKTOP-9HH5B62\Test\_DB (DESKTOP-9HH5B62\unknit (58)) - Microsoft SQL Server Management Studio

```

SELECT * FROM (
SELECT [DealerCode],[DealerName],[InvoiceNor],[CREATIONDATE], [CREATIONDATETIME],Rank()
over (Partition BY DealerCode
ORDER BY DealerCode,CREATIONDATE desc,CREATIONDATETIME desc) AS Rank
FROM [Test_DB].[dbo].[orders]
) rs
where rank = 1

```

DealerCode	DealerName	InvoiceNor	CREATIONDATE	CREATIONDATETIME	Rank
9121000036	ACC Limited	6834536927	2019-12-12 00:00:00.000	2019-12-12 07:27:00.000	1
9121000369	ACC Limited	6834539341	2019-12-12 00:00:00.000	2019-12-12 12:39:00.000	1
9121000369	ACC WH-Vkaanagar	6834536141	2019-12-12 00:00:00.000	2019-12-12 01:24:00.000	1
9121000384	ACC WH-DELHI SSB	6834526125	2019-12-11 00:00:00.000	2019-12-11 05:59:00.000	1
9121000385	ACC WH-DELHI RAJAPARK	6834526028	2019-12-11 00:00:00.000	2019-12-11 05:17:00.000	1
9121000386	ACC WH-Jalandhar	6834523416	2019-12-10 00:00:00.000	2019-12-10 18:59:00.000	1
9121002003	ACC LIMITED	6834529792	2019-12-11 00:00:00.000	2019-12-11 13:49:00.000	1
9121004015	SHAH KIRTILAL HARILAL	6834491220	2019-12-07 00:00:00.000	2019-12-07 15:02:00.000	1
9121004016	SHRI RAM AGRO	6834536924	2019-12-12 00:00:00.000	2019-12-12 07:26:00.000	1
9121004029	SHREE VISHWAKARMA SALES AGENCY	6834518981	2019-12-10 00:00:00.000	2019-12-10 14:01:00.000	1
9121004054	SHRI PAGLAM CEMENT CORPORATION	6834534981	2019-12-11 00:00:00.000	2019-12-11 20:36:00.000	1

Query executed successfully. DESKTOP-9HH5B62 (15.0 RTM) DESKTOP-9HH5B62\unknit ... Test\_DB 00:00:00 9,973 rows

#### Query 4. FIND AVERAGE AMOUNT OF EACH DEALER

```

SELECT [DealerCode],
avg([GroPrice]*[QTY]) as avgAmount
FROM [Test_DB].[dbo].[orders]
group by [DealerCode]
order by [DealerCode]

```

SQLQuery5.sql - DESKTOP-9HH5B62\Test\_DB (DESKTOP-9HH5B62\unknit (65)) - Microsoft SQL Server Management Studio

```

SELECT [DealerCode],
avg([GroPrice]*[QTY]) as avgAmount
FROM [Test_DB].[dbo].[orders]
group by [DealerCode]
order by [DealerCode]

```

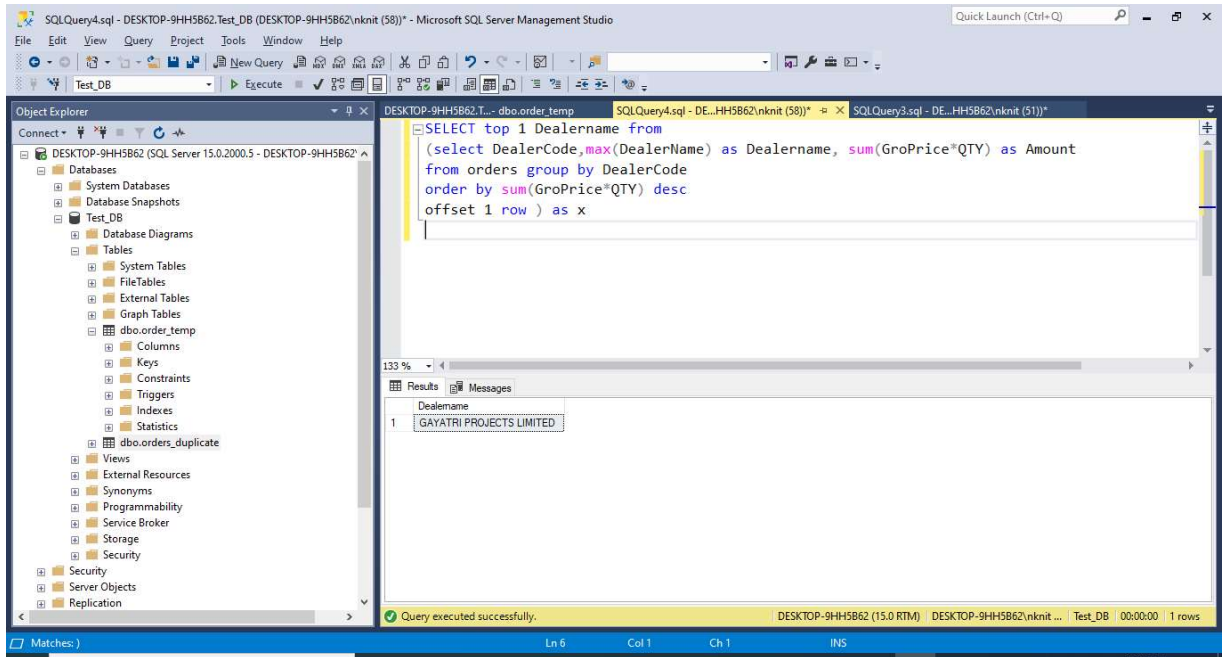
DealerCode	avgAmount
9121000001	3967091.44333333
9121000025	4618558.232
9121000036	9333896.4625
9121000061	8889527.17
9121000323	4216572.92307692
9121000331	3268134.7826087
9121000345	10076998.2857143
9121000346	5195240
9121000348	7661465
9121000349	2235600
9121000351	7208035.45714286

Query executed successfully. DESKTOP-9HH5B62 (15.0 RTM) DESKTOP-9HH5B62\unknit ... Test\_DB 00:00:00 9,466 rows



#### Query 5. FIND 2ND HIGHEST BILLED AMOUNT DEALER NAME

```
SELECT top 1 Dealername from
(select DealerCode,max(DealerName) as Dealername, sum(GroPrice*QTY) as Amount
from orders group by DealerCode
order by sum(GroPrice*QTY) desc
offset 1 row ) as x;
```



#### Query 6. FIND HOURLY AMOUNT BILLED

```
with temp as (
SELECT DealerCode,DealerName, CONVERT(DATETIME, CONVERT(CHAR(8), [CREATIONDATE], 112) + '
' + CONVERT(CHAR(8), [CREATIONDATETIME], 108)) as saletime,
[GroPrice]*[QTY] as amount
FROM [orders]), temp2 as(
select DealerCode,max(DealerName) as DealerName,CONVERT(DATE,temp.saletime) AS CallDate,
DATEPART(HOUR,temp.saletime) AS CallHour,
avg(amount) as avgamt
from temp
group by DealerCode,CONVERT(DATE,temp.saletime), DATEPART(HOUR,temp.saletime))
select DealerCode,max(DealerName) as DealerName,avg(avgamt) as HourlyAvgAmt from temp2
group by DealerCode
```

Comments:

1) --CONVERT(CHAR(8), [CREATIONDATETIME], 108) : this function will convert the given datetime to hh:mm:ss as string.

2) -- CONVERT(CHAR(8), [CREATIONDATE], 112) : this function will convert the given datetime to yyyy-mm-dd as string.

3) -- CONVERT(DATETIME, CONVERT(CHAR(8), [CREATIONDATE], 112) + ' ' + CONVERT(CHAR(8), [CREATIONDATETIME], 108)) : this function will convert the resulted string of format 'yyyy-mm-dd hh:mm:ss' (2019-12-01 01:28:00.000) to DATETIME

The screenshot displays the Microsoft SQL Server Management Studio interface. The 'Object Explorer' on the left shows the database structure for 'DESKTOP-9HH5B62'. The 'Query Editor' in the center contains a T-SQL query that uses a temporary table 'temp' to calculate the hourly average amount for each dealer. The query is as follows:

```
with temp as (
SELECT DealerCode, DealerName, CONVERT(DATETIME, CONVERT(CHAR(8), [CREATIONDATE], 112) + ' ' + 
[GroPrice]*[QTY] as amount
FROM [orders]), temp2 as(
select DealerCode, max(DealerName) as DealerName, CONVERT(DATE, temp.saletime) AS CallDate,
DATEPART(HOUR, temp.saletime) AS CallHour,
avg(amount) as avgamt
from temp
group by DealerCode, CONVERT(DATE, temp.saletime), DATEPART(HOUR, temp.saletime))
select DealerCode, max(DealerName) as DealerName, avg(avgamt) as HourlyAvgAmt from temp2 group by DealerCode
```

The 'Results' pane at the bottom shows the output of the query, which is a table with three columns: DealerCode, DealerName, and HourlyAvgAmt. The table contains 11 rows of data.

DealerCode	DealerName	HourlyAvgAmt
9121000001	ACC LIMITED	3947921.5375
9121000345	ACC LTD-WH-AMRITSAR-AARVEE	10104754.83333333
9121000346	ACC LTD-WH-JALLANDHAR-RV	5841115.625
9121000348	ACC WH-Bhatinda -Bachan Tea & Co	7661465
9121000360	ACC MANGALORE	4099775.57758621
9121000361	ACC-KOCHI-FOR-BCTC	2876390.76923077
9121000405	ACC LIMITED	5670000
9121002010	ACC LIMITED	2656589.6462963
9121004022	RADHAKRISHNA TRADERS	170000
9121004033	SHIV STEEL CORPORATION	257593.25
9121004034	SAJID CEMENT SUPPLIERS	3211610

The status bar at the bottom indicates that the query was executed successfully, returning 9,466 rows.

## Query 7 . FIND DATEWISE TOTAL QUATITY

```
select CreationDate, sum(QTY) as quantity from orders group by CreationDate order by CreationDate
```

The screenshot displays the Microsoft SQL Server Enterprise Manager interface. The left pane shows the 'Object Explorer' with the 'Test\_DB' database selected. The right pane shows the 'Query Editor' with the following SQL query:

```
select CreationDate, sum(QTY) as quantity from orders group by CreationDate order by CreationDate
```

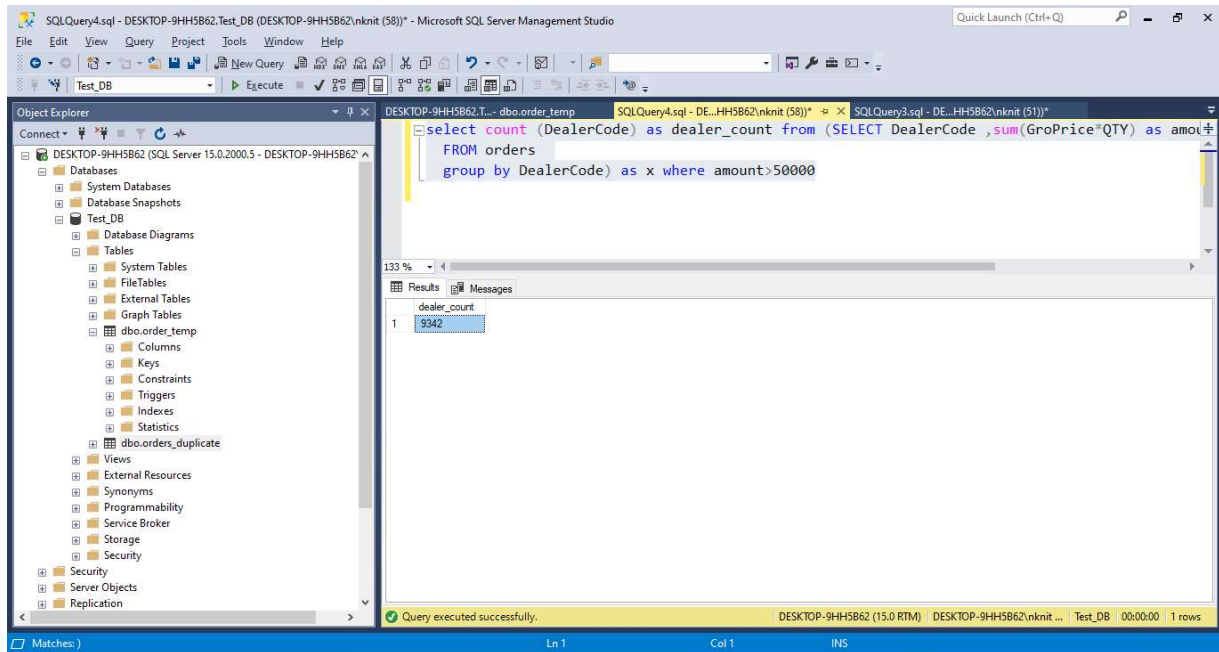
The query results are displayed in the 'Results' pane, showing a table with two columns: 'CreationDate' and 'quantity'. The results are as follows:

CreationDate	quantity
2019-12-01 00:00:00.000	38297.368
2019-12-02 00:00:00.000	49729.21
2019-12-03 00:00:00.000	61794.202
2019-12-04 00:00:00.000	68979.364
2019-12-05 00:00:00.000	66560.99
2019-12-06 00:00:00.000	73497.147
2019-12-07 00:00:00.000	87242.8660000001
2019-12-08 00:00:00.000	56527.027
2019-12-09 00:00:00.000	70445.362
2019-12-10 00:00:00.000	75534.346
2019-12-11 00:00:00.000	81296.0820000001
2019-12-12 00:00:00.000	57961.716

The status bar at the bottom indicates that the query was executed successfully, returning 12 rows.

Query 8. FIND TOTAL NO OF DEALER WHO HAVE AMOUNT GREATER THAN 50000

```
select count (DealerCode) as dealer_count from (SELECT DealerCode ,sum(GroPrice*QTY) as amount
FROM orders group by DealerCode) as x where amount>50000;
```



The screenshot displays the Microsoft SQL Server Management Studio interface. The 'Object Explorer' on the left shows the database structure for 'DESKTOP-9HH5B62'. The 'Query Editor' in the center contains the following SQL query:

```
select count (DealerCode) as dealer_count from (SELECT DealerCode ,sum(GroPrice*QTY) as amount
FROM orders group by DealerCode) as x where amount>50000;
```

The 'Results' pane at the bottom shows the output of the query, which is a single row with the value 9342 under the column 'dealer\_count'.

dealer_count
9342

The status bar at the bottom indicates that the query was executed successfully and returned 1 row.

QUERY 9. PUT A LABEL **TOP** IF DEALER AMOUNT IN MONTH IS GEATER THAN 10000 ELSE LABEL **AVG** AND SHOW LIST

```
with temp as (
SELECT [DealerCode],
       month([CREATIONDATE]) as MonthOfSale,
       sum([GroPrice]*[QTY]) as Amount
FROM [orders]
group by [DealerCode],
month([CREATIONDATE])
)
select [DealerCode],monthOfSale,amount,IIF(amount > 10000, 'TOP', 'AVG') as Label
from temp
```

The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor contains the following SQL code:

```
with temp as (
SELECT [DealerCode],
       month([CREATIONDATE]) as MonthOfSale,
       sum([GroPrice]*[QTY]) as Amount
FROM [orders]
group by [DealerCode],
month([CREATIONDATE])
)
select [DealerCode],monthOfSale,amount,IIF(amount > 10000, 'TOP', 'AVG') as Label
from temp
```

The Object Explorer on the left shows the database structure, including tables like `API`, `BasePrice`, `CREATIONDATE`, `CREATIONDATETIME`, `DealerCode`, `DealerName`, `DistChannel`, `Division`, `GroPrice`, `InvoiceDate`, `InvoiceNor`, `MRP`, `MaterialDescription`, `MaterialGrp2`, `MaterialNumber`, `Month`, `NSP`, `OfficerName`, `Officercode`, `OrderCreationdate`, and `QTY`.

The Results pane at the bottom displays the output of the query, showing 11 rows of data with columns: DealerCode, monthOfSale, amount, and Label. All rows are labeled 'TOP'.

DealerCode	monthOfSale	amount	Label
9125703893	12	960000	TOP
9121014124	12	32151600	TOP
9121019730	12	3695740	TOP
9121008537	12	28909700	TOP
9121024835	12	6688560	TOP
9121013119	12	8854100	TOP
9121020694	12	1516360	TOP
9121011026	12	3039460	TOP
9121024047	12	4212000	TOP
9121004914	12	28080600	TOP
9121020112	12	2409400	TOP

The status bar at the bottom indicates that the query was executed successfully, returning 9,466 rows.

## QUERY 10. LIST DEALER NAME WHOSE NAME STARTS WITH B

```
select DealerName from orders
where DealerName like 'B%'
```

The screenshot displays the Microsoft SQL Server Management Studio interface. The query editor shows the following SQL query:

```
select DealerName from orders
where DealerName like 'B%'
```

The Object Explorer on the left shows the database structure, including the 'dbo.orders' table. The Results pane at the bottom shows the output of the query, listing 11 dealer names that start with 'B':

DealerName
1 BAGAWAS CEMENT HOUSE
2 BHAGWATI ENTERPRISE
3 BINDRABAN & SONS
4 BHATTI TRADING COMPANY
5 BHAGWATI TRADERS
6 BHANDARY ENTERPRISES
7 BABU BUILDERS
8 BABLU PAL
9 BANSAL ENTERPRISES
10 BHAGWATI IRON STORE
11 BHARAT AGENCIES

The status bar at the bottom indicates that the query was executed successfully, returning 5,682 rows.

## Assessment 2

```
CREATE TABLE `shop` (  
  `shopId` int NOT NULL,  
  `shopName` varchar(255) DEFAULT NULL,  
  PRIMARY KEY (`shopId`),  
  KEY `shop_index1` (`shopId`)  
)
```

```
INSERT INTO shop (shopId,shopName)  
values  
(101,'abc'),  
(102,'as'),  
(103,'ac'),  
(104,'ab'),  
(105,'bc'),  
(106,'asd'),  
(107,'asw'),  
(108,'abc'),  
(109,'qw'),  
(110,'nb');
```

```
Select * from shop;
```

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' panel with a tree view of the database structure, including 'sakila', 'stock\_db', 'product', 'shop', 'stockavailability', 'Views', 'Stored Procedures', 'Functions', 'sys', and 'world'. The main query editor shows a query: `select * from shop;`. The 'Result Grid' displays the results of the query, showing 10 rows of data with columns 'shopId' and 'shopName'. The 'Output' panel at the bottom shows the execution log, including the time and duration of the query execution.

shopId	shopName
101	abc
102	as
103	ac
104	ab
105	bc
106	asd
107	asw
108	abc
109	qw
110	nb

#	Time	Action	Message	Duration / Fetch
18	23:36:29	Apply changes to stock_availability	Changes applied	
19	23:36:39	truncate stockavailability	0 row(s) affected	1.344 sec
20	23:36:42	call stock_db.stock_availability("0000-00-00")	1 row(s) returned	1.546 sec / 0.000 sec
21	23:39:50	select * from stockavailability LIMIT 0, 1000	9 row(s) returned	0.000 sec / 0.000 sec
22	23:44:42	select * from shop LIMIT 0, 1000	10 row(s) returned	0.031 sec / 0.000 sec



```

CREATE TABLE `product` (
  `productId` int NOT NULL,
  `shopId` int NOT NULL,
  `price` decimal(10,2) DEFAULT NULL,
  `maxAvail` int DEFAULT NULL,
  `date` date DEFAULT NULL,
  PRIMARY KEY (`productId`),
  KEY `shopId` (`shopId`),
  KEY `product_index1` (`productId`,`shopId`),
  CONSTRAINT `product_ibfk_1` FOREIGN KEY (`shopId`) REFERENCES `shop` (`shopId`)
)

```

```

INSERT INTO product(productId,shopId,price,maxAvail,date)

```

Values

```

(1,101,302,10,'25022021')
(2,102,302,10,'2021-02-25'),
(3,103,300,10,'2021-02-25'),
(4,104,3,10,'2021-02-25'),
(5,105,302,0,'2021-02-25'),
(6,106,302,10,'2021-02-25'),
(7,107,302,10,'2021-02-25'),
(8,108,302,10,'2021-02-25'),
(9,109,302,10,'2021-02-25'),
(10,110,302,10,'2021-02-25');

```

```

select * from product;

```

The screenshot shows the MySQL Workbench interface. The 'Query Editor' contains the query `select * from product;`. The 'Result Grid' displays the following data:

productId	shopId	price	maxAvail	date
1	101	302.00	10	2021-02-25
2	102	302.00	10	2021-02-25
3	103	300.00	10	2021-02-25
4	104	3.00	10	2021-02-25
5	105	302.00	0	2021-02-25
6	106	302.00	10	2021-02-25
7	107	302.00	10	2021-02-25
8	108	302.00	10	2021-02-25
9	109	302.00	10	2021-02-25
10	110	302.00	10	2021-02-25

The 'Action Output' pane shows the following log:

#	Time	Action	Message	Duration / Fetch
19	23:36:39	truncate stockavailability	0 row(s) affected	1.344 sec
20	23:36:42	call stock_db.stock_availability('0000-00-00')	1 row(s) returned	1.546 sec / 0.000 sec
21	23:39:50	select * from stockavailability LIMIT 0, 1000	9 row(s) returned	0.000 sec / 0.000 sec
22	23:44:42	select * from shop LIMIT 0, 1000	10 row(s) returned	0.031 sec / 0.000 sec
23	23:45:44	select * from product LIMIT 0, 1000	10 row(s) returned	0.032 sec / 0.000 sec

```

CREATE TABLE `stockavailability` (
  `stkId` int NOT NULL,
  `productId` int NOT NULL,
  `stkAvail` int DEFAULT NULL,
  `price` decimal(10,2) DEFAULT NULL,
  `date` date DEFAULT NULL,
  PRIMARY KEY (`stkId`),
  KEY `StockAvailability_index1` (`stkId`,`productId`)
)

```

Initially stockavailability table is blank and as required the data will be inserted from product table using stored procedure.

#### Stored procedure for inserting data in stockavailability :-

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `stock_availability`(prodId varchar(50),Date
datetime)
BEGIN
Declare stock varchar(50);
DECLARE is_complete,quantity_in_stock,qty INTEGER default 0;
Declare stock_cursor CURSOR FOR
select productId from product;
declare continue handler for not found SET is_complete = 1;
OPEN stock_cursor;
FETCH stock_cursor into stock;
REPEAT
    SELECT maxAvail INTO quantity_in_stock
    FROM product
    WHERE productId = stock;
IF quantity_in_stock = 0
THEN
Select concat('Error Code:5000 - Item out Of Stock for product Id'," ", stock) as error;
ELSE
set @query=concat("Insert Into stockavailability(stkId, productId, stkAvail, price, date)
select concat(productId,shopId) as stkId,productId,maxAvail,price,date from product where
maxAvail !=0 and productId = ",stock,";");
prepare stmt from @query ;
execute stmt ;
deallocate prepare stmt;
end if;
FETCH stock_cursor into stock;
    UNTIL is_complete = 1
    END REPEAT;
CLOSE stock_cursor;

END

```

Now Calling the procedure using the statement :

```
call stock_db.stock_availability('', '0000-00-00');
```

Calling the above procedure data has been inserted in the stockavailability table for 9 shops.

The data has not been inserted for productId = '5' as the maxAvail=0.

As required the shop with maxAvail = 0 or the shop with no product availability the data will not be inserted in the table-stockavailability and error message will be displayed as shown below:

**Output:** Error Code:5000 - Item out Of Stock for product Id 5.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' panel with a tree view showing the 'sakila' database, 'stock\_db' (containing 'product', 'shop', 'stockavailability', 'Views', and 'Stored Procedures'), 'sys', and 'world'. The main window shows a query titled 'stock\_availability - Routine' with the following SQL code:

```
1 call stock_db.stock_availability('', '0000-00-00');
2
3
```

The 'Result Grid' below the query shows an error message:

error
Error Code:5000 - Item out Of Stock for product Id 5

The bottom panel shows the 'Output' tab with an 'Action Output' table:

#	Time	Action	Message	Duration / Fetch
19	23:36:39	truncate stockavailability	0 row(s) affected	1.344 sec
20	23:36:42	call stock_db.stock_availability('', '0000-00-00')	1 row(s) returned	1.546 sec / 0.000 sec
21	23:39:50	select * from stockavailability LIMIT 0, 1000	9 row(s) returned	0.000 sec / 0.000 sec
22	23:44:42	select * from shop LIMIT 0, 1000	10 row(s) returned	0.031 sec / 0.000 sec
23	23:45:44	select * from product LIMIT 0, 1000	10 row(s) returned	0.032 sec / 0.000 sec

Now checking whether the data has been inserted into the table stockavailability.

**Select \* from stockavailability;**

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'sakila' and 'stock\_db' expanded. The 'stock\_availability' table is selected. The main query editor shows the query 'select \* from stockavailability;'. The 'Result Grid' displays the following data:

stkid	productid	stkAvail	price	date
1101	1	10	302.00	2021-02-25
2102	2	10	302.00	2021-02-25
3103	3	10	300.00	2021-02-25
4104	4	10	3.00	2021-02-25
5106	6	10	302.00	2021-02-25
7107	7	10	302.00	2021-02-25
8108	8	10	302.00	2021-02-25
9109	9	10	302.00	2021-02-25
10110	10	10	302.00	2021-02-25
NULL	NULL	NULL	NULL	NULL

The bottom panel shows the 'Action Output' with the following log:

#	Time	Action	Message	Duration / Fetch
17	23:35:32	call stock_db.stock_availability('','0000-00-00')	1 row(s) returned	0.515 sec / 0.000 sec
18	23:36:29	Apply changes to stock_availability	Changes applied	
19	23:36:39	truncate stockavailability	0 row(s) affected	1.344 sec
20	23:36:42	call stock_db.stock_availability('','0000-00-00')	1 row(s) returned	1.546 sec / 0.000 sec
21	23:39:50	select * from stockavailability LIMIT 0, 1000	9 row(s) returned	0.000 sec / 0.000 sec

As seen data has been inserted for the shops having products availability that is for 9 shops.