**STUDENT DATABASE: Data Cleaning and Exploration**

I have developed a simulated student database, focusing on demonstrating my proficiency in data cleaning. Following the cleaning process, I delve into data exploration, utilizing aggregate functions, nested queries, and window functions to address real world inquiries.

**About the database**:

* ‘Studentdata’ database has two tables- student\_address and grades
* Student\_address table has 4 columns i.e., student id (primary key), name, address and date of enrollment and data for 20 students. Some of the values in this table are duplicate.
* Grades table has 4 columns i.e., student id (primary key), name, subject and grade. This table has grades for English, physics, chemistry and biology for each student and has 80 rows. Some of the data (student names) is missing from this table.

**Data Cleaning**:

* Change data format- datetime format of date\_enrolled column was changed to date format by adding a new column ‘date\_enrolled\_converted’ and values from date\_enrolled column were added to the new column after converting the format to date. The original ‘date\_enrolled’ column was then dropped to remove duplicate information.
* Breaking out address into individual columns- streetaddress, city, province and postalcode columns were added to the student\_address table. The data in address column was broken down into substrings using **substring\_index and nested substring\_index**, then each substring was assigned to the appropriate column. The address column was then dropped.
* Removing duplicate entries- some of the entries in student\_address table had duplicate data, but a new student\_id was assigned to the duplicate entries. To find the duplicates, data was **grouped by** name and results were filtered to only show entries with name **count** greater than 1. Duplicate entries were then deleted by performing an **inner join** on student\_address table where name was same but student\_id was different.
* Changing abbreviations- **case when** statement was used to change subject abbreviations to full subject names in grades table.
* Populating missing data- to solve the issue of missing data from grades table, a new table ‘grades\_new’ was created with same columns as grades table. Data was then inserted into the new table by joining grades table and student\_address table on student\_id. Old grades table was then dropped.

**Data Exploration**:

The following questions were answered using MySQL queries:

* Name the student with highest grades in each subject.:

The query used selects the name, subject and grades from the ‘grades\_new’ table for the maximum grade in each subject. It uses a **subquery** with the **IN** clause to filter the results based on the **max** grade in each subject. The **group by** in subquery ensures that we get maximum grades for each subject.

* What are the total grades scored by each student? :

The query used retrieves the total grades for each student from the ‘grades\_new’ table, **grouping** the results by both student\_id and name. It calculates the **sum** of grades for each student and orders the output in **descending order** showcasing the highest totals first.

* What is the average grade for each subject?

AVG function is used to calculate the average grade, rounded to 2 decimal places and then results are grouped by subject

* Rank students based on score in each subject

The **DENSE\_RANK()** window function is used with the **PARTITION BY** clause to assign ranks separately for each subject. The output provides a summary of students’ ranking within each subject based on their grades.

* Create separate columns for grades for each subject.

The SQL query used summarizes the grades for each student in English, Biology, Physics, and Chemistry from the "grades\_new" table. It utilizes **conditional aggregation** with CASE WHEN statements to categorize grades based on subjects. The **COALESCE** function is employed to handle potential NULL values, and the results are **grouped by** student\_name, providing a breakdown of their scores in the specified subjects.

* How many students scored less than average grades in each subject?

The query utilizes a correlated subquery to calculate the average grade for each subject in the inner query. The outer query then counts the students in each subject from the ‘grades\_new’ table whose grades fall below the corresponding subject’s average. The results of this query show the count of students below average in each subject.

* Assign students to retest for the subjects they scored below 45.

The query creates a column named retest using a **CASE WHEN** statement, **concatenating** 'retest for' with the subject for students whose grades are below 45. The result includes the student\_name and the corresponding retest message, showing students who qualify for a retest in specific subjects. The additional condition ensures that the retest column is not null.