**MySQL Workbench Query Execution Documentation**

This documentation summarizes my practical experience executing SQL queries in MySQL Workbench.

---------------------------------------------------------------------------------------------------------------------

**Question: What is the average number of children per customer, rounded to the nearest whole number?**

SELECT ROUND(AVG(TotalChildren)) FROM sports.customer\_lookup

**Description:**

Uses AVG() aggregation function to compute the average and returns scalar result. Use when you need a quick summary statistic about customer demographics

**Question: Which customers have the maximum number of children?**

SELECT CustomerKey, Prefix, FirstName, LastName, TotalChildren

FROM sports.customer\_lookup

WHERE TotalChildren = (SELECT max(TotalChildren) FROM sports.customer\_lookup)

**Description:**

Uses a subquery to find the maximum value in TotalChildren column. Main query returns all customers who have this maximum number of children. Subquery executes first to determine the filter condition.

Use when you need to find records with extreme values (maximum/minimum).

**Question: Which customers have an annual income between 50,000 and 80,000?**

SELECT CustomerKey, Prefix, FirstName, LastName, AnnualIncome

FROM sports.customer\_lookup

WHERE AnnualIncome >= 50000 AND AnnualIncome <= 80000

**Description:**

Uses comparison operators (>= and <=) to define the rangeCould also be written using BETWEEN operator.

**Question: Which married customers who own homes have annual incomes of $90,000 or more?**

SELECT CustomerKey, Prefix, FirstName, LastName, AnnualIncome, MaritalStatus, HomeOwner

FROM sports.customer\_lookup

WHERE MaritalStatus = "M" AND HomeOwner = "Y" AND AnnualIncome >= 90000

**Description:**

Combines multiple conditions with AND operators. For targeted marketing campaigns.

**Question: Which customers have prefixes other than "MR.", "MRS.", or "MS."?**

SELECT \* FROM sports.customer\_lookup

WHERE Prefix != "MR." AND Prefix != "MRS." AND Prefix != "MS."

**Description:**

Uses inequality operators (!=) to exclude standard prefixes. Could also be written with NOT IN operator. For data quality checks

**Question: Which customers have no prefix specified?**

SELECT \* FROM sports.customer\_lookup WHERE Prefix = ""

**Description:**

Specifically looks for empty string values in the Prefix field. Different from NULL values (which would use IS NULL). When you need to identify missing information.

**Question: How many columns does the product\_lookup table have?**

SELECT COUNT(\*) AS total\_columns

FROM information\_schema.columns

WHERE table\_name = "product\_lookup"

**Description:**

Queries the metadata about the database schema. Uses the information\_schema system database. Returns a count of columns for the specified table. When exploring an unfamiliar database

**----Join-----**

These queries demonstrate different types of joins between product categories and subcategories tables.

When to use joins:

LEFT JOIN: When you want all records from the primary table with optional related data

RIGHT JOIN: Rarely used - typically restructure query to use LEFT JOIN instead

INNER JOIN: When you only want records with matches in both tables

FULL OUTER JOIN: When you want all records from both tables regardless of matches

**1.Left Join**

SELECT product\_subcategories\_lookup.ProductSubcategoryKey,

product\_subcategories\_lookup.SubcategoryName,

product\_subcategories\_lookup.ProductCategoryKey,

product\_categories\_lookup.CategoryName

FROM product\_subcategories\_lookup

LEFT JOIN product\_categories\_lookup

ON product\_subcategories\_lookup.ProductCategoryKey = product\_categories\_lookup.ProductCategoryKey

**Description:**

Returns all records from the left table (subcategories) with matching records from the right table. If no match, right table columns contain NULL

**2.Right Join**

SELECT product\_categories\_lookup.CategoryName,

product\_categories\_lookup.ProductCategoryKey,

product\_subcategories\_lookup.SubcategoryName

FROM product\_categories\_lookup

RIGHT JOIN product\_subcategories\_lookup

ON product\_categories\_lookup.ProductCategoryKey = product\_subcategories\_lookup.ProductCategoryKey

**Description:**

Returns all records from the right table (subcategories) with matching records from the left table. If no match, left table columns contain NULL

**3.Inner Join**

SELECT product\_categories\_lookup.CategoryName,

product\_categories\_lookup.ProductCategoryKey,

product\_subcategories\_lookup.SubcategoryName

FROM product\_categories\_lookup

INNER JOIN product\_subcategories\_lookup

ON product\_categories\_lookup.ProductCategoryKey = product\_subcategories\_lookup.ProductCategoryKey

**Description:**

Returns only records that have matches in both tables. Most common type of join for combining related data

**4.Simulated Full Outer Join**

SELECT c.CategoryName, c.ProductCategoryKey, s.SubcategoryName

FROM product\_categories\_lookup c

LEFT JOIN product\_subcategories\_lookup s

ON c.ProductCategoryKey = s.ProductCategoryKey

**5.UNION**

SELECT c.CategoryName, c.ProductCategoryKey, s.SubcategoryName

FROM product\_categories\_lookup c

RIGHT JOIN product\_subcategories\_lookup s

ON c.ProductCategoryKey = s.ProductCategoryKey;

**Description:**

Combines LEFT and RIGHT joins with UNION to simulate FULL OUTER JOIN. Returns all records from both tables, with NULLs where no match exists

**6.Self Join**

Are there any customers who share the same email address?

SELECT \*

FROM customer\_lookup AS t1

JOIN customer\_lookup AS t2

ON t1.EmailAddress = t2.EmailAddress

WHERE t1.CustomerKey != t2.CustomerKey

**Description:**

Joins a table to itself to find duplicate email addresses. Uses table aliases (t1, t2) to distinguish between instances. To find duplicate values in a table. For hierarchical data (e.g., employee-manager relationships)

**Question: How many customers are there in each occupation category?**

SELECT COUNT(customer\_lookup.CustomerKey) AS no\_of\_customers,

customer\_lookup.Occupation

FROM customer\_lookup

GROUP BY Occupation

SELECT COUNT(customer\_lookup.CustomerKey) AS no\_of\_customer\_key,

customer\_lookup.Occupation

FROM customer\_lookup

GROUP BY Occupation

HAVING no\_of\_customer\_key > 4000

**Description:**

Groups customers by occupation and counts them. HAVING clause filters groups after aggregation. Different from WHERE which filters before aggregation.

**Question: Which customers have placed orders with 2 or more items?**

SELECT customer\_lookup.CustomerKey, customer\_lookup.FirstName, customer\_lookup.LastName

FROM customer\_lookup

WHERE EXISTS (

SELECT \* FROM sales\_data

WHERE customer\_lookup.CustomerKey = sales\_data.CustomerKey

AND OrderQuantity >= 2

)

**Description:**

EXISTS returns true if the subquery returns any rows. More efficient than JOIN when you just need to check existence. Returns each customer only once, even with multiple qualifying orders. When you want to avoid duplicate rows from a JOIN

**Question: Which customers have first names starting with 'D'?**

SELECT \*

FROM customer\_lookup

WHERE customer\_lookup.FirstName LIKE "d%"

**Description:**

LIKE performs pattern matching. "%" is a wildcard matching any sequence of characters. When you need partial matches rather than exact matches

**Question: Which customers have occupations of "Manual" or "Skilled Manual"?**

SELECT \*

FROM customer\_lookup

WHERE customer\_lookup.Occupation IN ("Manual", "Skilled Manual")

**Description:**

IN checks if a value matches any in a list. More concise than multiple OR conditions

**Question: Which customers have annual incomes between 40000 and 70000?**

SELECT \*

FROM customer\_lookup

WHERE customer\_lookup.AnnualIncome BETWEEN 40000 AND 70000

**Description:**

BETWEEN is inclusive of both endpoints. More readable than using >= and <=

**Question: Which customers have incomes matching any of the incomes below $20,000?**

SELECT \*

FROM customer\_lookup

WHERE customer\_lookup.AnnualIncome = ANY (

SELECT customer\_lookup.AnnualIncome

FROM customer\_lookup

WHERE customer\_lookup.AnnualIncome < 20000

)

**Description:**

ANY returns true if the comparison is true for any value in the subquery. Similar to IN but more flexible with comparison operators.

NOTE: ANY takes first matching or true condition without taking duplicates from the subquery results, which means gives unique records

**Question: Which products have ALL of their orders with quantity > 0 and ProductKey = 310?**

SELECT product\_lookup.ProductName, product\_lookup.ProductKey

FROM product\_lookup

WHERE product\_lookup.ProductKey = ALL (

SELECT sales\_data.ProductKey

FROM sales\_data

WHERE sales\_data.OrderQuantity > 0 AND sales\_data.ProductKey = 310

)

**Description:**

ALL requires the comparison to be true for ALL values in the subquery. In this case, only returns products where every order meets the conditions. For strict validation requirements

NOTE: ALL, this will only return the product name and product key which matches ALL OF THE RESULTS IN THE SUBQUERY

**Question: How to create a new table combining product and sales data?**

CREATE TABLE product\_sales\_combo (

ProductKey INT,

ProductName TEXT,

OrderQuantity INT

)

INSERT INTO product\_sales\_combo

SELECT product\_lookup.ProductKey, product\_lookup.ProductName, sales\_data.OrderQuantity

FROM product\_lookup

JOIN sales\_data

ON product\_lookup.ProductKey = sales\_data.ProductKey

ALTER TABLE product\_sales\_combo

ADD COLUMN RowId INT AUTO\_INCREMENT PRIMARY KEY

**Description:**

Used to combine and store data for later use. Populates the new table with data from JOIN. Adds unique identifier to each row.

**Question: Show orders placed in January 2020 with quantities equal to 1**

SELECT sales\_data.OrderDate, sales\_data.OrderQuantity

FROM sales\_data

WHERE OrderDate BETWEEN '2020-01-01' AND '2020-01-31'

AND OrderQuantity = 1;

**Description:**

Used to analyze single-item purchases during a given month. This can help identify customer behavior such as trial purchases, or validate marketing campaigns that may have driven initial low-quantity sales.

**Question: Find min and max date in the sales\_data table**

SELECT MIN(OrderDate), MAX(OrderDate) FROM sales\_data;

**Description:**

Useful for understanding the time range of your dataset — often the first thing to check before performing time-based analytics or setting up dashboards with filters.

**Question: List returns from the "United Kingdom" territory**

SELECT territory\_lookup.Country, COUNT(returns\_data.ReturnQuantity)

FROM territory\_lookup

JOIN returns\_data

ON territory\_lookup.SalesTerritoryKey = returns\_data.TerritoryKey

WHERE territory\_lookup.Country = "United Kingdom";

**Description:**

Helpful for regional product performance analysis or identifying problematic territories with high return rates, which could lead to action items like quality checks, shipping review, or customer service improvements.

**Question: Find customers born before 1970 with "Bachelors" education**

SELECT CustomerKey, FirstName, LastName, BirthDate

FROM customer\_lookup

WHERE BirthDate < '1970-01-01' AND EducationLevel = 'Bachelors';

**Description:**

Targeted marketing campaigns, demographic profiling, or for understanding customer segmentation based on age and education level.

**Question: Display product names with their category names**

SELECT

product\_categories\_lookup.CategoryName,

product\_subcategories\_lookup.SubcategoryName,

product\_lookup.ProductName

FROM product\_subcategories\_lookup

JOIN product\_lookup

ON product\_subcategories\_lookup.ProductSubcategoryKey = product\_lookup.ProductSubcategoryKey

JOIN product\_categories\_lookup

ON product\_subcategories\_lookup.ProductCategoryKey = product\_categories\_lookup.ProductCategoryKey;

**Description:**

Used for catalog building, product filtering in e-commerce platforms, or understanding product taxonomy for reporting or analytics.

**Question: List all sales with customer names and product names**

SELECT

sales\_data.OrderNumber,

customer\_lookup.FirstName,

product\_lookup.ProductName

FROM sales\_data

JOIN customer\_lookup

ON sales\_data.CustomerKey = customer\_lookup.CustomerKey

JOIN product\_lookup

ON sales\_data.ProductKey = product\_lookup.ProductKey;

**Description:**

Basic but powerful for CRM analytics, sales reports, or combining order history with customer and product info for dashboards.

**Question: Show returns with product details**

SELECT p.ProductName, p.ProductPrice, r.ReturnQuantity

FROM returns\_data r

JOIN product\_lookup p

ON r.ProductKey = p.ProductKey;

**Description:**

Supports return analysis, helps identify high-return products (potential quality issues), and helps in inventory and refund planning.

**Question: Find orders with customer emails and territory countries**

SELECT s.OrderNumber, c.FirstName, c.EmailAddress, t.Country

FROM sales\_data s

JOIN customer\_lookup c

ON s.CustomerKey = c.CustomerKey

JOIN territory\_lookup t

ON s.TerritoryKey = t.SalesTerritoryKey;

**Description:**

Useful for regional email campaigns, support escalations, or understanding customer distribution across geographies.

**Question: List products that have never been returned**

SELECT product\_lookup.ProductKey, product\_lookup.ProductName, returns\_data.ReturnQuantity

FROM product\_lookup

LEFT JOIN returns\_data

ON product\_lookup.ProductKey = returns\_data.ProductKey

WHERE returns\_data.ReturnQuantity IS NULL;

**Description:**

Useful for identifying top-performing or reliable products, can influence promotions, restocking decisions, and supply chain optimization.