**Madiha Aimon Tappal**

[**madihaaimon@gmail.com**](mailto:madihaaimon@gmail.com)

**Day - 5**

**Data Engineering Batch – 1**

**What Is Data Cleaning?**

The process of data cleaning (also called data cleansing) involves identifying any inaccuracies in a dataset and then fixing them. It’s the first step in any analysis and it includes deleting data, updating data, and finding inconsistencies or things that just don’t make sense.

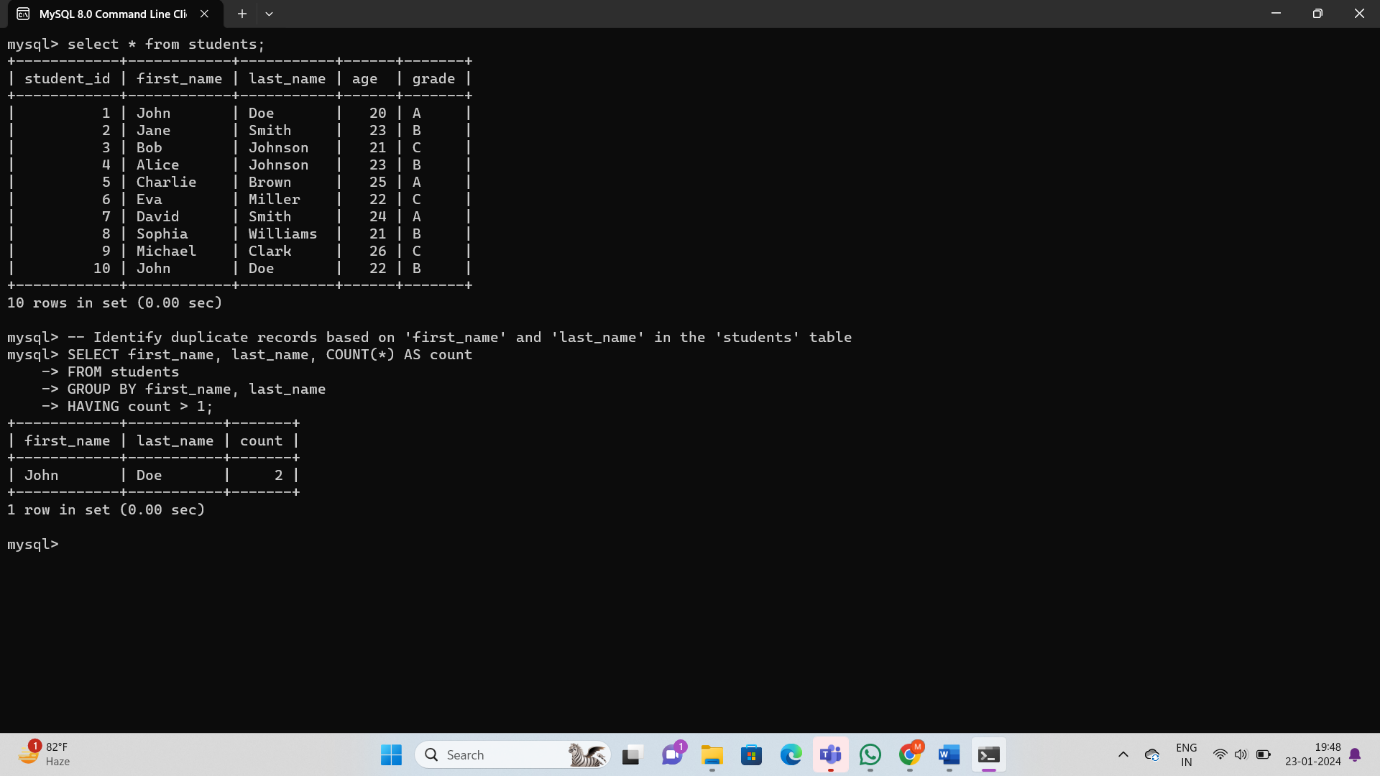
### **How and When to Delete Data**

Sometimes you will encounter scenarios in which you need to remove data from your dataset. This could be because the data is not pertinent to what you are analysing or it is duplicated or inaccurate.

#### **Deleting Duplicate Data**

The first thing we’ll do is look for any data that needs to be deleted. This could be because of duplicates or because the data is not relevant.

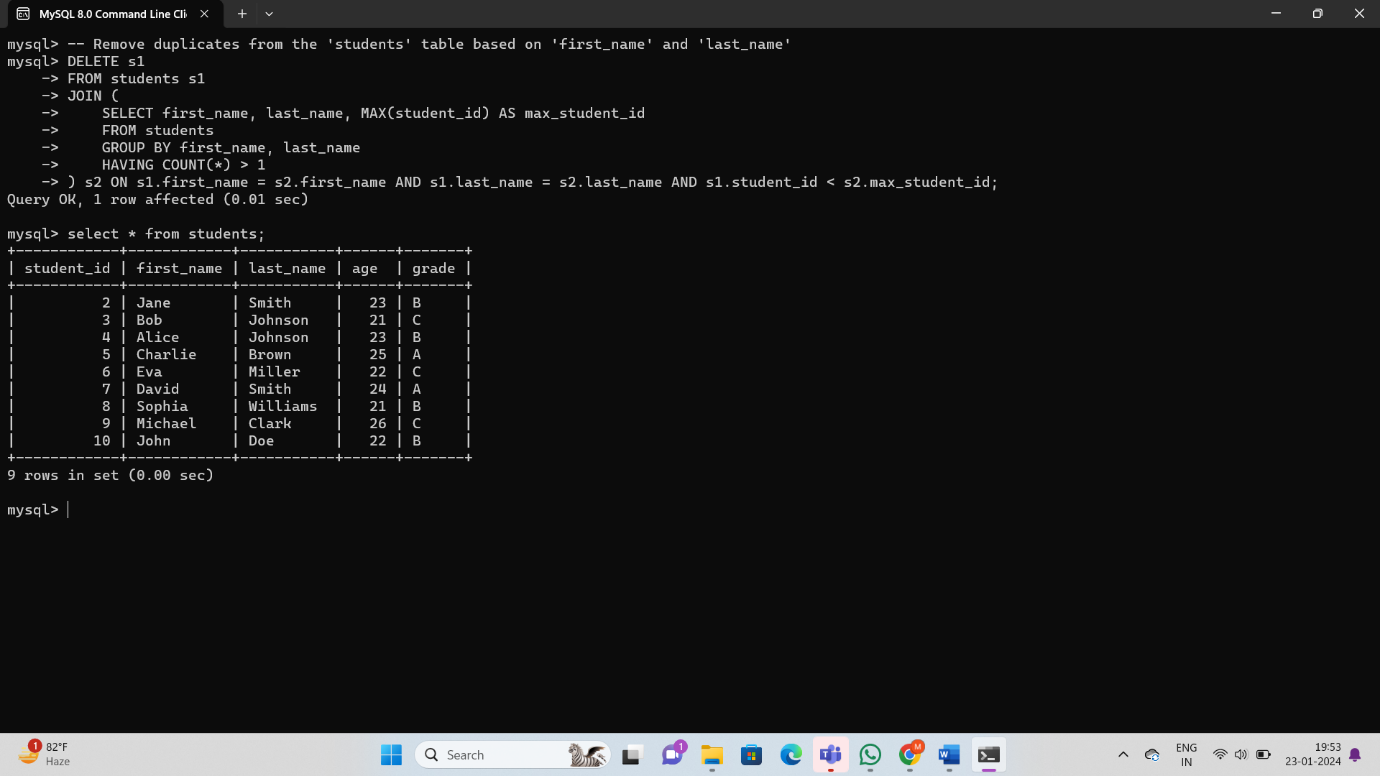
use the **GROUP BY** and **HAVING** clauses to identify duplicate names. This query is going to count the number of times each name exists in the database using **GROUP BY**. Then it uses the **HAVING**clause to filter the results for only those names that exist more than once.



#### This **DELETE** statement uses a similar logic with **ROW\_NUMBER ()** to keep the record with the lowest student\_id among the duplicates.

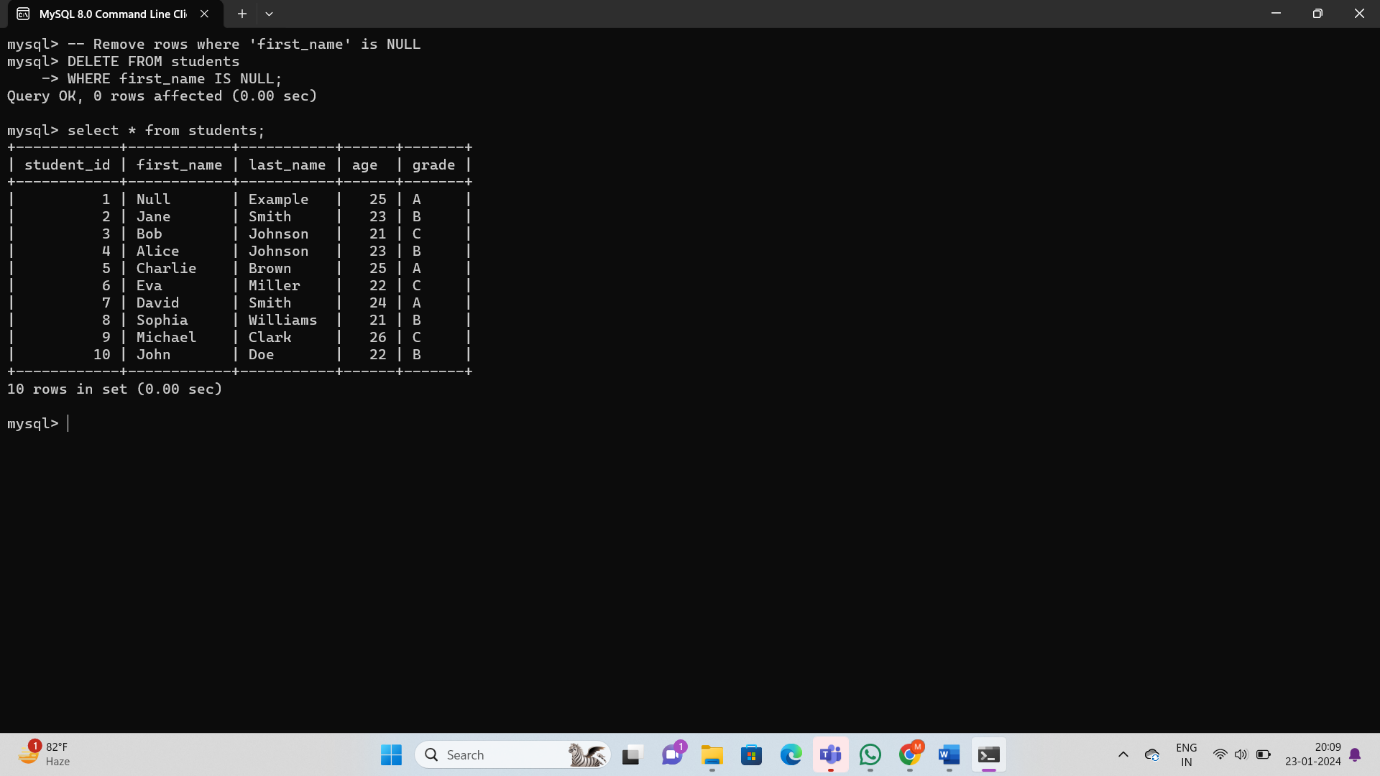
#### **Ordering Data Before Deletion**

In this example, the rows for Toughtam are the same (with the exception of id), so we are not ordering the rows by anything other than how they appear in the database. Many times, you’ll find that rows are duplicated, but maybe the fields are not identical. If this is the case, you can add an **ORDER BY** clause after the **PARTITION BY**.



In this example, the **MAX (student\_id)** is used to determine the record with the highest student\_id among the duplicates, and the **ORDER BY** clause ensures that the records are considered in that order during the deletion process.

#### **Removing NULL Values**

**NULL** indicates a missing value, depending on the implications of NULL values in your data, you can either remove those rows or you can update them.

# **SQL transformation**

Use the SQL transformation to call a stored procedure or function in a relational database or to processes SQL queries midstream in a pipeline. The transformation can call a stored procedure or function, process a saved query, or process a query that you create in the transformation SQL editor.

The SQL transformation can process the following types of SQL statements:

**Stored procedure or stored function**

A stored procedure is a precompiled collection of database procedural statements and optional flow control statements, similar to an executable script. Stored procedures reside in the database and run within the database. A stored function is similar to a stored procedure, except that a function returns a single value.

When the SQL transformation processes a stored procedure or function, it passes input parameters to the stored procedure or function. The stored procedure or function passes the return value or values to the output fields of the transformation.

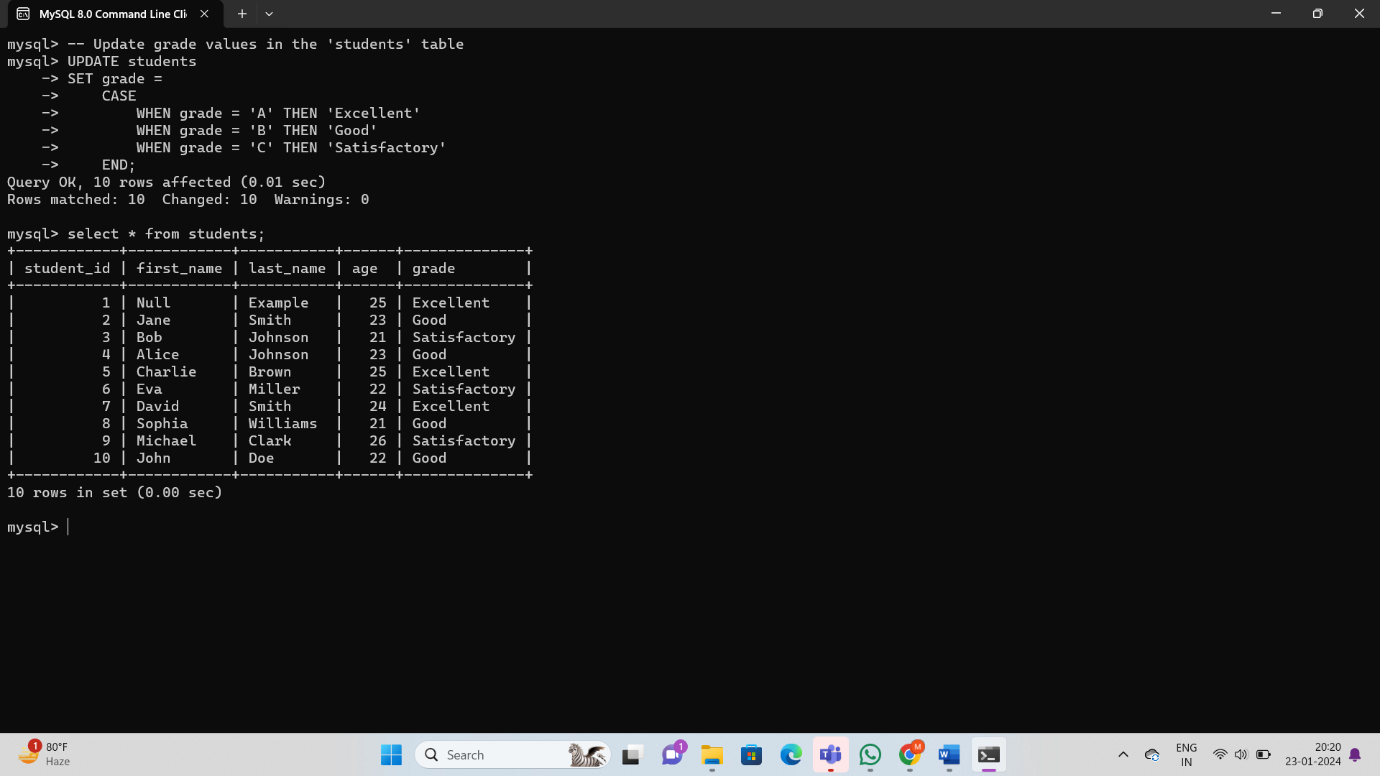
**Saved or user-entered query**

You can configure the SQL transformation to process a saved query that you create in Data Integration or you can enter a query in the SQL editor. The SQL transformation processes the query and returns rows and database errors. You can pass strings or parameters to the query to define dynamic queries or change the selection parameters. You can output multiple rows when the query has a SELECT statement.

Transformation queries are used to modify or transform data within a database. The nature of these queries can vary based on the specific transformation or modification you want to apply to your data.

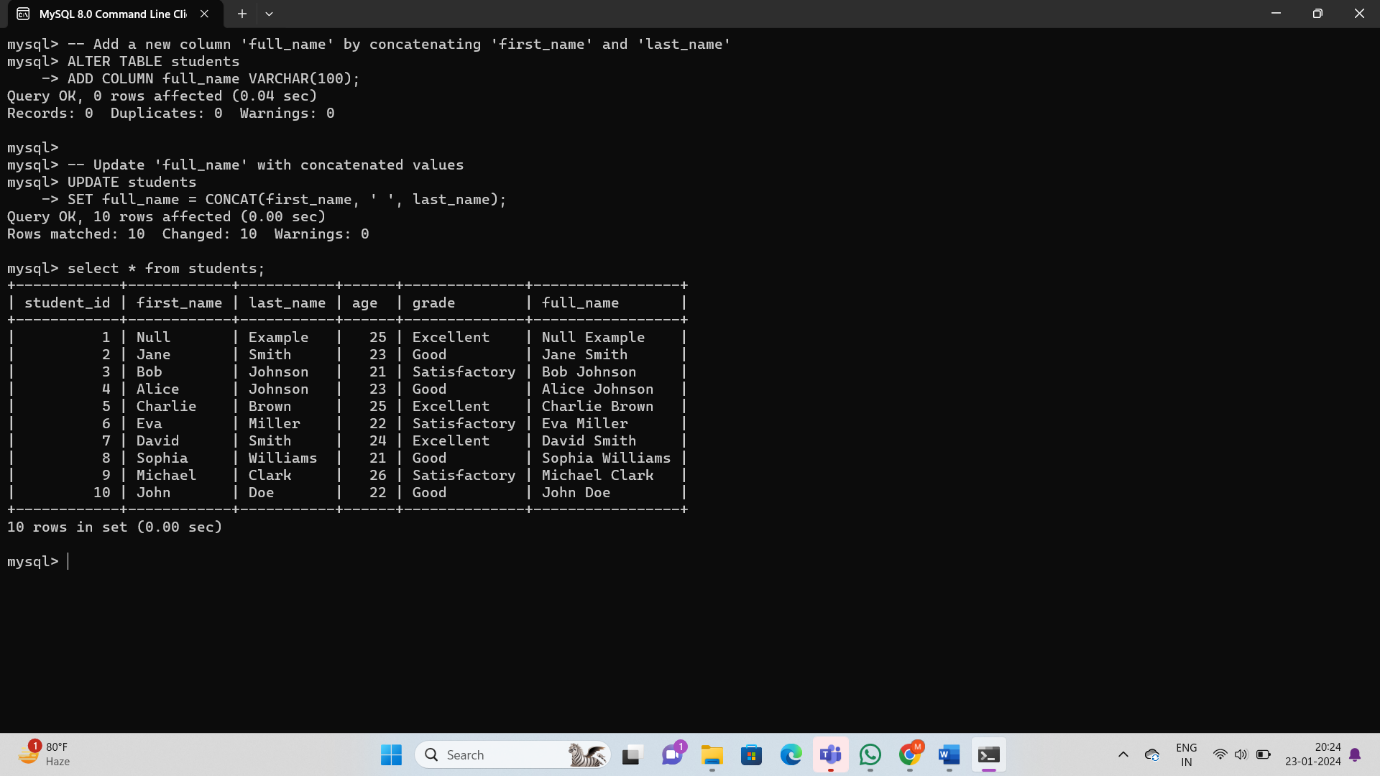
### **Update Grade Values:**

Let's say you want to update the grade values in the student’s table. For example, you want to convert all 'A' grades to 'Excellent,' 'B' grades to 'Good,' and 'C' grades to 'Satisfactory.'



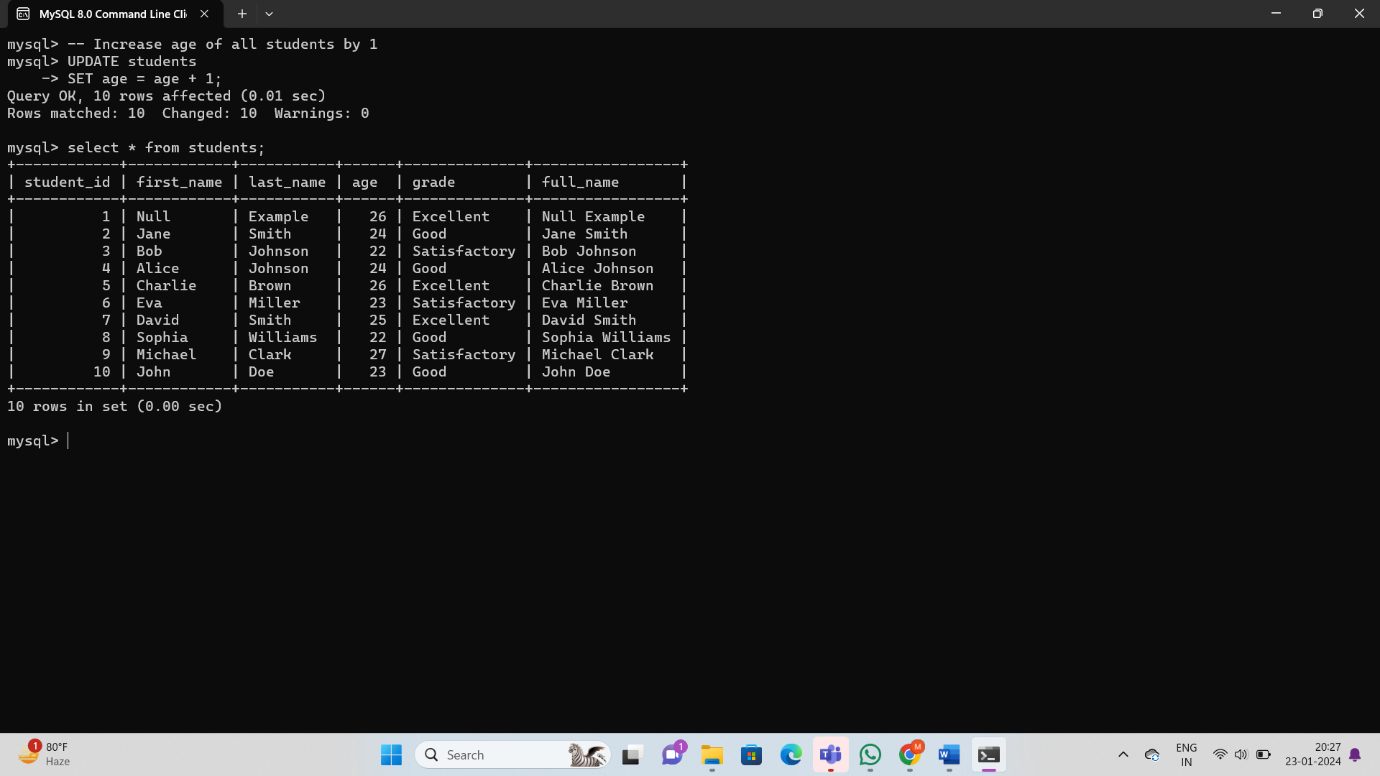
### **Concatenate First Name and Last Name:**

If you want to create a new column that contains the full name by concatenating the **first\_name** and **last\_name** columns:



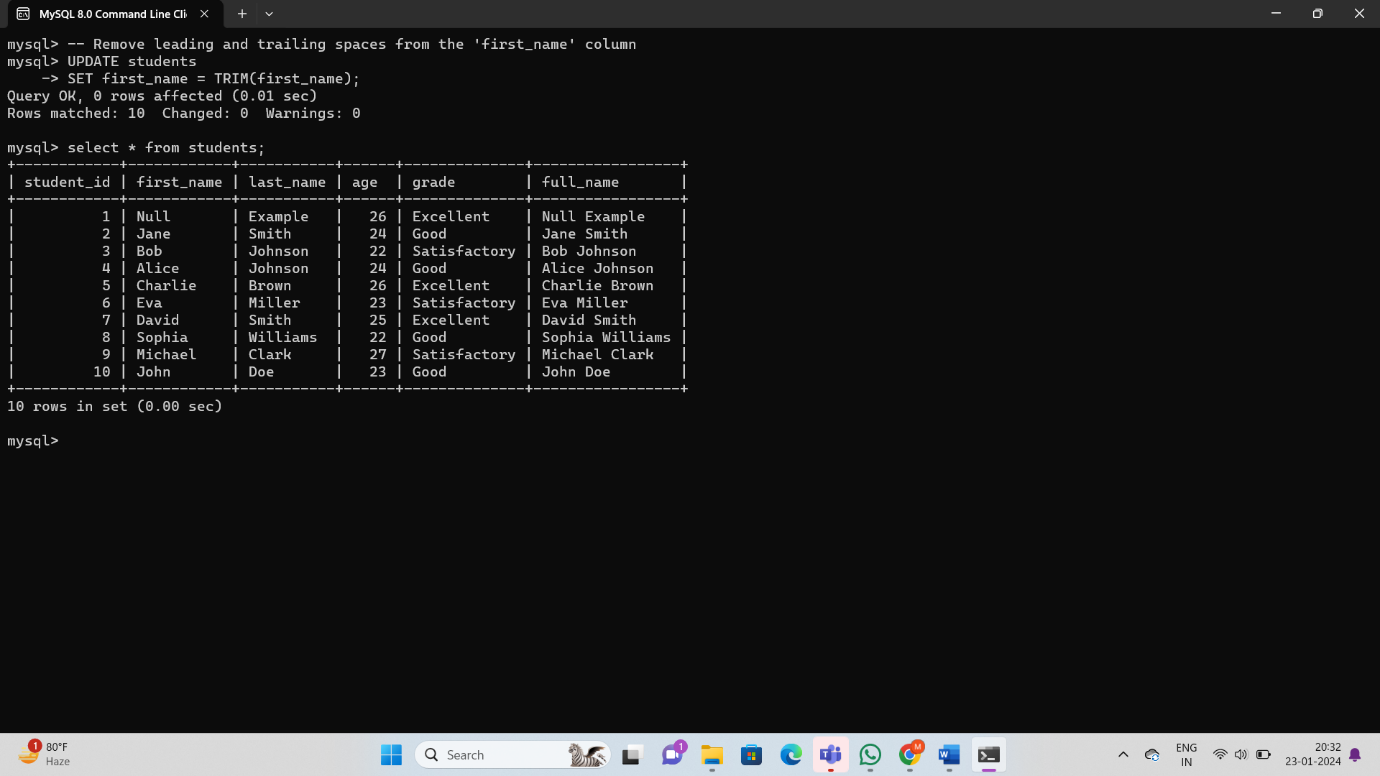
### **Increase Age by 1:**

If you want to increase the age of all students by 1:



### **Remove Leading and Trailing Spaces:**

If you want to remove leading and trailing spaces from the First\_name column:

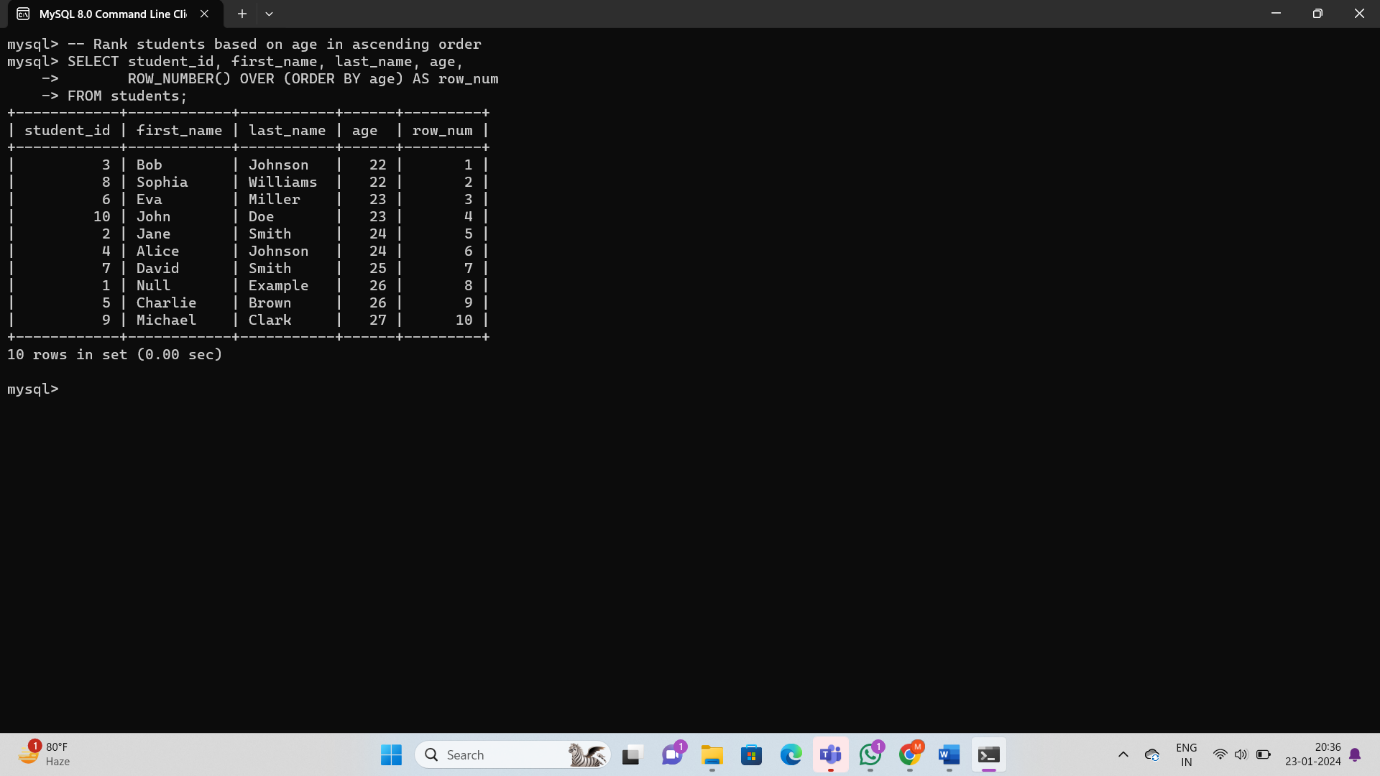


**Ranking Function: -**

Ranking functions in SQL are used to assign a rank to each row within a result set based on the values in one or more columns. The ranking can be assigned in ascending or descending order.

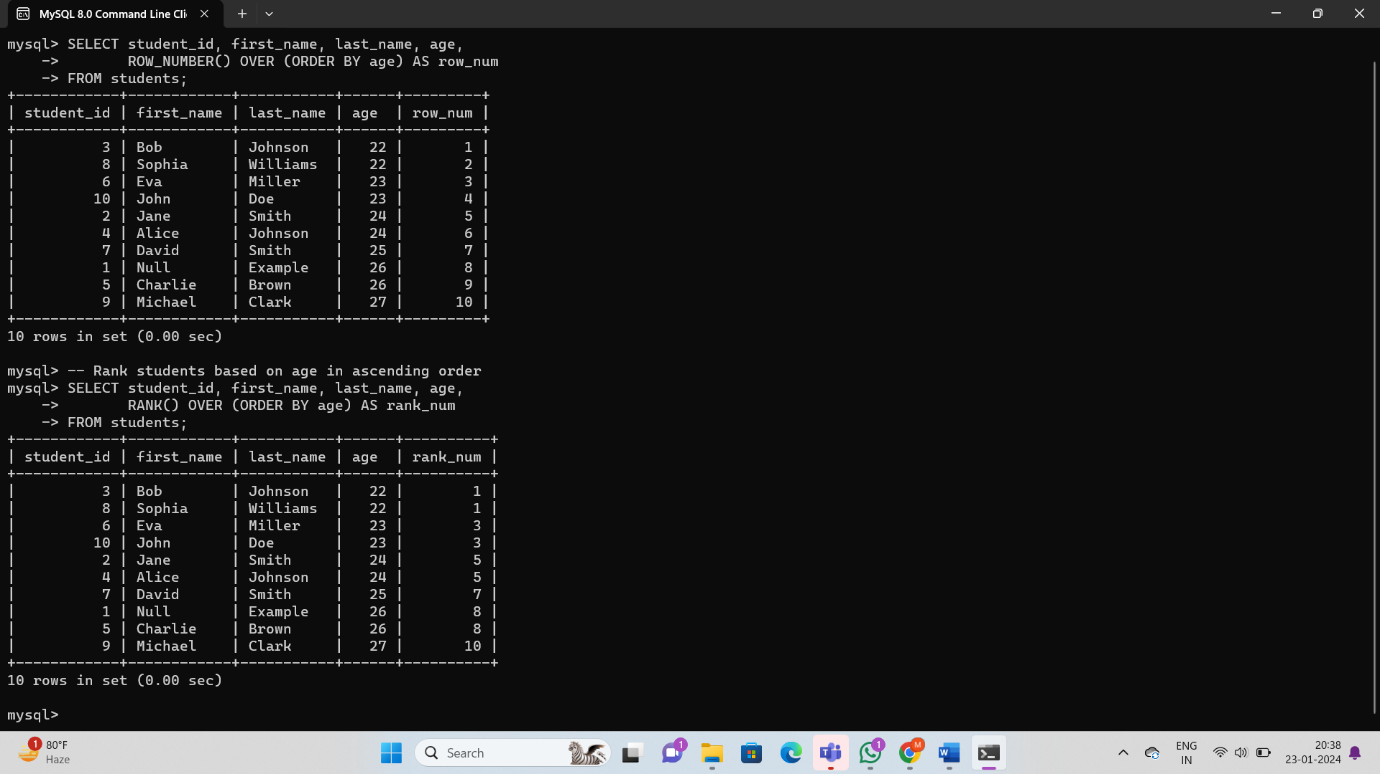
### 1. **ROW\_NUMBER ()**

ROW\_NUMBER () assigns a unique number to each row, starting from 1.



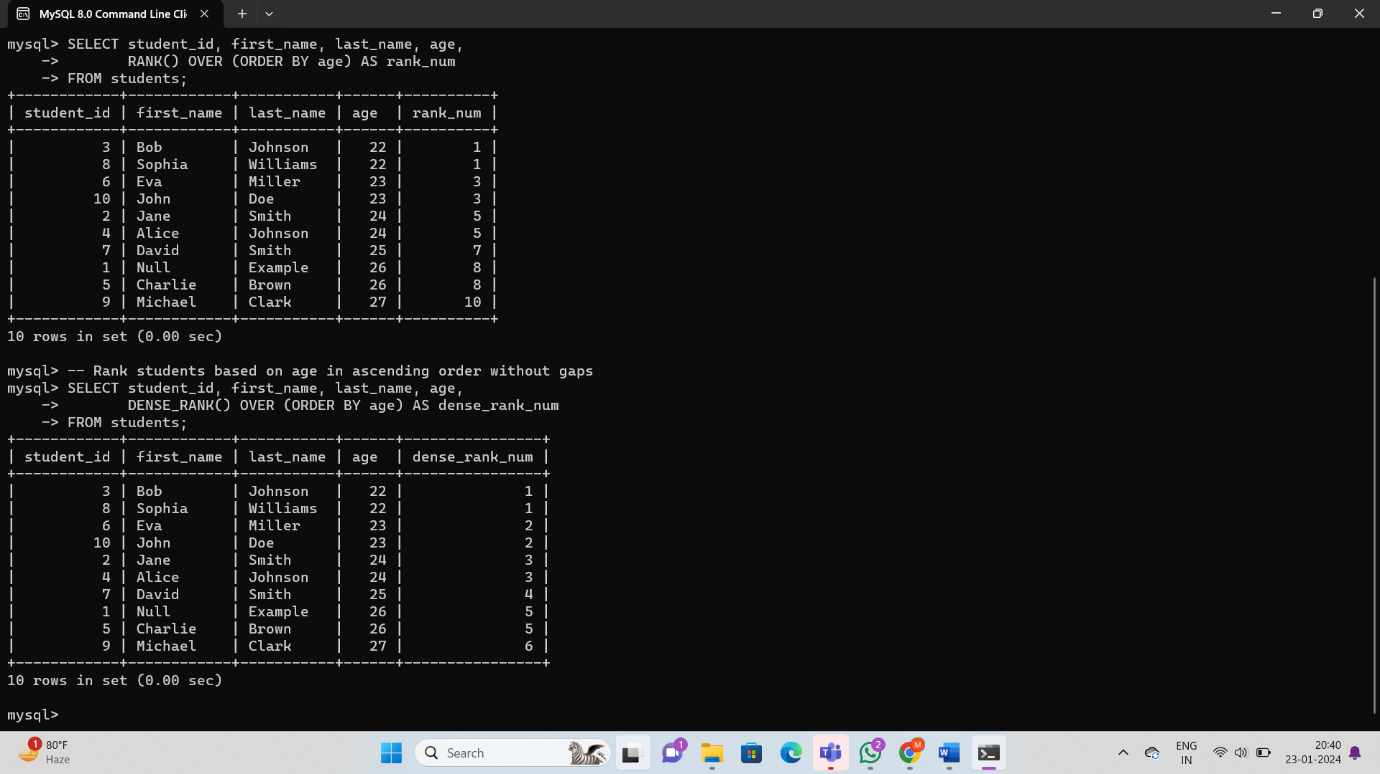
### 2. **RANK** ()

**RANK ()** assigns a rank to each distinct row value, with ties receiving the same rank and leaving gaps.



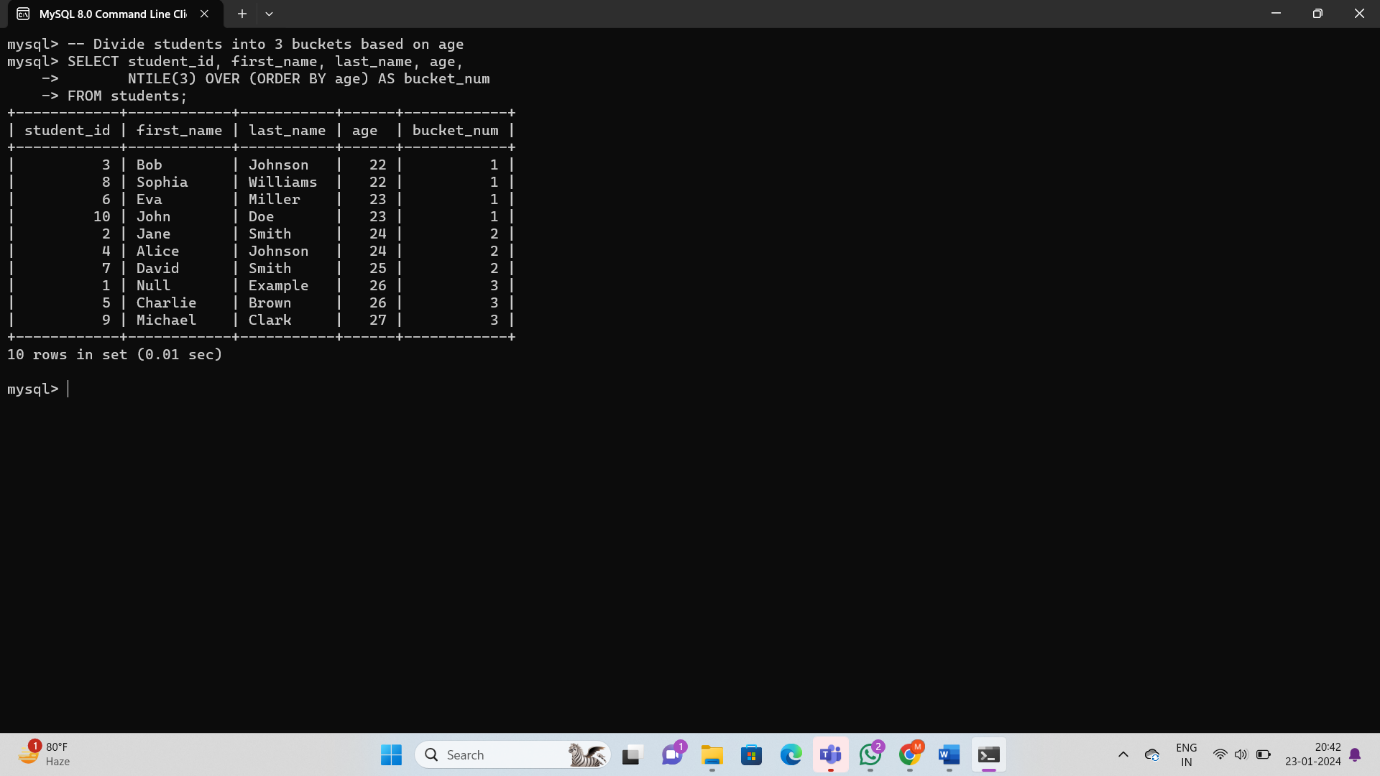
### 3. **DENSE\_RANK ()**

**DENSE\_RANK ()** is similar to **RANK ()**, but it doesn't leave gaps in rank numbers for ties.



### 4. **NTILE ()**

**NTILE(n)** divides the result set into n roughly equal parts and assigns a bucket number to each row.



# **Stored procedures in SQL**

Stored procedures are prepared SQL code that you save so you can reuse it over and over again. So if you have an SQL query that you write over and over again, save it as a stored procedure and call it to run it. You can also pass parameters to stored procedures so that the stored procedure can act on the passed parameter values.

Stored Procedures are created to perform one or more [DML](https://www.geeksforgeeks.org/dml-full-form/)operations on Database. It is nothing but the group of SQL statements that accepts some input in the form of parameters and performs some task and may or may not return a value.

**Syntax:**

Creating a Procedure

CREATE PROCEDURE procedure\_name  
(parameter1 data\_type, parameter2 data\_type, …)  
AS  
BEGIN  
   — SQL statements to be executed  
END

To Execute the procedure

EXEC procedure\_name parameter1\_value, parameter2\_value, ...

**Parameter Explanation**

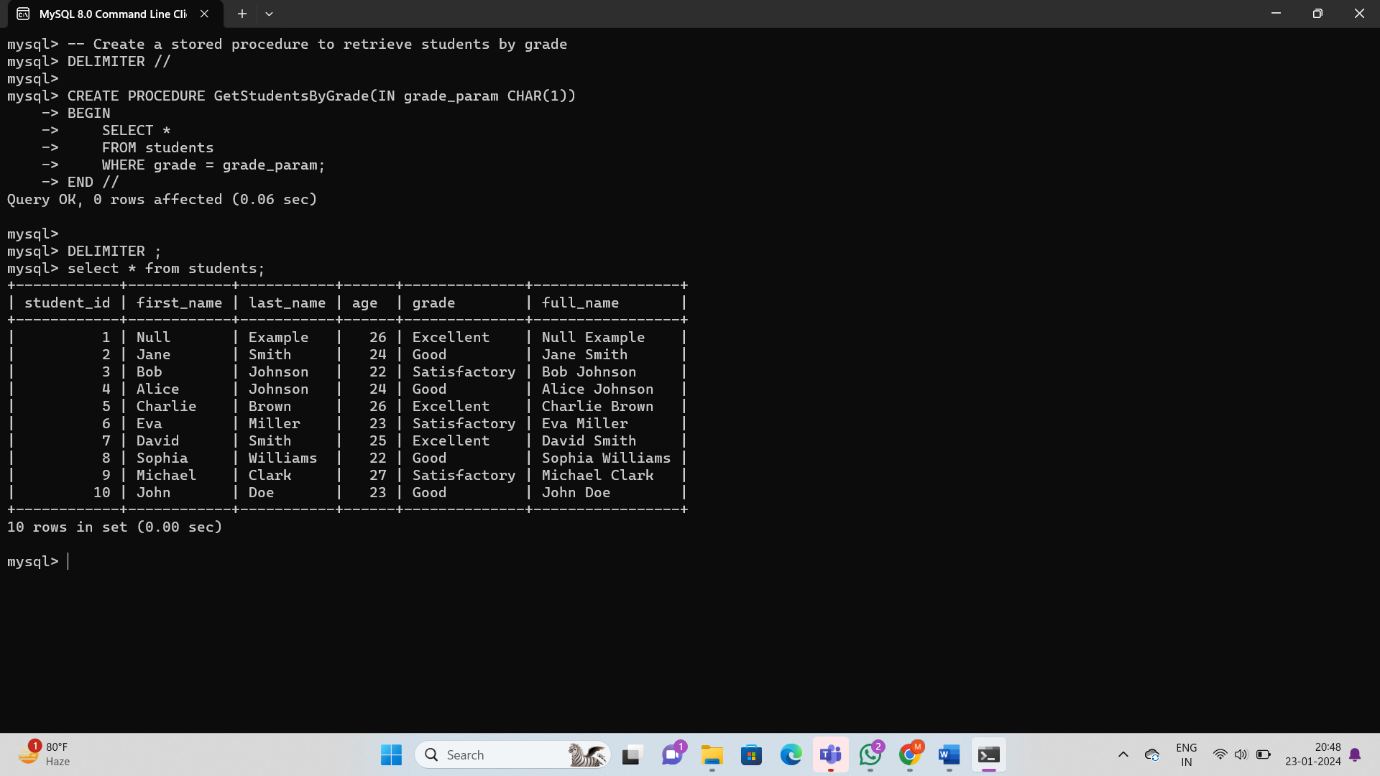
The most important part is the parameters. Parameters are used to pass values to the Procedure. There are different types of parameters, which are as follows:

1. **BEGIN**: This is what directly executes or we can say that it is an executable part.
2. **END**: Up to this, the code will get executed.

A stored procedure in MySQL is a set of SQL statements that can be stored in the database and executed later as a single unit. Here's an example of creating a simple stored procedure for retrieving students based on a specified grade:

In this example:

* GetStudentsByGrade is the name of the stored procedure.
* IN grade\_param CHAR (1) specifies an input parameter named grade\_param of type CHAR (1).
* BEGIN and END enclose the body of the stored procedure, which contains the SQL statements.
* The SELECT statement retrieves students based on the specified grade.



To call the stored procedure, you can use the following:

