DE Lab Week-5
MariaDB

Introduction:

MariaDB is a fork of the MySQL database management system. It is created by its original developers. This DBMS tool offers data processing capabilities for both small and enterprise tasks.
MariaDB is an improved version of MySQL. It comes with numerous inbuilt powerful features and many usabilities, security and performance improvements that you cannot find in MySQL.

Database
A database is a collection of data stored in some organized fashion.

DBMS :
Database Management System (DBMS) is a software for storing and retrieving users' data while considering appropriate security measures. It consists of a group of programs which manipulate the database.
DBMS does all the work of storing, retrieving, managing, and manipulating data. MariaDB is also a DBMS.

Table A structured list of data of a specific type.

Column A single field in a table. All tables are made up of one or more columns

Datatype A type of allowed data. Every table column has an associated data type that restricts (or allows) specific data in that column. Datatypes restrict the type of data that can be stored in a column.

Row A record in a table.

Primary key A column (or set of columns) whose values uniquely identify every row in a table.

SQL (Structured Query Language) SQL is a language designed specifically for communicating with databases.

Working with MariaDB

Issuing MariaDB SQL statements, and obtaining information about databases and Tables.

Selecting a Database

Syntax : USE crashcourse;

Learning About Databases and Tables

Information about databases, tables, columns.

SHOW DATABASES; Returns a list of available databases

SHOW TABLES; Returns a list of available tables in the currently selected database.

DESCRIBE customers; It returns a row for each field containing the field name, its datatype, whether NULL is allowed, key information, default value, and extra information (such as auto_increment for field cust_id).

Retrieving Data :

Using the SELECT statement to retrieve one or more columns of data from a table.

The SELECT Statement :

Retrieving Individual Columns:

```
SELECT prod_name
FROM products;
```

Statement uses the SELECT statement to retrieve a single column called prod_name from the products table.

Note : It is important to note that SQL statements are not case sensitive, so SELECT is the same as select, which is the same as Select.

Retrieving Multiple Columns:

To retrieve multiple columns from a table, the same SELECT statement is used. The only difference is that multiple column names must be specified after the SELECT keyword, and each column must be separated by a comma.

```
SELECT prod_id, prod_name, prod_price
```

FROM products;

The following SELECT statement retrieves three columns from the products table

Retrieving All Columns :

*SELECT **
FROM products;

When a wildcard (*) is specified, all the columns in the table are returned. The columns are in the order in which the columns appear in the table definition.

Retrieving Distinct Rows :
DISTINCT keyword:

SELECT DISTINCT vend_id
FROM products;

It return only distinct (unique) vend_id rows,

Note : If u apply distinct on multiple columns all rows would be retrieved unless both of the specified columns were distinct.

Limiting Results :
SELECT statements return all matched rows, possibly every row in the specified table. To return just the first row or rows, use the LIMIT clause

SELECT prod_name
FROM products
LIMIT 5;

The previous statement uses the SELECT statement to retrieve a single column. LIMIT 5 instructs MariaDB to return no more than five rows.

LIMIT 3,4 means 3 rows starting from row 4.

Using Fully Qualified Table Names:
It is also possible to refer to columns using fully qualified names (using both the table and column names)

SELECT products.prod_name
FROM products;

Sorting Retrieved Data

Using the SELECT statement's ORDER BY clause to sort retrieved data as needed.

Order By :
To explicitly sort data retrieved using a SELECT statement, the ORDER BY clause is used. ORDER BY takes the name of one or more columns by which to sort the output

```
SELECT prod_name
FROM products
ORDER BY prod_name;
```

Sorting by Multiple Columns
It is often necessary to sort data by more than one column. For example, if you are displaying an employee list, you might want to display it sorted by last name and first name (first sort by last name, and then within each last name sort by first name). This would be useful if there are multiple employees with the same last name.

```
SELECT prod_id, prod_price, prod_name
FROM products
ORDER BY prod_price, prod_name;
```

Specifying Sort Direction:
DESC : Sorting by descending order.

```
SELECT prod_id, prod_price, prod_name
FROM products
ORDER BY prod_price DESC;
```

Filtering Data

Retrieving just the data you want involves specifying search criteria, also known as a filter Condition.

Within a SELECT statement, data is filtered by specifying search criteria in the WHERE clause. The WHERE clause is specified right after the table name.

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price = 2.50;
```

This statement retrieves two columns from the products table, but instead of returning all rows, only rows with a prod_price value of 2.50 are returned.

Note : When using both ORDER BY and WHERE clauses, make sure ORDER BY comes after the WHERE; otherwise, an error will be generated.

The WHERE Clause Operators :

Operator	Description
=	Equality
<>	Nonequality
!=	Nonequality
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
BETWEEN	Between two specified values

Checking for a Range of Values
To check for a range of values, you can use the BETWEEN operator.

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price BETWEEN 5 AND 10;
```

Combining WHERE Clauses Using the AND Operator :
To filter by more than one column, you use the AND operator to append conditions to your WHERE clause.

```
SELECT prod_id, prod_price, prod_name
FROM products
WHERE vend_id = 1003 AND prod_price <= 10;
```

Using the IN Operator

The IN operator is used to specify a range of conditions, any of which can be matched. IN takes a comma-delimited list of valid values, all enclosed within parentheses.

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id IN (1002,1003)
ORDER BY prod_name;
```

The SELECT statement retrieves all products made by vendor 1002 and vendor 1003.

Using Wildcard Filtering

Wildcards are Special characters used to match parts of a value. Wildcard searches can be performed using the LIKE operator for sophisticated filtering of retrieved data.

Using the LIKE Operator :
Using wildcards, you can create search patterns that can be compared against your data.

The Percent Sign (%) Wildcard

The most frequently used wildcard is the percent sign (%). Within a search string, % means match any number of occurrences of any character.
For example, to find all products that start with the word jet, you can issue the following SELECT statement

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE 'jet%
```

This example uses a search pattern of 'jet%'. When this clause is evaluated, any value that starts with jet is retrieved. The % tells MariaDB to accept any characters after the word jet, regardless of how many characters there are.

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '%anvil%';
```

The search pattern '%anvil%' means match any value that contains the text anvil anywhere within it, regardless of any characters before or after that text.

The Underscore (_) Wildcard

Another useful wildcard is the underscore (_). The underscore is used just like %, but instead of matching multiple characters, the underscore matches just a single character.

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '_ ton anvil';
```

Summarizing Data

We will learn what the SQL aggregate functions are and how to use
them to summarize table data.
It is often necessary to summarize data without actually retrieving it all, and MariaDB provides special functions for this purpose.

- Determining the number of rows in a table (or the number of rows that meet some condition or contain a specific value)
- Obtaining the sum of a group of rows in a table
- Finding the highest, lowest, and average values in a table column(either for all rows or for specific rows)

Aggregate functions :
Functions that operate on a set of rows to calculate and return a single value.

SQL Aggregate Functions:

- AVG() Returns a column's average value
- COUNT() Returns the number of rows in a column
- MAX() Returns a column's highest value
- MIN() Returns a column's lowest value
- SUM() Returns the sum of a column's values

The AVG() Function
AVG() is used to return the average value of a specific column by counting both
the number of rows in the table and the sum of their values.

```
SELECT AVG(prod_price) AS avg_price
FROM products;
```

the average price of all products in the products table. avg_price is an alias.

The COUNT() Function :
Using COUNT(), you can determine the number of rows in a table or the number of rows that match a specific criterion.
COUNT() can be used two ways:

- Use COUNT(*) to count the number of rows in a table, whether columns contain values or NULL values.
- Use COUNT(column) to count the number of rows that have values in a specific column, ignoring NULL values.

```
SELECT COUNT(*) AS num_cust
FROM customers;
```

COUNT(*) is used to count all rows, regardless of values.

Note
NULL Values Column rows with NULL values in them are ignored by the COUNT() function if a column name is specified, but not if the asterisk (*) is used.

Grouping Data

We will learn how to group data so you can summarize subsets of table contents. This involves two new SELECT statement clauses: the GROUP BY clause and the HAVING clause.

Grouping enables you to divide data into logical sets so you can perform aggregate calculations on each group .

Creating Groups
Groups are created using the GROUP BY clause in your SELECT statement

```
SELECT vend_id, COUNT(*) AS num_prods
FROM products
GROUP BY vend_id;
```

Filtering Groups
In addition to being able to group data using GROUP BY, MariaDB also allows you to filter which groups to include and which to exclude. Using the HAVING clause.

HAVING is similar to WHERE. In fact, all types of WHERE clauses you learned about thus far can also be used with HAVING. The only difference is that WHERE filters rows and HAVING filters groups.

```
SELECT cust_id, COUNT(*) AS orders
FROM orders
GROUP BY cust_id
HAVING COUNT(*) >= 2;
```

A list of all customers who have made at least two orders.

SELECT Clause Ordering

Table 13.2 `SELECT` Clauses and Their Sequence

Clause	Description	Required
`SELECT`	Columns or expressions to be returned	Yes
`FROM`	Table to retrieve data from	Only if selecting data from a table
`WHERE`	Row-level filtering	No
`GROUP BY`	Group specification	Only if calculating aggregates by group
`HAVING`	Group-level filtering	No
`ORDER BY`	Output sort order	No
`LIMIT`	Number of rows to retrieve	No

Working with Subqueries **(Nested Query)**

SQL also enables you to create subqueries. Queries that are embedded into other queries.

```
SELECT cust_id
FROM orders
WHERE order_num IN (SELECT order_num
FROM orderitems
WHERE prod_id = 'TNT2');
```

Subqueries are always processed starting with the innermost SELECT statement and working outward.

Inserting Data

We will learn how to insert data into tables using the SQL INSERT statement.

INSERT is used to insert (add) rows to a database table.

Inserting Complete Rows

Basic INSERT syntax, which requires that you specify the table name and the values to be inserted into the new row.

```
INSERT INTO Customers
VALUES(NULL,
'Pep E. LaPew','100 Main Street', 'Los Angeles','CA', '90046', 'USA',NULL,NULL);
```

The safer (and unfortunately more cumbersome) way to write the INSERT statement is as follows:
Inserting Complete Rows

```
INSERT INTO customers(cust_name,cust_address,
cust_city,cust_state,cust_zip,cust_country,cust_contact,cust_email)
VALUES('Pep E. LaPew','100 Main Street','Los
Angeles','CA','90046','USA',NULL,NULL);
```

Inserting Retrieved Data (INSERT SELECT)

INSERT can be used to insert the result of a SELECT statement into a table.

```
INSERT INTO
customers(cust_id,cust_contact,cust_email,cust_name,cust_address,cust_city,cu
st_state,cust_zip,cust_country)
SELECT
cust_id,cust_contact,cust_email,cust_name,cust_address,cust_city,cust_state,cu
st_zip,cust_country
FROM custnew;
```

Instead of listing the VALUES to be inserted, the SELECT statement retrieves them from custnew. Each column in the SELECT corresponds to a column in the specified columns list.

Updating and Deleting Data

We will learn how to use the UPDATE and DELETE statements to enable you to further manipulate your table data.

Updating Data
To update (modify) data in a table the UPDATE statement is used. UPDATE can be used in two ways:

- To update specific rows in a table
- To update all rows in a table

```
UPDATE customers
SET cust_email = 'elmer@fudd.com'
WHERE cust_id = 10005;
```

Note :
The UPDATE statement finishes with a WHERE clause that tells MariaDB which row to update. Without a WHERE clause, MariaDB would update all the rows in the customers table with this new e-mail address.

Deleting Data
To delete (remove) data from a table, the DELETE statement is used. DELETE can be used in two ways:

- To delete specific rows from a table
- To delete all rows from a table

Deletes a single row from the customers table

```
DELETE FROM customers
WHERE cust_id = 10006;
```

Note : If the WHERE clause were omitted, this statement would delete every customer in the table.

Note : Never execute an UPDATE or a DELETE without a WHERE clause unless you really do intend to update and delete every row.

Creating and Manipulating Tables

Basic Table Creation
To create a table using CREATE TABLE, you must specify the following information:

- The name of the new table specified after the keywords CREATE TABLE.
- The name and definition of the table columns separated by commas.

Creating the customers table.

```
CREATE TABLE customers
(
cust_id int NOT NULL AUTO_INCREMENT,
cust_name char(50) NOT NULL ,
cust_address char(50) NULL ,
cust_city char(50) NULL ,
cust_state char(5) NULL ,
cust_zip char(10) NULL ,
cust_country char(50) NULL ,
cust_contact char(50) NULL ,
cust_email char(255) NULL ,
PRIMARY KEY (cust_id)
);
```

Primary Key

Primary key values must be unique. That is, every row in a table must have a unique primary key value. If a single column is used for the primary key, it must be unique; if multiple columns are used, the combination of them must be unique.

Updating Tables

To update table definitions, the ALTER TABLE statement is used.
Adding a column to a table:

```
ALTER TABLE vendors
ADD vend_phone CHAR(20);
```

This statement adds a column named vend_phone to the vendors table. The datatype must be specified.

Deleting Tables

Deleting tables (actually removing the entire table, not just the contents) is easy—arguably too easy. Tables are deleted using the DROP TABLE statement:

```
DROP TABLE customers2;
```

There is no confirmation, nor is there an undo—executing the statement permanently removes the table.

Renaming Tables

To rename a table, use the RENAME TABLE statement as follows:

```
RENAME TABLE customers2 TO customers;
```

REFERENCE :

- MariaDB Crash Course by Ben Forta (Book)