SUMMARY: ADVANCED CONCEPTS IN JOINS

SESSION OVERVIEW:

By the end of this session, the students will be able to:

- Understand other types of JOINs.
- Understand the concepts of IFNULL and COALESCE.
- Perform advanced queries using different other functions.

KEY TOPICS AND EXAMPLES:

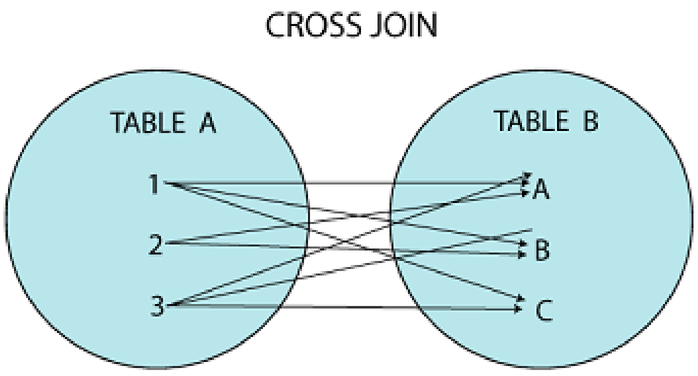
This session is more inclined toward the practical understanding of joins along with using other features which we have learned in our previous sessions. The practical understanding of this topic will help you generate insights from the dataset.

Understanding other types of JOINs:

1. **CROSS JOIN:**

   MySQL CROSS JOIN is used to combine all possibilities of two or more tables and returns the result that contains every row from all contributing tables. The CROSS JOIN is also known as CARTESIAN JOIN, which provides the Cartesian product of all associated tables. The Cartesian product can be explained as all rows present in the first table multiplied by all rows present in the second table.

   We can understand it with the following visual representation where CROSS JOIN returns all the records from table A and table B, and each row is the combination of rows of both tables.



Basic syntax:

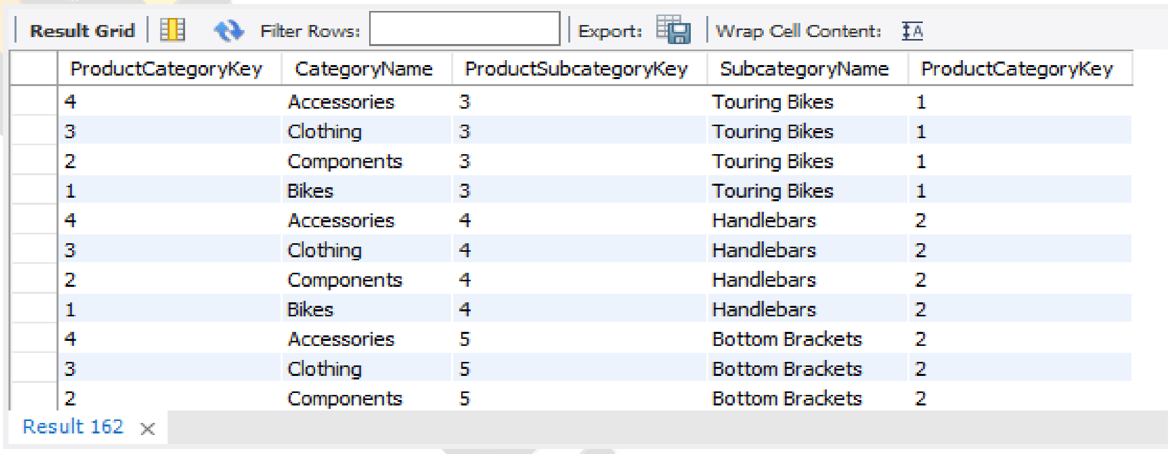
```
SELECT *
```

```
FROM table1
CROSS JOIN table2;
```

Example:

```
SELECT *
FROM product_categories
CROSS JOIN product_subcategories;
```

Output:



ProductCategoryKey	CategoryName	ProductSubcategoryKey	SubcategoryName	ProductCategoryKey
4	Accessories	3	Touring Bikes	1
3	Clothing	3	Touring Bikes	1
2	Components	3	Touring Bikes	1
1	Bikes	3	Touring Bikes	1
4	Accessories	4	Handlebars	2
3	Clothing	4	Handlebars	2
2	Components	4	Handlebars	2
1	Bikes	4	Handlebars	2
4	Accessories	5	Bottom Brackets	2
3	Clothing	5	Bottom Brackets	2
2	Components	5	Bottom Brackets	2

IMPORTANT NOTE:
It's important to use cross joins judiciously. Cross joins are not as commonly used in industry standards as other types of joins like inner joins, left joins, and so on. This is because cross-joins can potentially produce very large result sets, especially when used with tables that have a large number of rows.

In most cases, cross joins are used intentionally and with caution, often in specific scenarios where the Cartesian product is explicitly needed, such as generating all possible combinations for some analysis or calculation. However, they are not typically used for regular join operations where the goal is to match rows between tables based on some common criteria.

2. SELF JOIN:

A SELF JOIN is a join that is used to join a table with itself. In the previous sections, we have learned about the joining of the table with the other tables using different JOINS, such as INNER, LEFT, RIGHT, and CROSS JOIN. However, there is a need to combine data with other data in the same table itself. In that case, we use Self Join.

We can perform Self Join using table aliases. The table aliases allow us not to use the same table name twice with a single statement. If we use the same table name more than one time in a single query without table aliases, it will throw an error.

Basic syntax:

```
SELECT columns
FROM table1 alias1
JOIN table1 alias2 ON alias1.column_name = alias2.column_name;
```

Note:

- ***table1 is the name of the table you're joining.***
- ***alias1 and alias2 are aliases for the same table (table1). These aliases are used to differentiate between the two instances of the same table in the query.***
- ***column_name is the column you're using to join the table to itself. This column should relate to the same entity (e.g., employee ID, manager ID) within the table.***

Example:

```
SELECT e.employee_name AS employee, m.employee_name AS manager
FROM employees e
JOIN employees m ON e.manager_id = m.employee_id;
```

This query performs a self-join on the employees table to retrieve the names of employees along with the names of their managers. It uses aliases (e and m) to differentiate between the employee and manager in the join. The join condition (e.manager_id = m.employee_id) matches employees with their corresponding managers based on the manager_id and employee_id columns. The result is a table showing each employee along with their manager.

Understanding the concepts of IFNULL and COALESCE:

1. **Concepts of IFNULL:**

   The IFNULL function in SQL is used to handle null values in a database. It takes two arguments: the first argument is an expression that will be evaluated, and the second argument is the value that will be returned if the first argument is null.

Basic syntax:

```
IFNULL(expression, value_if_null)
```

Syntax using SELECT statement:

```
SELECT IFNULL(expression, value_if_null) AS alias
FROM table_name;
```

NOTE:

- ***IFNULL(expression, value_if_null): This is the IFNULL() function itself, where the expression is the value or column that you want to evaluate, and value_if_null is the value that will be returned if the expression evaluates to NULL.***
- ***AS alias: This is an optional part that allows you to give an alias (a different name) to the result of the IFNULL() function.***
- ***FROM table_name: This specifies the table from which you want to select data.***

2. **Concepts of COALESCE:**

The COALESCE() function in MySQL is used to return the first non-null value in a specified series of expressions. If this function evaluates all values of the list are null, or it does not find any non-null value, then it returns NULL.

Basic Syntax:

```
COALESCE(expr1, expr2, ...)
```

Syntax with SELECT statement:

```
SELECT COALESCE(column_name, value_if_null)
FROM table_name;
```

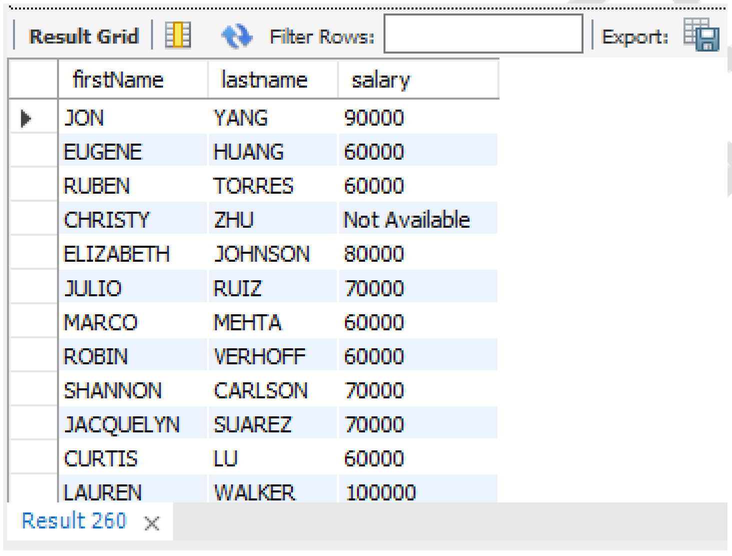
NOTE:

- ***column_name is the name of the column you want to check for null values.***
- ***value_if_null is the value that will be returned if the column_name is null.***

Example 1:

```
SELECT c.firstName, c.lastname, COALESCE(c.annualincome, 'Not
Available') AS salary
FROM customers c
LEFT JOIN sales_2015 s5 ON s5.customerkey = c.customerkey;
```

Output:



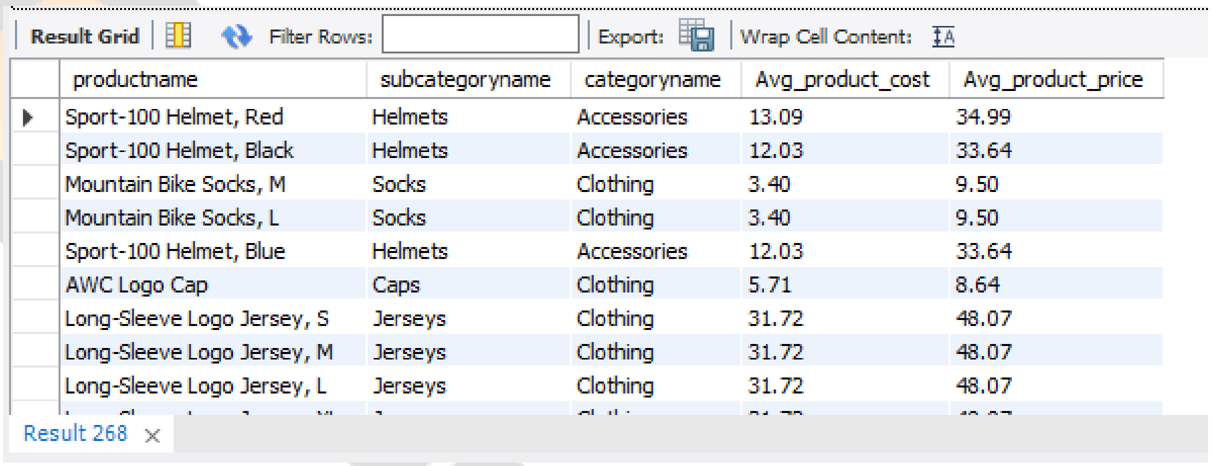
firstName	lastname	salary
JON	YANG	90000
EUGENE	HUANG	60000
RUBEN	TORRES	60000
CHRISTY	ZHU	Not Available
ELIZABETH	JOHNSON	80000
JULIO	RUIZ	70000
MARCO	MEHTA	60000
ROBIN	VERHOFF	60000
SHANNON	CARLSON	70000
JACQUELYN	SUAREZ	70000
CURTIS	LU	60000
LAUREN	WALKER	100000

Example 2:

```
SELECT p.productname, ps.subcategoryname, pc.categoryname,
       ROUND(CAST(AVG(COALESCE(p.productcost, 0)) AS DECIMAL(5,2)), 2)
AS Avg_product_cost,
       ROUND(CAST(AVG(COALESCE(p.productprice, 0)) AS DECIMAL(5,2)), 2)
AS Avg_product_price
FROM products AS p
```

```
JOIN product_subcategories AS ps ON
p.ProductSubcategoryKey=ps.ProductSubcategoryKey
JOIN product_categories AS pc ON
ps.ProductCategoryKey=pc.ProductCategoryKey
GROUP BY p.productname, ps.subcategoryname, pc.categoryname;
```

Output:



productname	subcategoryname	categoryname	Avg_product_cost	Avg_product_price
Sport-100 Helmet, Red	Helmets	Accessories	13.09	34.99
Sport-100 Helmet, Black	Helmets	Accessories	12.03	33.64
Mountain Bike Socks, M	Socks	Clothing	3.40	9.50
Mountain Bike Socks, L	Socks	Clothing	3.40	9.50
Sport-100 Helmet, Blue	Helmets	Accessories	12.03	33.64
AWC Logo Cap	Caps	Clothing	5.71	8.64
Long-Sleeve Logo Jersey, S	Jerseys	Clothing	31.72	48.07
Long-Sleeve Logo Jersey, M	Jerseys	Clothing	31.72	48.07
Long-Sleeve Logo Jersey, L	Jerseys	Clothing	31.72	48.07

Differences between IFNULL and COALESCE functions:

IFNULL()	COALESCE()
IFNULL() function takes only two expressions. After doing an evaluation, it returns the first expression if this expression is not NULL; otherwise, it returns the second expression.	The COALESCE() function returns the first non-null value in a specified series of expressions. If this function evaluates all values of the list are null, or it does not find any non-null value, then it returns NULL.

FUN TIME:

In this activity, you will be presented with a scenario involving multiple tables representing different aspects of a business. All the queries will be explained to you step-wise during the session by the instructor. The dataset that we are going to use for this insight generation purpose will be the same as we have used throughout the module which has been mentioned at the beginning of the module.

1. Total Return Quantity by Product Subcategory and Category:

Tracking the total return quantity by product subcategory and category is important for several reasons:

- **Identifying Product Performance:** It helps in identifying which product categories or subcategories have higher return rates. This information can be used to evaluate the quality, design, or marketing of these products.

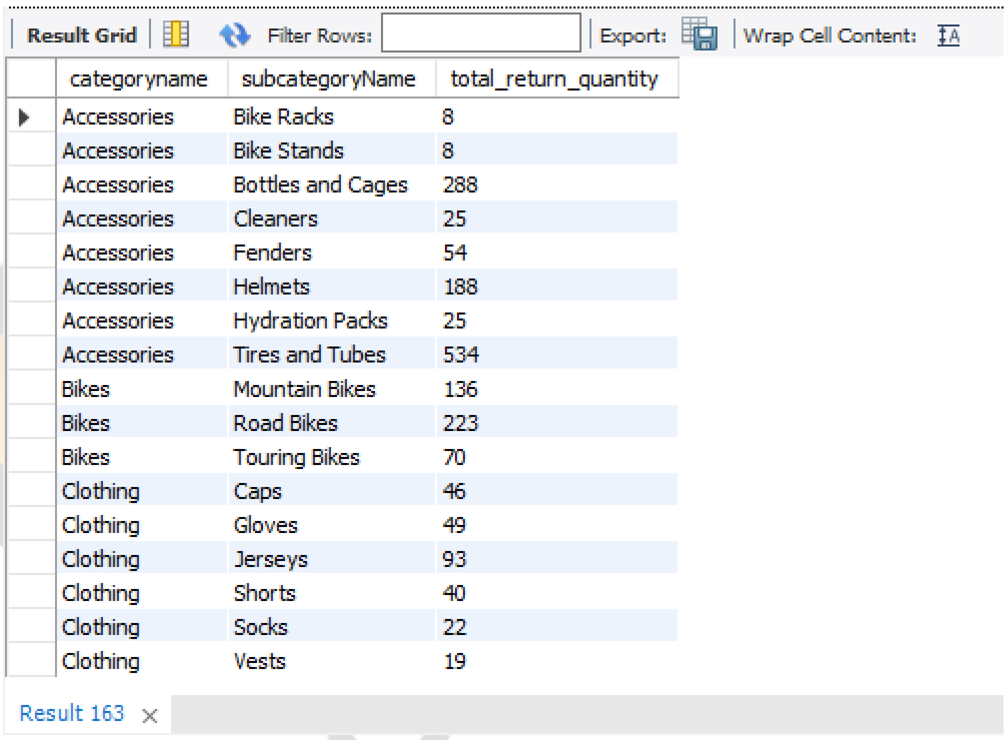
- **Improving Product Quality:** High return rates can indicate potential issues with product quality or functionality. By tracking returns, a company can identify areas for improvement and take corrective actions to enhance product quality.

- **Optimizing Inventory Management:** Understanding return patterns can help in optimizing inventory levels. For products with high return rates, inventory levels can be adjusted to minimize excess stock and associated costs.

```sql
SELECT pc. categoryname, ps.subcategoryName, SUM(r.returnQuantity) AS
total_return_quantity
FROM returns r
JOIN products p ON r.productkey = p.ProductKey
JOIN product_subcategories ps ON p.productsubcategorykey =
ps.productsubcategorykey
JOIN product_categories pc ON ps.productcategorykey =
pc.productcategorykey
GROUP BY pc. categoryname, ps.subcategoryName
ORDER BY pc. categoryname, ps.subcategoryName;
```

Description:
This query calculates the total return quantity for each product subcategory within its respective category. It joins the returns table with the products, product_subcategories, and product_categories tables to link return quantities with their corresponding subcategories and categories. The results are grouped by category and subcategory and then ordered alphabetically by category and subcategory names.

Output:


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

categoryname	subcategoryName	total_return_quantity
Accessories	Bike Racks	8
Accessories	Bike Stands	8
Accessories	Bottles and Cages	288
Accessories	Cleaners	25
Accessories	Fenders	54
Accessories	Helmets	188
Accessories	Hydration Packs	25
Accessories	Tires and Tubes	534
Bikes	Mountain Bikes	136
Bikes	Road Bikes	223
Bikes	Touring Bikes	70
Clothing	Caps	46
Clothing	Gloves	49
Clothing	Jerseys	93
Clothing	Shorts	40
Clothing	Socks	22
Clothing	Vests	19

Result 163

2. Territories with a Total Return Quantity Greater Than 200:

Tracking territories with a total return quantity greater than 200 can provide valuable insights and benefits for a company:

- **Identifying Problematic Regions:** Territories with high return quantities may indicate underlying issues such as product dissatisfaction, poor product fit for the market, or ineffective sales strategies in those regions. By identifying these problematic regions, a company can take targeted actions to address the root causes.

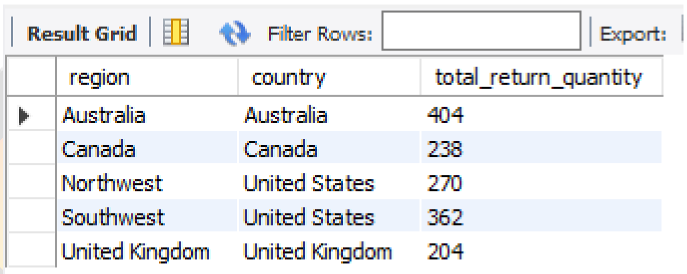
- **Improving Marketing and Sales Strategies:** Understanding return patterns in different territories can help a company tailor its marketing and sales strategies. It can lead to more targeted advertising, promotions, and product offerings that better resonate with customers in those regions, ultimately reducing return rates.

```sql
SELECT t.region, t.country, SUM(r.returnQuantity) AS
total_return_quantity
FROM returns r
JOIN territories t ON r.territorykey = t.salesterritorykey
GROUP BY t.region, t.country
HAVING total_return_quantity > 200
ORDER BY t.region, t.country;
```

Description:
This query calculates the total return quantity for each territory and retrieves territories where the total return quantity is greater than 200. It joins the returns table with the territories table using the

territorykey and salesterritorykey columns. The results are grouped by region and country, and the HAVING clause filters out territories where the total return quantity is not greater than 200. The territories are then ordered alphabetically by region and country names.

Output:



region	country	total_return_quantity
Australia	Australia	404
Canada	Canada	238
Northwest	United States	270
Southwest	United States	362
United Kingdom	United Kingdom	204

3. **Top 5 Product Subcategories by Total Return Quantity:**

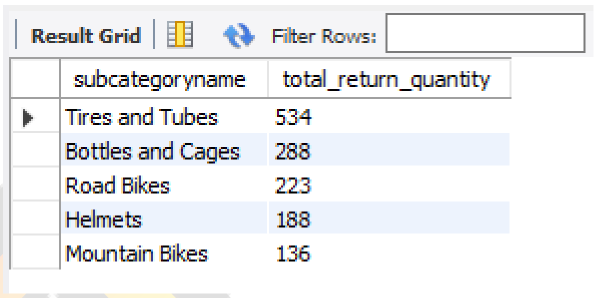
   Tracking the top 5 product subcategories by total return quantity can provide several benefits for a company:

   - **Identifying Problematic Product Lines:** It helps in identifying which product subcategories have the highest return rates. This information can be used to investigate and address potential issues such as product quality, design flaws, or customer dissatisfaction.
   - **Financial Impact:** Returns can have a significant financial impact on a company. By tracking return rates by product subcategory, companies can better understand the financial implications of returns and take steps to minimize their impact on profitability.

```sql
SELECT ps.subcategoryname, SUM(r.returnQuantity) AS
total_return_quantity
FROM returns r
JOIN products p ON r.productkey = p.ProductKey
JOIN product_subcategories ps ON p.productsubcategorykey =
ps.productsubcategorykey
JOIN product_categories pc ON ps.productcategorykey =
pc.productcategorykey
GROUP BY ps.subcategoryname
ORDER BY total_return_quantity DESC
LIMIT 5;
```

Description:
This query calculates the total return quantity for each product subcategory and retrieves the top 5 subcategories with the highest return quantities. It joins the returns table with the products, product_subcategories, and product_categories tables to link return quantities with their corresponding subcategories. The results are grouped by subcategory, and the subcategories are ordered in descending order of total return quantity. The LIMIT 5 clause is used to limit the output to the top 5 subcategories.

Output:


Result Grid | Filter Rows:

subcategoryname	total_return_quantity
Tires and Tubes	534
Bottles and Cages	288
Road Bikes	223
Helmets	188
Mountain Bikes	136

4. Products with No Returns:

Tracking products with no returns can provide valuable insights and benefits for a company:

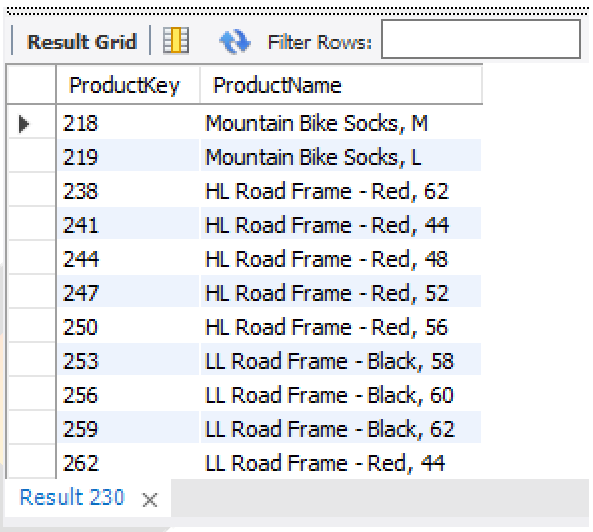
- **Product Quality Assurance:** Products with consistently no returns can indicate high product quality and customer satisfaction. Tracking these products can help a company identify successful product features and use them as benchmarks for future product development.
- **Identifying Sales Opportunities:** Products with no returns may indicate high customer demand or unique selling propositions. By tracking these products, a company can identify sales opportunities and allocate resources accordingly.
- **Cost Reduction:** Products with no returns incur lower costs associated with returns processing, restocking, and potential product damage. By focusing on products with no returns, a company can reduce overall operational costs.

```
SELECT p.ProductKey, p.ProductName
FROM products p
LEFT JOIN returns r ON p.ProductKey = r.productkey
WHERE r.returnQuantity IS NULL;
```

Description:

This query retrieves the ProductKey and ProductName of products that have not been returned. It uses a LEFT JOIN between the products table (p) and the returns table (r) on the ProductKey column. The WHERE clause filters the result to include only rows where returnQuantity is NULL, indicating no returns for that product.

Output:



ProductKey	ProductName
218	Mountain Bike Socks, M
219	Mountain Bike Socks, L
238	HL Road Frame - Red, 62
241	HL Road Frame - Red, 44
244	HL Road Frame - Red, 48
247	HL Road Frame - Red, 52
250	HL Road Frame - Red, 56
253	LL Road Frame - Black, 58
256	LL Road Frame - Black, 60
259	LL Road Frame - Black, 62
262	LL Road Frame - Red, 44

Result 230

5. Total Product Cost and Price by Category and Subcategory:

NOTE:
The below query is not considered to be the optimized form of using UNION or UNION ALL in SQL. The purpose of including this as an example is to help you understand the difference between an optimized query and an unoptimized query.

Tracking total product cost and price by category and subcategory can provide several benefits for a company:

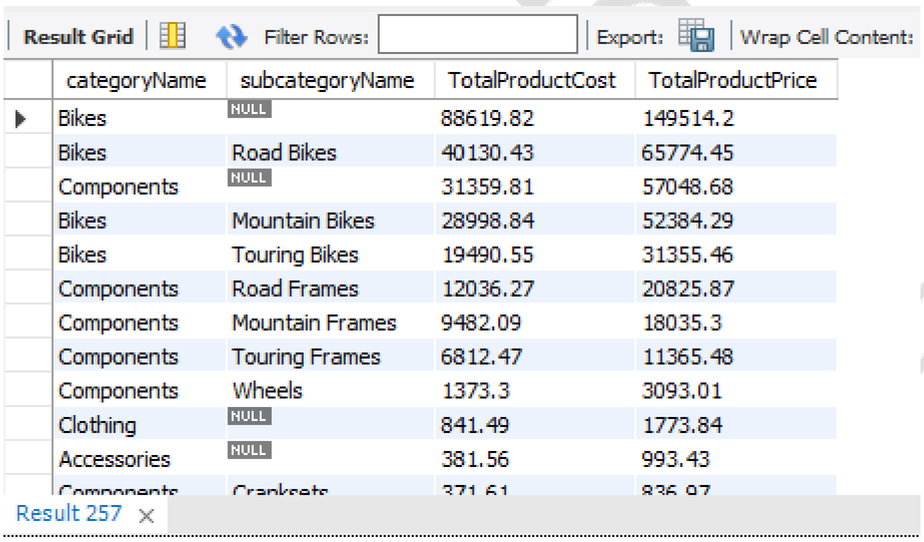
- **Profitability Analysis:** By comparing total product cost and price, a company can analyze the profitability of different product categories and subcategories. This information can help in pricing strategies and identifying areas for cost reduction or pricing adjustments.
- **Budgeting and Planning:** Tracking total product cost and price by category and subcategory can provide valuable information for budgeting and planning purposes. It can help in setting pricing targets, forecasting revenues, and managing costs effectively.
- **Product Development:** Insights from total product cost and price can inform product development strategies. Companies can focus on developing products in categories and subcategories that are more profitable or have higher demand.

```
SELECT pc.categoryName, ps.subcategoryName,
       ROUND(SUM(COALESCE(p.productCost, 0)), 2) AS TotalProductCost,
       ROUND(SUM(COALESCE(p.productPrice, 0)), 2) AS TotalProductPrice
FROM Products p
JOIN Product_Subcategories ps ON p.productSubcategoryKey =
ps.productSubcategoryKey
JOIN Product_Categories pc ON ps.productCategoryKey =
pc.productCategoryKey
GROUP BY pc.categoryName, ps.subcategoryName
```

```
UNION

SELECT pc.categoryName, NULL AS subcategoryName,
       ROUND(SUM(COALESCE(p.productCost, 0)), 2) AS TotalProductCost,
       ROUND(SUM(COALESCE(p.productPrice, 0)), 2) AS TotalProductPrice
FROM Products p
JOIN Product_Subcategories ps ON p.productSubcategoryKey =
ps.productSubcategoryKey
JOIN Product_Categories pc ON ps.productCategoryKey =
pc.productCategoryKey
GROUP BY pc.categoryName
ORDER BY TotalProductCost DESC;
```

Description:
This query calculates the total product cost and price for each product subcategory within its respective category. It uses the COALESCE function to handle NULL values in the productCost and productPrice columns, replacing them with 0. The results are grouped by category and subcategory, and then by category alone. The UNION operator is used to combine the results into a single list, with an additional row for each category showing the total cost and price for all products in that category. The final result is ordered by total product cost in descending order.

Output:



categoryName	subcategoryName	TotalProductCost	TotalProductPrice
Bikes	NULL	88619.82	149514.2
Bikes	Road Bikes	40130.43	65774.45
Components	NULL	31359.81	57048.68
Bikes	Mountain Bikes	28998.84	52384.29
Bikes	Touring Bikes	19490.55	31355.46
Components	Road Frames	12036.27	20825.87
Components	Mountain Frames	9482.09	18035.3
Components	Touring Frames	6812.47	11365.48
Components	Wheels	1373.3	3093.01
Clothing	NULL	841.49	1773.84
Accessories	NULL	381.56	993.43
Components	Cranksets	371.61	836.97

6. **Top 5 Customers by Total Sales Quantity:**

Tracking the top 5 customers by total sales quantity can provide several benefits for a company:

- **Identifying Key Customers:** It helps in identifying the most valuable customers who contribute significantly to sales volume. This information can be used to tailor marketing efforts and provide personalized services to these customers.
- **Customer Relationship Management:** By tracking the top customers, a company can focus on building strong relationships with them. This can lead to increased customer loyalty and repeat business.
- **Sales Performance Analysis:** Insights from the top customers can help in analyzing sales performance. Companies can identify trends, patterns, and opportunities for growth based on the behavior of these key customers.

NOTE:
The below query is not considered to be the optimized form of using UNION or UNION ALL in SQL. The purpose of including this as an example is to help you understand the difference between an optimized query and an unoptimized query.

The following query is not optimized for the following reasons:

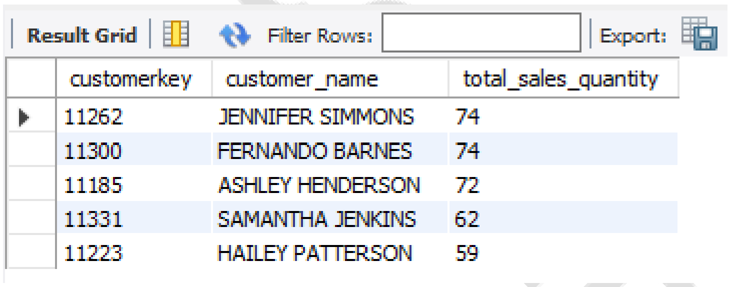
- ***Repetition of the CONCAT Function:*** *The CONCAT(c.firstname, ' ', c.lastname) expression is repeated in each subquery. While this does not impact the correctness of the query, it could be optimized for readability and maintainability by using a common table expression (CTE) or a subquery to avoid repetition.* ***(The concepts of CTE will be covered in the upcoming sessions)***

- ***Joining with the Same Table Multiple Times:*** *Each subquery joins the sales table with the customers table. While this is necessary to retrieve the customer name, it could be optimized by using subqueries or nested queries.* ***(The concepts of subquery or nested query will be covered in the upcoming sessions)***

```
SELECT c.customerkey,
CONCAT(c.firstname, ' ', c.lastname) AS customer_name,
SUM(s_2015.orderquantity) AS total_sales_quantity
FROM sales_2015 s_2015
JOIN customers c ON s_2015.customerkey = c.customerkey
GROUP BY c.customerkey, customer_name

UNION ALL

SELECT c.customerkey,
CONCAT(c.firstname, ' ', c.lastname) AS customer_name,
SUM(s_2016.orderquantity) AS total_sales_quantity
FROM sales_2016 s_2016
JOIN customers c ON s_2016.customerkey = c.customerkey
GROUP BY c.customerkey, customer_name

UNION ALL

SELECT c.customerkey,
CONCAT(c.firstname, ' ', c.lastname) AS customer_name,
SUM(s_2017.orderquantity) AS total_sales_quantity
FROM sales_2017 s_2017
JOIN customers c ON s_2017.customerkey = c.customerkey
GROUP BY c.customerkey, customer_name
ORDER BY total_sales_quantity DESC
LIMIT 5;
```

Description:
This query retrieves the total sales quantity for each customer across three different sales tables (sales_2015, sales_2016, sales_2017). It uses UNION ALL to combine the results from each year and then orders the combined result set by total_sales_quantity in descending order, limiting the output to the top 5 customers.

Output:



```
customerkey  customer_name      total_sales_quantity
11262        JENNIFER SIMMONS   74
11300        FERNANDO BARNES    74
11185        ASHLEY HENDERSON   72
11331        SAMANTHA JENKINS   62
11223        HAILEY PATTERSON   59
```