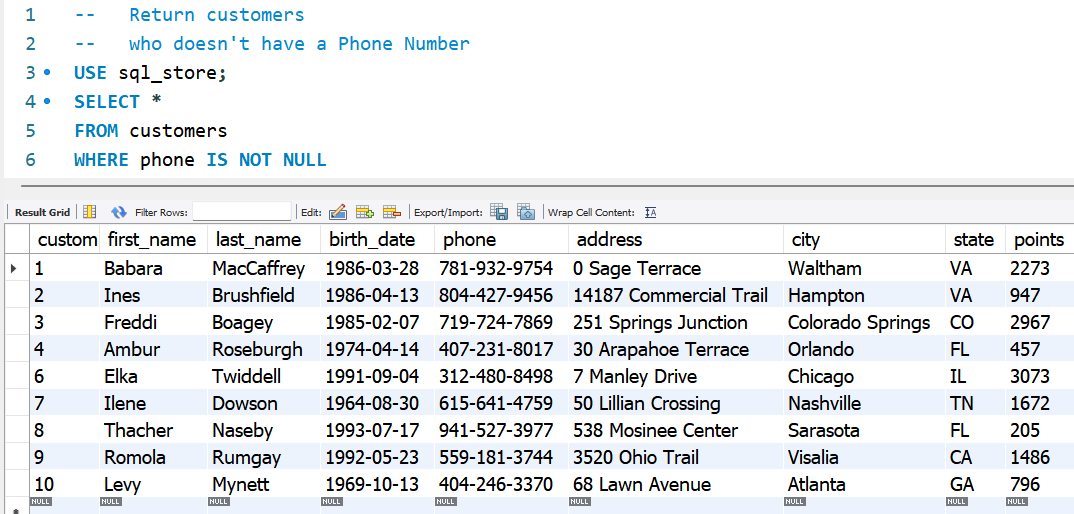
## **IS NULL Operator**

### **Returns customers who doesn’t have a phone number**

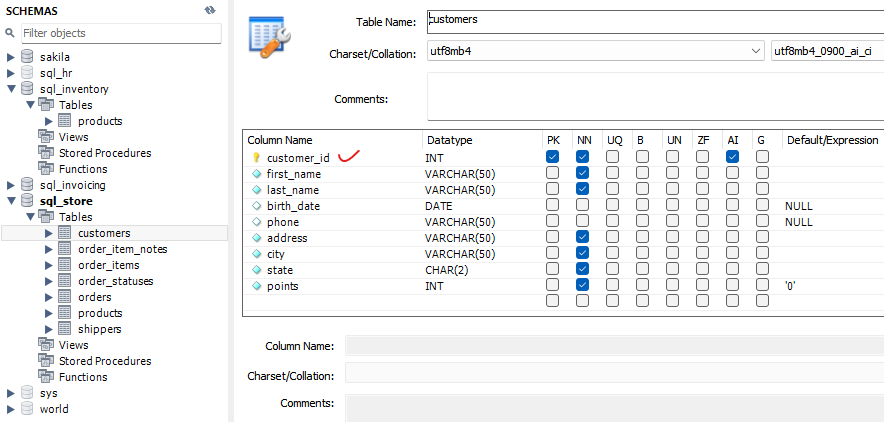
## **IS NOT NULL Operator**

### **Returns customers who has a phone number.**

****

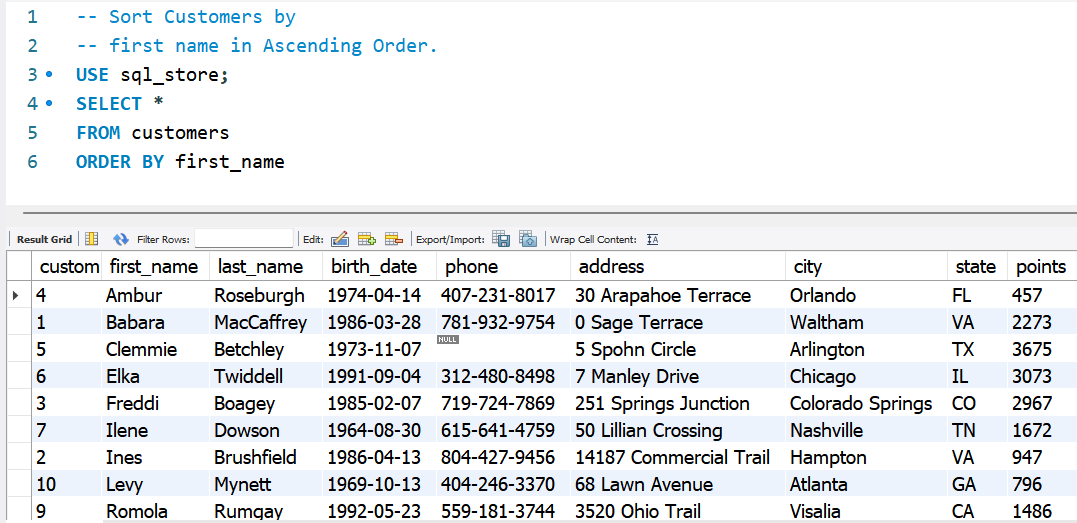
## **ORDER BY Clause**

### **Customer ID column is the default order by column as seen below.**

****

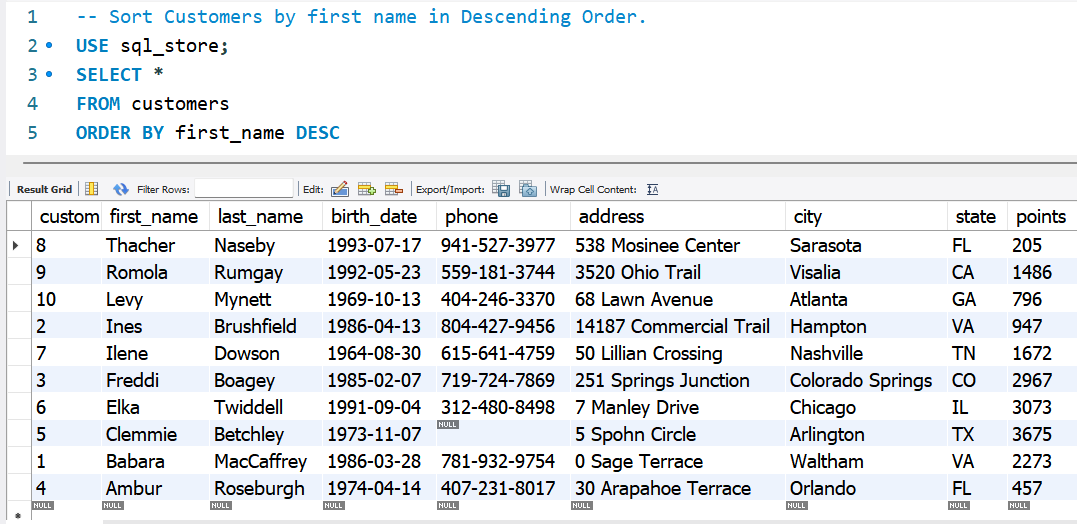
## **ORDER BY first name**

### **Sort the customers table by first name in Ascending order.**

****

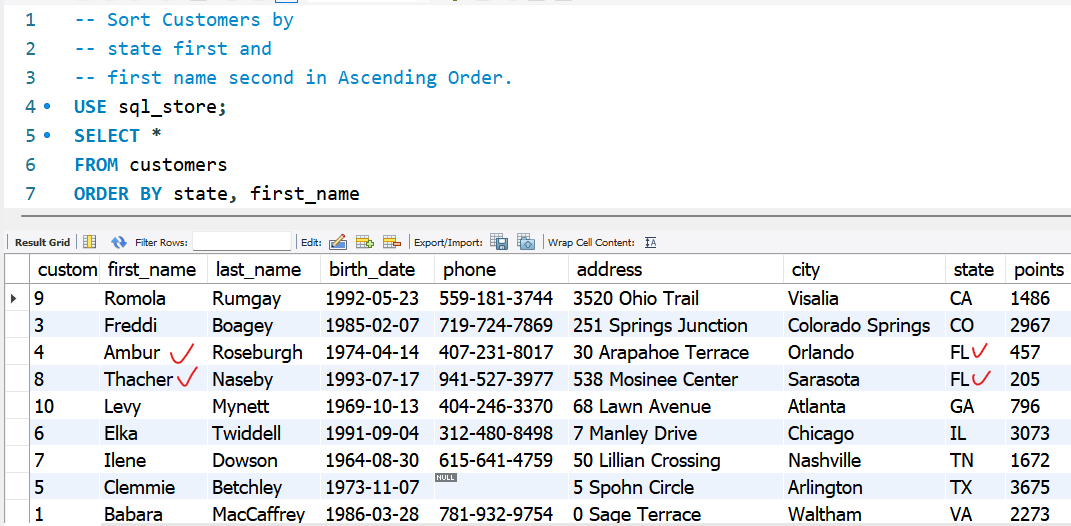
## **ORDER BY first name DESC**

### **Sort the customers table by first name in Descending order.**

****

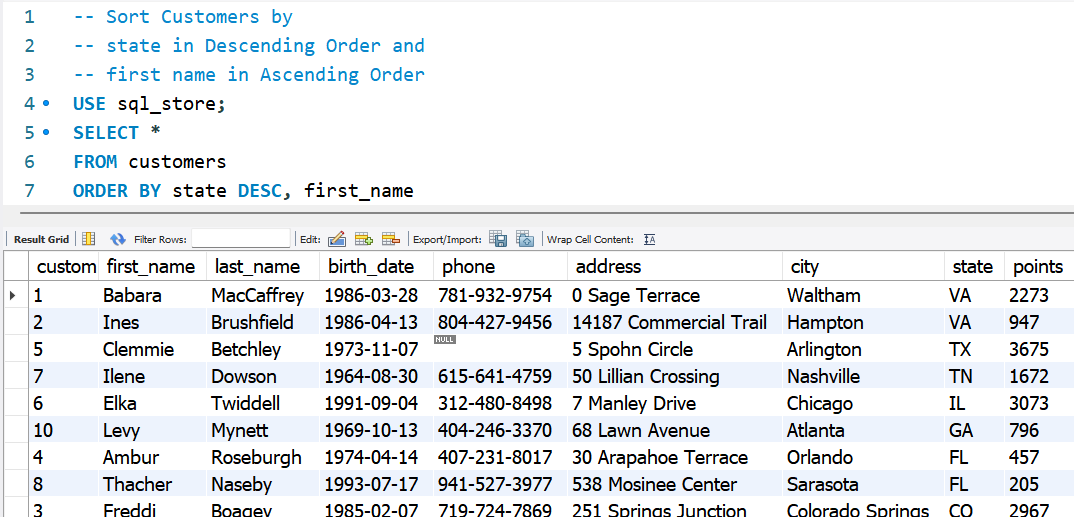
## **ORDER BY state, first name**

### **Sort customers by state first in Ascending order and by first name second in Ascending order.**

****

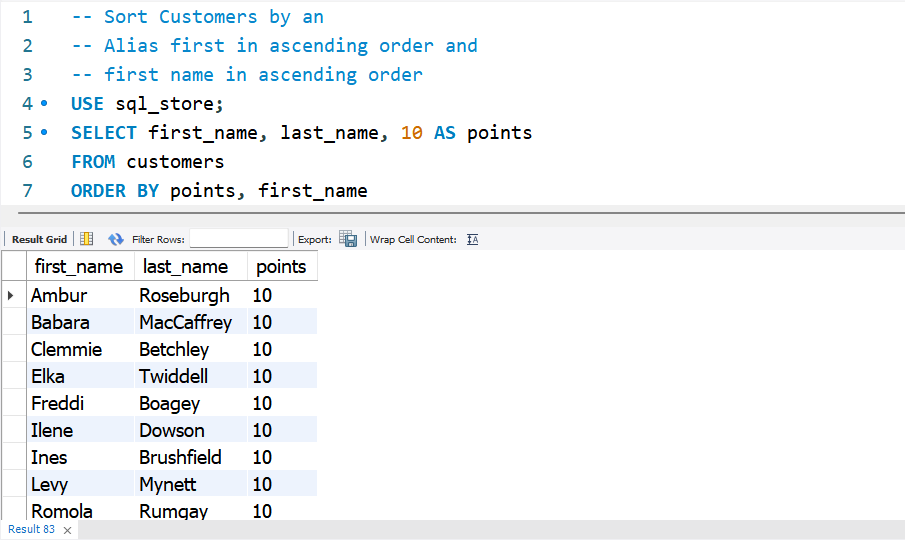
## **ORDER BY state DESC, first name**

### **Sort the customers table by state in Descending order and first name in Ascending order.**

****

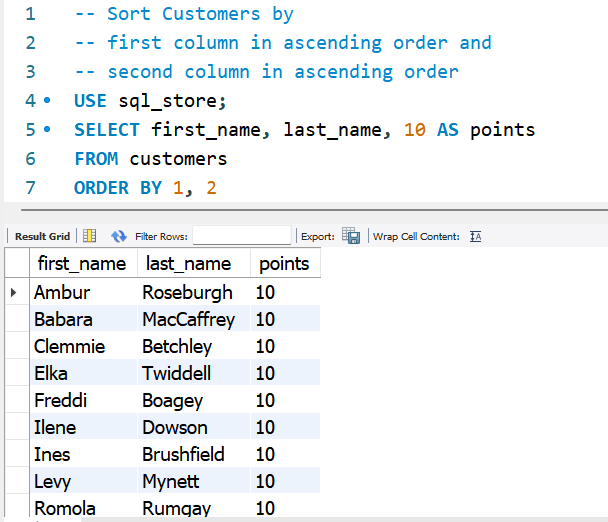
## **ORDER BY an Alias (10 AS points) first**

### **Sort the customers table by an Alias first in Ascending order and secondly by first name in Ascending order.**

****

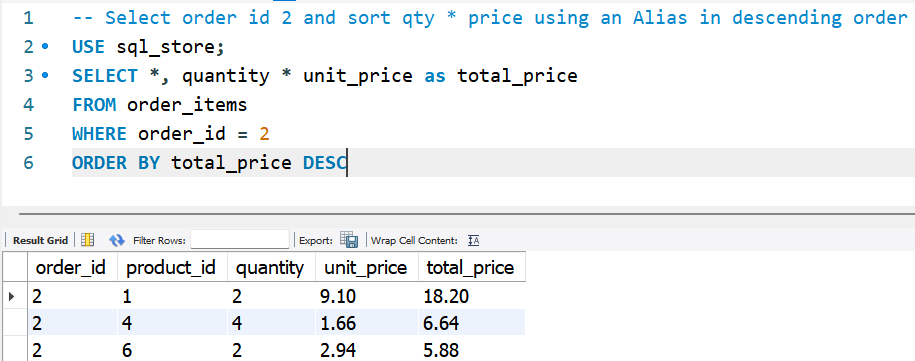
## **ORDER BY column numbers 1, 2**

### **Sort customers by first column in ascending order and second column in ascending order.**

****

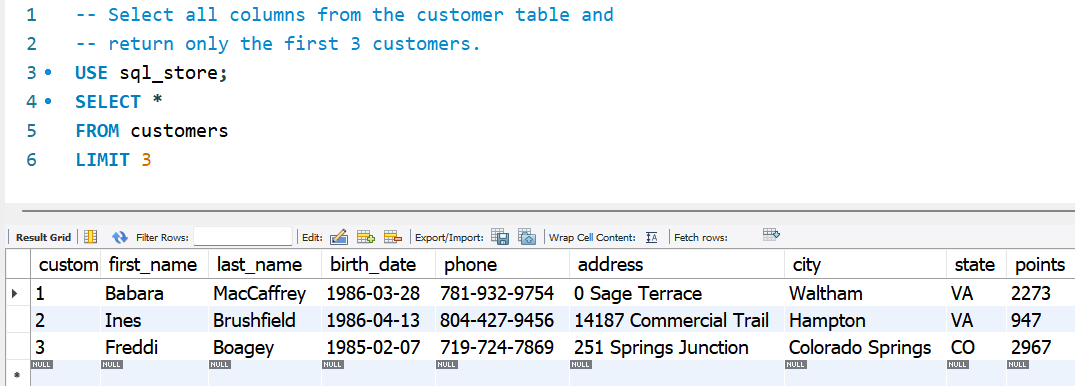
## **ORDER BY quantity \* price = total price (ALIAS) in descending**

### **Select order id 2, multiply qty \* unit price and create a ‘total price’ ALIAS column, sort said column in descending order.**

****

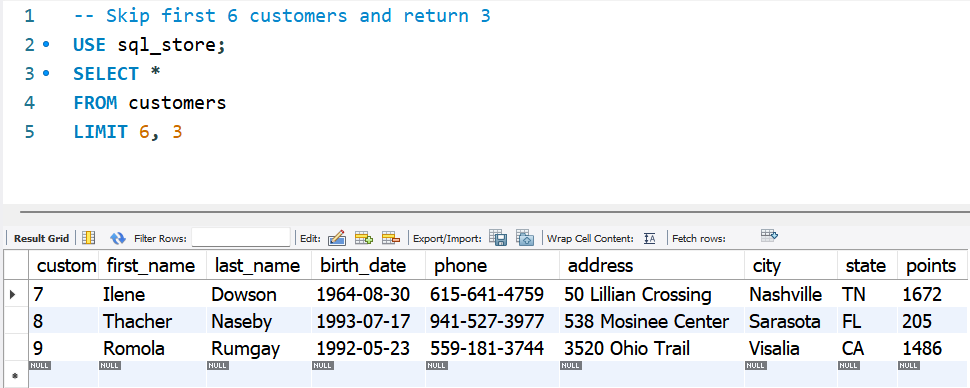
## **LIMIT CLAUSE**

### **Return only the first 3 customers from the customers’ table.**



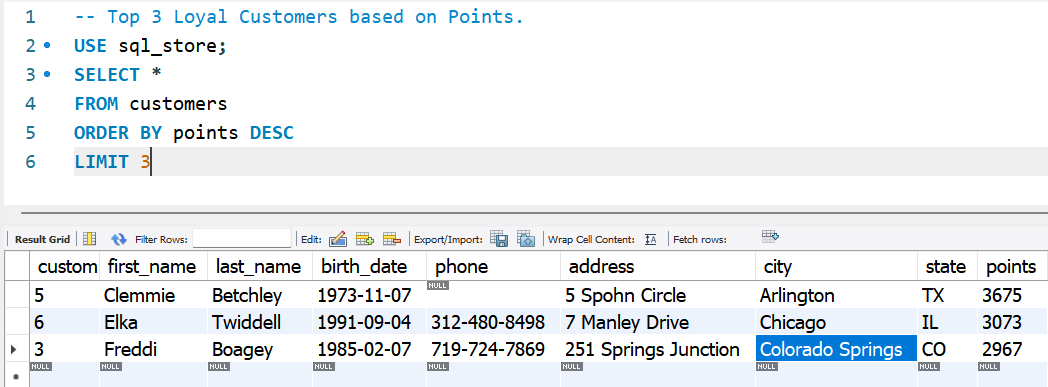
## **LIMIT CLAUSE**

### **Skip the first 6 customers and return 3.**



## **LIMIT CLAUSE – Top 3 Loyal Customers**

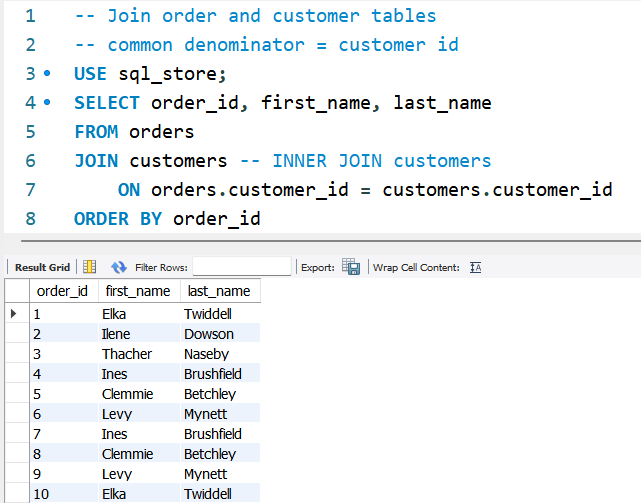
### **Top 3 Loyal Customers based on Points.**



# **Retrieving Data From Multiple Tables**

## **INNER Joins**

### **Join order table and customer table using a common denominator; customer id.**

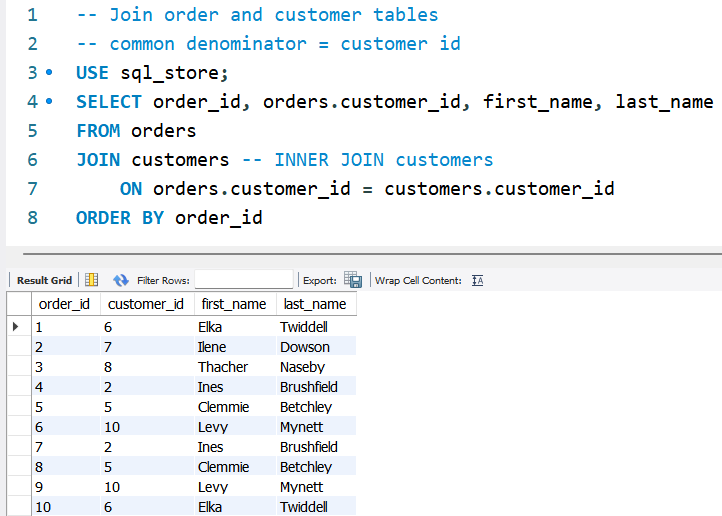


## **INNER Joins**

**NOTES**

1. Because we have Customer ID in both the orders as well as the customers table, when returning said column, we need to mention from which table said column in taken, in the SELECT statement.
2. Instead of JOIN customers, we can use INNER JOIN customers.

### **Because we have the Customer ID column in both the orders as well as the customers table, we need to mention from which table said column is taken, in the SELECT statement.**

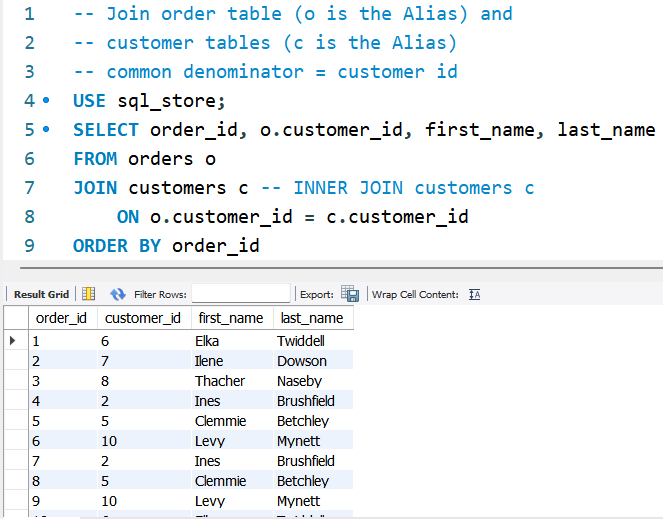


## **INNER Joins**

**NOTES**

1. Use O as the Alias for the orders table.
2. Use C as the Alias for the customers’ table.
3. All the places where the words orders and customers needs to get replaced by o and c respectively.

### **Rewriting the code where ‘O’ as the Alias for the Order Table and ‘C’ as the Alias for the Customers Table.**

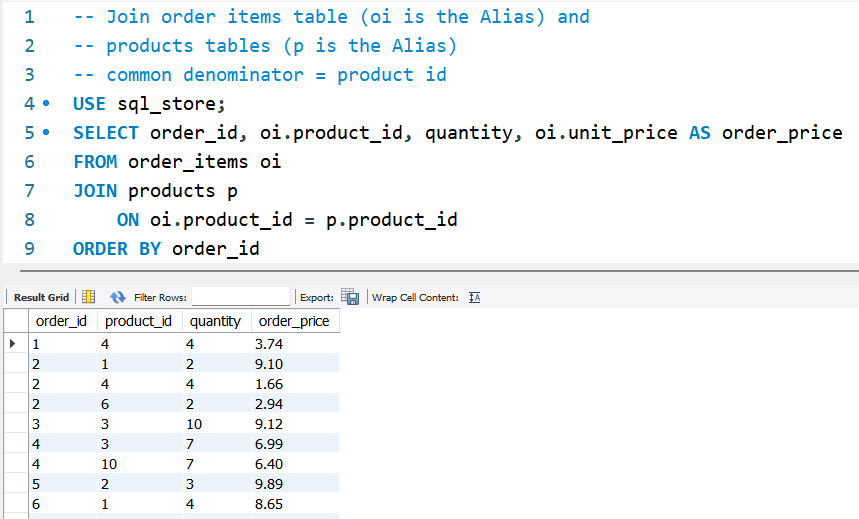


## **INNER Joins**

**NOTES**

1. There are 2 order price columns – one price when the customer ordered the product – hence appearing in the order items table and second price appearing as the current price of the product appearing in the products table.
2. We have selected the price at the point of making the order. Hence, we have taken the order price from the order items table; oi.unit\_price AS order\_price

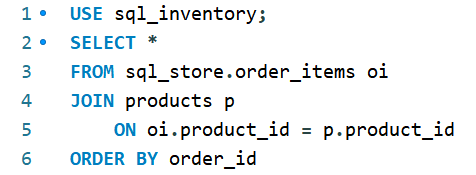
### **Exercise – Join order items (oi as Alias) and products(p) using the denominator product id**



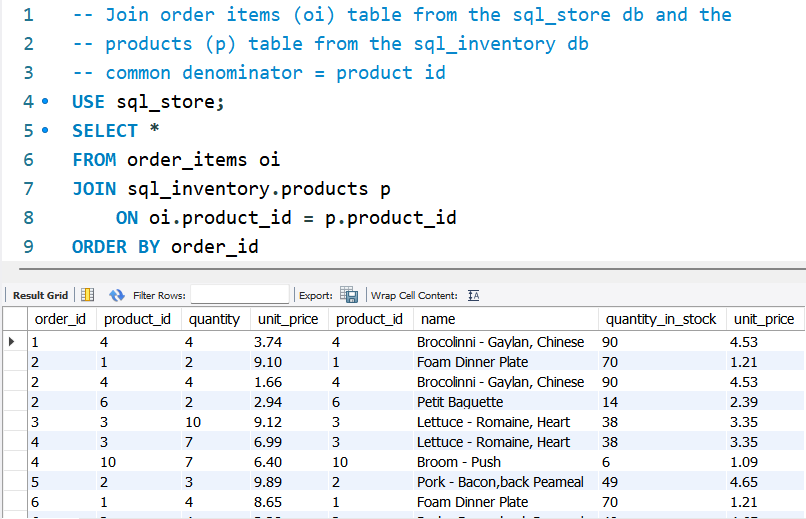
## **JOIN Across DATABASES**

**NOTES**

1. If we are to USE the SQL Inventory Database, the code can be rewritten as follows which will return the same results.

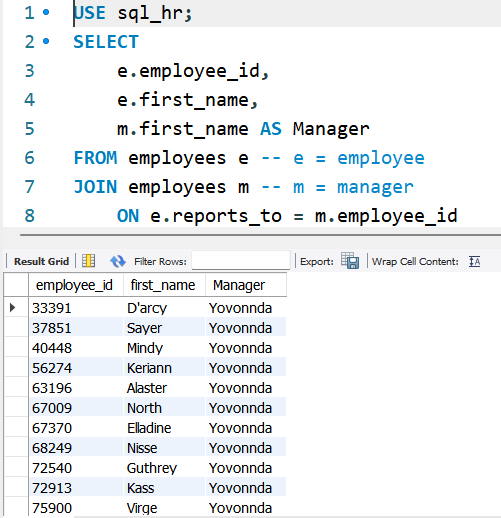


### **Exercise – Join order items (oi) table from the SQL store database and the products (p) table from the SQL inventory database.**



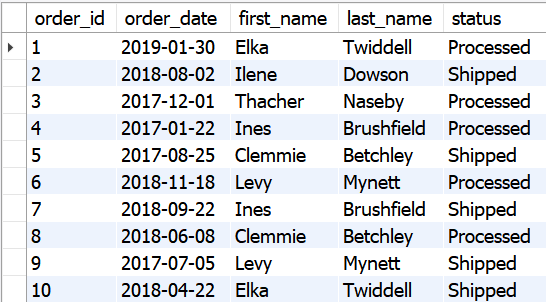
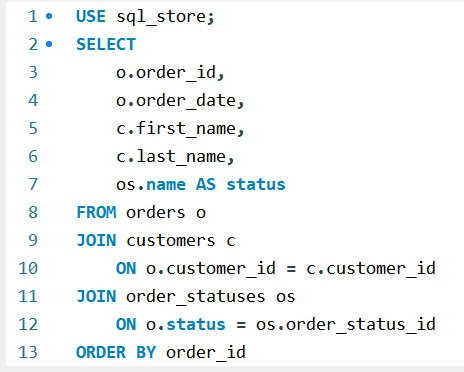
## **JOIN A Tables with itself**

### **SELF JOIN TABLES.**



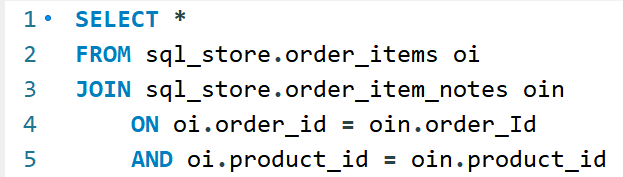
## **JOIN Multiple tables**

### **Self Join Multiple Tables.**



## **COMPOUND JOIN statements – ON, and**

### **Exercise.**

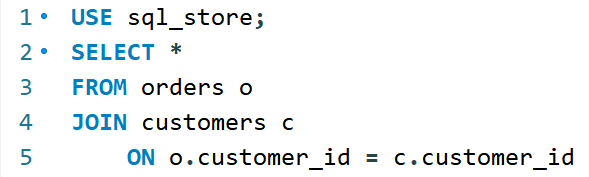


## **IMPLICIT JOIN SYNTAX**

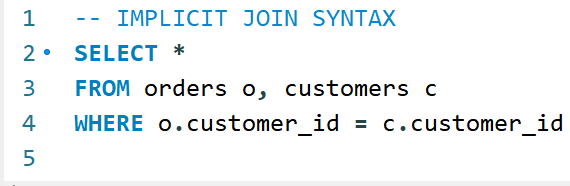
**NOTES**

1. If you forget to type in the WHERE clause,
2. it will result in a CROSS JOIN leading to 100's of lines where
3. every record in the orders table is matched with every table in the customers table

### **We can use FROM, WHERE - instead of JOIN, ON.**



.

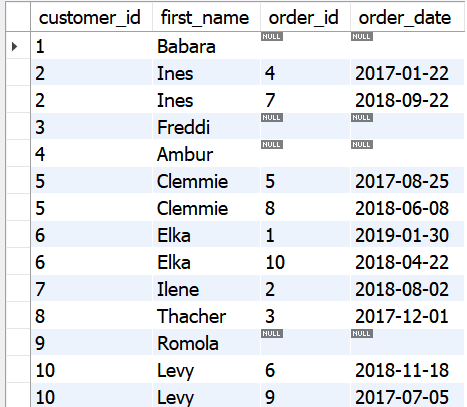
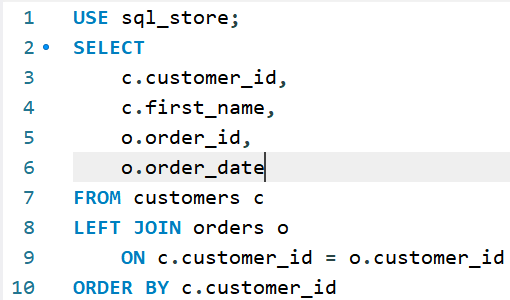


## **OUTER JOIN: LEFT JOIN**

**NOTES**

1. By Left joining the customer table, customers who haven’t’ placed orders (order id = null) will also return in the results; giving us a complete picture of all the customers who placed/didn’t placed orders.

### **Join everything on the left side table ignoring the JOIN condition.**

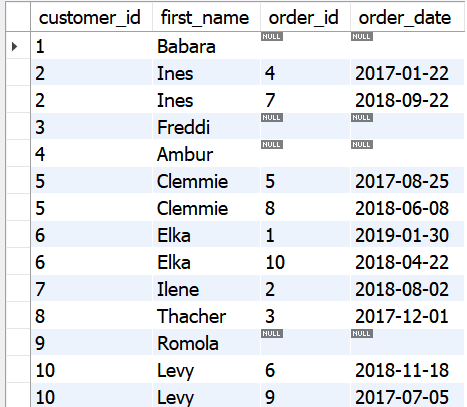
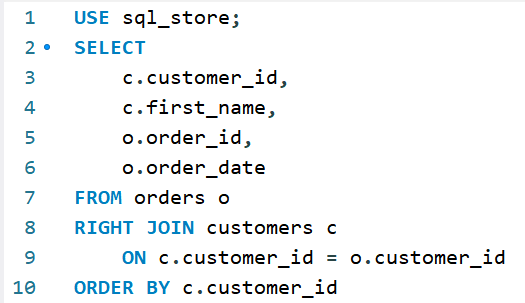
 

## **OUTER JOIN: RIGHT JOIN**

**NOTES**

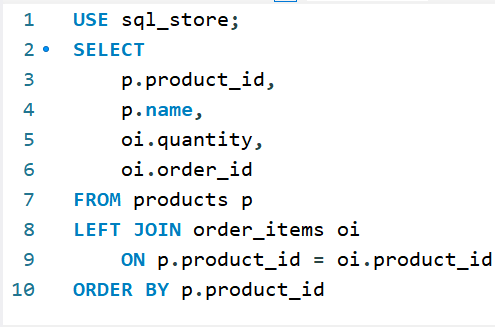
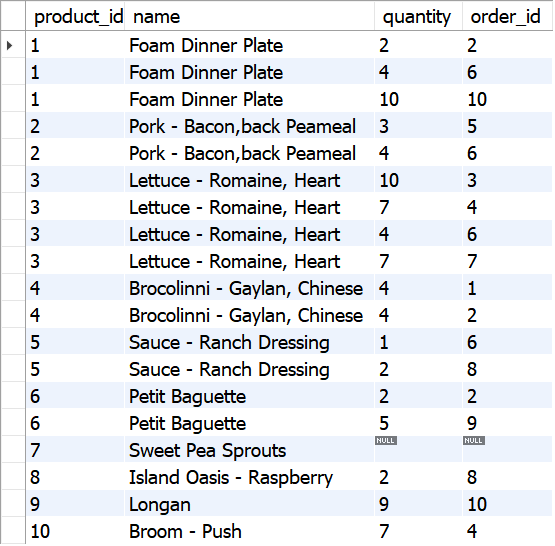
You can swap the position of the tables; customers table on the right and orders table on the left and do a RIGHT JOIN of the customer table to get the same results as above in the LEFT join.

### **Join everything on the right side table ignoring the JOIN condition.**

## **OUTER JOIN: LEFT JOIN Exercise**

### **Exercise – Find out which products were never ordered – Answer is product id 7.**



## **OUTER JOIN between multiple tables**

**NOTES**

1. Customer table is at the far left.
2. Products table is in the middle.
3. Shippers table is at the far right.

**FROM customers c**

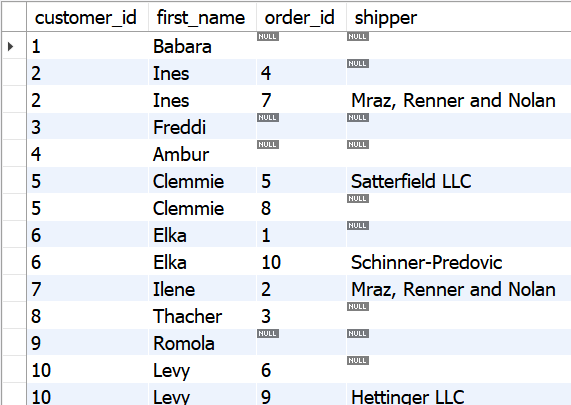
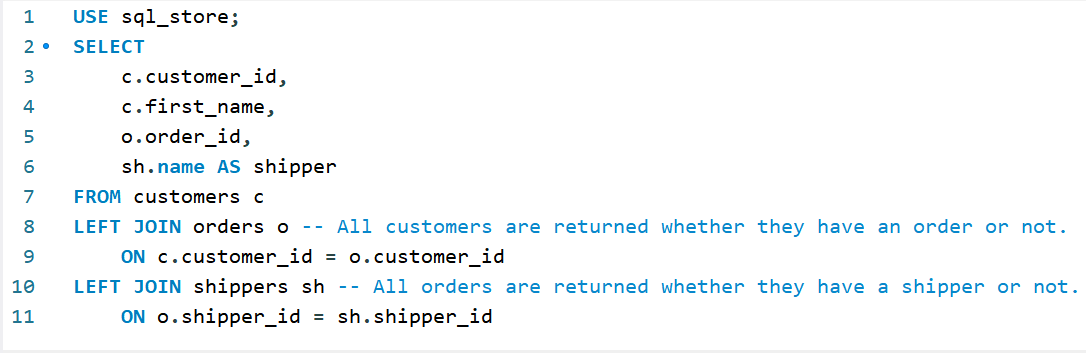
**LEFT JOIN orders o**

* customers table = left side table.
* orders table = right-side table.
* All customers are returned (because we left join everything from the left table; customer table) here irrespective whether they have an order or not.

**LEFT JOIN shippers**

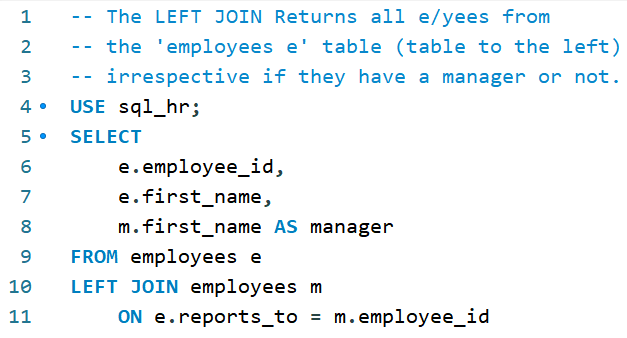
* orders table = left side table.
* shippers table = right table.
* All orders are returned (because we left join everything from the left table; orders table) here irrespective whether they have a shipper name or not.

### **Exercise – All customers with/without an order as well as with/without a shipper.**

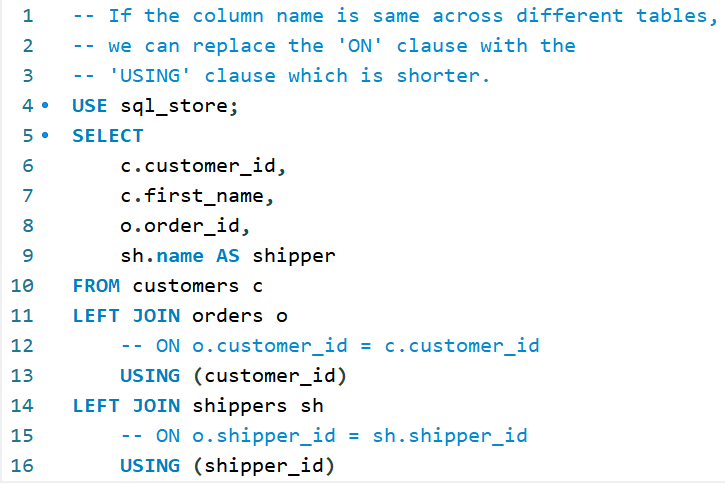
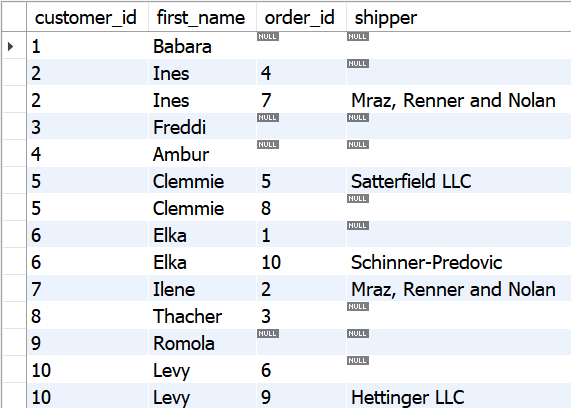
## **SELF OUTER JOIN within the same table**

### **Exercise – Self Outer Join employees to their managers.**



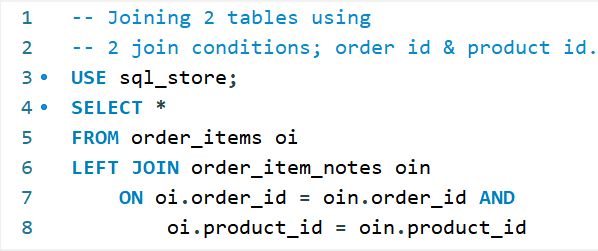
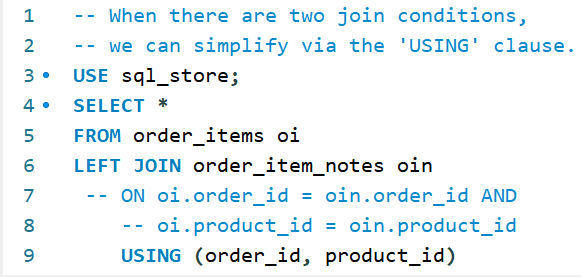
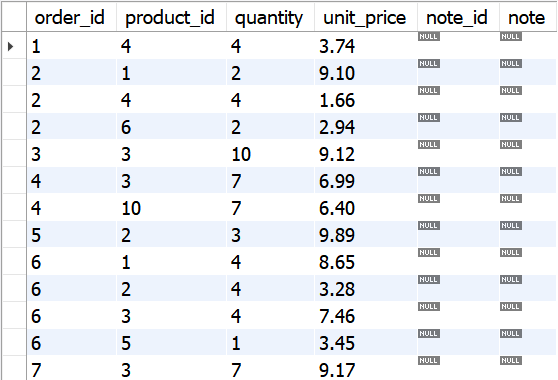
## **USING CLAUSE – replace ‘ON’ with ‘USING’**

### **IF the column name is SAME across different tables, we can replace the 'ON' clause with the 'USING' clause which is shorter and simpler.**



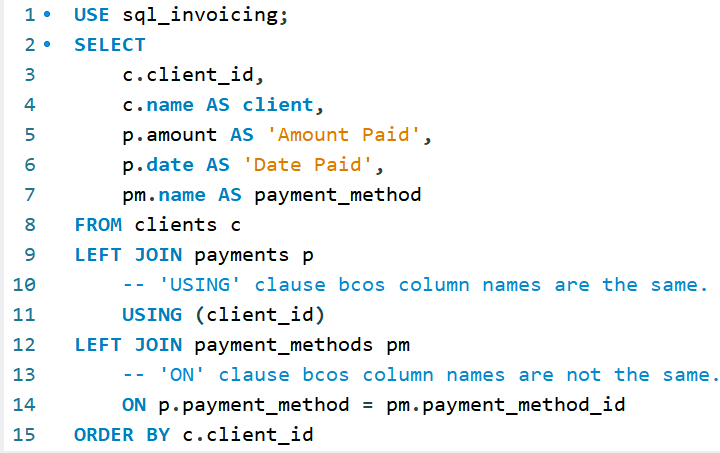
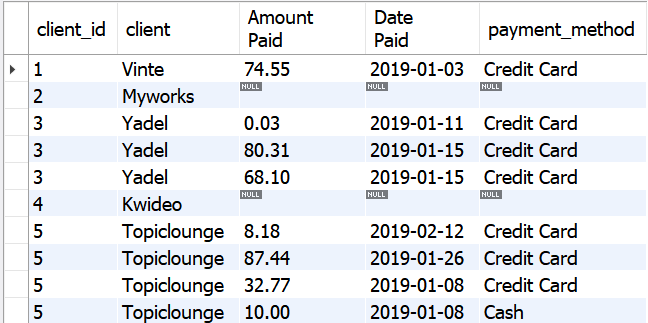
## **USING CLAUSE – JOINING 2 CONDITIONS VIA ‘USING’**

### **When there are two join conditions, we can simplify via the ‘USING’ clause.**



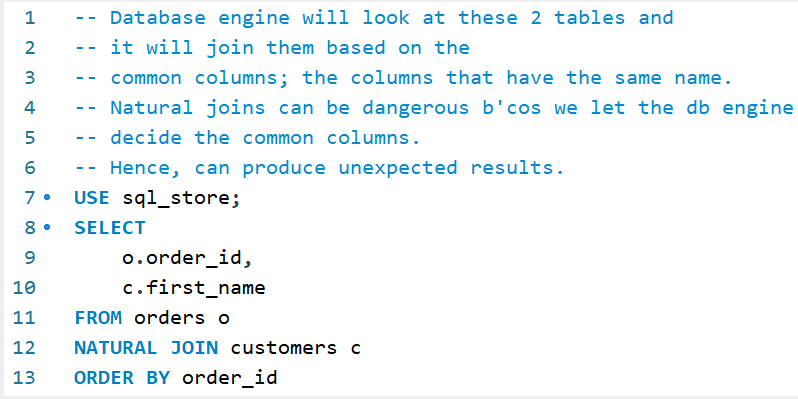
## **ON & USING CLAUSES**

### **Exercise – ON/USING clauses when column names are ‘not same’/same.**



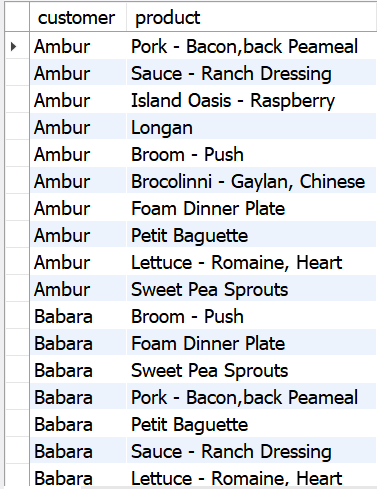
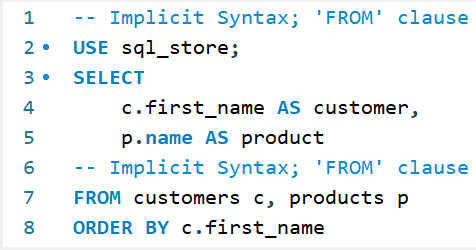
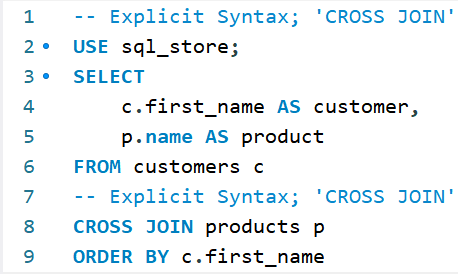
## **NATURAL JON**

### **Letting the database engine join tables based on columns with the same name.**

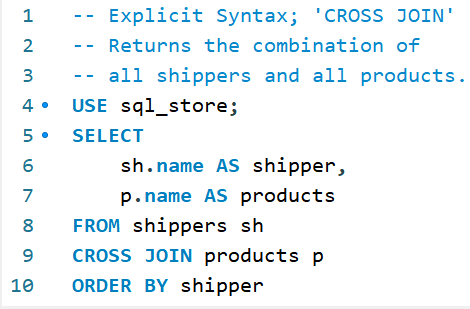
## **CROSS JOIN**

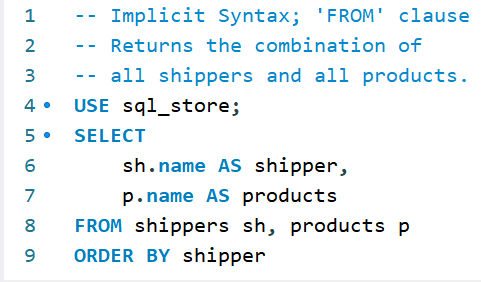
### **Cross Join: Join every record from the 1st table with every record in the 2nd table.**

## **CROSS JOIN - Exercise**

### **Exercise - Cross Join to Return the combination of all shippers and all products using the Explicit syntax; CROSS JOIN and the Implicit syntax; FROM clause.**



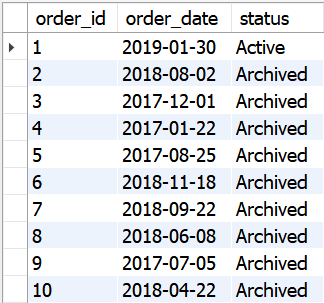
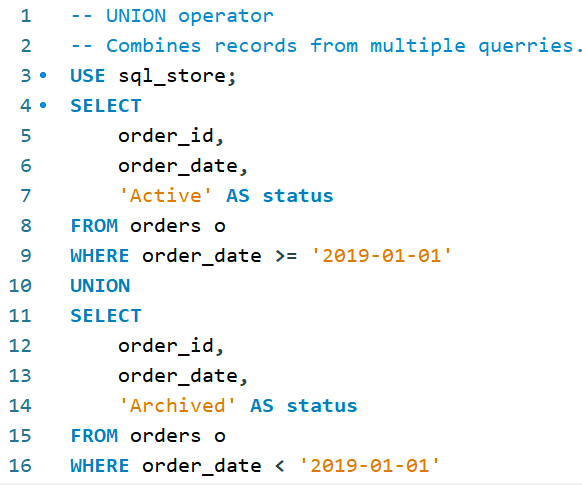


## **UNION OPERATOR**

### **EXAMPLE 1 - Combine results from Multiple Queries.**

**Query 1 – Filter out orders older than 2019-01-01 and update the status as Archived.**

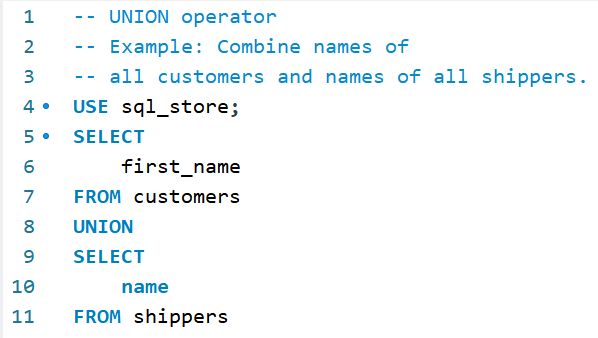
**Query 2 – Filter out orders on or before 2019-01-01 and update the status as Active.**

**Use the UNION operator to combine the two queries as follows.**

## **UNION OPERATOR**

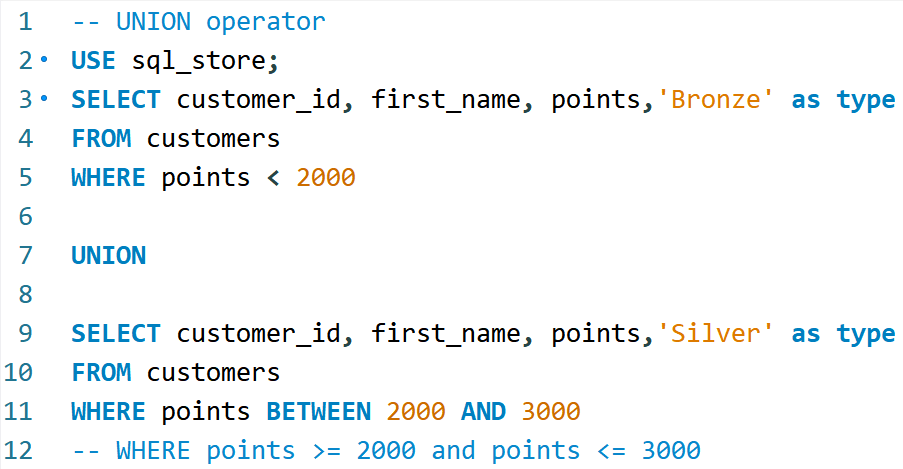
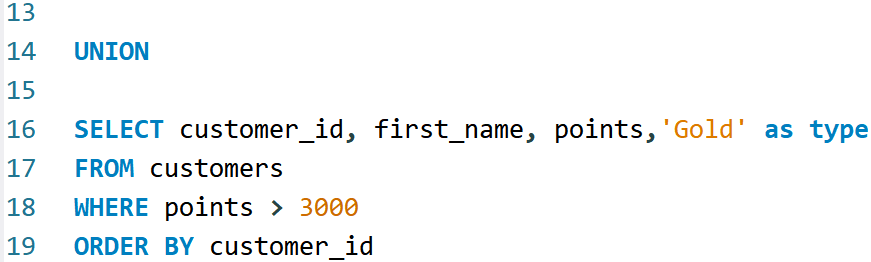
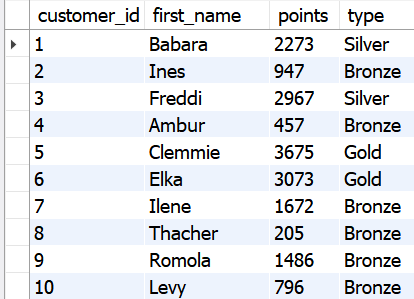
### **EXAMPLE 2 – Combine names of all customers and names of all shippers.**

**NOte – name of the column is based on the first query.**



## **UNION OPERATOR Example**

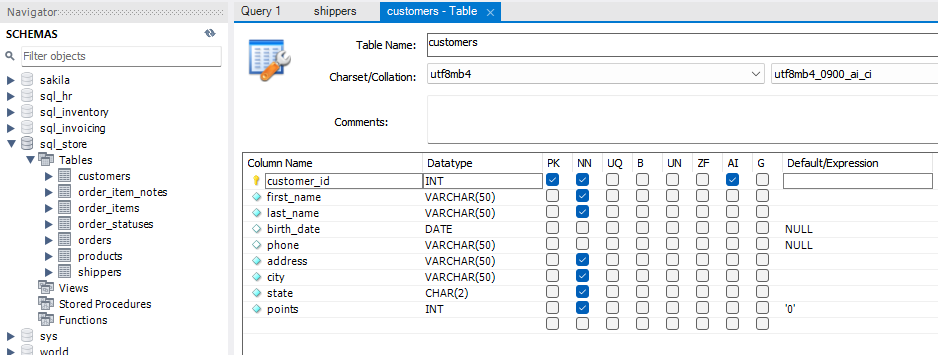
### **EXAMPLE 3 – Gold, Silver, Bronze based on the number of points.**



# **INSERTING, UPDATING AND DELETING DATA**

## **INSERTING A ROW**

### **Customers Table – Design Mode.**

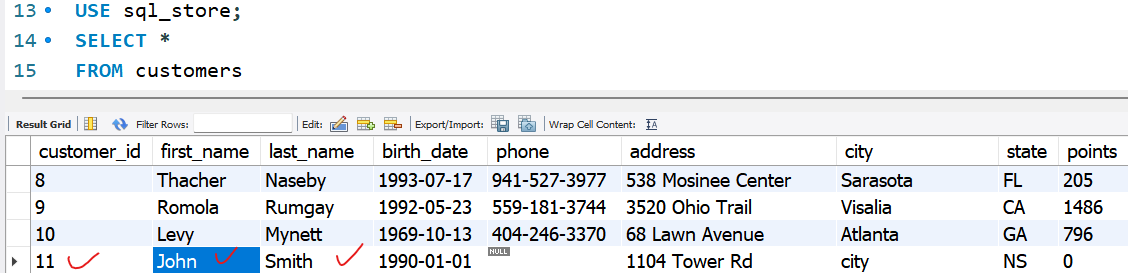
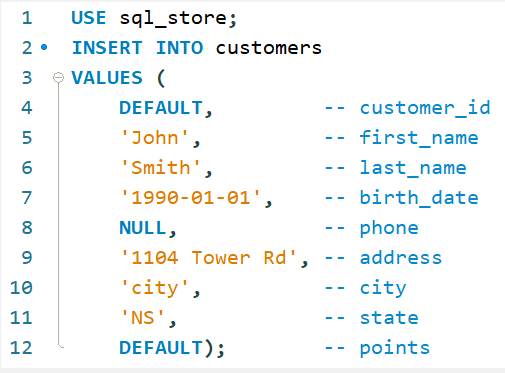
* **AI = Auto increment**
* **PK = primary key**
* **nn = not null**
* **b = is binary column**

## **INSERTING A ROW**

**NOTES**

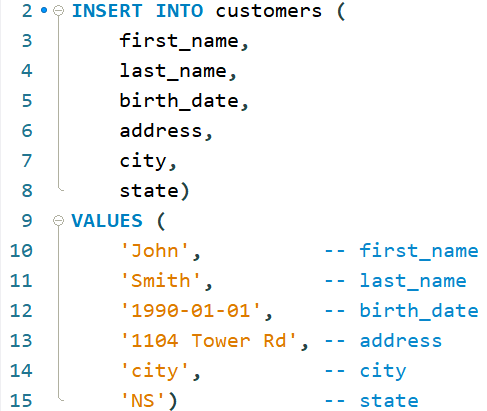
1. **customer\_id** - mySQL will generate a Unique Value for the customer ID based on the AI (auto increment) feature.
2. **first\_name, last\_name, address, city, state** 
   1. NN (Not Null) feature has been enabled/checked.
   2. Hence a string should be entered here.
   3. Strings and Dates should be enclosed within single '' or double "" quotes.
3. **birth\_date, phone** = is optional because NN (Not Null) is not checked. You can be enter value within single '' or double "" quotes. Alternatively, you can either enter **NULL** or **DEFAULT**. Notice the Default expression is DEFAULT here.
4. **points** - can either use an explicit value (Ex - 2000) or leave it as **DEFAULT**. Notice the Default expresion is 0 here.

### **Method 1 - Inserting a row into the customers’ table.**

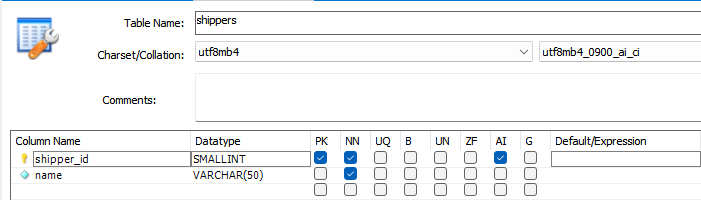
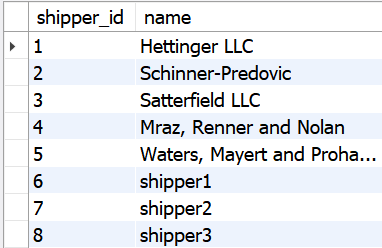
## **INSERTING A ROW**

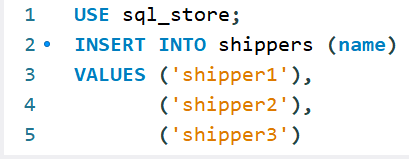
### **Method 2 - Inserting a row into the customers’ table. We don’t have to supply DEFAULT or NULL values here.**



## **INSERTING Multiple rows**

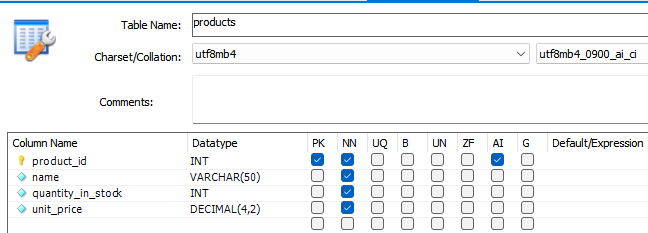
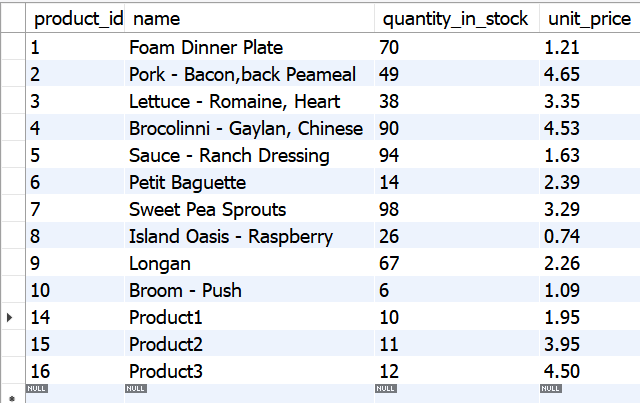
### **Inserting multiple shipper names into the shippers’ table. Note that the shipper id is auto incremented – ref to design mode. Hence, no value is needed there.**

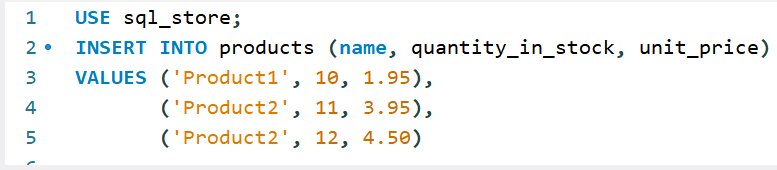




## **INSERTING Rows Example**

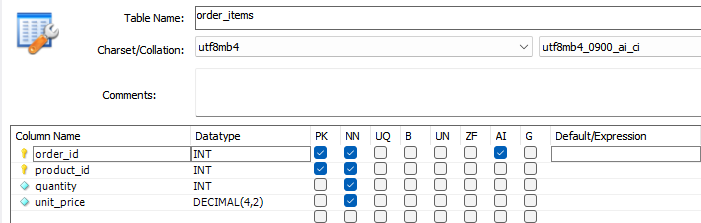
### **Example – Insert 3 rows in the products table. Note – Product id is an AI column.**

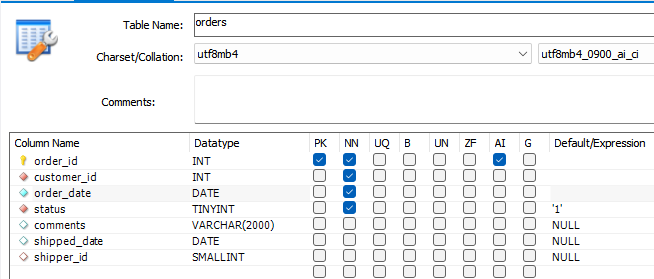




## **INSERTING hierarchical rows**

### **Example – Insert 2 orders into the order\_items table and thereafter update the order table.**





**NOTES**

**Order Items Table**

1. When you create an order, it firstly gets created in the order items table.
2. We need the next in line order\_id to create the order.
3. For that purpose we use the function LAST\_INSERT\_ID.
4. This function returns the next in line order id (note that the order\_id column in the order\_items table is set to Auto Increment (AI).
5. We create two orders in the order\_items table for the below values.

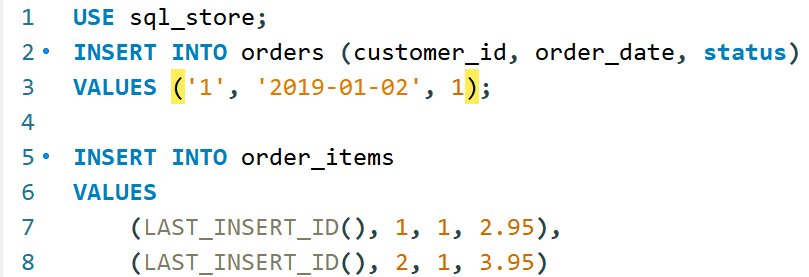
* (LAST\_INSERT\_ID(), 1, 1, 2.95),
* (LAST\_INSERT\_ID(), 2, 1, 3.95)

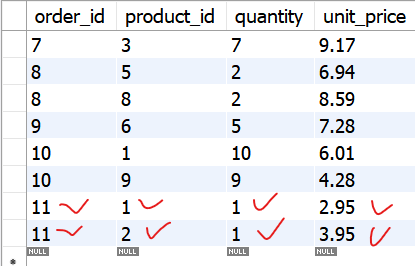
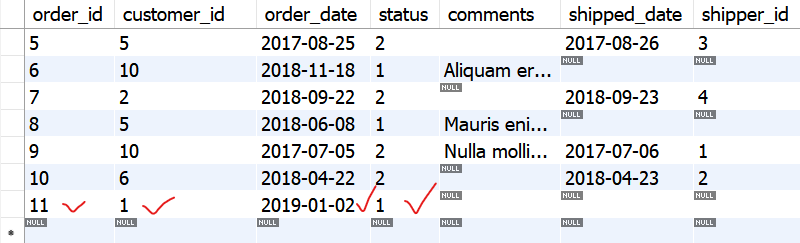
**Order Table.**

1. order\_id - will be picked up from the order items table
2. customer\_id, order\_date, status is entered per the below code.

## **INSERTING hierarchical rows**

### **Example – Insert 2 orders into the order\_items table and thereafter update the order table.**

**SQL Code**

**order\_items table** **orders table**

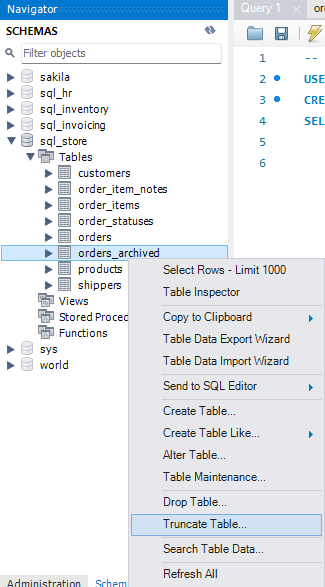
## **CREATING a copy of a table**

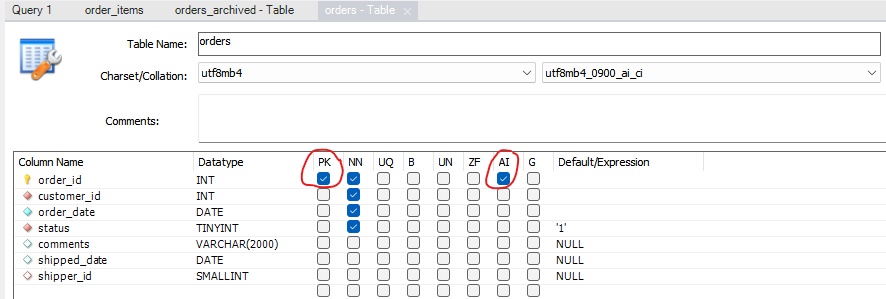
### **STEP 1 – Create a copy of the orders table and, rename it as orders\_archived.**

**NOTES**

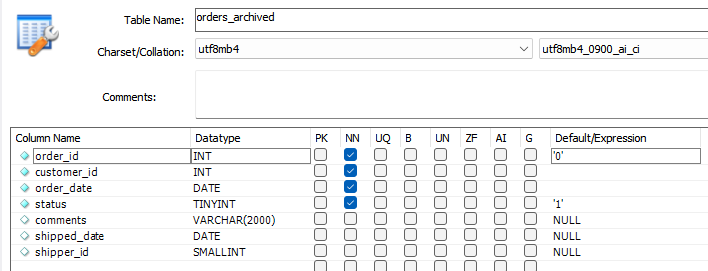
**Orders Archived Table**

* Notice that there is no primary key (PK) and no auto increment (AI) column in the table copy created.
* Right click on and click **Truncate** to delete all data from the table.





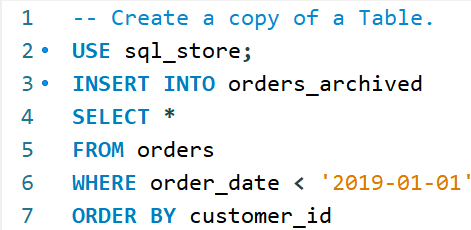
Orders Table Settings

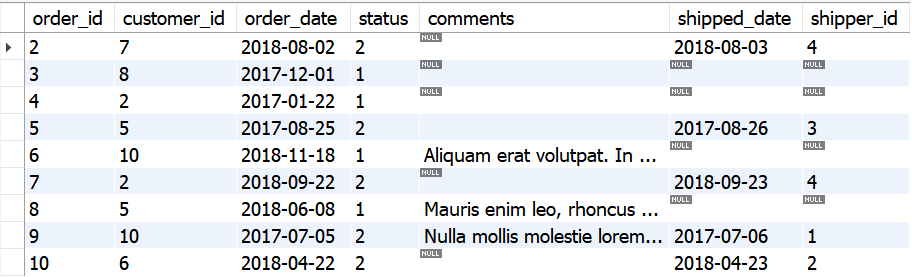


Orders Archived Settings

## **STORE data in the Created Table**

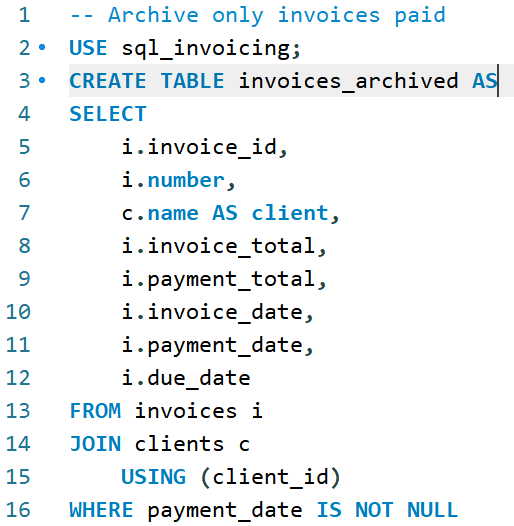
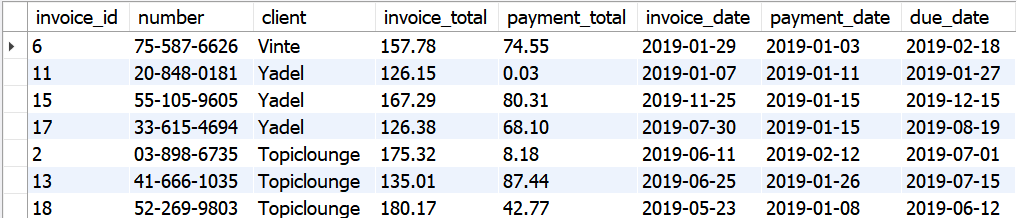
### **STEP 2 – Select orders with order dates < 2019-01-01 and archive it in the orders archived table.**





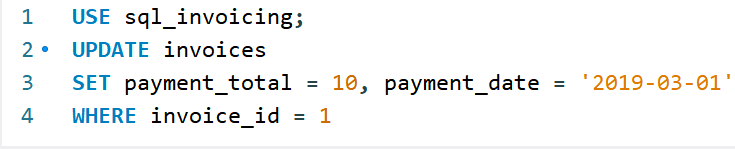
## **Creating a copy of a table - EXAMPLE**

### **Example – Create a new table to archive all invoices paid.**

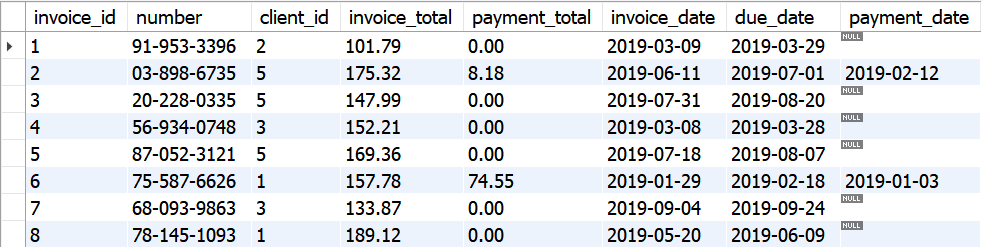


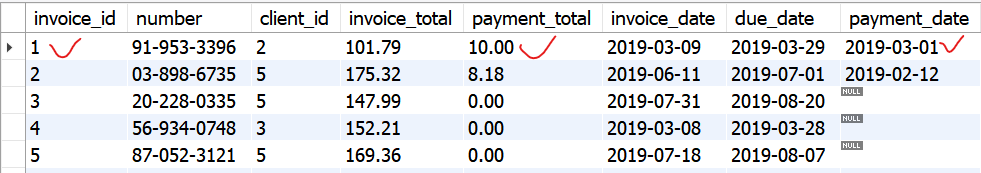
## **UPDATING a single row**

### **Update invoice id 1 for Payment total and Payment date.**

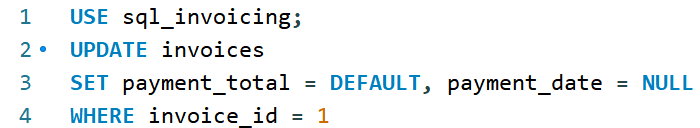
****

**invoice table before**



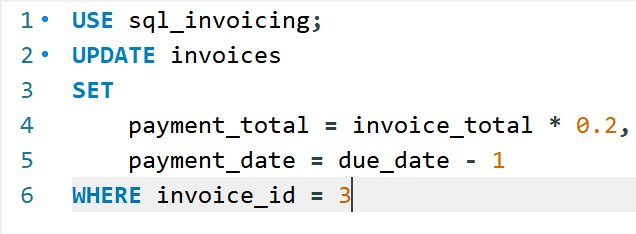
**invoice table AFTER**

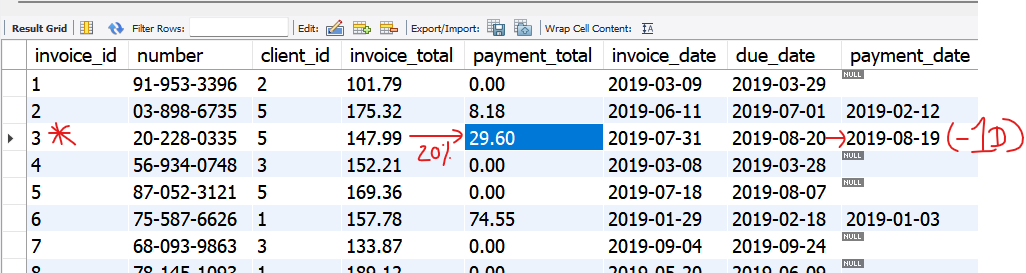
### **To UNDO changes – SET payment total = Default and payment date = None.**

****

## **UPDATING a single row from other columns**

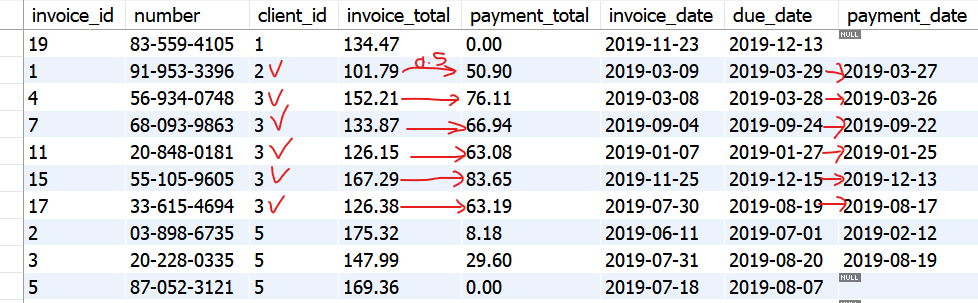
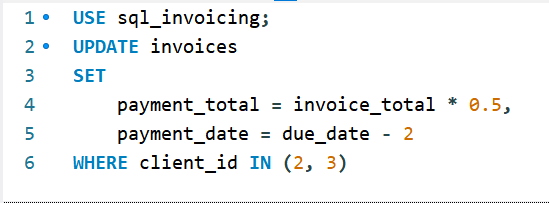
### **SET payment total = invoice total \* 20% and payment date = due date less 1 day.**





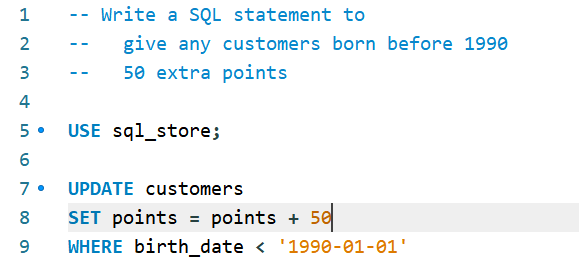
## **UPDATING Multiple Rows**

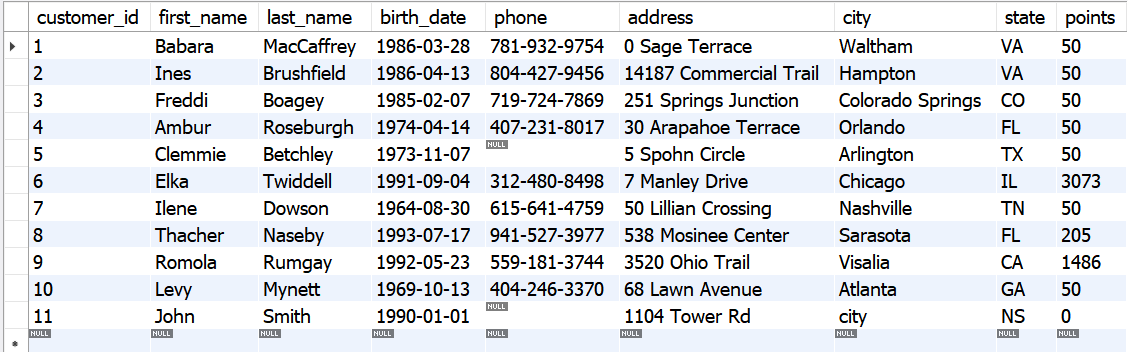
### **SET payment total = invoice total \* 50% and payment date = due date less 2 days for client 2 and client 3 having multiple rows.**



## **UPDATING Multiple Rows Exercise**

### **Exercise – Write a SQL st. to give all customers born before 1990, 50 extra points.**

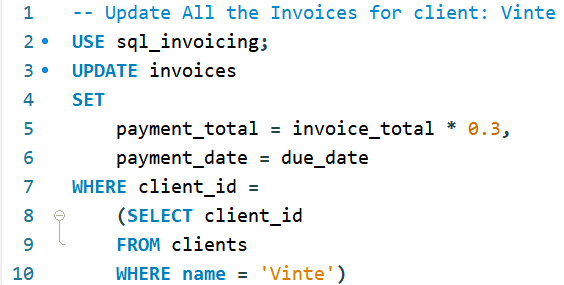


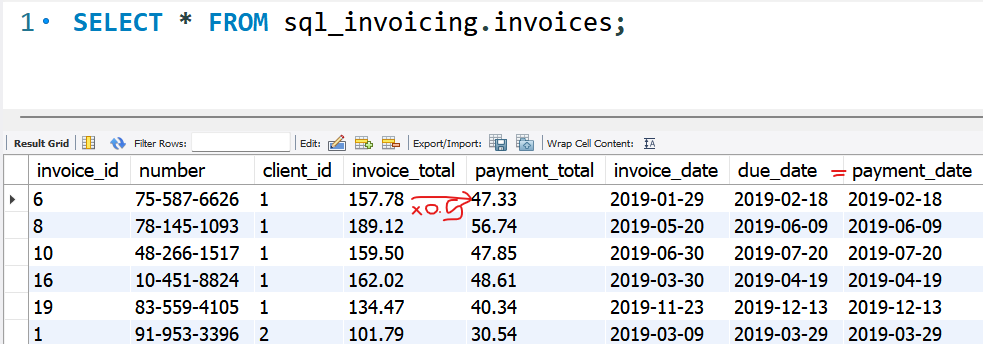
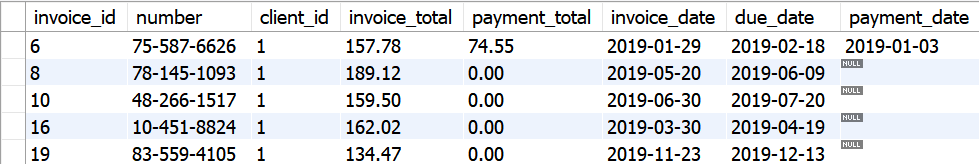


## **USING SUBQUERIES in updates.**

### **DEFINITION: A Subquery is a SELECT statement within another SELECT Statement.**

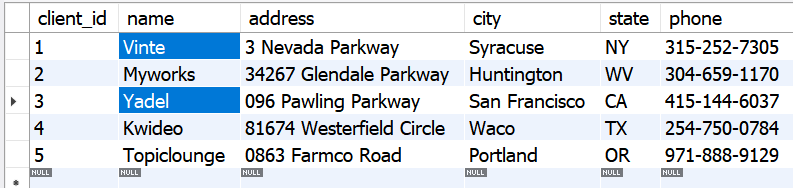
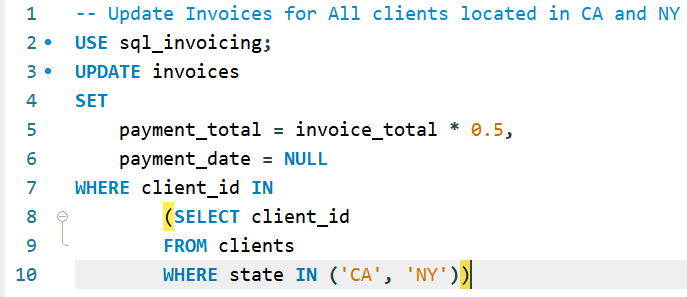
### **Exercise – Update ALL Invoices of client: Vinte. Run invoices table to see results.**



**BEFORE AFTER** 

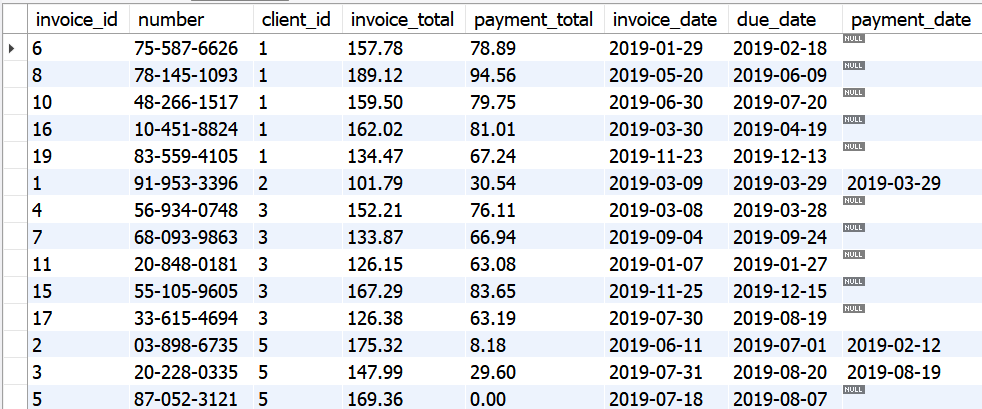
## **USING SUBQUERIES in updates.**

### **Exercise – Update Invoices for all clients in CA and NY. Note how the IN Operator has been used.**

****

**Notes**

* Client 1 Vinte and Client 3 Yadel are the clients living in NY and CA respectively.
* Hence you will see in the Invoice table that Client 1 and Client 3 has been updated.
* Left table = client table
* Bottom table = invoices table



## **USING SUBQUERIES in updates.**

**Notes**

* Before Updating All Invoices Where Payment Date is NULL,
* You will want to see the Invoice List where the Payment Date is NULL.
* Hence, you should firstly run:

**SELECT \***

**FROM invoices**

**WHERE payment\_date IS NULL**

* Once you are satisfied with the invoice list where payment date is NULL, you can then run the following SQL query

**USE sql\_invoicing;**

**UPDATE invoices**

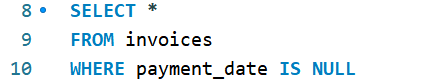
**SET**

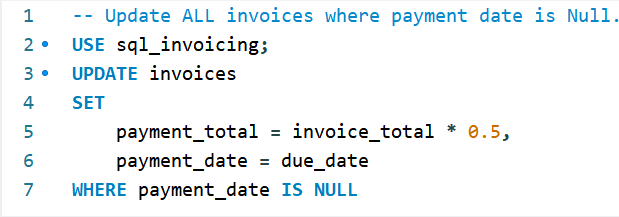
**payment\_total = invoice\_total \* 0.5**

**payment\_date = due\_date**

**WHERE payment\_date IS NULL**

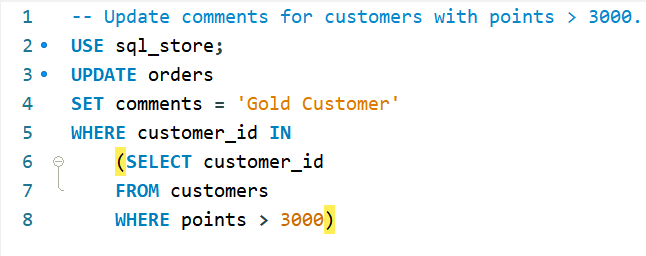
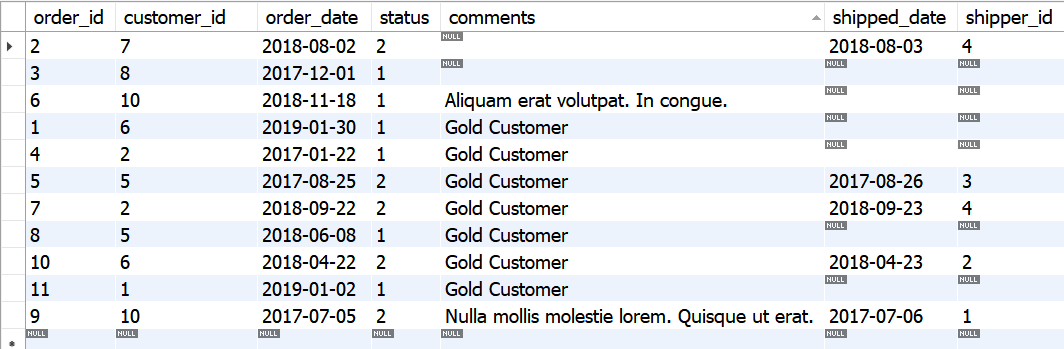
### **Exercise – Update ALL Invoices Where payment date is NULL.**





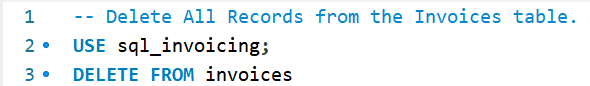
## **USING SUBQUERIES in updates.**

### **Exercise – Update comments in the orders table for customers in the customers table with points >3000.**

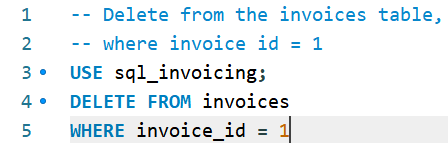


## **DELETING ROWS.**

### **DELETE ALL ROWS from the invoices table in the sql invoicing database.**

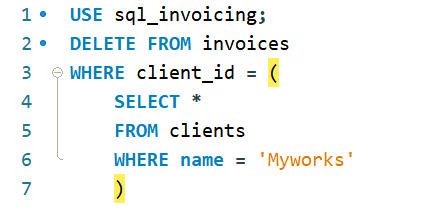


### **DELETE A ROW from the invoices table where invoice id = 1.**



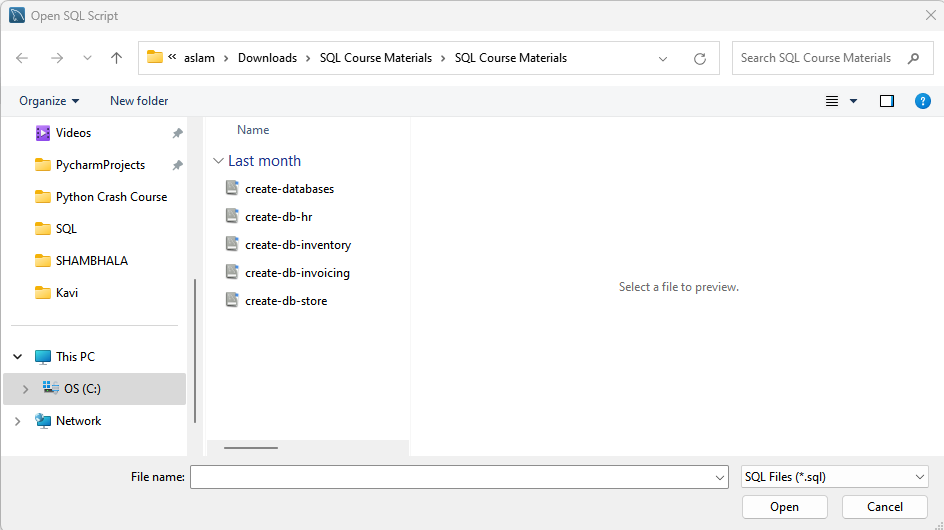
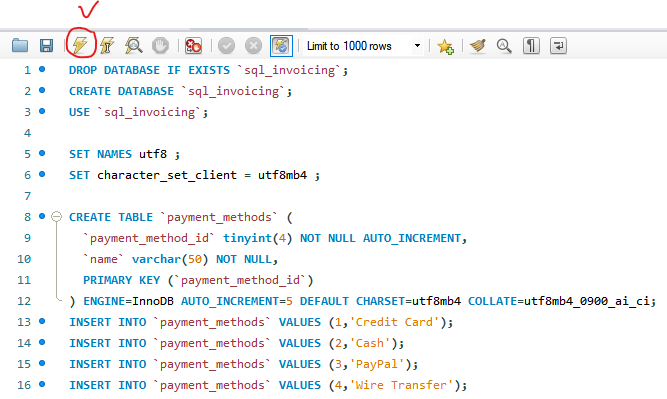
## **DELETING ROWS using Sub Queries.**

### **DELETE A Row from the invoices table where customer: ‘Myworks’ using a Subquery.**



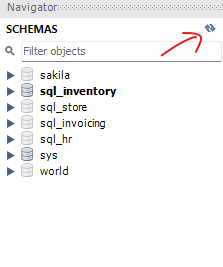
## **RESTORING the Databases.**

### **File -> Open SQL Script.**



**Notes**

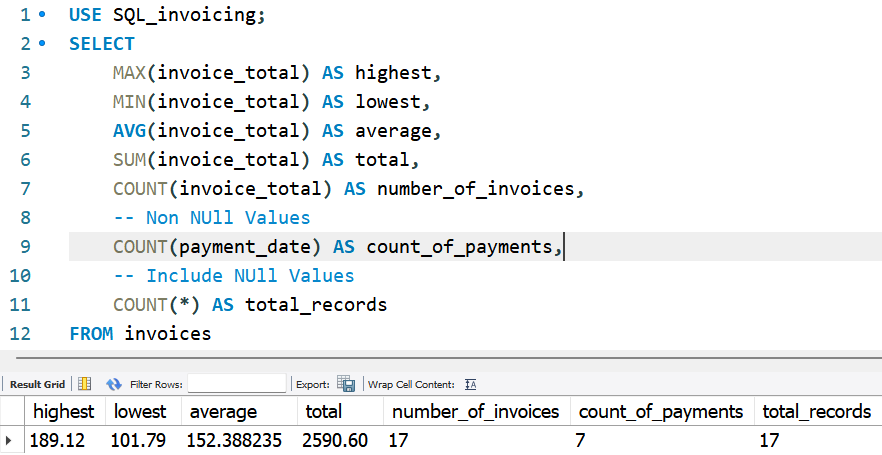
1. Go to **File**, Click **Open** SQL Script.
2. It will open the following window from where you lastly ran the SQL database.
3. Select ‘**create-database**’ and click **Open**.
4. It will load up a SQL code on MySQL.
5. **Execute** the SQL code and it will Restore All databases.
6. Click the Refresh  button on the Navigator Panel over to the left to view the databases.



# **SUMMARISING DATA IN MY SQL**

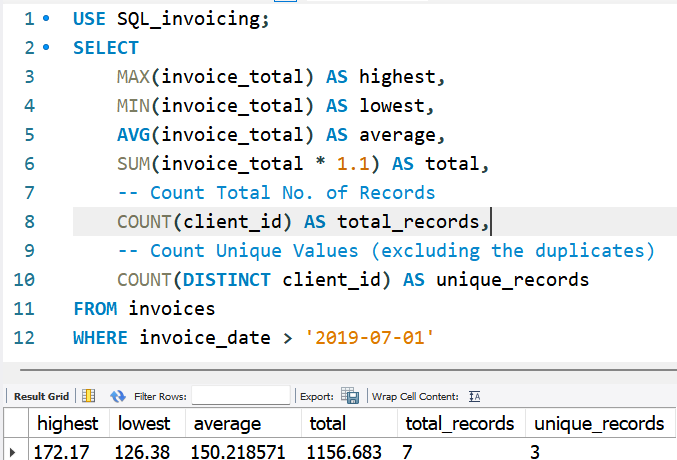
## **AGGREGATE Functions.**

### **Aggregate Functions = Queries that summarise data.**

1. **MAX( )**
2. **MIN( )**
3. **AVG( )**
4. **COUNT( ) – Non Null Values**
5. **COUNT(\*) – All Values**
6. **SUM( )**

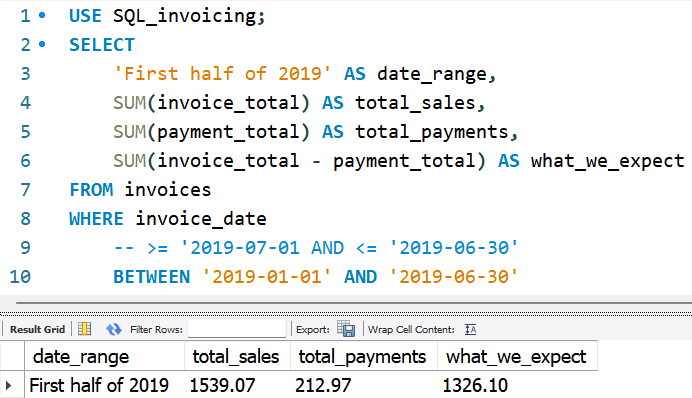
## **AGGREGATE Functions with Conditions**

### **Aggregate Functions Where Invoice Date > 2019-07-01.**



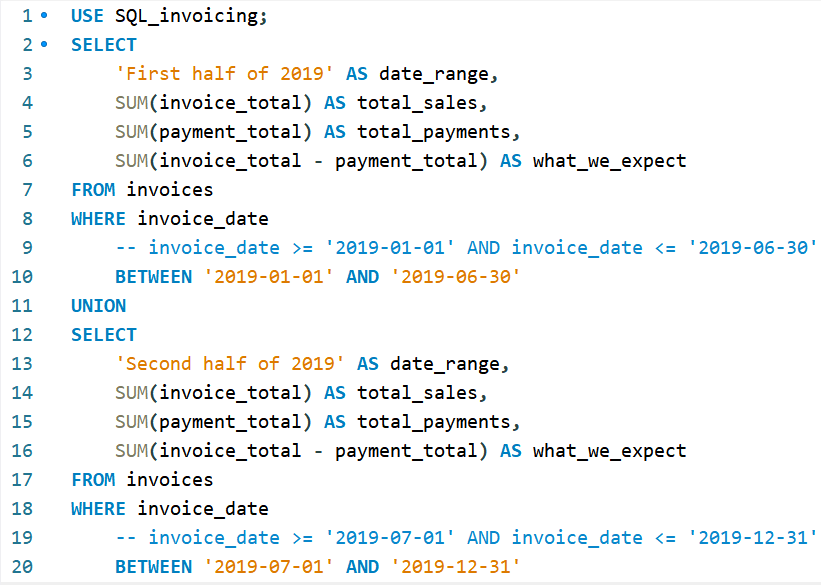
## **AGGREGATE Functions Exercise**

### **Aggregate Functions Exercise.**



## **AGGREGATE Functions Exercise**

### **Aggregate Functions Exercise.**

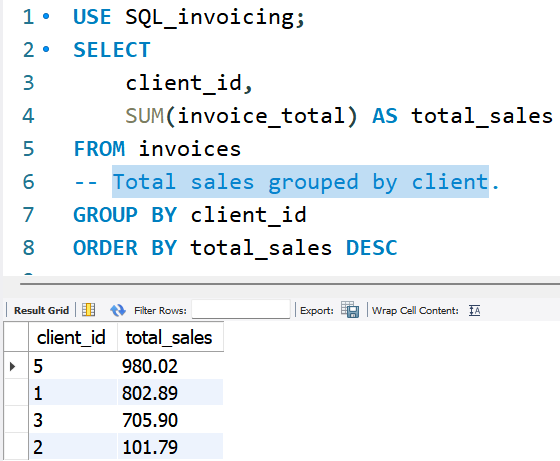
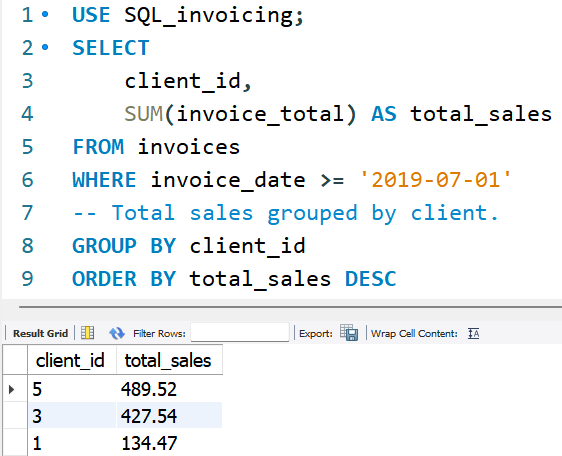


## **AGGREGATE Functions Exercise**

### **Summarise data using the Aggregate Function.**

## **GROUP BY Clause**

### **Total sales grouped by client.**



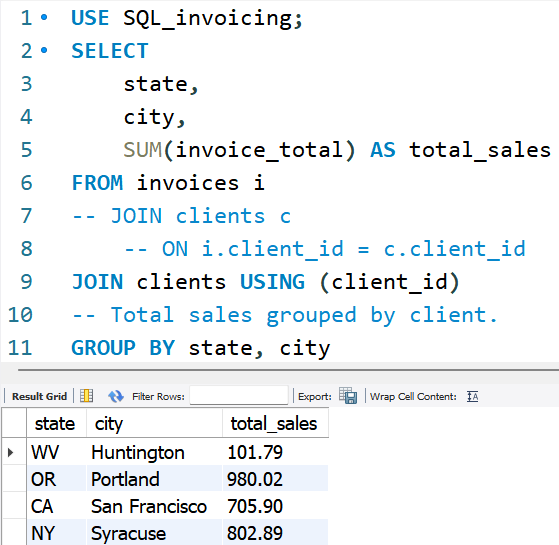
**NOTE**

**GROUP BY** Clause is always

1. AFTER the
   1. **FROM** and
   2. **WHERE** Clause and
2. BEFORE the
   1. **ORDER BY** clause.

## **GROUP BY Clause – State, City**

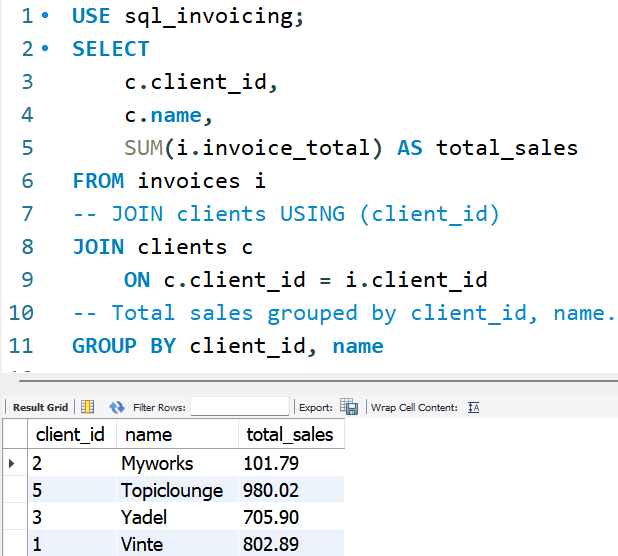
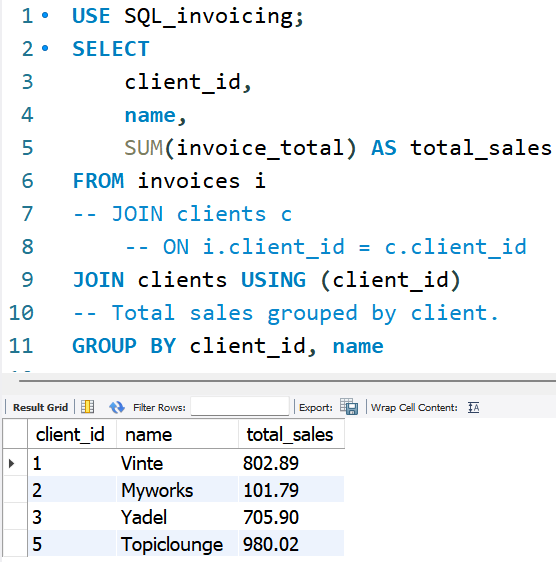
### **1. Total sales grouped by state and city.**



## **GROUP BY Clause – client id, name**

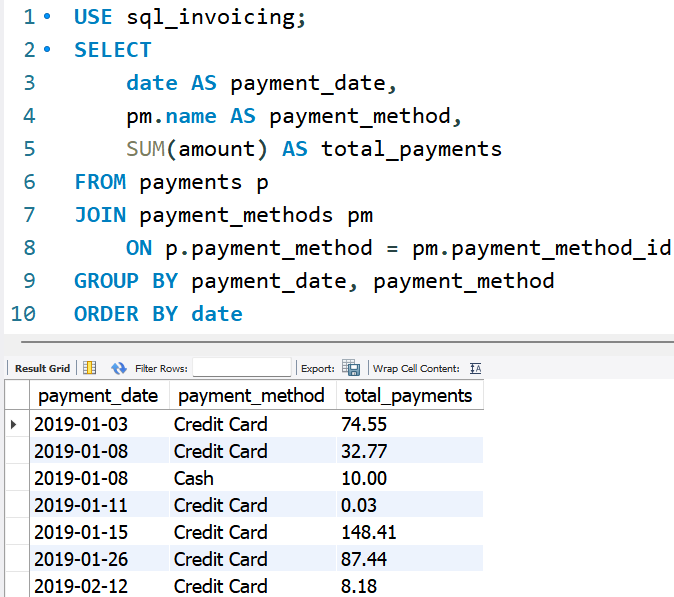
### **Total Sales grouped by client id, name using JOIN and USING clause vs. ON clause.**

**1. JOIN USING CLAUSE 2. ON CLAUSE**

****

## **GROUP BY Clause – pmt date, pmt method**

### **Total Payments Grouped By payment date and payment method.**



## **HAVING Clause to filter data AFTER the data is GROUPED**

**NOTE**

Question:

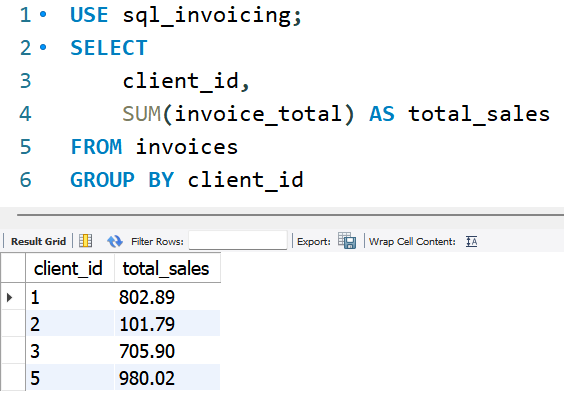
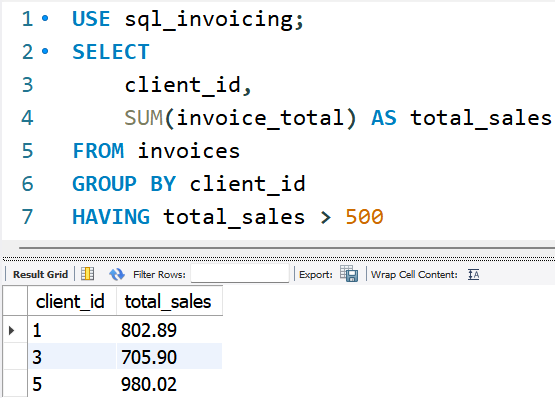
* What if we want to include clients with Total Sales of more than $500.
* Note – That Total Sales gets calculated at line 6 where it is subtotaled for each Client ID.
* That means we want to exclude Client ID 2 which has 101.79 Total Sales.
* How can we do this?
* WHERE > 500 clause
  1. after the FROM invoices

(line 5) and

* 1. before the GROUP BY Client ID (line 6),
  2. cannot be used because it is at line 6 that the Total Sales is calculated based on Client ID.
* The workaround that is to use a HAVING clause after the GROUP BY clause.
* **When we want to FILTER DATA BEFORE GROUPING our rows, we can use a WHERE Clause.**
* **When we want to Filter Data AFTER GROUPING our rows, we can use a HAVING Clause.**

### **1. Get the Total Sales grouped by Client ID.**

### **2. Thereafter, Filter out Total Sales > 500.**



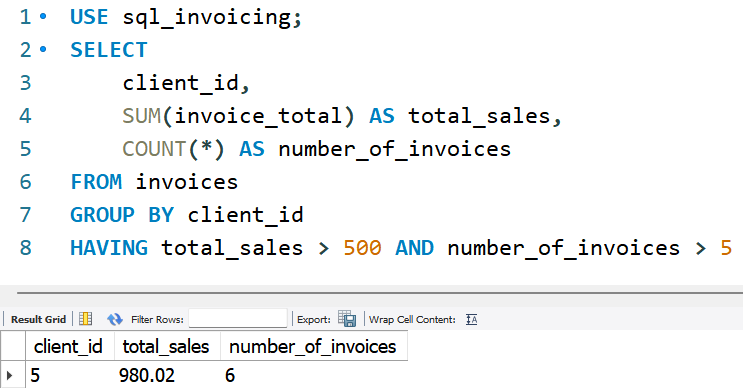
## **HAVING Clause to filter data AFTER the data is GROUPED**

### **1. Get the Total Sales grouped by Client ID, and**

**NOTES**

1. Columns that we use in the HAVING clause should be already included in the SELECT Clause.
2. Columns that we use the WHERE clause doesn’t need to be in the SELECT Clause

### **2. Thereafter, Filter out Total Sales > 500 AND number of invoices > 5.**

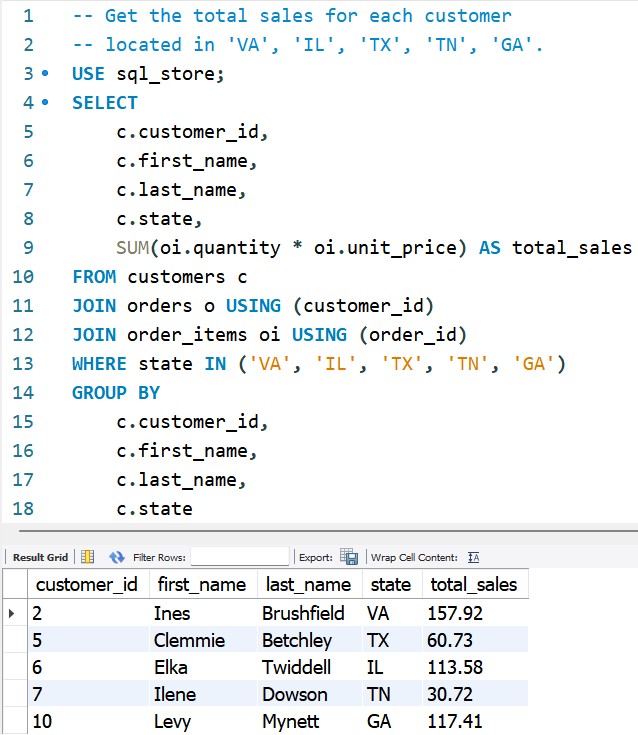


## **HAVING Clause Example**

**NOTES**

1. As a RULE OF THUMB, whenever you have a SUM function in a SELECT statement, you should GROUP by all the columns in the SELECT clause.
2. Columns in the SELECT statement
   1. c.customer\_id,
   2. c.first\_name,
   3. c.last\_name,
   4. c.state
3. Columns in the GROUP BY statement
   1. c.customer\_id,
   2. c.first\_name,
   3. c.last\_name,
   4. c.state

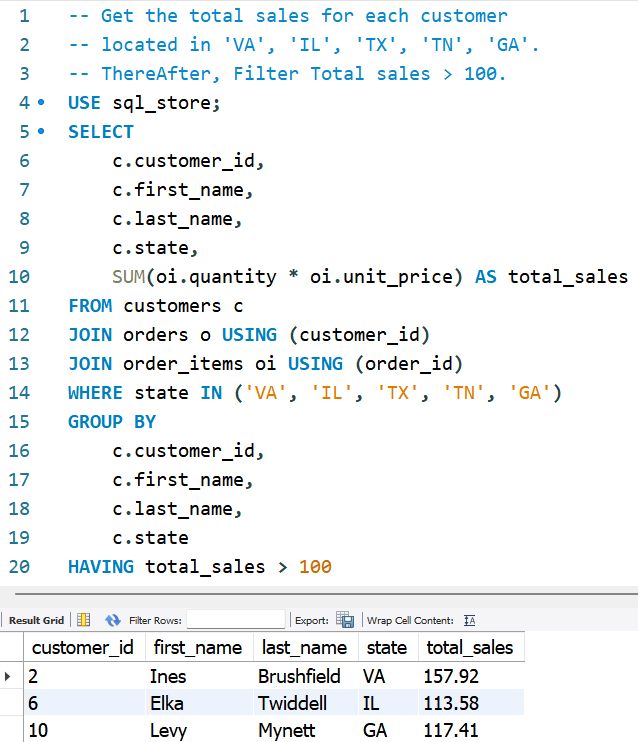
### **1. Calculate Total Sales for each customer located in VA, IL, TX, TN, GA.**



## **HAVING Clause Example continued…**

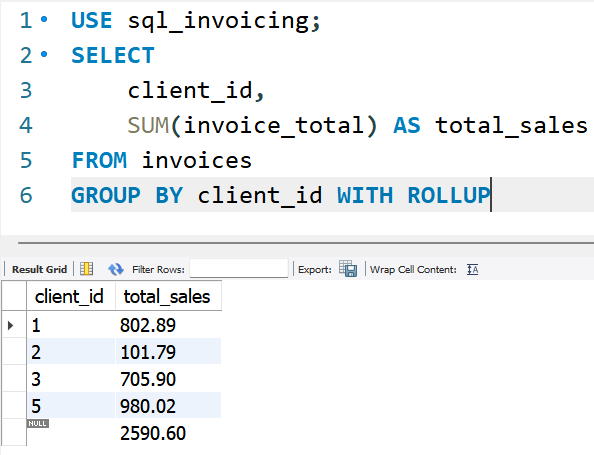
### **1. Calculate Total Sales for each customer located in VA, IL, TX, TN, GA.**

### **2. Thereafter, filter for total sales > 100.**



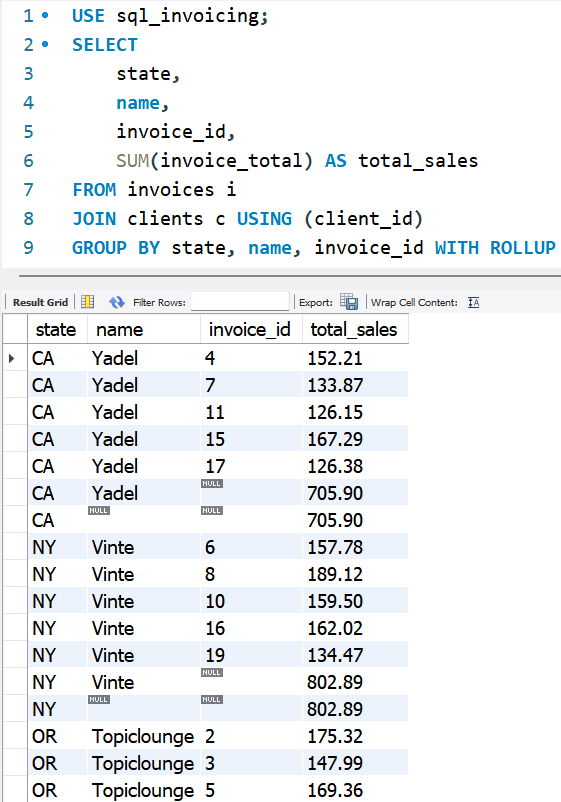
## **ROLL UP Operator: Sub-Total value columns**

### **The Roll Up Operator only works on columns that Aggregate values.**

****

## **ROLL UP Operator**

### **Sub-total ‘Total sales’ per State first and per Name second.**



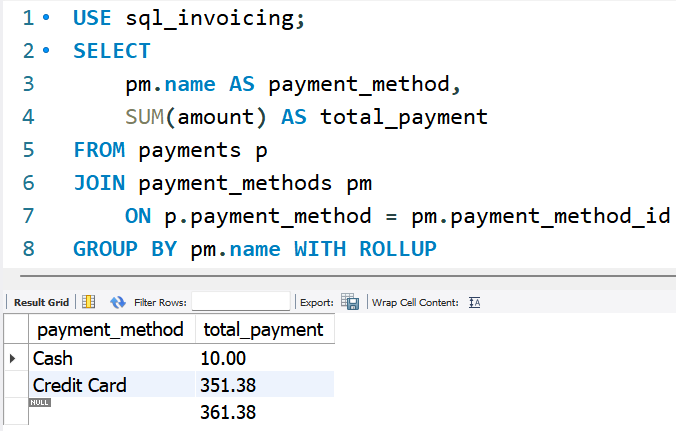
## **ROLL UP Operator Exercise**

**NOTE**

**IMPORTANT:**

When Using the GROUP BY statement you need to use the ACTUAL NAME of the COLUMN and NOT the ALIAS NAME.

### **Sub-total ‘Total sales’ per State first and per Name second.**



# **WRITING COMPLEX QUERY**