Lab: Write SQL Data Manipulation Language

This lab exercise focuses on hands-on practice with SQL Data Manipulation Language (DML), using the Sales LT database within MySQL Workbench. The exercise is divided into sections, each covering various SQL operations that learners will perform by writing and running SQL queries. The main goal is to help learners understand how to manipulate and retrieve data from a relational database.

Please note that this lab is **not graded**, by completing this lab, learners will gain practical experience in querying and manipulating data using SQL. We encourage you to try this lab exercise independently. The SQL script for this exercise can be found [here](https://classroom.emeritus.org/courses/11616/files/3946781?wrap=1)[Download here](https://classroom.emeritus.org/courses/11616/files/3946781/download?download_frd=1).

**MODULE 1: BASIC QUERIES**

We'll use the **Sales LT** database in this lab, so let's start by exploring it in MySQL Work Bench

* Start MY SQL Work Bench. Open the SQL File “[Adventureworks-MySQL Setup](https://classroom.emeritus.org/courses/11616/files/4085517?wrap=1" \t "_blank)[Download Adventureworks-MySQL Setup](https://classroom.emeritus.org/courses/11616/files/4085517/download?download_frd=1)” Highlight the entire script, 3000++ rows. Execute the entire file to install the database.
* In the panel, select the **Sales LT** connection by selecting the name. This will connect to the SQL Server instance and show the objects in the **Sales LT** database.
* Expand the **SalesLT.Product** table and then expand its **Columns** folder to see the columns in this table. Each column has a name, a data type, an indication of whether it can contain *null* values, and in some cases an indication that the columns is used as a primary key (PK) or foreign key (FK).
* Right-click the **SalesLT.Product** table and use the **SELECT**option to create and run a new query script that retrieves all rows from the table.
* Review the query results, - each row representing a product that is sold by the fictitious *Adventure Works Cycles* company.
* Close the **SQLQuery\_1** pane that contains the query and its results.
* Explore the other tables in the database, which contain information about product details, customers, and sales orders. The tables are related through primary and foreign keys, as shown here:

**SECTION 1: Use SELECT queries to retrieve data**

Now that you've had a chance to explore the **Sales LT** database, it's time to dig a little deeper into the product data it contains by querying the **Product** table.

* Create a new query (you can do this clicking on New SQL
* In the new **SQL Query\_…** pane, ensure that the **Sales LT** database is selected
* In the query editor, enter the following code: SELECT \* FROM SalesLT.Product;
* Use the **Run** button to run the query, and after a few seconds, review the results.
* In the query editor, enter a new query to display the 3 columns as shown.

      SELECT Name, StandardCost, ListPrice

      FROM SalesLT.Product;

* Use the **Run** button to re-run the query, and review the results, which this time include only the **Name**, **StandardCost**, and **ListPrice** columns for all products.
* Create a new query to include an expression that results in a calculated column

      SELECT Name, ListPrice - StandardCost

      FROM SalesLT.Product;

* Rerun the query, note that the results include the **Name** column and an unnamed column of the result of subtracting the **StandardCost** from the **ListPrice**.
* Modify the query to assign names to the columns in the results.

      SELECT Name AS ProductName, ListPrice - StandardCost AS Markup

       FROM SalesLT.Product;

* Rerun the query. The results now include columns **ProductName** and **Markup**. The **AS** keyword has been used to assign an *alias* for each column in the results.
* Create a new query with the following code, which also includes an expression that produces a calculated column in the results:

       SELECT ProductNumber,

      Color,

      Size,

      CONCAT(Color , ', ' , Size) AS Color\_Size,

      ListPrice - StandardCost AS ProductDetails

       FROM SalesLT.Product;

* Run the query. The , operator in the calculated **ProductDetails** column is used to *concatenate* the **Color** and **Size** column values (with a literal comma between them). The behavior of this operator is determined by the data types of the columns - had they been numeric values, the **+** operator would have *added* them.

Note also that some results are *NULL* - we'll explore NULL values later.

**Section 2: Work with data types**

As you just saw, columns in a table are defined as specific data types, which affects the operations you can perform on them.

* Replace the existing query with the following code, and run it:

SELECT CONCAT(ProductID , ': ' , Name) AS ProductName

FROM SalesLT.Product;

* Note that this query may return an error. **CONCAT**can be used to *concatenate* text-based values, or *add* numeric values; but in this case there's one numeric value (**ProductID**) and one text-based value (**Name**), so it's unclear how the CONCAT function should be applied.
* Modify the query and re-run it:

SELECT CONCAT(CAST(ProductID AS char(5)), ': ' , Name) AS ProductName

          FROM SalesLT.Product;

* Note **CAST** function change the numeric **ProductID** column into a **char** (variable-length character data) value so it can be concatenated with other text-based values.
* We can use the function **DATE\_FORMAT** that can be useful for formatting date and time values when converting them to text-based data. Replace the existing query and run it.

      SELECT SellStartDate,

                  DATE\_FORMAT(SellStartDate, '%Y-%m-%d %H:%i:%s') AS ConvertedDate,

                  DATE\_FORMAT(SellStartDate, '%Y-%m-%dT%H:%i:%s') AS ISO8601FormatDate

       FROM SalesLT.Product;

* Create the new query with the following code, and run it.

SELECT Name, CAST(Size AS SIGNED) AS NumericSize

FROM SalesLT.Product;

* CAST(Size AS SIGNED) attempts to cast the Size column to a signed integer in MySQL.
* If Size contains non-numeric data, or if it cannot be directly converted to a signed integer, MySQL will return 0 or NULL depending on its strict SQL mode configuration.
* Modify the query to use a **CAST** function, as shown here.

      SELECT Name,

       CASE WHEN Size REGEXP '^[0-9]+$' THEN CAST(Size AS UNSIGNED)

       ELSE NULL

       END AS NumericSize

      FROM SalesLT.Product;

* Run the query and note that the numeric **Size** values are converted successfully to integers, but that non-numeric sizes are returned as *NULL*.

**Section 3: Handle NULL values**

We've seen some examples of queries that return *NULL* values. *NULL* is generally used to denote a value that is *unknown*. Note that this is not the same as saying the value is *none* - that would imply that you *know* that the value is zero or an empty string!

View SalesLT.Product Table using

SELECT NAME, Size FROM SalesLT.Product;

* Modify the existing query as shown here:

SELECT Name, IFNULL(CAST(Size AS SIGNED), 0) AS NumericSize

FROM SalesLT.Product;

* Run the query and view the results. The **IFNULL** function replaces *NULL* values with the specified value, so in this case, sizes that are not numeric (and therefore can't be converted to integers) are returned as **0**. In this example, the **IFNULL** function is applied to the output of the inner **CAST** function.
* Create a new query with the following code to handle *NULL* values for **Color** and **Size** values in the table:

       SELECT ProductNumber,

         CONCAT( IFNULL(Color, '') , ', ' , IFNULL(Size, '')) AS ProductDetails

       FROM SalesLT.Product;

The **IFNULL** function replaces *NULL* values with a specified literal value. Sometimes, you may want to achieve the opposite result by replacing an explicit value with *NULL*. To do this, you can use the **NULLIF** function.

* Try the following query, which replaces the **Color** value "Multi" to *NULL*.

SELECT Name, NULLIF(Color, 'Multi') AS SingleColor

FROM SalesLT.Product;

**Task: Suppose you want to track the status of a product's availability based on the dates recorded when it was first offered for sale or removed from sale. Find the first non-NULL date for product selling status.**

**COALESCE** function: In some scenarios, you might want to compare multiple columns and find the first one that isn't *NULL*.

* + A product that is currently available has a **SellStartDate**, but the **SellEndDate**value will be *NULL*.
  + A product that is no longer sold, a date is entered in its **SellEndDate column.**
* The previous query returns the last date on which the product selling status was updated, but doesn't actually tell us the sales status itself.

SELECT Name, COALESCE(SellEndDate, SellStartDate) AS StatusLastUpdated

FROM SalesLT.Product;

* To determine that, we'll need to check the dates to see if the **SellEndDate**is *NULL*. To do this, you can use a **CASE** expression in the **SELECT** clause to check for *NULL* **SellEndDate** values.

**CASE** expression has two variants: a *simple* **CASE** that evaluates a specific column or value, or a *searched* **CASE** that evaluates one or more expressions.

In this example, our **CASE** expression must determine if the **SellEndDate** column is *NULL*. Typically, when you are trying to check the value of a column you can use the **=** operator;

For example the predicate **SellEndDate = '01/01/2005'** returns **True** if the **SellEndDate** value is *01/01/2005*, and **False** otherwise.

However, when dealing with *NULL* values, the default behavior may not be what you expect. To check to see if a value is *NULL*, you must use the **IS NULL** predicate; and conversely to check that a value is not *NULL* you can use the **IS NOT NULL** predicate.

* Run the following query, which includes *searched* **CASE** that uses an **IS NULL** expression to check for *NULL* **SellEndDate** values.

      SELECT Name,

       CASE

        WHEN SellEndDate IS NULL THEN 'Currently for sale'

        ELSE 'No longer available'

        END AS SalesStatus

      FROM SalesLT.Product;

The previous query used a *searched* **CASE** expression, and includes one or more **WHEN…THEN** expressions with the values and predicates to be checked.

An **ELSE** expression provides a value to use if none of the **WHEN** conditions are matched, and the **END** keyword denotes the end of the **CASE** expression, which is aliased to a column name for the result using an **AS** expression.

In some queries, it's more appropriate to use a *simple* **CASE** expression that applies multiple **WHERE…THEN** predictes to the same value.

* Run the following query to see an example of a *simple* **CASE** expression that produced different results depending on the **Size** column value.

SELECT Name,

    CASE Size

        WHEN 'S' THEN 'Small'

        WHEN 'M' THEN 'Medium'

        WHEN 'L' THEN 'Large'

        WHEN 'XL' THEN 'Extra-Large'

        ELSE IFNULL(Size, 'n/a')

    END AS ProductSize

     FROM SalesLT.Product;

* Review the query results and note that the **ProductSize** column contains the text-based description of the size for *S*, *M*, *L*, and *XL* sizes; the measurement value for numeric sizes, and *n/a* for any other sizes values.

The End

**CHALLENGE**

Now that you've seen some examples of SELECT statements that retrieve data from a table, it's time to try to compose some queries of your own.

**Tip**: Try to determine the appropriate queries for yourself. If you get stuck, suggested answers are provided at the end of this lab.

**Challenge 1: Retrieve customer data**

Adventure Works Cycles sells directly to retailers, who then sell products to consumers. Each retailer that is an Adventure Works customer has provided a named contact for all communication from Adventure Works. The sales manager at Adventure Works has asked you to generate some reports containing details of the company’s customers to support a direct sales campaign.

1. Retrieve customer details
   * Familiarize yourself with the **SalesLT.Customer** table by writing a SQL query that retrieves all columns for all customers.
2. Retrieve customer name data
   * Create a list of all customer contact names that includes the title, first name, middle name (if any), last name, and suffix (if any) of all customers.
3. Retrieve customer names and phone numbers
   * Each customer has an assigned salesperson. You must write a query to create a call sheet that lists:
     + The salesperson
     + A column named **CustomerName** that displays how the customer contact should be greeted (for example, *Mr Smith*)
     + The customer’s phone number.

**Challenge 2: Retrieve customer order data**

As you continue to work with the Adventure Works customer data, you must create queries for reports that have been requested by the sales team.

1. Retrieve a list of customer companies
   * You have been asked to provide a list of all customer companies in the format *Customer ID* : *Company Name* - for example, *78: Preferred Bikes*.
2. Retrieve a list of sales order revisions
   * The **SalesLT.SalesOrderHeader** table contains records of sales orders. You have been asked to retrieve data for a report that shows:
     + The sales order number and revision number in the format () – for example SO71774 (2).
     + The order date converted to ANSI standard *102* format (*yyyy.mm.dd* – for example *2015.01.31*).

**Challenge 3: Retrieve customer contact details**

Some records in the database include missing or unknown values that are returned as NULL. You must create some queries that handle these NULL values appropriately.

1. Retrieve customer contact names with middle names if known
   * You have been asked to write a query that returns a list of customer names. The list must consist of a single column in the format *first last* (for example *Keith Harris*) if the middle name is unknown, or *first middle last* (for example *Jane M. Gates*) if a middle name is known.
2. Retrieve primary contact details
   * Customers may provide Adventure Works with an email address, a phone number, or both. If an email address is available, then it should be used as the primary contact method; if not, then the phone number should be used. You must write a query that returns a list of customer IDs in one column, and a second column named **PrimaryContact** that contains the email address if known, and otherwise the phone number.

**IMPORTANT**: In the sample data provided, there are no customer records without an email address. Therefore, to verify that your query works as expected, run the following **UPDATE** statement to remove some existing email addresses before creating your query:

UPDATE SalesLT.Customer

SET EmailAddress = NULL

WHERE CustomerID % 7 = 1;

 3. Retrieve shipping status

* + You have been asked to create a query that returns a list of sales order IDs and order dates with a column named **ShippingStatus** that contains the text *Shipped* for orders with a known ship date, and *Awaiting Shipment* for orders with no ship date.

**IMPORTANT**: In the sample data provided, there are no sales order header records without a ship date. Therefore, to verify that your query works as expected, run the following UPDATE statement to remove some existing ship dates before creating your query.

UPDATE SalesLT.SalesOrderHeader

SET ShipDate = NULL

WHERE SalesOrderID > 71899;