ORMEDIAN RESEARCH INSTITUTE

TOPIC:

DATABASE

SUBTOPIC: SQL FUNDAMENTALS (PART 3)

By:

ADEKOLA OLUWASEUN O.

DECEMBER 26, 2022

CONTENT COVERED

- *TABLE USED FOR REFERENCES*
- *UNDESTANDING WHAT SQL JOIN IS ALL ABOUT*
- *COMMON TYPES OF SQL JOINS*
- *IMPLEMENTATION OF DIFFERENT SQL JOINS*
- *UNDERSTANDING WHAT UNION AND UNION ALL ARE, AND THEIR IMPLEMENTATION IN SQL*
- *IDENTICAL DATA GROUPING IN SQL (Using GROUP BY) WITH IMPLEMENTATION*
- *APPLICATION OF GROUP BY statement WITH HAVING CLAUSE IN SQL*
- *HOW EXISTS CONDITION IS USED IN SQL.*

IN THIS PRESNTATION, THE FOLLOWING TABLES WILL BE USED AS OUR REFERENCES AT EVERY LEVEL OF WRITING QUERIES

TABLENAME: ECE501

STUDENTID	SCORE	DEPARTMENT	LEVEL
160211012	65	ECE	500
160221014	89	MECH	500
160231044	89	CPE	500
160211059	89	ECE	500
160221088	53	MECH	500
160211014	88	ECE	500

TABLENAME: ECE507

STUDENTID	SCORE	CURRENT_CGPA	LEVEL	DEPARTMENT
160221015	89	4.67	500	ECE
160211012	89	4.67	500	ECE
160211014	83	4.6	500	MECH
160231044	81	4.6	500	CPE
160211059	89	4.63	500	ECE
160211088	89	4.6	500	ECE

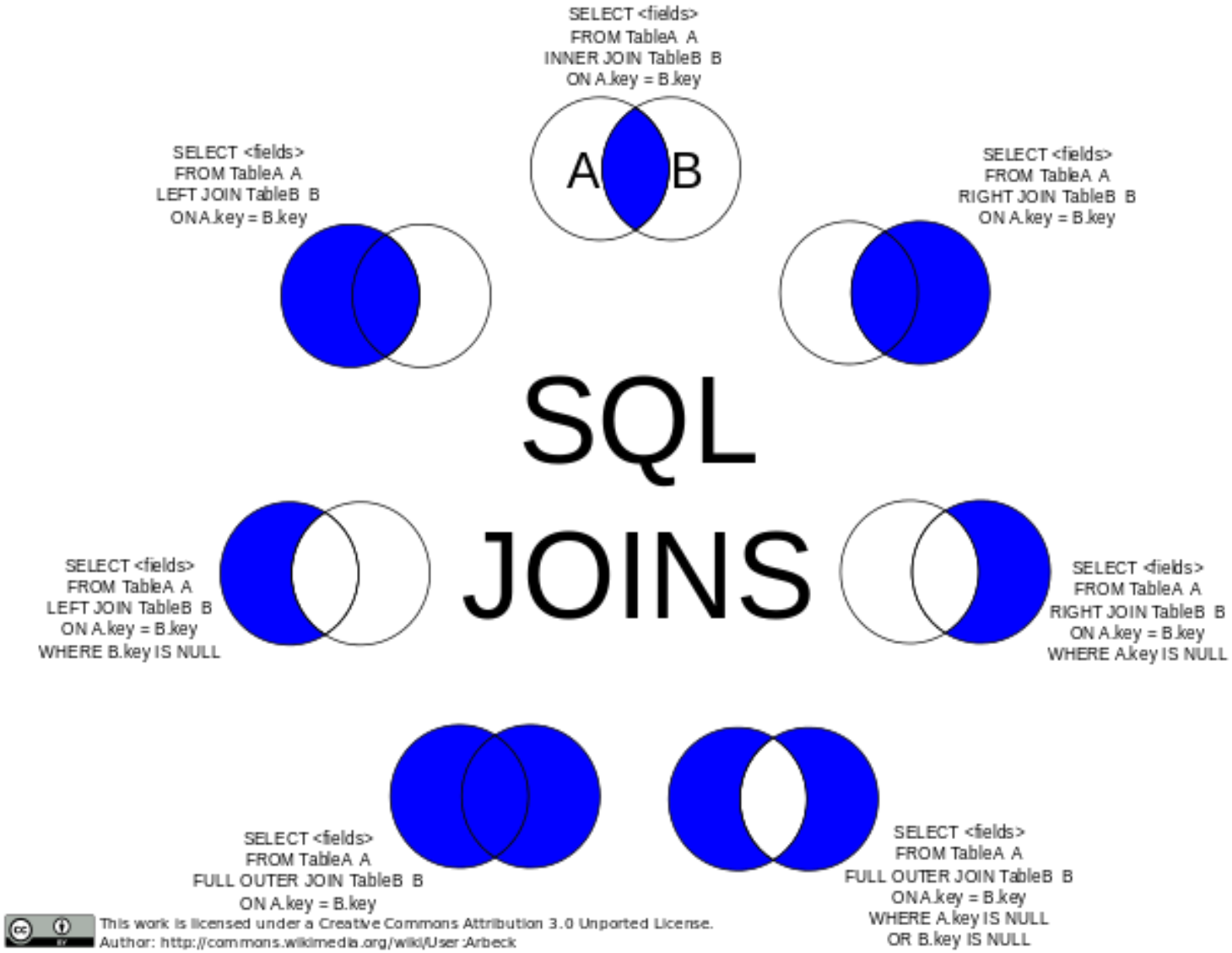
UNDERSTANDING SQL JOIN

SQL JOIN: *Clause is used to combine rows from two different tables together when they have some field (Column) names in common.*

There are different types of join Statements

- ***INNER JOIN****: This captures a result-set which is a combination of all the matching records between the two tables where a specified condition is satisfied.*
- ***LEFT JOIN****: this is used to join all rows of table on the left side of the join and matching rows for table on the right side of the join. Null is outputted where the right side table has no matching records.*
- ***RIGHT JOIN****: this is used to join all rows of table on the right side of the join and matching rows for table on the left side of the join. Null is outputted where the left side table has no matching records.*
- ***SELF JOIN****: This is a query that is used to join a table to itself. By doing so, the table appears as the combination of the same copy of table. It is also regarded as regular join. For self join to be implemented, one of the two tables requires to be given an alias.*
- ***FULL JOIN****: this combines the result of both LEFT JOIN and RIGHT JOIN together from the rows of both tables involved. NULL is returned for the rows with no matching results.*

PICTORIAL ILLUSTRATION OF JOIN IN SQL



This work is licensed under a Creative Commons Attribution 3.0 Unported License.
Author: http://commons.wikimedia.org/wiki/User:Arbeck

IMPLEMENTATION OF INNER JOIN

EXAMPLE:

Let's do the INNER JOIN of the ECE501 and ECE507 Tables to capture the scores of both results using the STUDENTID as the common field.

CODE EXAMPLE

```
SELECT ECE501.SCORE, ECE507.SCORE FROM ECE501 JOIN ECE507 ON ECE507.STUDENTID = ECE501.STUDENT;
```

Suppose we have another table, we can continue the trend that way to join the three tables together as long as conditions have been satisfied.

IMPLEMENTATION OF LEFT JOIN

SCENARIO

Let's say we need to use LEFT JOIN to ensure that ECE501 STUDENTID and their SCORES as well only the ECE507 SCORES are to be returned in the result-set where both tables have common STUDENTID.

CODE EXAMPLE:

```
SELECT ECE501.STUDENTID, ECE501.SCORE, ECE507.SCORE FROM ECE501 LEFT JOIN ECE507 ON ECE501.STUDENTID=ECE507.STUDENTID;
```

We can also use ORDER BY keyword to rearrange any column in order of one's desire. Assuming we need it to be sorted in ascending order of the ECE501 SCORES. The code can further be modified to:

```
SELECT ECE501.STUDENTID, ECE501.SCORE, ECE507.SCORE FROM ECE501 LEFT JOIN ECE507 ON
ECE501.STUDENTID=ECE507.STUDENTID OREDER BY ECE501.SCORE ASC;
```

IMPLEMENTATION OF RIGHT JOIN

SCENARIO

Let's say we need to use RIGHT JOIN to ensure that ECE507 STUDENTID and their SCORES as well only the ECE501 SCORES are to be returned in the result-set where both tables have common STUDENTID.

CODE EXAMPLE:

```
SELECT ECE501.STUDENTID, ECE501.SCORE, ECE507.SCORE FROM ECE501 LEFT JOIN ECE507 ON
ECE501.STUDENTID=ECE507.STUDENTID;
```

We can also use ORDER BY keyword to rearrange any column in order of one's desire. Assuming we need it to be sorted in descending or ascending order of the ECE501 SCORES. The code can further be modified to:

```
SELECT ECE507.STUDENTID, ECE507.SCORE, ECE501.SCORE FROM ECE501 RIGHT JOIN ECE507 ON
ECE501.STUDENTID=ECE507.STUDENTID ORDER BY ECE507.SCORE DESC;
```

FULL JOIN IN SQL

SCENARIO 1

Let's say we need to join the table of the ECE501 to ECE507 using the FULL JOIN strategy;

CODE EXAMPLE: *This will return all records of combined table regardless of matching or no matching;*

SELECT *FROM ECE501 FULL JOIN ECE507;

SCENARIO 2

Let's say we need to join the records of ECE501 table and ECE507 table together such that the condition on which records are returned is strictly based on the STUDENTID;

SELECT *FROM ECE501 FULL OUTER JOIN ECE507 ON ECE501.STUDENTID=ECE507.STUDENTID;

Note: *this second scenario works well on SQL server, but might not work on MySQL.*

IMPLEMENTATION OF SELF JOIN IN SQL

Let's assume we need to make use of only the table named ECE507 to implement the SELF JOIN. In this case we want to join the table ECE507 to itself.

Assume that we need to join the following columns: SCORES, DEPARTMENT of ECE507 table to SCORES, DEPARTMENT, LEVEL columns of the same table.

CODE EXAMPLE

```
SELECT A.SCORE, A.DEPARTMENT, B.SCORE, B.DEPARTMENT, B.LEVEL FROM ECE507 A, ECE507 B WHERE A.STUDENTID=B.STUDENTID;
```

UNDESTANDING THE CONCEPT OF UNION OPERATOR IN SQL

The use of UNION Operator in SQL is not only limited to just one call request. In fact UNION Operator can further be expanded to perform a more unique functionality. Categories of UNION Operator in SQL are:

(a) ***UNION:*** *it is used for combining the result-set of two different select statements together under the condition that these statements have the same number of columns, such columns have the same data types, and these select statement columns must also have the same order. UNION Operator returns distinct values by default.*

(a) ***UNION ALL****: this is used to allow duplicate values to be returned in the result-set unlike in the case of the UNION operator which restrict the duplication of values in the result-set.*

IMPLEMENTATION OF UNION IN SQL

SCENARIO

Let's say we need to output the UNION between the LEVEL field from both ECE501 and ECE507 without values duplication.

CODE EXAMPLE:

```
SELECT LEVEL FROM ECE501 UNION SELECT LEVEL FROM ECE507;
```

SCENARIO

Assuming there is need for us to obtain the distinct values of departments from both ECE501 and ECE507 tables with the condition that the result must be in order of the department.

CODE EXAMPLE:

```
SELECT DEPARTMENT FROM ECE501 UNION SELECT DEPARTMENT FROM ECE507 ORDER BY DEPARTMENT;
```

IMPLEMENTATION OF UNION OPERATOR

Continuation…….

There are cases where UNION operator can still be used with WHERE clause to generate a result-set that will require distinct values from tables involved.

SCENARIO:

Let's say we need to generate the union result-set containing only the scores and departments from ECE501 and ECE507 for only where student's department is ECE.

CODE EXAMPLE:

```
SELECT SCORE, DEPARTMENT FROM ECE501 WHERE DEPARTMENT ='ECE' UNION SELECT SCORE, DEPARTMENT FROM ECE507 WHERE DEPARTMENT ='ECE';
```

IMPLEMENTATION OF UNION ALL IN SQL

SCENARIO

If our motive is to output result-set that has duplication of values of SCORE, DEPARTMENT and STUDENTID where department is strictly ECE. We can achieve this by the use of UNION ALL.

CODE EXAMPLE

```
SELECT SCORE, DEPARTMENT, STUDENTID FROM ECE501 WHERE DEPARTMENT='ECE' UNION ALL SELECT SCORE, DEPARTMENT, STUDENTID FROM ECE507 WHERE DEPARTMENT='ECE';
```

Other functionalities can be used to further modify both union and union all to generate more beautiful results but this depends on the need of the user.

GROUP BY Statement IN SQL

GROUP BY: *This can be regarded as the functionality in SQL that will enable rows having the same column values to be summarized. Or we can simply say that it is use to arrange identical data into groups.*

It is also worth noting that GROUP BY Statement are used with aggregate function in some cases.

Typical aggregate function includes the following: ***AVG(), COUNT(), MAX(), MIN(), SUM().***

For example, in the ECE 501 table, we can see that we have more that one student from both ECE and MECH departments. Let's assume we want to know the number of those in ECE, those in CPE and those in MECH. The GROUP BY statement is one beautiful statement in SQL that we can use to achieve that.

IMPLEMENTATION OF GROUP BY statement

CODE EXAMPLE:

```
SELECT COUNT(STUDENTID), DEPARTMENT FROM ECE501 GROUP BY DEPARTMENT;
```

The code written above groups student based on their departments. Although, this might not really be in ascending order or descending order. If we wish to have a more ordered groups of data either well arranged in ascending or descending order.

Ascending Order of groups:

```
SELECT COUNT(STUDENTID), DEPARTMENT FROM ECE501 GROUP BY DEPARTMENT ORDER BY COUNT(STUDENTID) ASC;
```

Descending order of groups:

```
SELECT COUNT(STUDENTID), DEPARTMENT FROM ECE501 GROUP BY DEPARTMENT ORDER BY COUNT(STUDENTID) DESC;
```

Using GROUP BY statement with HAVING clause

*The **HAVING** Clause enables us to specify conditions that filter which group results appear in the results.*

***GROUP BY** statement becomes important to use with **HAVING** clause because WHERE clause cannot fit into some queries that require the use of aggregate function. HAVING clause is used instead.*

A good use case is when we want to group students based on their department but we only need the result-set of a department with specified number of student as the given condition for which result-set must be returned.

We may only need the departments that have at least 2 students. In the table named ECE501, we found out that only 3 ECE, 1 CPE and 2 MECH students are there. To write some queries that will output groups with 2 or more students, we will need to use the HAVING clause.

IMPLEMENTATION OF '*HAVING*' CLAUSE IN SQL

SCENARIO

Let's Assume that we need the result-set that will contain only the group of students of departments where student count is two or more. Use the table named ECE501 as reference.

CODE EXAMPLE

```
SELECT COUNT(STUDENTID), DEPARTMENT FROM ECE501 GROUP BY DEPARTMENT HAVING COUNT(STUDENTID)>=2;
```

USE OF EXISTS IN SQL

EXISTS: *this is an operator or condition in SQL that is used to check for the existence of records in a given subquery. The result of exists is usually a Boolean i.e. TRUE/FALSE.*

It can be used in a SELECT, UPDATE, INSERT or DELETE statement.

IMPLEMENTATION OF EXISTS CONDITION IN SQL

SCENARIO 1

Let's say we need to return the list of students from ECE501 only based on where these same set of students have the same score in ECE507. So we need to check if those who made a particular score in ECE501 exist in ECE507.

CODE EXAMPLE

```
SELECT STUDENTID, SCORE, DEPARTMENT FROM ECE501 WHERE EXISTS (SELECT STUDENTID FROM ECE507 WHERE ECE501.SCORE=ECE507.SCORE);
```

SCENARIO 2

Lets assay we need to output all records of ECE507 that exist only where STUDENTID of ECE501 and ECE507 matches.

```
SELECT *FROM ECE507 WHERE EXISTS (SELECT *FROM ECE501 WHERE ECE507.STUDENTID=ECE501.STUDENTID);
```

THANKS FOR VIEWING

NOTE: ***Subsequent topics under SQL will be discussed in Part 4***

Also, all Implementations as far as this presentation is concerned are in line with MySQL Syntax. All have been tested on XAMPP Server.

PART 4 PRESENETATION WILL COVER THE FOLLOWING: NULL FUNCTIONS, SQL CASE, SQL ANY, ALL, SQL SELECT INTO, SQL INSERT INTO, SQL STORED PROCEDURES, SQL OPERATORS AND SQL COMMENTS