ADT HOMEWORK 4 (hrupchan)

PART - 1

Data Modelling

Step 1 - Locate data in CSV format (for example, Kaggle, governmental, CDC, yelp reviews etc).

Step 2- Use the ARROWS tool to create a schema with nodes, relations, and properties

Step 3- Write Part 1 description (WORD document) and include the figure schema from Arrows: Namely, 1) what data you are using, 2) data origin (who collected and provided it), and 3) schema screenshot

1) What data you are using ?

   The dataset chosen for this assignment is from Kaggle and the name of the data set is Student performance in exams, The dataset's shape has 8 columns and 1000 observations.

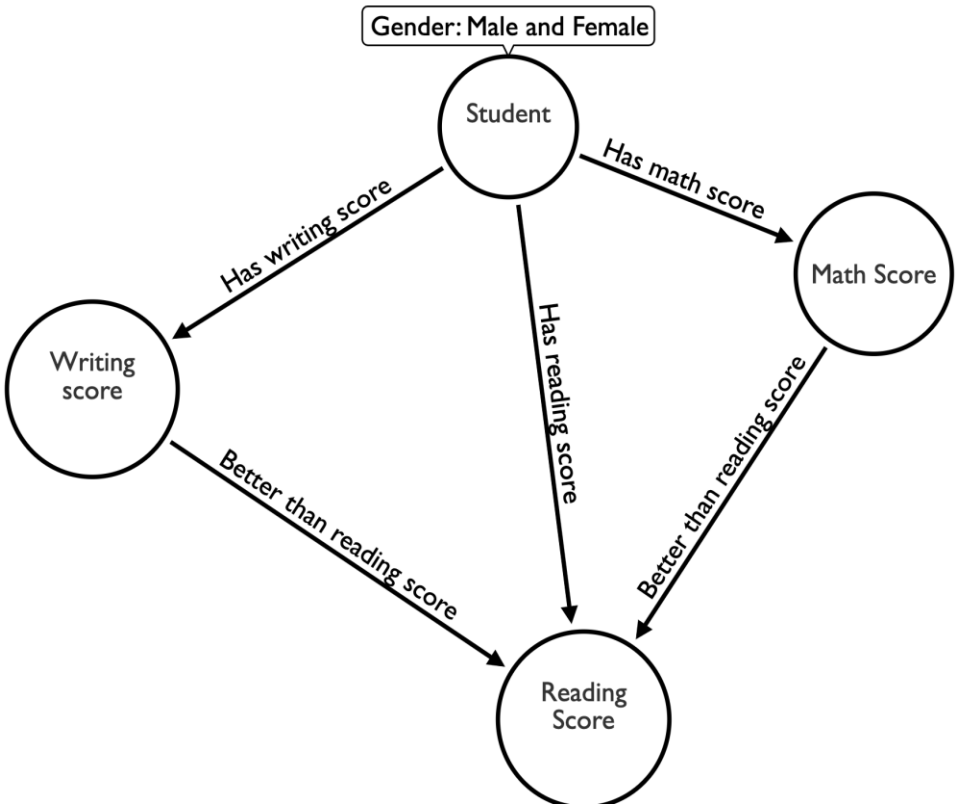
2) Data Origin

   The dataset is from:
   Kaggle's Student Performance in Exams and it consists of 8 columns which are:

   - **gender:** the student's gender (male or female),
   - **race/ethnicity**: the student's race/ethnicity (group A, group B, group C, group D, or group E),
   - **parental level of education**: the highest level of education achieved by the student's parent(s) (some high school, high school, some college, associate degree, bachelor's degree, or master's degree),
   - **lunch:** whether the student received free/reduced-price lunch or paid for lunch (free/reduced or standard),
   - **test preparation course:** Whether test preparation course was completed (completed) or not (none).
   - **math score**: the student's score on a math exam (out of 100),
   - **reading score**: the student's score on a reading exam (out of 100),
   - **writing score**: the student's score on a writing exam (out of 100).

   The dataset was originally created by a user named "JAKKI SESHAPANPU", who compiled the data from various sources such as U.S. Department of Education and the National Center for Education Statistics. It contains information about student demographics, test scores, and other variables related to academic performance.

3.1) First Schema:

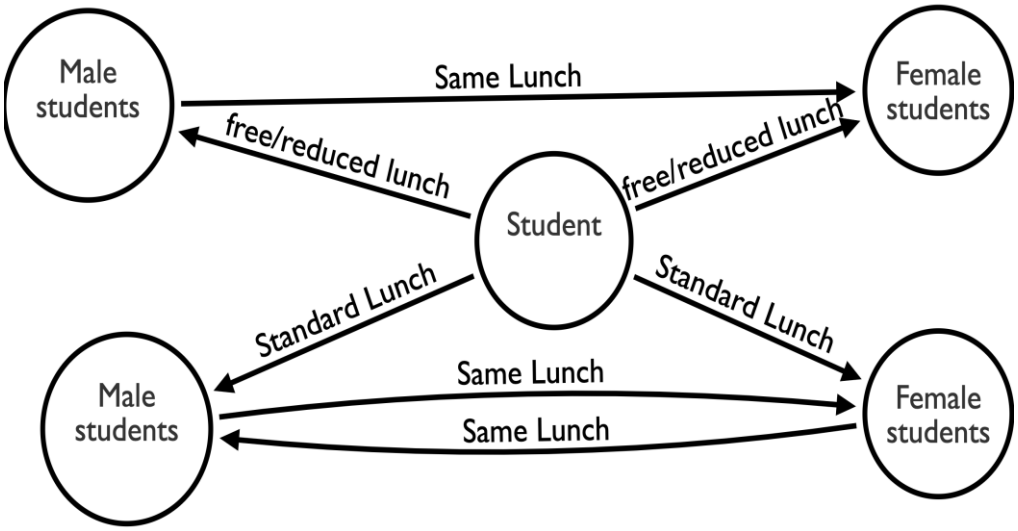


The first schema is formed by selecting student as the first node and Gender as the property of the student node, where the gender is specified as male and female, and then selecting the other three nodes as math score, reading score, and writing score, and then establishing a relationship between the nodes as follows: The student node is linked to all three nodes by the connection "HAS MATH/READING/WRITING SCORE"; this relation indicates that all students have a math, reading, and writing score.

The second relationship is established between the math score and the reading score, which are taken as the other two nodes of the schema. The math score node is connected to the reading score node with the relation "BETTER THAN READING SCORE," which denotes students who have a higher math score than a higher reading score.

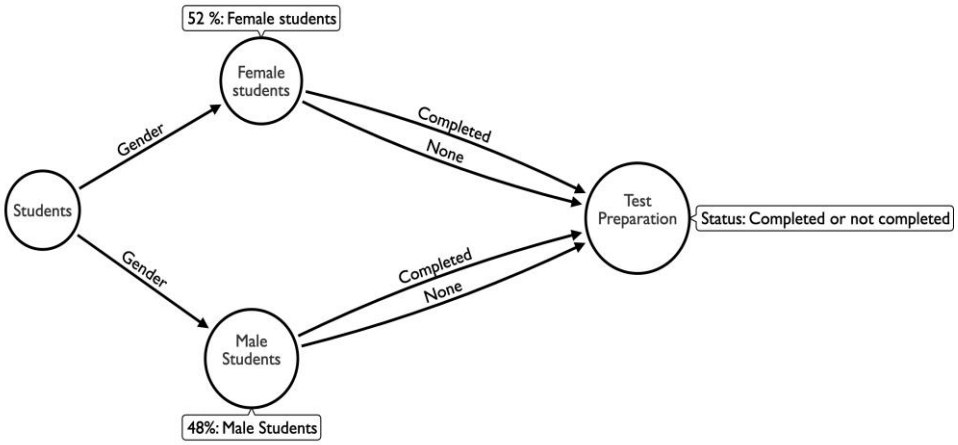
The third relationship is established between the writing score and the reading score, which are taken as the other two nodes of the schema, the math score node is connected to the reading score node with the relation "BETTER THAN READING SCORE", this relation denotes students who have a higher writing score than a higher reading score.

3.2) Second schema:



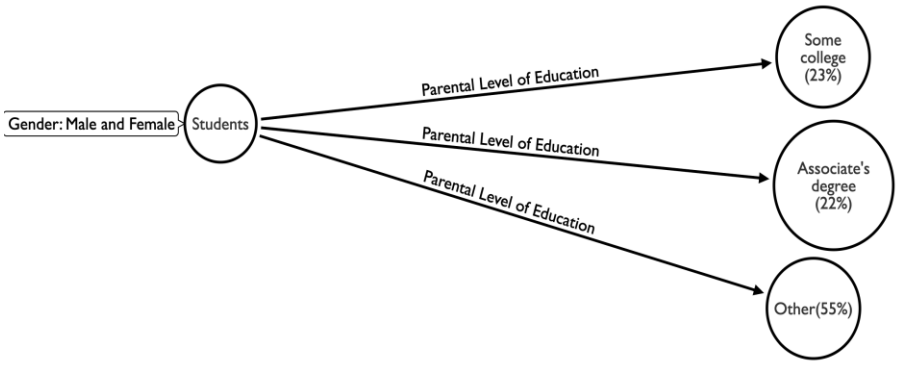
The relationship between the male students and female students is based on the "**SAME LUNCH**" in the second schema, which is generated by using the student as the first node and splitting the students into female students and male students as two independent nodes. First, all male and female students are removed from the node **STUDENT** with the lunch type of **STANDARD LUNCH** separately. Next, all male and female students who have the same lunch type as the rest of the students are placed under the relation of **SAME LUNCH**.

3.3) Third schema:



The second schema is formed by taking student as the first node and dividing the students into two separate nodes based on gender. The third node formed is the test preparation node, and the above schema shows the relationship between males and females who have completed the test preparation course and males and females who haven't.

3.4) Fourth schema:



The nodes formed are student nodes (**Gender**: male and female) and three other nodes are some **college node**, **associate's degree node**, and other node (types of **parental level of education**), and the students are associated to the other three nodes through the parental level of education relationship.

PART - 2

Data Import

Loading data:

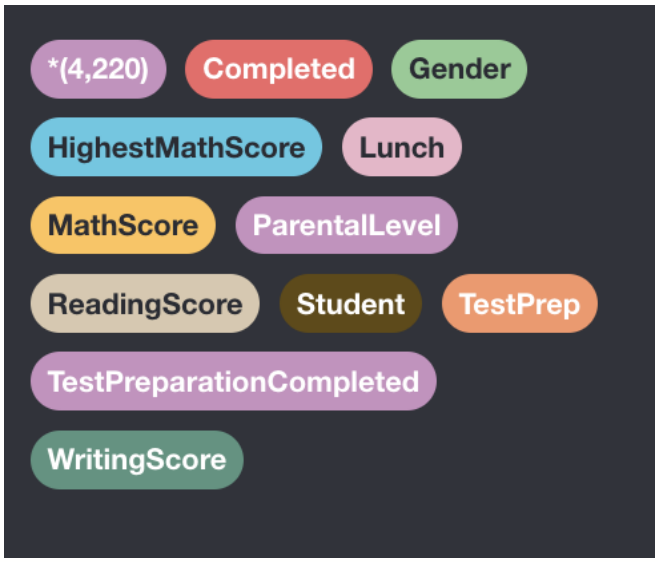
Load from a CSV file:

```
LOAD CSV WITH HEADERS FROM
"D:/ADT/StudentsPerformance.csv"' AS row
with row
return row
```

This Cypher code loads a CSV file from the supplied file directory with headers and assigns each row to the variable row. Then, without conducting any extra actions, it returns each row of the CSV file as a result.

Nodes:

The nodes formed in the Student Performance in Exams are Student , TestPrep , MathScore, ReadingScore , WritingScore, Parental Level of Education , lunch.



2.1) Student node :

```
LOAD CSV WITH HEADERS FROM
"D:/ADT/StudentsPerformance.csv " AS row
CREATE (:Student {
gender: row. Gender,
race_ethnicity: row["race/ethnicity"],
```

```
parental education: row["parental level of education"],
lunch: row. Lunch,
test_preparation_course: row["test preparation course"],
math_score: toInteger(row["math score"]),
reading_score: toInteger(row["reading score"]),
writing_score: toInteger(row["writing score"])
})
```

The student node is formed in this case, and it has features like gender, parents level of education, lunch, test preparation course, math score, reading score, and writing score. The CREATE clause constructs a node for each row in the CSV file and assigns values from the row's associated columns to the node's properties. The toInteger function is used to convert the math, reading, and writing scores from strings to integers before assigning them to the appropriate attributes.

2.2) TestPrep node : The TestPrep node has properties as none and completed

```
CREATE (:Test_preparation_course {name: "none"})
CREATE (:Test_preparation_course {name: "completed"})
```

2.3) Gender node : with the property as male and female

```
CREATE (:Gender {name: "male"})
CREATE (:Gender {name: "female"})
```

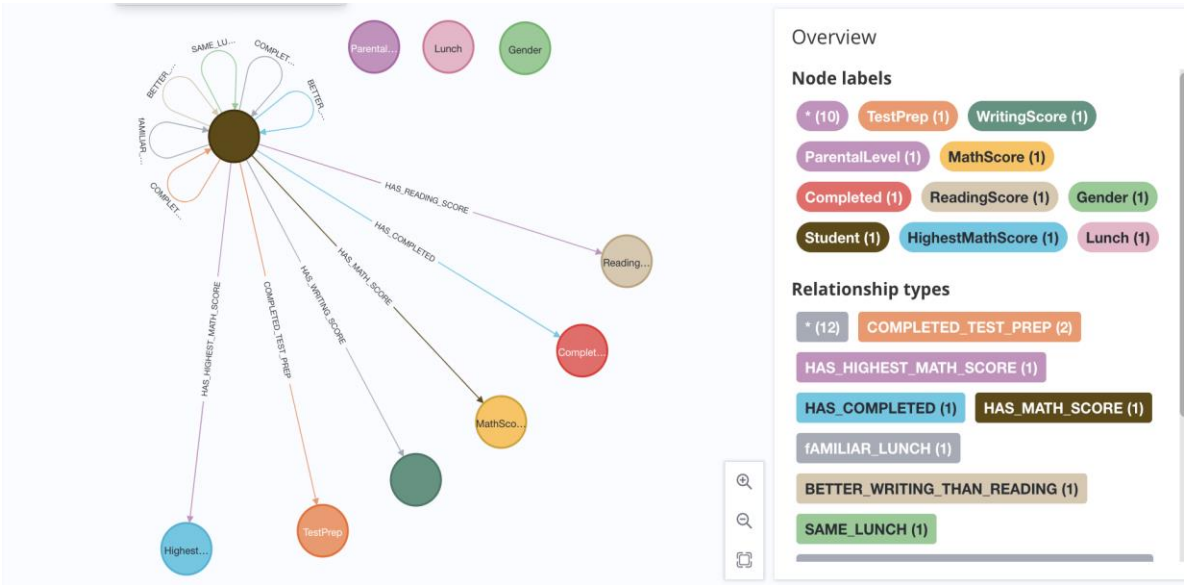
2.4) Parental level education node : property as some college, associate’s degree and other

```
CREATE (:ParentalLevel {name: "some college"})
CREATE (:ParentalLevel {name: "Other"})
CREATE (:ParentalLevel {name: "associate's degree"})
```

2.5) lunch node : with properties as standard and free/reduced

```
CREATE (:Lunch {name: "free/reduced"})
CREATE (:Lunch {name: "standard "})
```

Relationship :



The graph represents all the nodes and Relationship types

1)

```
// a relation between students who have worse reading score than math score
MATCH (s:Student)
WHERE s.reading_score < s.math_score
MERGE (s)-[:BETTER MATH_THAN_READING]→(s)
```

The code will match all student nodes whose math score is higher than their reading score and create a relationship of type BETTER_MATH_THAN_READING from each matched student node to itself. The relationship implies that the student is better at math than reading.

2)

```
// a relation between students who have better writing score than reading score
MATCH (s:Student)
WHERE s.reading_score < s.writing_score
MERGE (s)-[:BETTER_WRITING_THAN_READING]->(s)
```

The code is used to establish a link between pupils who performed higher in writing than in reading. First, the algorithm selects all pupils who have greater writing scores than reading scores. The WHERE clause examines a student's writing_score and reading_score and returns only those students whose writing_score is higher than their reading_score.

The algorithm then establishes a connection between the chosen students and themselves, showing that their writing score was higher than their reading score. The [:BETTER WRITING THAN READING] identifies the kind of relationship, and the MERGE clause establishes the relationship. The node with which the relationship will be established is indicated by the (s) at the end of the code.

3)
// relation between students that have highest math score in the whole class

```
MATCH (s:Student)
WITH MAX(s.math_score) AS high_math_score
MATCH (s:Student)
WHERE s.math_score = highest_ math _score
WITH COLLECT(s) AS math_topper_students
UNWIND math_topper_students AS topper
MERGE (topper)-[:HAS_HIGHEST_MATH_SCORE]->(:HighestMathScore)
```

4)
//relation between male and female students who have completed the test preparation course

```
MATCH (male:Student (gender: male', test prep: 'completed})
MATCH (female:Student (gender: female', test prep: 'completed'})
MERGE (male)-[:COMPLETED_TEST_PREP]->(female)
```

In order to establish a connection between male and female students who have finished the exam preparation course, the aforementioned code is a Cypher query. Starting with the pattern (male: Student gender:'male', TestPrep: 'completed'), the query matches all male students who have successfully completed the test preparation course. Then, using the pattern (female: Student gender: 'female', TestPrep: 'completed'), it matches all female students who have successfully completed the test preparation course.

Finally, using the MERGE clause, the query establishes a COMPLETED TEST PREP relationship between each male and female student who has finished the test preparation course: (male)-[:COMPLETED TEST PREP]->(female). The ability to identify pairs of students who have finished the test preparation course and maybe compare their results to those who did not can be studied using this relationship.

5)
// Students that have completed status in all the three courses

```
MATCH (stu:Student)
WHERE stu.reading_score IS NOT NULL AND stu.writing_score IS NOT NULL AND
stu.math_score IS NOT NULL
MERGE (comp:Completed {name: "Completed"})
MERGE (stu)-[:HAS_COMPLETED]->(comp)
```

By adding a new node with the name "Completed," the code above establishes a relationship between the nodes representing the students who have successfully finished all three tests (math, reading, and writing).

The first indication that the students have finished all three tests is that the code matches all nodes with the label "Student" where the math, reading, and writing scores are not null. The MERGE clause is then used to produce a new node with the label "Completed." A "Completed"

node will be reused if one already exists. The HAS COMPLETED relationship type is then used to establish a connection between the student node and the "Completed" node. This form of connection suggests that the student has finished all three tests.

6)

// This query builds a collection of those pupils after first filtering out the male and female students who have regular lunch individually. The SAME LUNCH connection is then established between each pair of students by matching each female student with each male student in their respective collections.

```
MATCH (f:Student)
WHERE f.gender "female" AND flunch "standard"
WITH COLLECT(F) AS female_standard
MATCH (m:Student)
WHERE m.gender male "AND m. Lunch standard"
```

7)

// a link between students whose parents hold associate's degrees in education

```
MATCH (s:Student)
WHERE s.`parental_education` = "associate's degree"
WITH COLLECT(s) AS associate_students
UNWIND associate_students AS student
MERGE (student)-[:HAS_ASSOCIATE_EDUCATION]->(:AssociateEducation)
```

The above code creates a relationship between the student(s) with the highest math score in the dataset and a node labelled "HighestMathScore" using the Cypher query language in Neo4j.

The code first finds the maximum math score achieved by any student in the dataset using the MAX function. Then it finds all the students who achieved that score using the MATCH clause. It collects those students into a list using the COLLECT function.

Next, it uses the UNWIND clause to iterate over each student in the list and creates a new relationship between the student and a new node labelled "HighestMathScore" using the MERGE clause. This relationship is labelled HAS_HIGHEST_MATH_SCORE.

In summary, the code identifies the student(s) with the highest math score in the dataset and creates a relationship between them and a new node to signify this achievement.

NODES EXPLAINATION

1) Using information from a CSV file placed at the supplied file path, numerous nodes and relations have been constructed in the Neo4j database, similar to how the student node is formed. The code constructs a **Student** node for each row in the CSV file after reading the data using the LOAD CSV function. The node with the properties **gender, parental education, lunch, test_preparation_course, math_score, reading_score,** and **writing**_score is created using the **CREATE** command. The string values of the math, reading, and writing scores are

converted to integers using the **toInteger** function. This code assists in structuring the student data storage in the Neo4j database, facilitating data querying and analysis.

2) There is now a TestPrep node. The two possible values for the test preparation course column in the StudentsPerformance.csv file can be represented by this node. To show if a specific student finished the test preparation course or not, they can be linked to the Student nodes using relationships.

3) The "Gender" node with the "name" property. It specifically produces a node with the property "name" set to "female" and a second node with the property set to "male". These nodes can be used to classify or organize other nodes, such students, into groups according to their gender. This can be advantageous for data analysis and querying, as well as for establishing associations between nodes based on gender.

4) The first code generates three parental level nodes: "some college", "associate's degree", and "Other". These nodes can be used to form relationships with pupils who have each type of parental level. This is useful for queries such as determining the number of students with a specific parental level or the average score of students with a specific parental level.

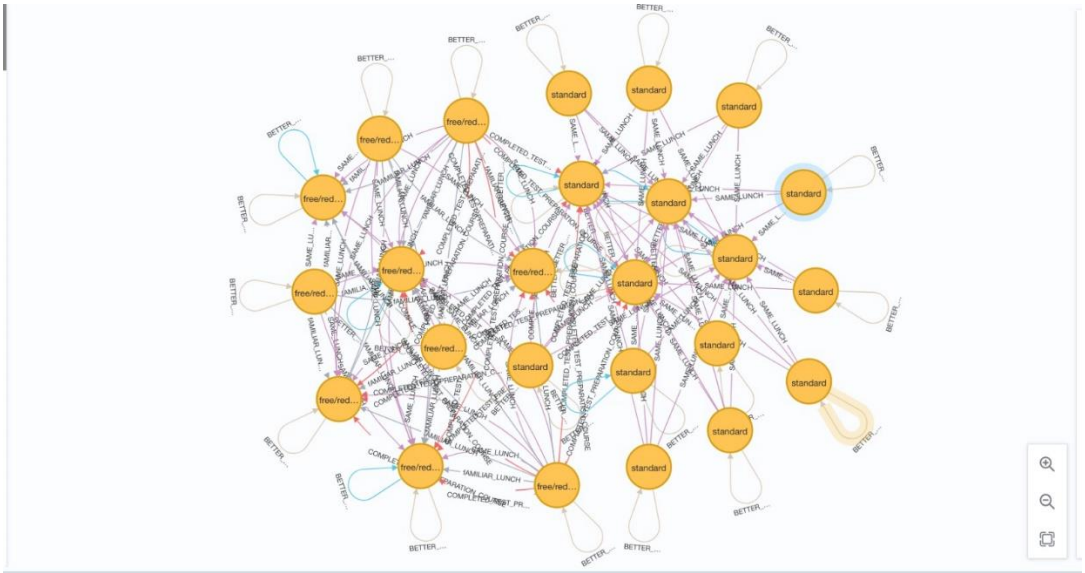
5) The second function generates two meal nodes: "standard" and "free/reduced". These nodes can be utilized to form relationships with the students who eat each sort of lunch. This can be useful for queries such as determining the number of students who eat a specific type of lunch or the average score of students who eat a specific sort of lunch.

Part - 3

Graph Exploration

1) **A relation between students who have worse reading score than math score**

   This tells the relationship between students that have writing score better than the reading score and here the nodes are the students and the relationship between the students formed is of has writing score better

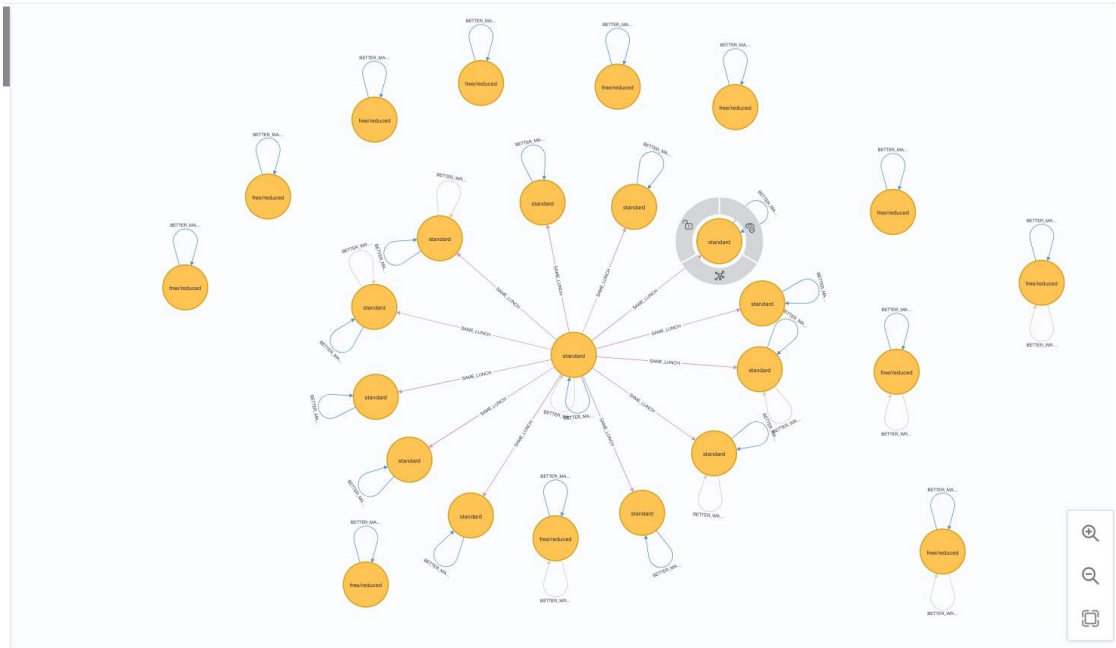


p


```
{
  "start": {
    "identity": 0,
    "labels": [
      "Student"
    ],
    "properties": {
      "lunch": "standard",
      "writing_score": 74.0,
      "gender": "female",
      "reading_score": 72.0,
      "test_preparation_course": "none",
      "math_score": 72.0,
      "parent_education": "bachelor's degree"
    },
    "elementId": "0"
```


2) a relation between students who have better writing score than reading score.

This tells the relationship between students that have math score better than the reading score and here the nodes are the students and the relationship between the students formed is of has writing score better.

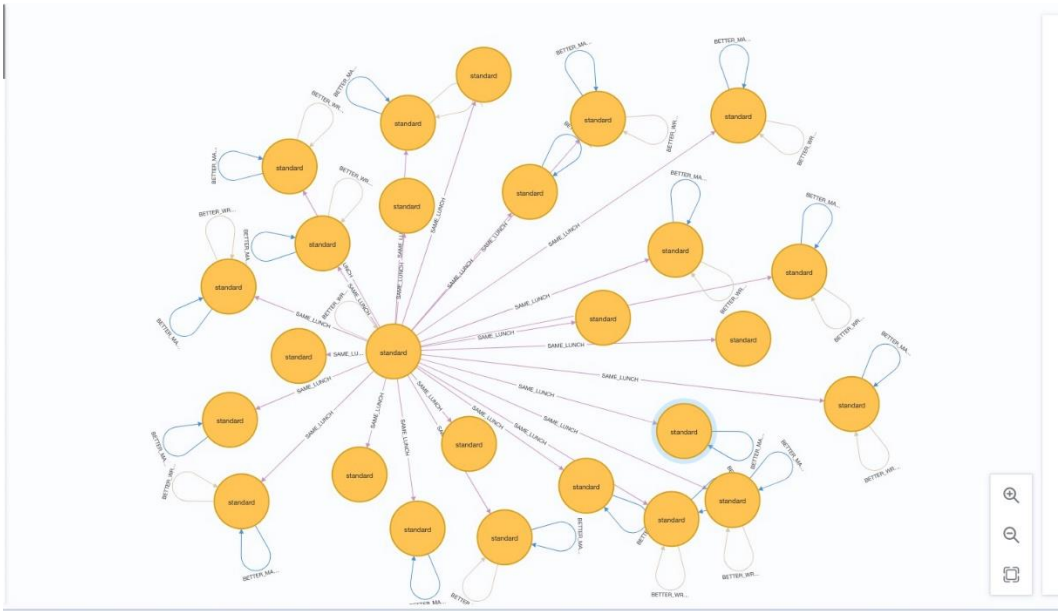


p


```
{
  "start": {
    "identity": 10,
    "labels": [
      "Student"
    ],
    "properties": {
      "lunch": "standard",
      "writing_score": 52.0,
      "reading_score": 54.0,
      "gender": "male",
      "test_preparation_course": "none",
      "math_score": 58.0,
      "parent_education": "associate's degree"
    },
    "elementId": "10"
```


3) The count of male and female students have the same lunch that is the standard lunch or the free/reduced lunch which is put under as same lunch.

The student node created forms a relationship as same lunch between the students which means that the count of students that have the same lunch as either standard lunch or the free/reduced lunch are connected to each other by the respective relationship lines.



p

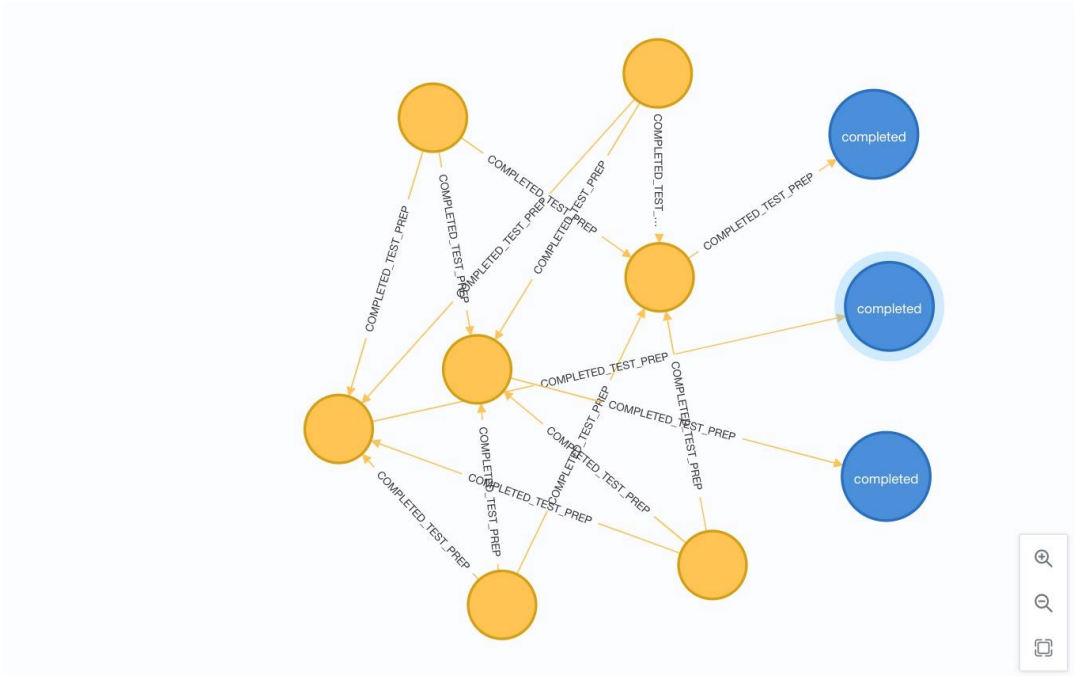

```
      },
      "elementId": "0"
    },
    "relationship": {
      "identity": 1512,
      "start": 0,
      "end": 919,
      "type": "SAME_LUNCH",
      "properties": {

      },
      "elementId": "1512",
      "startNodeElementId": "0",
      "endNodeElementId": "919"
    },
    "end": {
```


started streaming 25 records after 2 ms and completed after 10 ms

4) The relation between male and female students who have completed the test preparation course.

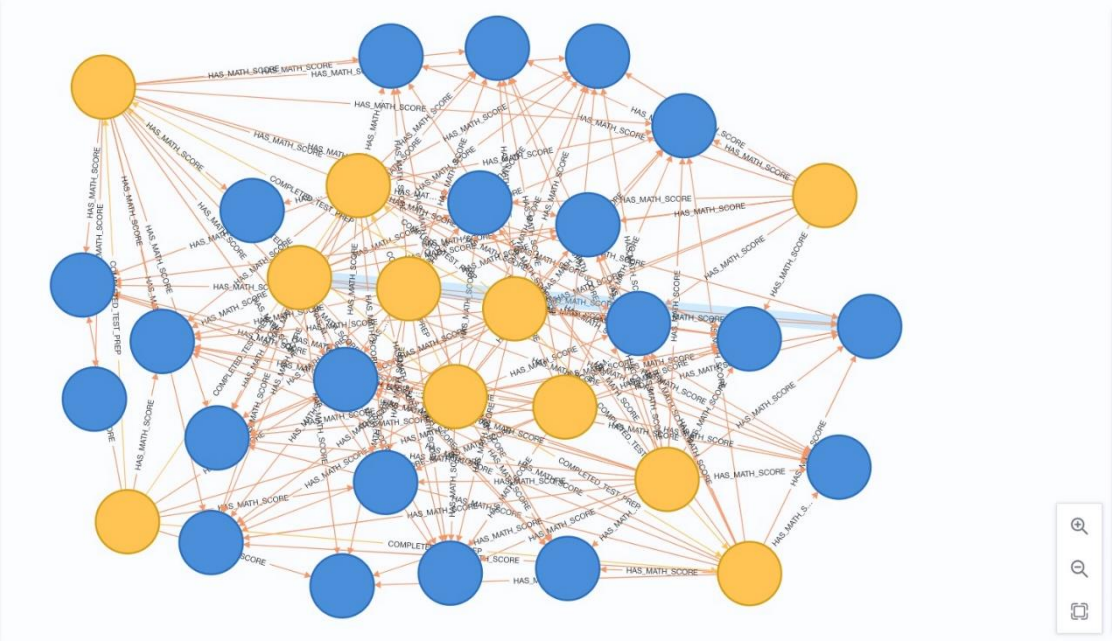
Here the relationship is created between students and the TestPrep which tells the students have completed the test preparation course and the relationship is denoted by completed as the relation.




```
p
    ],
    "properties": {
      "lunch": "standard",
      "writing_score": 88.0,
      "reading_score": 90.0,
      "gender": "female",
      "test_preparation_course": "completed",
      "math_score": 69.0,
      "parent_education": "some college"
    },
    "elementId": "1"
  },
  "end": {
    "identity": 132,
    "labels": [
      "Student"
```

rted streaming 25 records after 1092 ms and completed after 1113 ms.

5) **The count and relation between students that have highest math score in the whole class.**

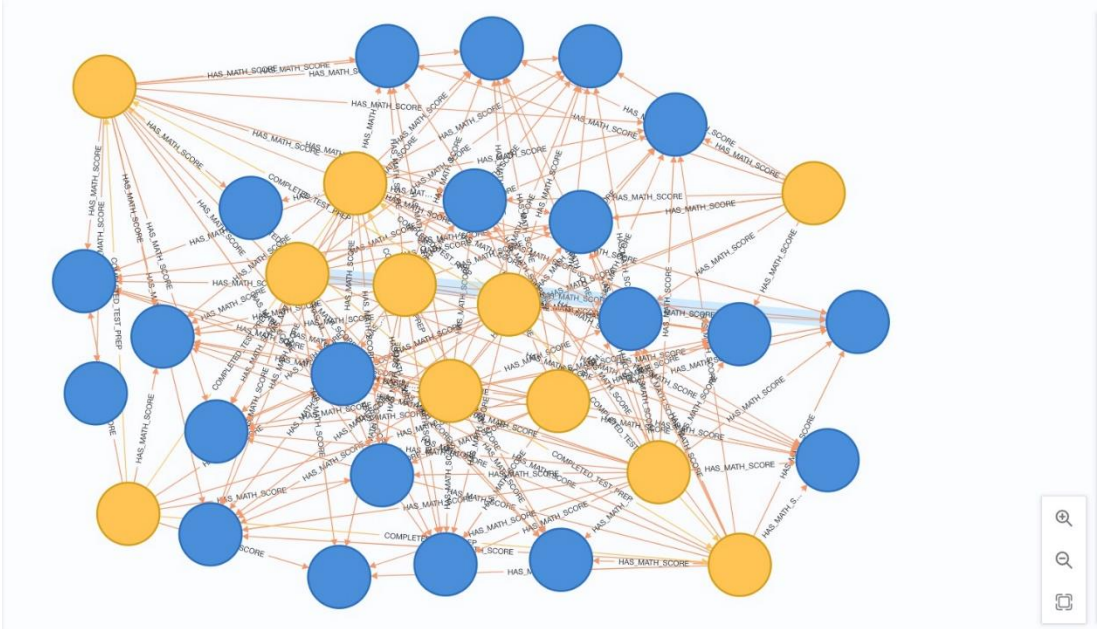


```
p
```


```
      "relationship": {
        "identity": 0,
        "start": 1000,
        "end": 1012,
        "type": "HAS_MATH_SCORE",
        "properties": {

        },
        "elementId": "0",
        "startNodeElementId": "1000",
        "endNodeElementId": "1012"
      },
      "end": {
        "identity": 1012,
        "labels": [
          "MathScore"
        ]
```

6) The count of students those who have high reading score.

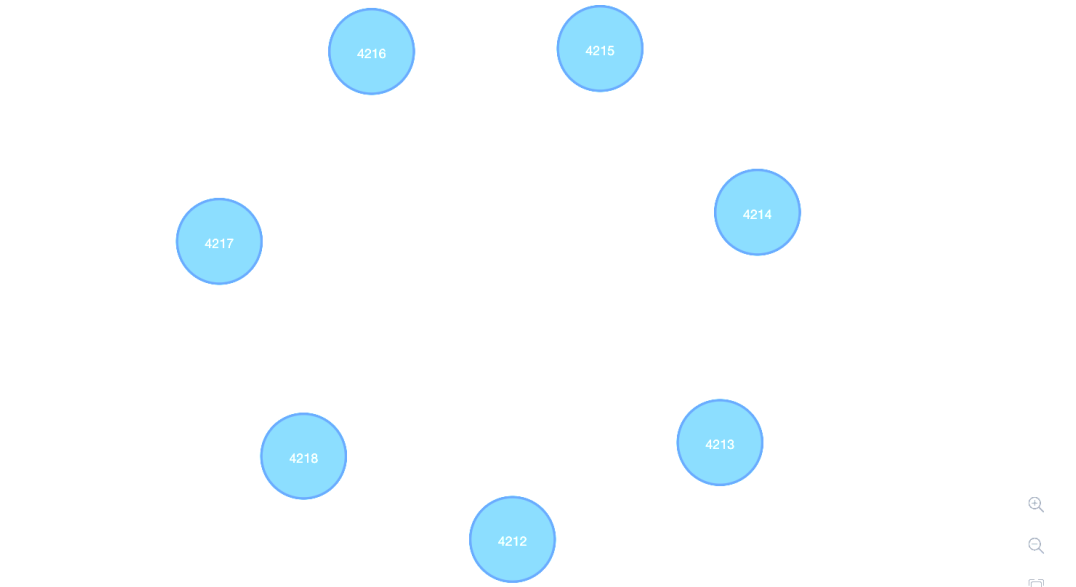


p


```
        "test_prep": "none"
      },
      "elementId": "1000"
    },
    "relationship": {
      "identity": 501,
      "start": 1000,
      "end": 1093,
      "type": "HAS_READING_SCORE",
      "properties": {

      },
      "elementId": "501",
      "startNodeElementId": "1000",
      "endNodeElementId": "1093"
    },
```

7) Count of students those who have the highest score as math score.



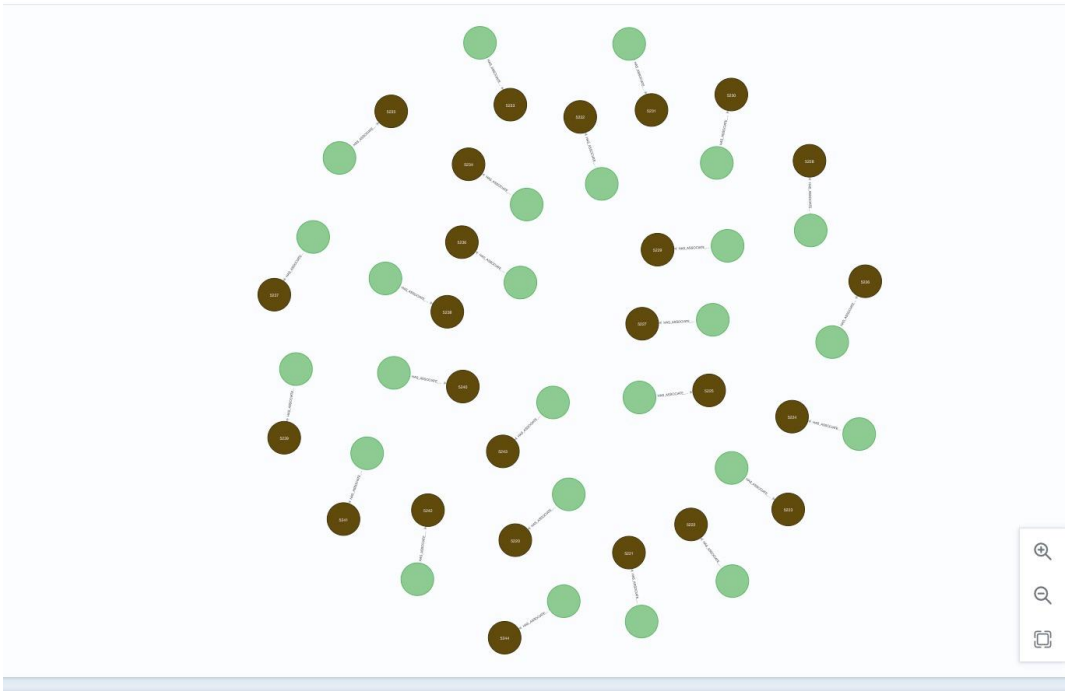
p


```
      "gender": "male",
      "test_preparation_course": "completed",
      "math_score": 100.0,
      "parent_education": "associate's degree"
    },
    "elementId": "149"
  },
  "end": {
    "identity": 4212,
    "labels": [
      "HighestMathScore"
    ],
    "properties": {

    },
    "elementId": "4212"
  },
```


8) A link between students whose parents hold associate's degrees in education.

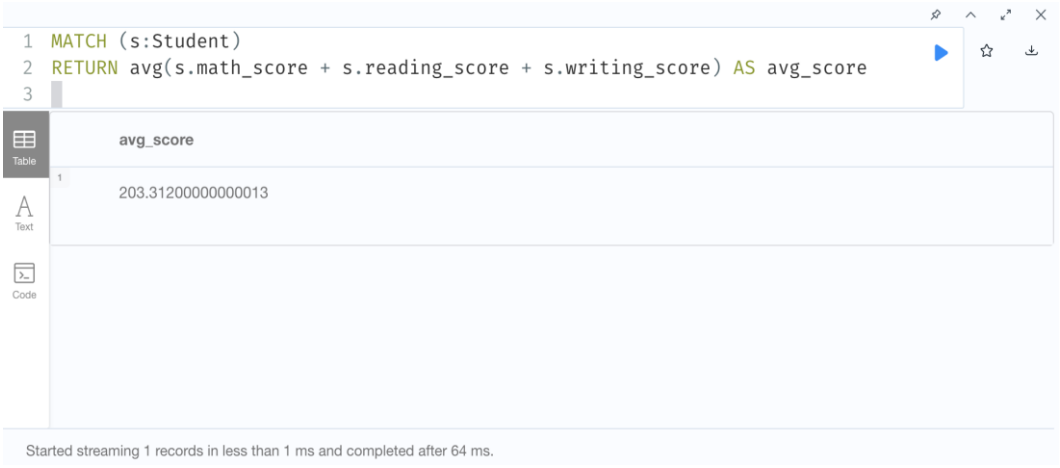
We begin by matching all pupils with "associate's degree" as their parental education. The students are then collected in a list called associate_students. Using the HAS_ASSOCIATE_EDUCATION connection, we unwind the list and merge each student with a node called AssociateEducation. This establishes a link between each student with a "associate's degree" and the node AssociateEducation.



- Performing basic cipher queries (e.g., aggregation, conditions, operators) and describe results.

1)

The average score of all students in the dataset is returned by this Cypher query. First, it uses the MATCH (s:Student) clause to match all nodes labeled as "Student." It then computes each student's average score by adding their math, reading, and writing scores and dividing the total by three. This is accomplished through the use of the avg() function, which computes the average of a set of variables. Dot notation is used to retrieve the scores; for example, s.math score accesses the math score attribute of the current student node.Finally, the query uses the RETURN clause to return the calculated average score, with the result named avg_score.



2)

This Cypher search returns the number of female students in the Student nodes of the network whose writing score is higher than their reading score.

Using the MATCH clause, it begins by matching all the nodes that have the label Student. Then, using the WHERE clause, it restricts the results to to the female students whose writing score is higher than their reading score.

Finally, using the RETURN clause and the count() function, it returns the total number of these students as females_high_writing_score.

```
// count the number of females who have high writing score than reading score
MATCH (s:Student)
WHERE s.gender = 'female' AND s.writing_score > s.reading_score
RETURN count(s) AS females_high_writing_score
```

	females_high_writing_score
1	217

3)

This Cypher command counts the number of pupils who are members of the race/ethnicity category C and then returns that figure.

4) By gender and test preparation status, this search returns the total number of students who have finished the test preparation course.

• The first line of the query defines a variable s to serve as a representation for all nodes that have the Student label.

• The variables to be retained for the following section of the query are listed in the WITH clause. Here, we preserve the gender and test_prep variables and use the COUNT() method to count the number of students who have the same gender and test_prep value.

• The IS NOT NULL condition is used in the WHERE clause to exclude any null entries from the test_prep variable. The RETURN clause also returns the count, gender, and test_prep variables for each set of students that satisfy the requirements.

```
MATCH (s:Student)
WITH s.gender AS gender, s.test_prep AS test_prep, COUNT(*) AS
count
WHERE test_prep IS NOT NULL
RETURN gender, test_prep, count
```

Table | Text | Code

	gender	test_prep	count
1	"female"	"none"	3
2	"female"	"completed"	3
3	"male"	"none"	2
4	"male"	"completed"	4

Started streaming 4 records in less than 1 ms and completed after 6 ms.

5)

The above query begins by matching all nodes with the Student label before grouping them with the WITH clause according to their race_ethnicity parameter. The COUNT aggregation function is then used to count the number of pupils in each racial/ethnic category and aliases the result as count. The WHERE clause then limits the results to racial/ethnic groupings whose values fall under group C or group D. The RETURN clause then uses the reduce function to add up the values in the list and alias it as total_count after using the collect function to build a list of the count values for each race and ethnicity category. The total number of pupils in groups C and D is the outcome.

```
MATCH (s:Student)
WITH s.race_ethnicity AS race, COUNT(*) AS count
WHERE race IN ['group C', 'group D']
RETURN reduce(total = 0, n IN collect(count) | total + n) AS total_count
```

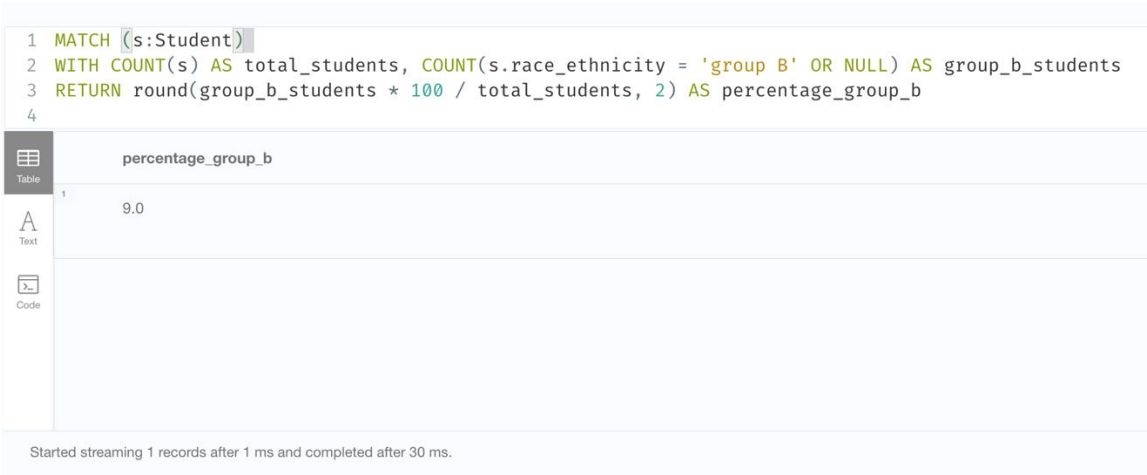
	total_count
1	581

arted streaming 1 records after 1 ms and completed after 23 ms.

6)

- The percentage of pupils in group B is determined using this query in the dataset's race/ethnicity field.

- It begins by matching all of the nodes with the label "Student." Then it uses COUNT(s) to count all of the pupils. Furthermore, it uses the formula COUNT(s.race_ethnicity = 'group B' OR NULL) to count the number of students that are members of that group.
- Because COUNT() does not count NULL values, the OR NULL is used here. Therefore, it returns NULL if there are no students in group B. When there are no students in group B, we ensure that the COUNT() function returns 0 by using OR NULL.
- The percentage of students in group B is then calculated by multiplying the number of students in group B by 100 and dividing that result by the total number of students. The result is rounded to two decimal places using the round() method.
- In general, this inquiry aids in understanding the distribution of students by race and ethnicity as well as the proportion of students in each group.



7)

- Finding all of the students whose parental_education attribute is "associate's degree" is done using the MATCH clause in this query. The count of these students is then determined using the WITH clause and is then assigned to the variable count_associate_degree.
- The MATCH clause is used once more in the next line to find every student, regardless of the parental_education attribute, and the variable total_students is given the count of every student.
- In the RETURN clause, the proportion of students whose parents have a "associate's degree" as their highest level of education is determined by dividing count_associate_degree by total_students and multiplying the result by 100.0.
- As a result, the query gives the proportion of all the pupils in the dataset whose parents have a "associate's degree" as their level of education.

```
MATCH (s:Student)
WHERE s.`parental_education` = "associate's degree"
WITH COUNT(s) AS count_associate_degree
MATCH (all:Student)
WITH count_associate_degree, COUNT(all) AS total_students
RETURN (count_associate_degree * 100.0 / total_students) AS percent_associate_degree
```

percent_associate_degree
11.033797216699801

Started streaming 1 records after 1 ms and completed after 7 ms.

8)

- The aim of this search is to determine the standard deviation of the test preparation course graduates' math, reading, and writing scores.
- To only include nodes that have successfully completed the exam preparation course, it begins by matching all nodes with the label "Student." It then computes the standard deviation for math scores using the stDev() function and assigns it to the variable math_stdev using the WITH clause. Like how it calculates and assigns the standard deviation for reading scores, it also computes and assigns the standard deviation for writing scores to writing_stdev. The three standard deviation values are returned at the end.

```
MATCH (s:Student)
WHERE s.test_preparation_course = 'completed'
WITH stDev(s.math_score) AS math_stdev,
     stDev(s.reading_score) AS reading_stdev,
     stDev(s.writing_score) AS writing_stdev
RETURN math_stdev, reading_stdev, writing_stdev
```

math_stdev	reading_stdev	writing_stdev
14.434594502762952	13.62884297306514	13.365978189919181

Started streaming 1 records in less than 1 ms and completed after 9 ms.