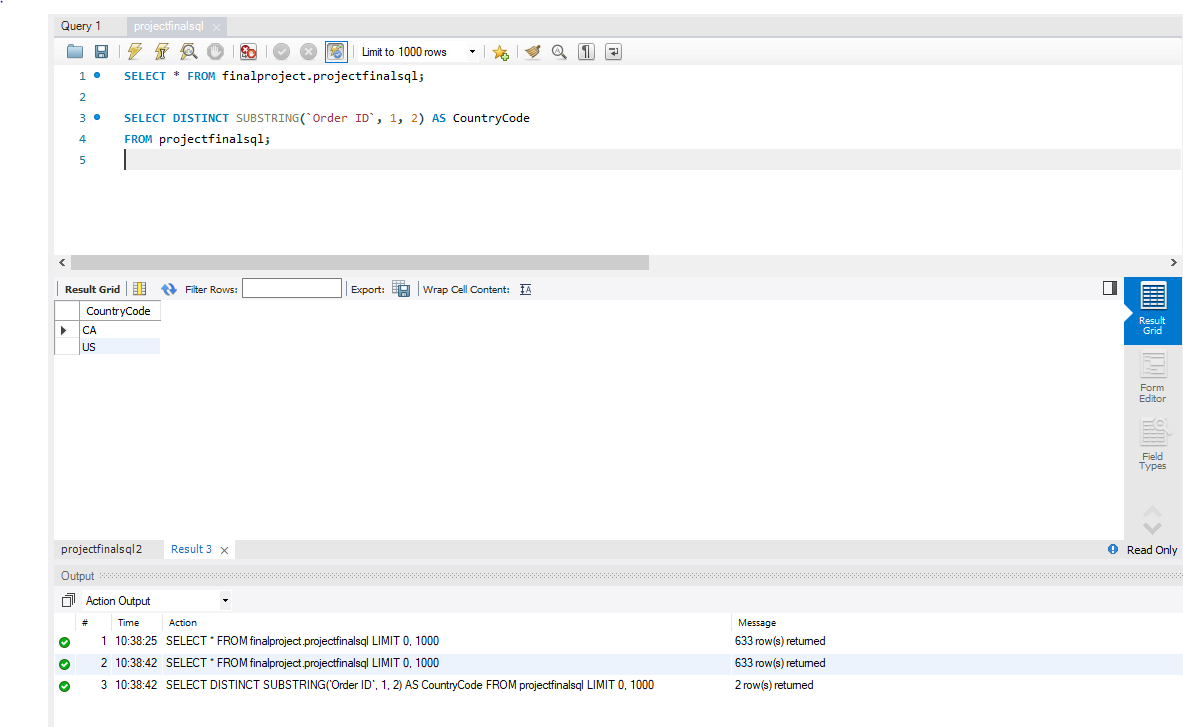
**1.Give the Country code from the order id column**

**Query**

SELECT DISTINCT SUBSTRING(`Order ID`, 1, 2) AS CountryCode

FROM projectfinalsql;

**Output:**

**2.Get the top 5 profit from sales for each city wise**

**Query**

SELECT City, MAX(Profit) AS TopProfit

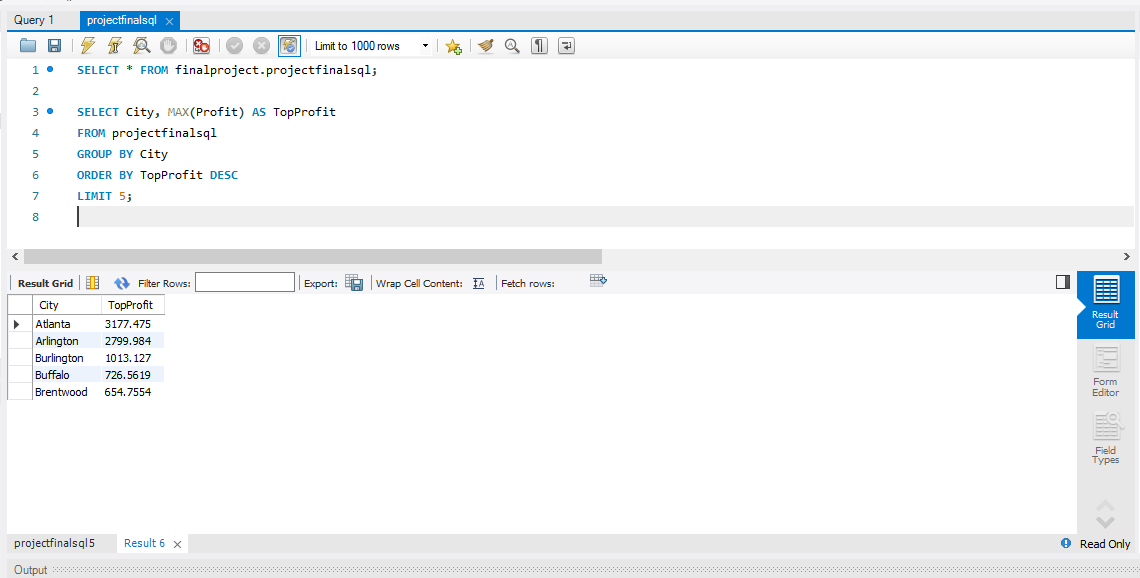
FROM projectfinalsql

GROUP BY City

ORDER BY TopProfit DESC

LIMIT 5;

**Output**



**3.Extract the maximum values using row number function**

**Query**

WITH RankedData AS (

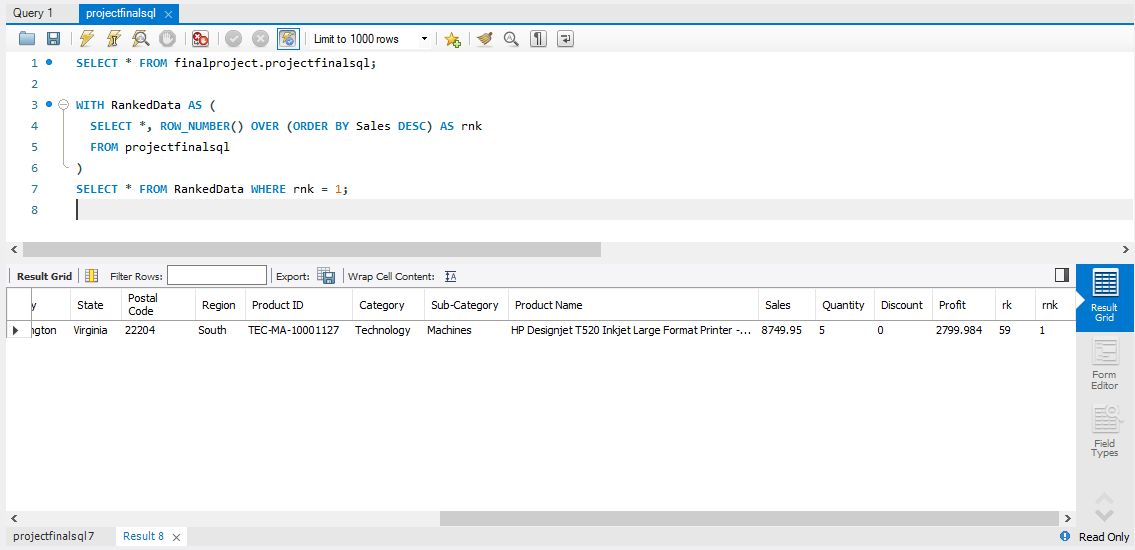
SELECT \*, ROW\_NUMBER() OVER (ORDER BY Sales DESC) AS rnk

FROM projectfinalsql

)

SELECT \* FROM RankedData WHERE rnk = 1;

**Output**



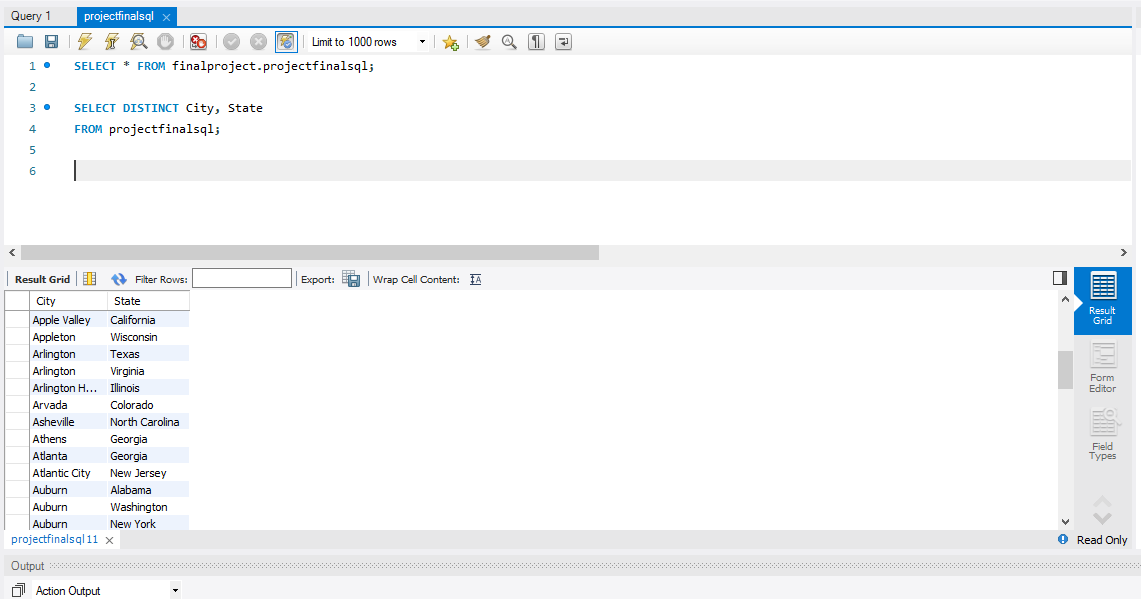
**4.remove duplicates from city , state**

**Query**

SELECT DISTINCT City, State

FROM projectfinalsql;

**Output**



**5.Segment wise sales which is greater than 500**

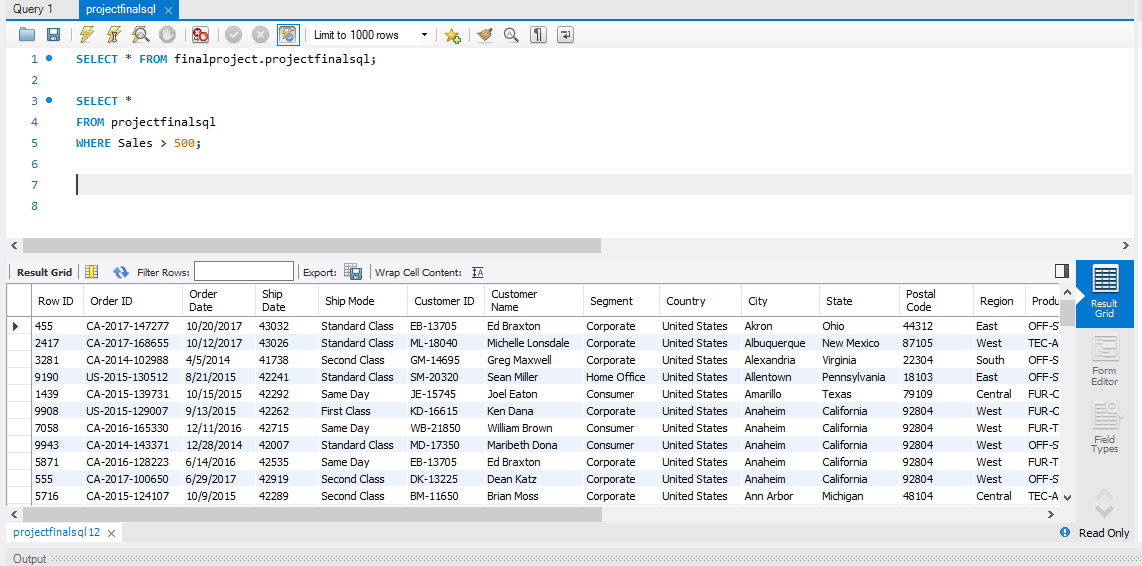
**Query**

SELECT \*

FROM projectfinalsql

WHERE Sales > 500;

**Output**



**6.Category Sales 0 - 100 as low , 101 - 500 medium , 500 - 2000 as high , > 2000 as best**

**Query**

SELECT \*,

CASE

WHEN Sales <= 100 THEN 'Low'

WHEN Sales <= 500 THEN 'Medium'

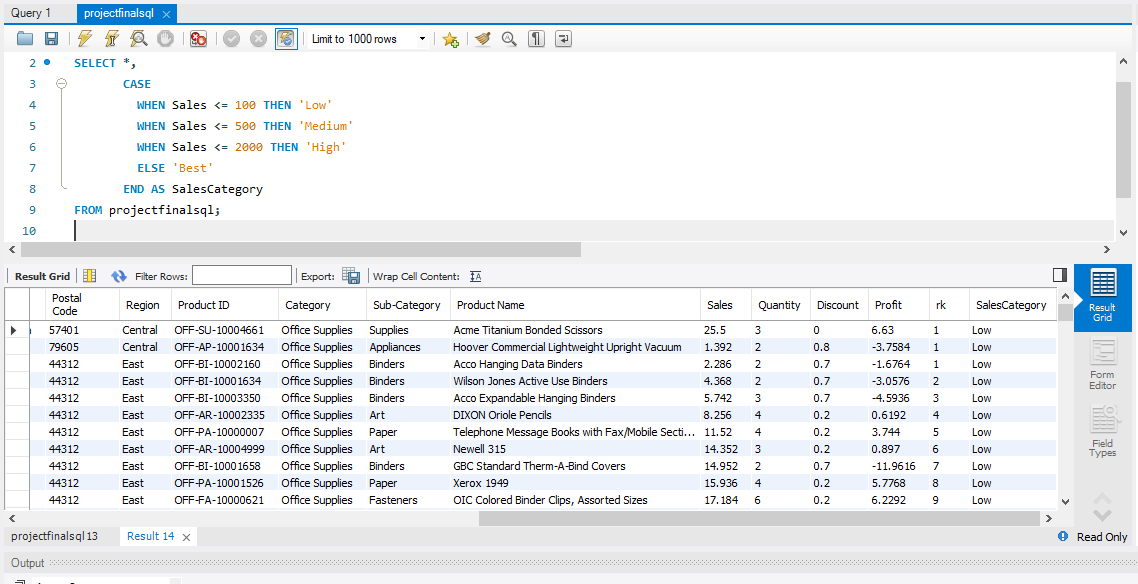
WHEN Sales <= 2000 THEN 'High'

ELSE 'Best'

END AS SalesCategory

FROM projectfinalsql;

**Output**



**7.city , region wise sum of sales , profits , count of sales & profits**

**Query**

SELECT City, Region,

SUM(Sales) AS TotalSales,

SUM(Profit) AS TotalProfits,

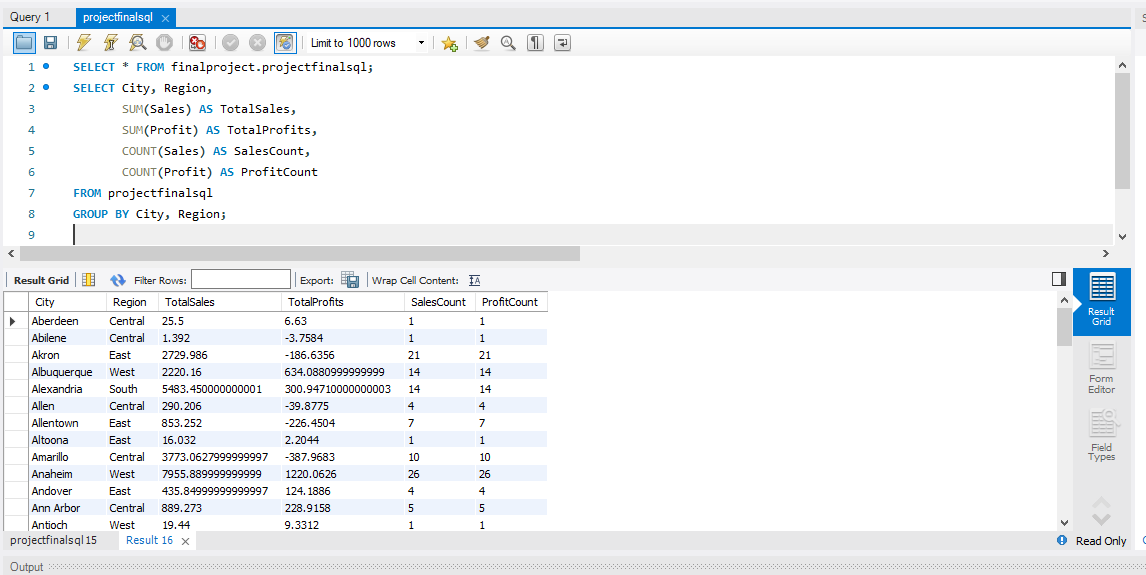
COUNT(Sales) AS SalesCount,

COUNT(Profit) AS ProfitCount

FROM projectfinalsql

GROUP BY City, Region;

**Output**



**8.Select all the records belongs to region east & west with sales > 500**

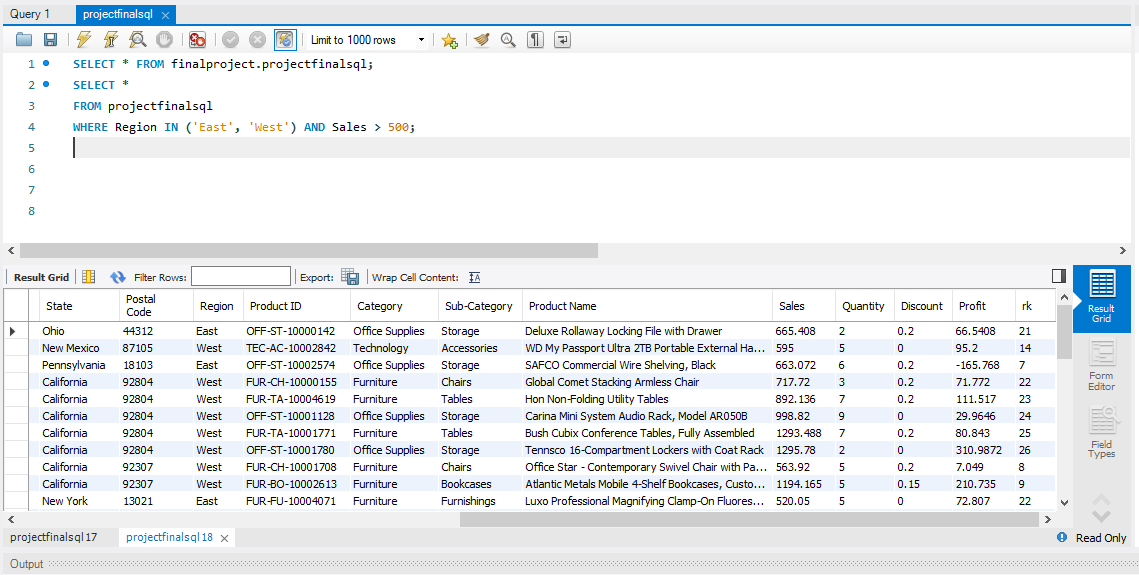
**Query**

SELECT \*

FROM projectfinalsql

WHERE Region IN ('East', 'West') AND Sales > 500;

**Output**



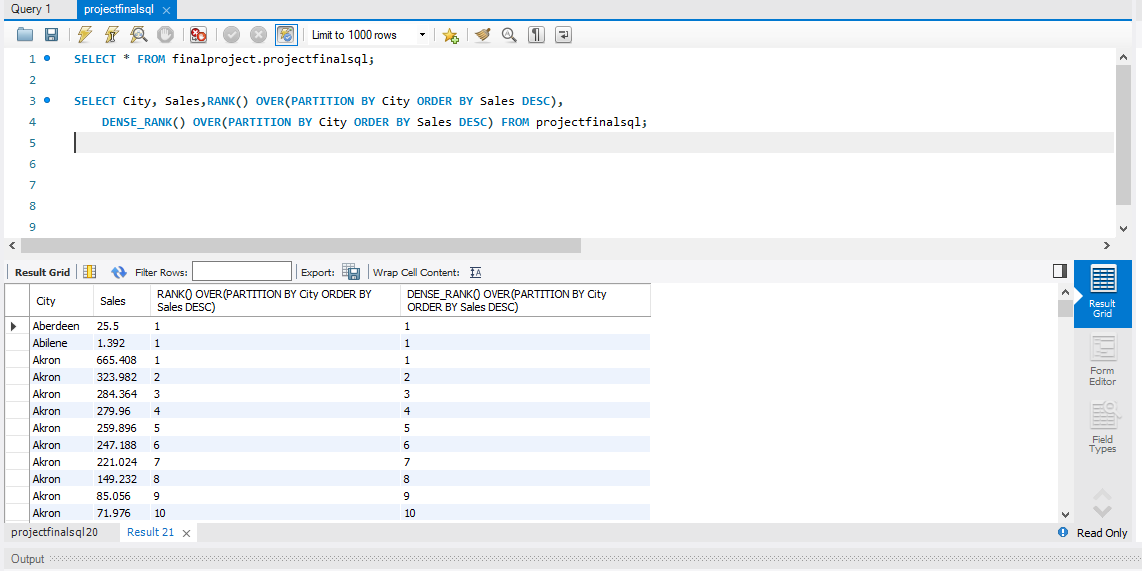
**9.Create rank , dense rank by city and sales**

**Query**

SELECT City, Sales,RANK() OVER(PARTITION BY City ORDER BY Sales DESC),

DENSE\_RANK() OVER(PARTITION BY City ORDER BY Sales DESC) FROM projectfinalsql;

**Output**



**10. describe about left , right , inner , self , cross joins & unions , subquries**

1. **Left Join:**
   * **Description:** A left join is used to combine rows from two or more tables based on a related column, returning all records from the left table and the matched records from the right table. If there is no match, NULL values are returned for columns from the right table.
2. **Right Join:**
   * **Description:** A right join is similar to a left join but returns all records from the right table and the matched records from the left table. If there is no match, NULL values are returned for columns from the left table.
3. **Inner Join:**
   * **Description:** An inner join is used to retrieve records that have matching values in both tables. It returns only the rows where there is a match between the specified columns in both tables.
4. **Self Join:**
   * **Description:** A self join is a regular join where a table is joined with itself. It is used when you want to combine rows within the same table based on related columns. It involves aliasing the table to distinguish between the two instances of the same table.
5. **Cross Join:**
   * **Description:** A cross join, or Cartesian join, returns the Cartesian product of the two tables. It combines every row from the first table with every row from the second table, resulting in a combination of all possible pairs of rows.
6. **Union:**
   * **Description:** The UNION operator is used to combine the results of two or more SELECT statements into a single result set. It removes duplicate rows from the combined result set. Each SELECT statement within the UNION must have the same number of columns with similar data types.
7. **Subqueries:**
   * **Description:** A subquery is a query nested within another query. It can be used in various parts of a SQL statement, such as SELECT, FROM, WHERE, or HAVING. Subqueries can be correlated, meaning they reference columns from the outer query, or non-correlated, where they are independent of the outer query.