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.