ORMEDIAN RESEARCH INSTITUTE

TOPIC:

DATABASE

SUBTOPIC: SQL FUNDAMENTALS (PART 2)

By:

ADEKOLA OLUWASEUN O.

DECEMBER 22, 2022

CONTENT COVERED

- *REFERENCED TABLES*
- *DATA ORDERING IN SQL*
- *RECORDS INSERTION, UPDATION AND DELETION*
- *USE OF MATH MODULES (MIN, MAX, SUM, and AVG) IN SQL*
- *UNDERSTANDING WHAT NULL MEANS IN SQL*
- *UNDERSTANDING HOW TO USE COUNT STATEMENT*
- *THE USE OF LIKE OPERATOR AND WHERE STATEMENT WITH WILDCARDS*
- *USE OF ALIASES FOR COLUMNS AND TABLES IN SQL.*

TABLE 1:

This is a table stored in a database, and it contains some records of students from three different department in the faculty of Engineering.
The name given to this table in my database is Students. This table name and its cells will referenced throughout every implementation part of this module.

S/N	FirstName	LastName	MatricNo	Department	Level	EntryMode	Age	Sex
1	*Oluwaseun*	*Adekola*	*160211014*	*ECE*	*500*	*UTME*	*26*	*Male*
2	*Abosede*	*Adedoyin*	*180231032*	*CPE*	*300*	*UTME*	*23*	*Female*
3	*Samuel*	*Adebayo*	*180211044*	*ECE*	*300*	*UTME*	*24*	*Male*
4	*Damilola*	*Osekita*	*170221022*	*MECH*	*400*	*DIRECT ENTRY*	*25*	*Female*
7	*Ayoola*	*Abiodun*	*170221032*	*MECH*	*400*	*DIRECT ENTRY*	*27*	*Male*
8	*Michael*	*Ogedengbe*	*160231056*	*CPE*	*500*	*UTME*	*29*	*Male*
9	*Taiwo*	*Adeoti*	*180211054*	*ECE*	*300*	*DIRECT ENTRY*	*26*	*Male*

TABLE 2: This is named in our database as Specstudents

***NOTE:* This table will also be needed later during this course.**

FirstName	LastName	Level
Peter	Bankole	400
Damilare	Onadipe	400
Tolu	Onatunde	500

ASCENDING AND DESCENDING ORDER OF DATA IN SQL

Order of data in the database could be very paramount is many cases. When putting into consideration what organization of data really is, how well data can be ordered for easy readability and interpretation matters too.

Let's take the Table 1 that is in slide number 2 of this presentation as our reference table. We may decide to organize data (values) entered into one or more of the columns in ascending or descending order. This can be easily achieved by using simple SQL queries to make these changes. ORDER BY clause is what does the ordering while we also use SQL keywords like ASC and DESC.

We can rearrange single or multiple columns at the same time.

Let's say we need to sort the "Age" field of Table 1 in descending order.

```
SELECT *FROM Students ORDER BY Age DESC;
```

INSERTION, UPDATION AND DELETION OF DATA IN SQL

INSERTION OF DATA: *The* ***INSERT*** *statement is used to enter new records into the table created in the SQL.*
Let us assume the Table 1 displayed previously in this presentation had no records at all.

INSERTION OF DATA IS OF TWO CASES:

CASE 1: *When there is need to insert values into all columns available in the table.*

CODE EXAMPLE

```
INSERT INTO Students VALUES (“Oluwaseun”, “Adekola”, 160211014, “ECE”, 500, “UTME”, 26, “Male”);
```

CASE 2: *When just few or some of the columns are to be filled with data but not all.*

CODE EXAMPLE
```
INSERT INTO Students (FirstName, LastName, MatricNo, Age) VALUES (“Oluwaseun”, “Adekola”, 160211014, 26);
```

NOTE: *Case 2 can also be used to achieve the objective of case 1, but it will be a waste of time to list all columns before inserting values into each of them because the table has a column of a given structure already.*

CONT’

***DELETION OF DATA**: **DELETE** statement is used to delete existing records from a table. It is the functionality of SQL that can be used to delete one or records from a table provided that an extra criteria such as WHERE clause is specified to make that happen. If WHERE clause is not specified, there is possibility of losing all records.*

CODE EXAMPLE:

```
DELETE FROM Students WHERE MatricNo = 160211014;
```

***UPDATION OF DATA**: **UPDATE** statement is used for changing the changing table records. It has the capability of changing the values stored in columns but cannot change the column name. Always specify WHERE Condition for the row to effect changes on. If WHERE clause is not specified, there is possibility that all values in that column will be assigned the same new value altogether.*

CODE EXAMPLE:

```
UPDATE Students SET Department=“MECH” WHERE MatricNo =160211014;
```

MIN, MAX, SUM and AVERAGE

MIN: *this is a statement that is used for obtaining the lowest value from a given field.*

For example*, we may wish to output the lowest age from the given reference table in this presentation.*

Let's say we need to obtain the minimum value that exist in the Age column.

CODE EXAMPLE:

```
SELECT MIN(Age) FROM Students;
```

MAX: *this is a statement that is used for obtaining the highest age from a given field.*

For example*, we may wish to output the highest age from the given reference table in this presentation.*

CODE EXAMPLE:

```
SELECT MAX(Age) FROM Students;
```

CONTINUATION

MIN, MAX, SUM and AVG

SUM: *This is a functionality that adds all values in a particular column together. This works on columns with numerical data.*

Example: *Obtain the total sum of values in the Age column of Table 1.*

CODE EXAMPLE:

```
SELECT SUM(Age) FROM Students;
```

AVG: *This functionality enables us to get the average of all values in a particular column.*

Example: *Lets find out what the average of the ages is.*

CODE EXAMPLE:

```
SELECT AVG(Age) FROM Students;
```

UNDERSTANDING WHAT NULL IS ALL ABOUT IN SQL

NULL *in SQL is just a placeholder that specifies that a particular field has no value in the database table. It is simply used to represent a missing value. If a given table has field which is optional as null but such field can later be updated.*

For example, *say we have this table sample; we have kept the VALUE field to be NULL here, hence its value is missing.*

PDNAME	QUANTITY	VALUE
VIJU	2	

Any column that has a null value can be tested for by using IS NULL keyword Clause to check.

COUNT STATEMENT

COUNT: *This is used to return the number of records in a table. It could also be used to return the total number of rows of a database table.*

CASE 1

Let us assume that we want to output the number of rows entries in the Student Table which is our reference table. We can implement this count statement on such table entries to obtain the desired result. Say we need to count number of item in LastName column.

CODE EXAMPLE

```
SELECT COUNT (LastName) FROM Students;
```

CASE 2

It could also be used to obtain the number of rows within a certain range of a particular column. Say we need to know only the number of students in 500 Level from the table.

CODE EXAMPLE

```
SELECT COUNT(Level) FROM Students WHERE Level=500;
```

UNDERSTANDING THE USE OF LIKE OPERATORS WITH WILDCARDS, and CHARLIST

LIKE: *This is used to search value in one or more columns by specifying some special character as the condition for searching.*

WILDCARDS: *Wildcards characters are used in place of one or more characters. Wildcards rely solely on LIKE operator and WHERE clause to function well. It uses percentage (%) to represent one or multiple character, underscore (_)to represent one or single character, closed square bracket [] to represent any character within the bracket.*

CHARLIST: *It is a form of wildcard that is used to specify a list of character or range of character out of which any of them must be found to be starting character of some column values in the table. If the false statement of this charlist is expected to be the output, then such set of charlist should be preceded by exclamation mark. i.e. [!charlist]*

NOTE: *Some queries having wildcards within them could only be responsive on ACCESS Or Oracle etc. Hence, not all of these wildcard and charlist pattern related queries work on MySQL.*

EXAMPLE 1

Say we need to select entries of a table containing only students whose surname starts with "AD"

CODE EXAMPLE

```
SELECT *FROM Students WHERE LastName LIKE "AD%";
```

More Code Examples Continuation…….

EXAMPLE 2

Assume we need to use wildcards to generate result set of students whose EntryMode is Direct Entry in the Student table.

CODE EXAMPLE

SELECT *FROM Students WHERE EntryMode LIKE “dir%”;

EXAMPLE 3

Let’s say we aim to output records of every student whose LastName has “ad” in any position.

CODE EXAMPLE

SELECT *FROM Students WHERE Lastname LIKE %ad%;

UNDERSTANDING WHAT SQL ALIASES ARE

Alias is a feature of SQL systems that enables it to give a temporary or alternative names to database objects.

Aliases are created with the use of ***AS*** *keyword. It's a way of making column names more readable at the result set level.*

CASE 1

For example, we may wish to temporarily change the column named MatricNO to Student_idNO in the result set.

CODE EXAMPLE

```
SELECT MatricNo AS Student_idNO FROM Students;
```

CASE 2:

Selection of Multiple columns and assigning different Aliases to them, say we need to give a temporary name

```
SELECT MatricNo AS Student_idNO, Department AS Discipline FROM Students;
```

CONT’

Understanding Aliases……………………

CASE 3:

There are several cases where may want to make the new alias have some space separated words e.g. course of study. This is a very special case that requires that such alias must be enclosed in a square bracket i.e. [alias], else it will not give the needed result. So we have space separated words as the new temporary name, so we must ensure the use of square bracket.

NOTE: In databases, [ ] can be replaced by “ ” if the former does not work. In this presentation double quote was used because it makes this work, while [] could not give result needed.

CODE EXAMPLE:

```
SELECT MatricNo AS Student_idNO, Department AS “Course of Study” FROM Students;
```

COMPLEX PART OF SQL ALIASES

Alias also has a very nice feature which we can use to combine more than one column together and assign just one alias to represent all the columns.

***For example**, in the student table we may decide to give the whole columns a single alias e.g Student Information. This approach has different way of formatting the code on different databases*

CODE EXAMPLE 1: for SQL Server

```
 SELECT FirstName, LastName+’, ‘MatricNo+’, ‘ Level+’, ‘ EntryMode+’, ‘ Age+’, Level AS “Student information” From
Students;
```

CODE EXAMPLE 2: This achieves similar result on MySQL.

```
SELECT CONCAT(FirstName,',  ', LastName,', ',MatricNo,', ',Level,',  ',EntryMode,',  ',Age,', ',Level) AS  “Student information”
FROM Students;
```

USING ALIASES FOR MULTIPLE TABLES

Different tables that reside in a single database can be given aliases to enable referencing any field of these tables easy. Such columns of these tables can also be referenced at the same to generate result set.

In this presentation, lets take the Students table and Specstudents table as the tables to be used for example in this case.

Let us assume that we want to generate result set which will be the combination of some of the field of Table 1 and Table 2. consider generating result set that will contain FirstName from Table1 , LastName From Table 2 where the Level in Table 1 is same as Table 2.

Solution approach is to first set shortened aliases for both tables . Let Students be st, and Specstudent be ss

CODE EXAMPLE

```
SELECT st.FirstName, ss.LastName FROM Students AS st, Specstudents AS ss WHERE st.Level = ss.Level;
```

THANKS FOR VIEWING

NOTE: ***Subsequent topics under SQL will be discussed in Part 3***
Also, all Implementations as far as this presentation is concerned are in line with MySQL Syntax. All have been tested on XAMPP Server.

PART 3 PRESENETATION WILL COVER EVERYTHIN ABOUT "JOIN " IN SQL