Database Management System

Lab Assignment

Course Name : Database Management System lab

Course Code : CSEL-2204

Submitted to-

Roksana Khanom
Assisant Professor
Department of CSE, JnU

Submitted by-

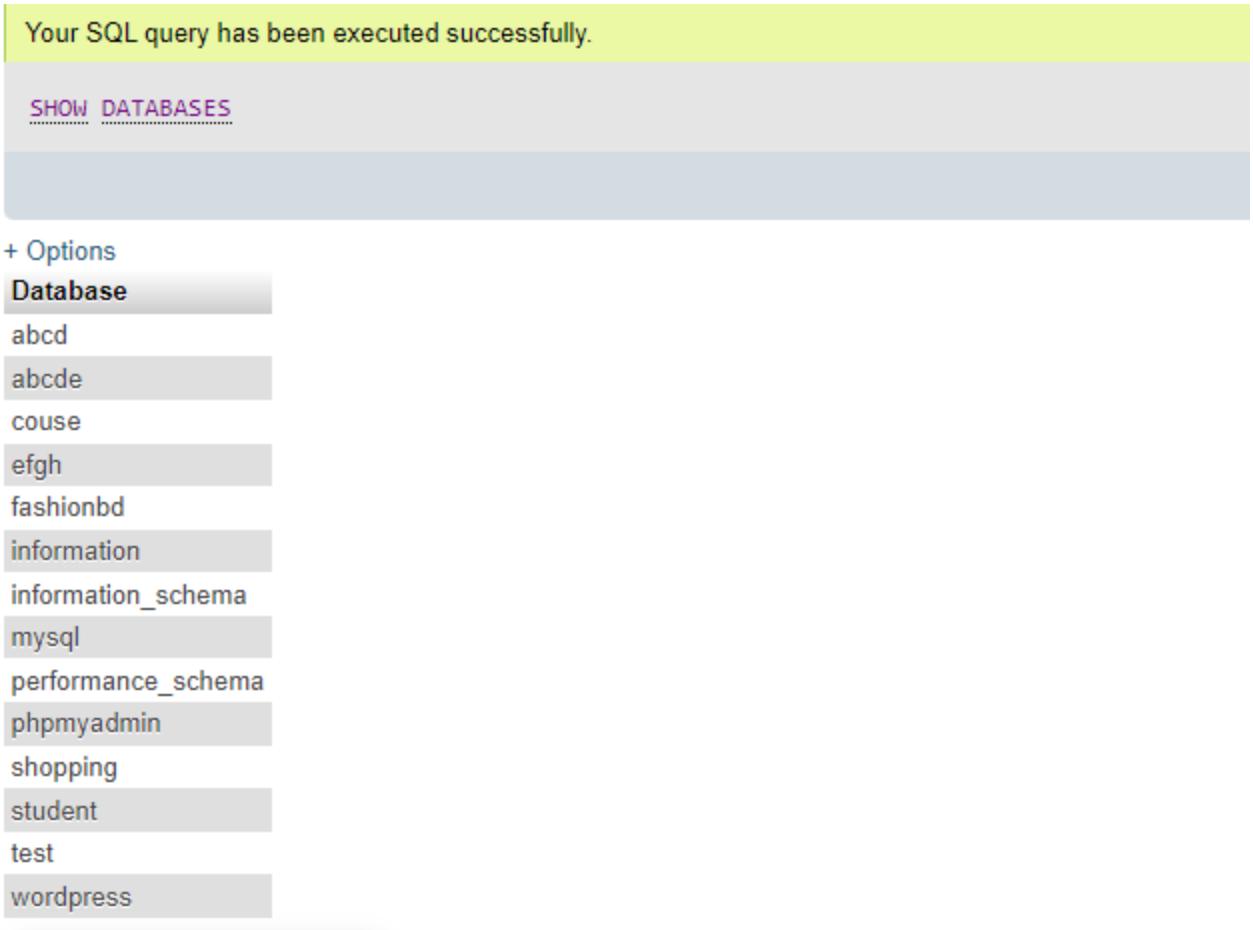
Susmita Rani Saha
Id- B18030547
Computer science and engineering

1. SQL CREATE DATABASE statement :

The CREATE DATABASE statement is used to create a new SQL database.For example,

```
CREATE DATABASE abcd;
```

This code will create a database named 'abcd' in localhost and we can see the database using 'show databases' query.

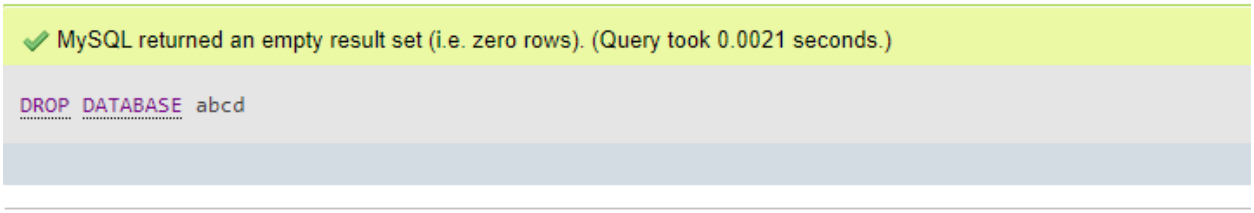


2. SQL DROP DATABASE statement :

The DROP DATABASE statement is used to drop SQL database.For example,

```
DROP DATABASE abcd;
```

This code will drop a database named 'abcd' in localhost.

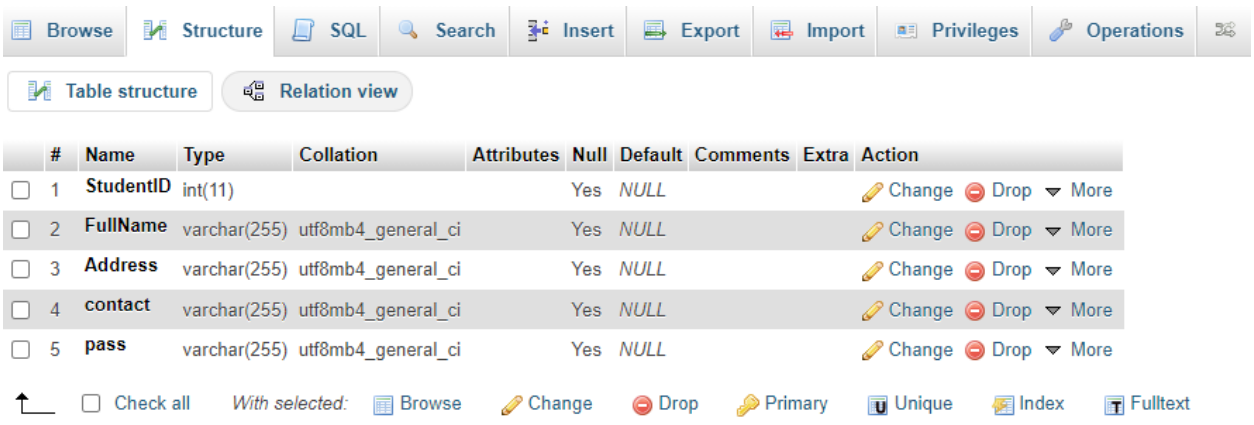


3. SQL CREATE TABLE statement :

The following example creates a table called "student" that contains five columns: StudentID, FullName, Address, Contact and pass. For example,

```
CREATE TABLE student (
    StudentID int,
    FullName varchar(255),
    Address varchar(255),
    contact varchar(255),
    pass varchar(255)
);
```

This code will create a table named ‘student’ in database named ‘abcd’ in localhost.

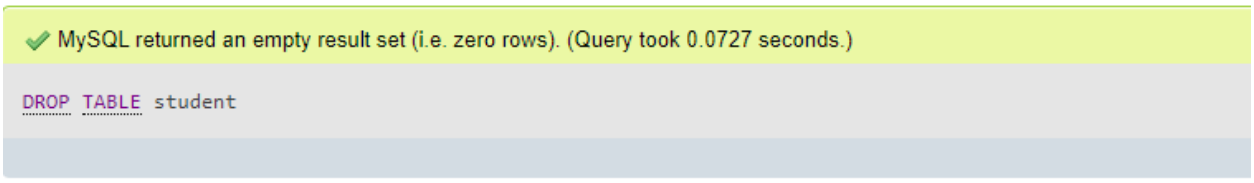


4. SQL DROP TABLE statement :

The DROP TABLE statement is used to drop an existing table in a database. For example,

```
DROP TABLE student;
```

This code will drop a table named 'student' in database named 'abcd' in localhost.



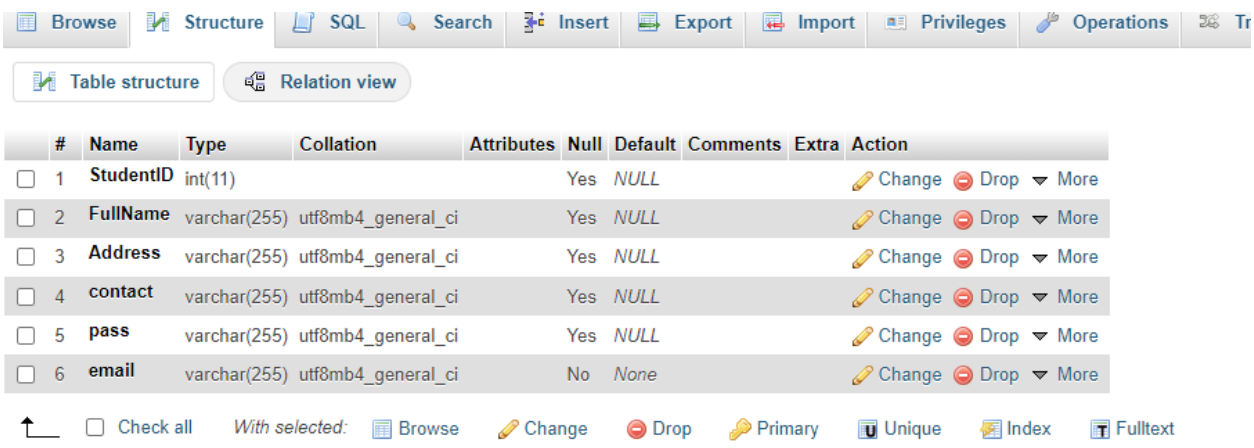
5. SQL ALTER TABLE – ADD COLUMN :

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

The ALTER TABLE-ADD COULMN statement is used to add a column in a table in a database. For example,

```
ALTER TABLE student
ADD email varchar(255);
```

This code will add a column named 'email' in table named 'student' in database named 'abcd' in localhost.

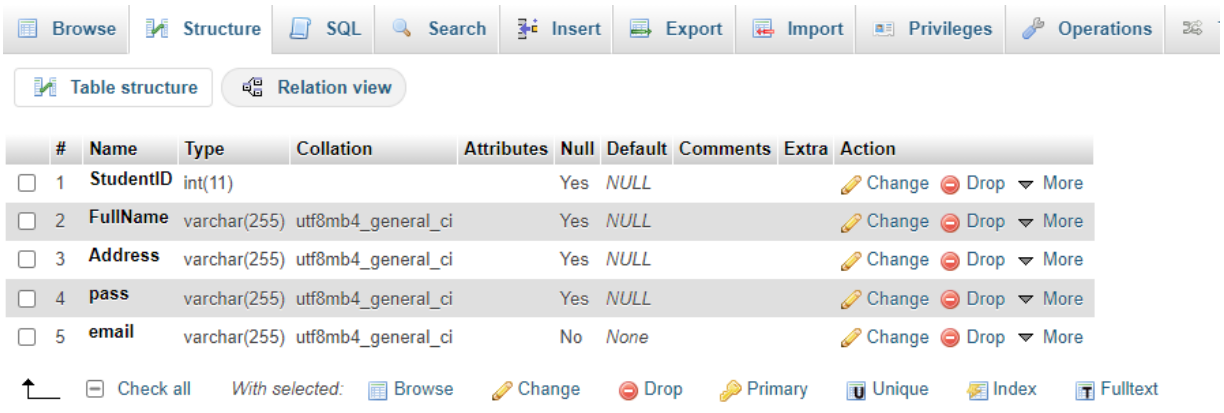


6. SQL ALTER TABLE – DROP COLUMN :

The ALTER TABLE-DROP COULMN statement is used to drop a column in a table in a database. For example,

```
ALTER TABLE student
DROP COLUMN contact;
```

This code will drop a column named 'contact' in table named 'student' in database named 'abcd' in localhost.

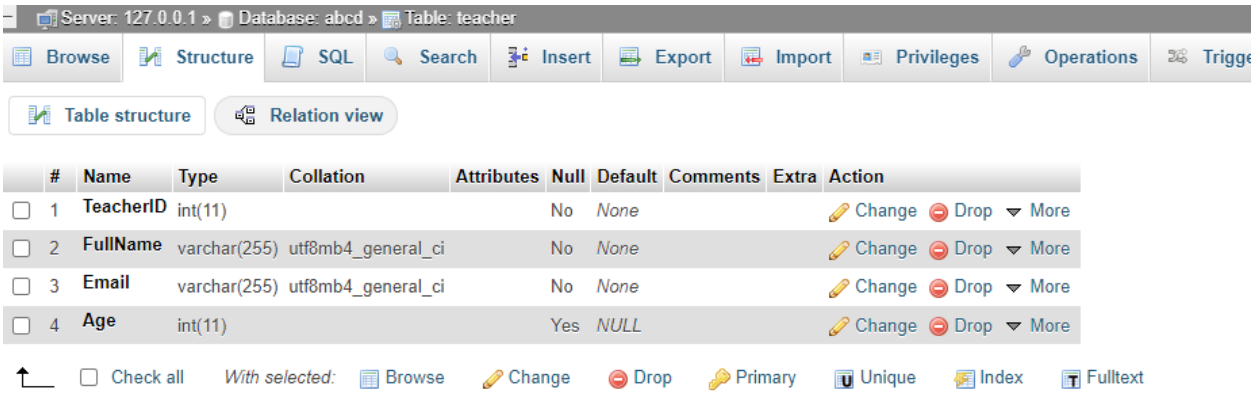


7. NOT NULL :

By default, a column can hold NULL values.The NOT NULL constraint enforces a column to NOT accept NULL values.This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL ensures that the "TeacherID", "FullName", and "email" columns will NOT accept NULL values when the "Teacher" table is created: For example,

```
CREATE TABLE Teacher (
    TeacherID int NOT NULL,
    FullName varchar(255) NOT NULL,
    Email varchar(255) NOT NULL,
    Age int
);
```

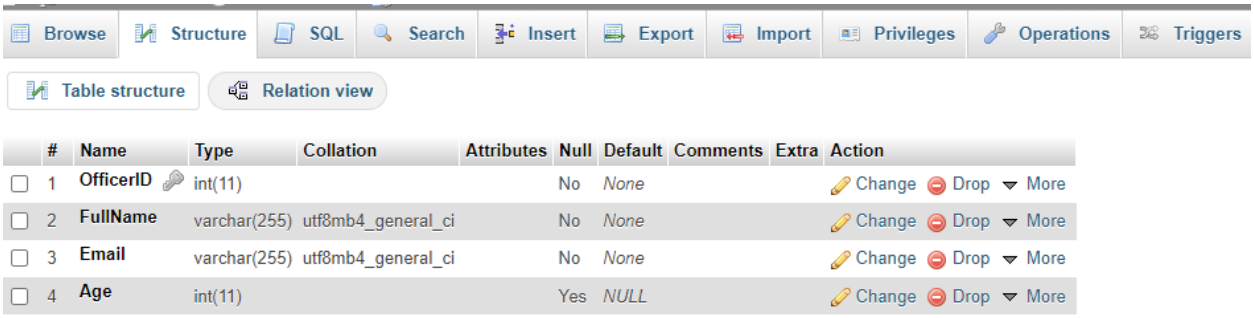


8. UNIQUE:

The UNIQUE constraint ensures that all values in a column are different.Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.A PRIMARY KEY constraint automatically has a UNIQUE constraint.However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

The following SQL creates a UNIQUE constraint on the "OfficerID" column when the "officer" table is created. For example,

```
CREATE TABLE Officer (
    OfficerID int NOT NULL UNIQUE,
    FullName varchar(255) NOT NULL,
    Email varchar(255) NOT NULL,
    Age int
);
```

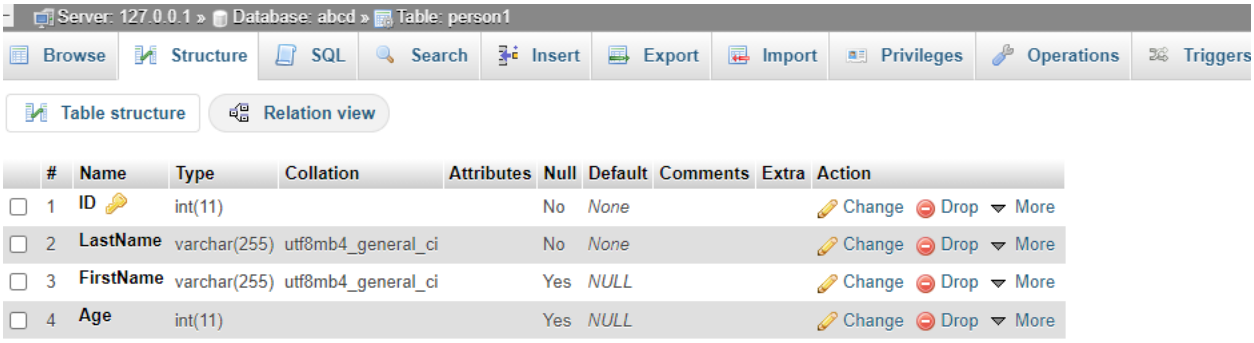


9.PRIMARY KEY ：

The PRIMARY KEY constraint uniquely identifies each record in a table.Primary keys must contain UNIQUE values, and cannot contain NULL values.A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

The following SQL creates a PRIMARY KEY on the "ID" column when the "Person1" table is created. For example:

```
CREATE TABLE Person1 (
                ID int NOT NULL,
                LastName varchar(255) NOT NULL,
                FirstName varchar(255),
                Age int,
                PRIMARY KEY (ID)
);
```

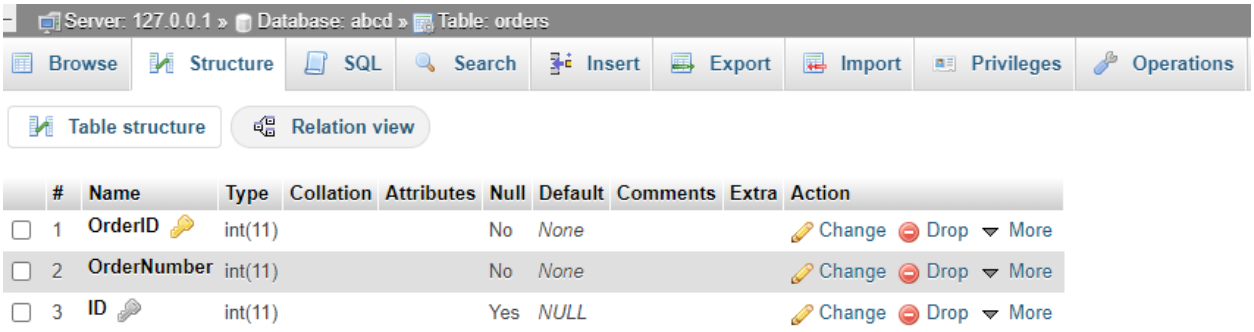


10. FOREIGN KEY:

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

The following SQL creates a FOREIGN KEY on the "ID" column when the "Orders" table is created.For example :

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    ID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (ID) REFERENCES Persons(ID)
);
```

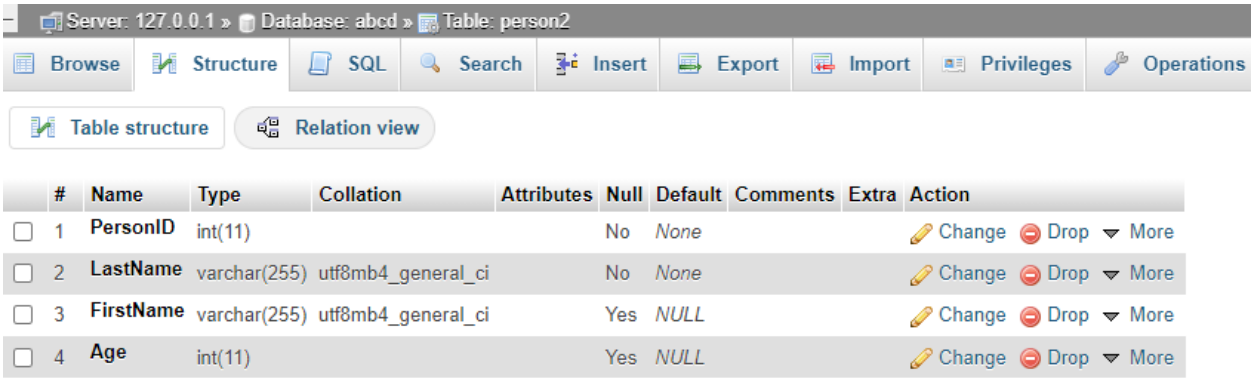


11. CHECK :

The CHECK constraint is used to limit the value range that can be placed in a column.If you define a CHECK constraint on a column it will allow only certain values for this column.If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

The following SQL creates a CHECK constraint on the "Age" column when the "Person2" table is created. The CHECK constraint ensures that the age of a person must be 18, or older.For example,

```
CREATE TABLE Person2 (
    PersonID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

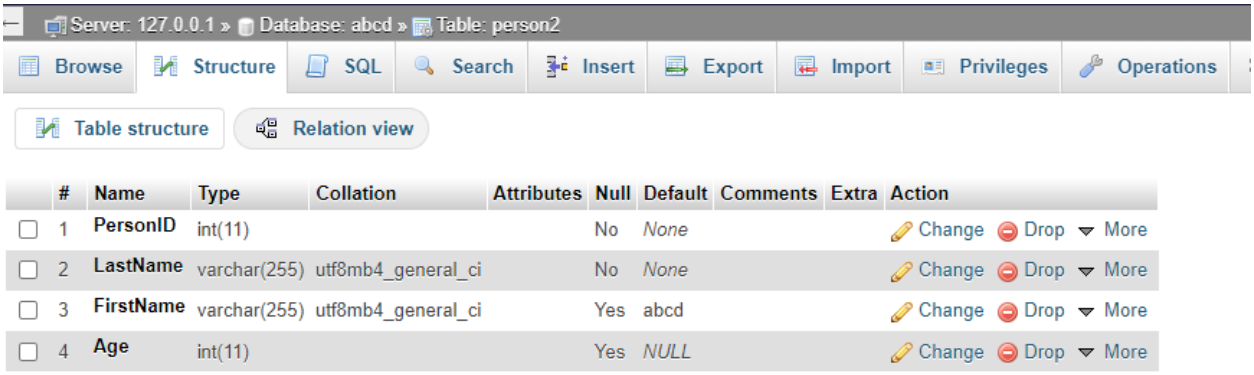


12. DEFAULT:

The DEFAULT constraint is used to set a default value for a column.The default value will be added to all new records, if no other value is specified.

To create a DEFAULT constraint on the "City" column when the table is already created. For example,

```
ALTER TABLE Person2
ALTER FirstName SET DEFAULT 'abcd';
```

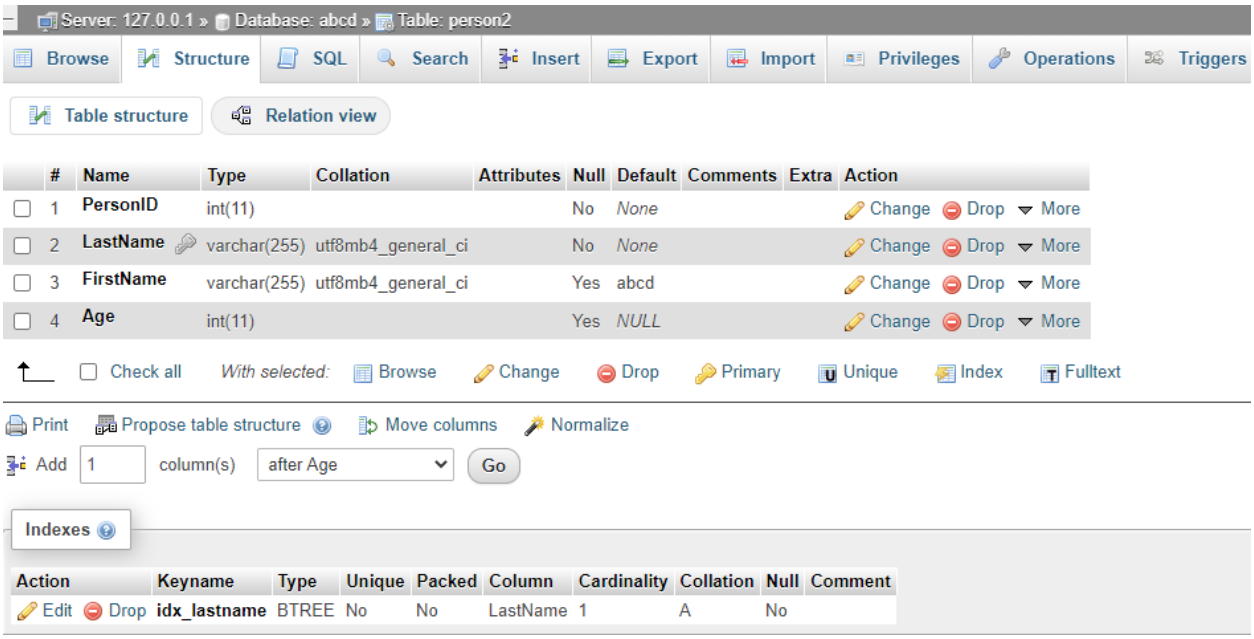


13. CREATE INDEX :

The CREATE INDEX statement is used to create indexes in tables.Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Person2" table. For example,

```
CREATE INDEX  idx_lastname
ON Person2 (LastName);
```



14. AUTO INCREMENT:

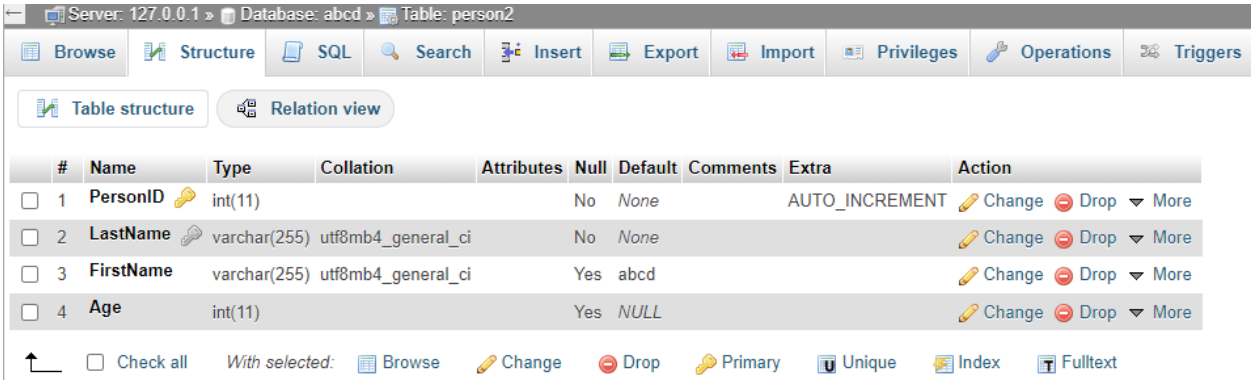
Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

The following SQL statement defines the "PersonId" column to be an auto-increment primary key field in the "Person2" table.

```
ALTER TABLE Person2

    CHANGE PersonId
    PersonId int(11) AUTO_INCREMENT;
```

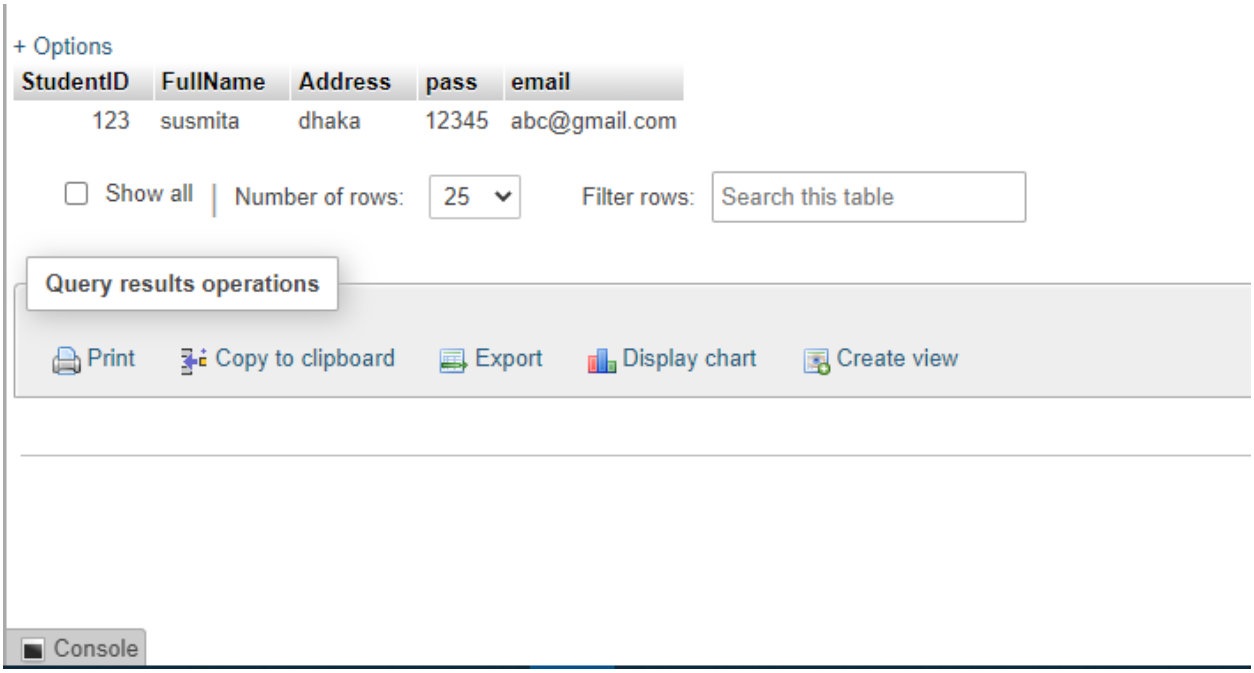


15. INSERT INTO:

The INSERT INTO statement is used to insert new records in a table.

The following SQL statement inserts a new record in the "student" table:

```
INSERT INTO `student`(`StudentID`, `FullName`, `Address`, `pass`, `email`)
VALUES (123,'susmita','dhaka','12345','abc@gmail.com')
```

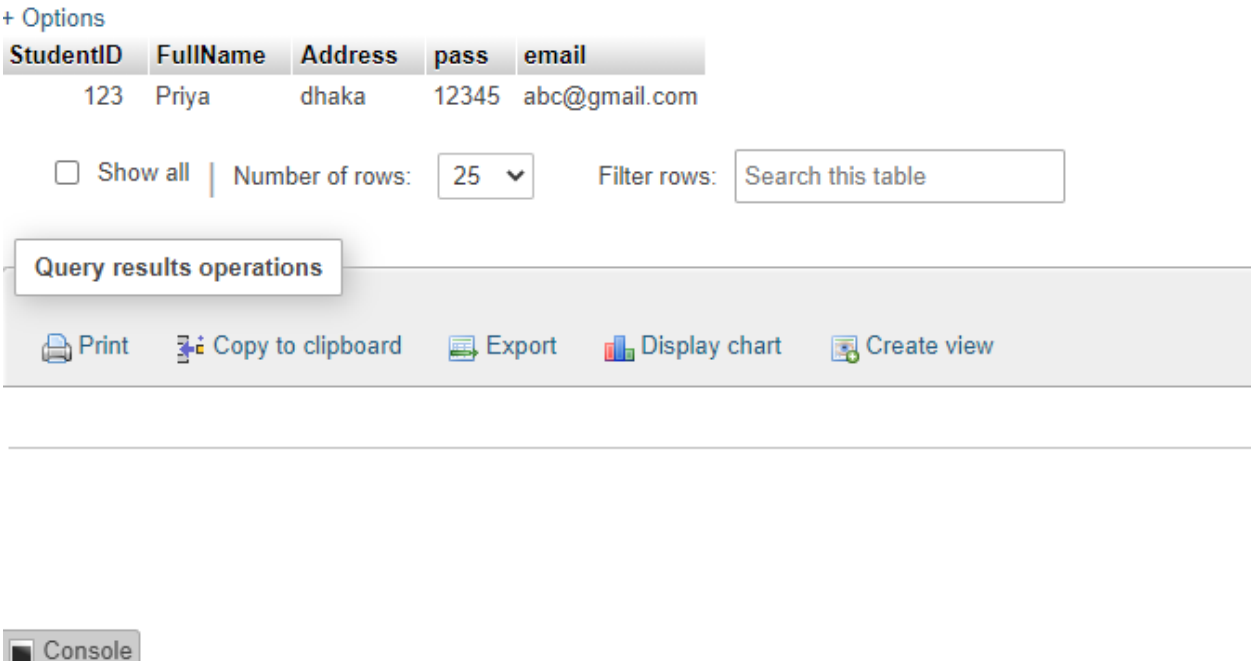


16. UPDATE :

The UPDATE statement is used to modify the existing records in a table.

The following SQL statement updates the first student (StudentID = 123) with a new name:

```
UPDATE student
SET FullName= 'Priya'
WHERE StudentID = 123;
```

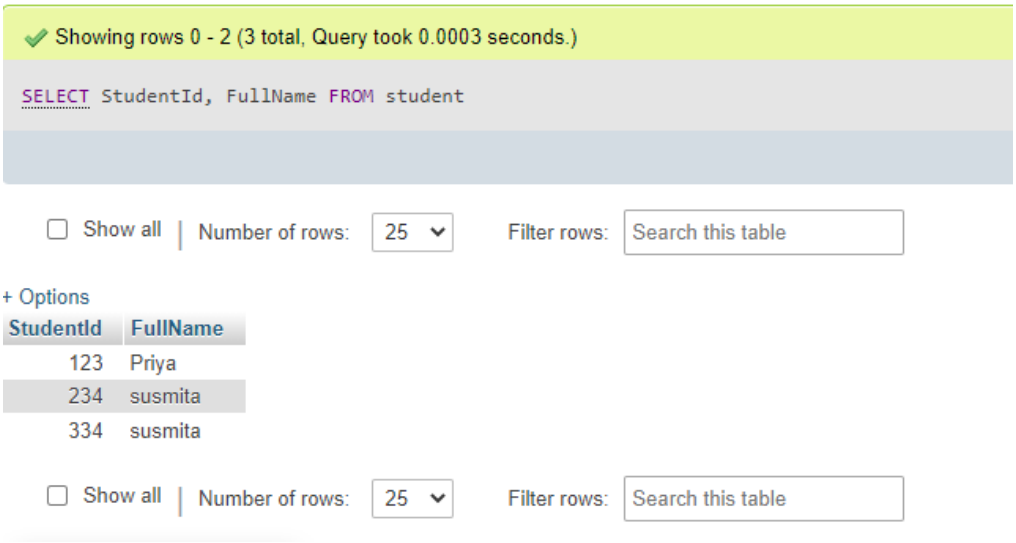


17. SELECT :

The SELECT statement is used to select data from a database.The data returned is stored in a result table, called the result-set.

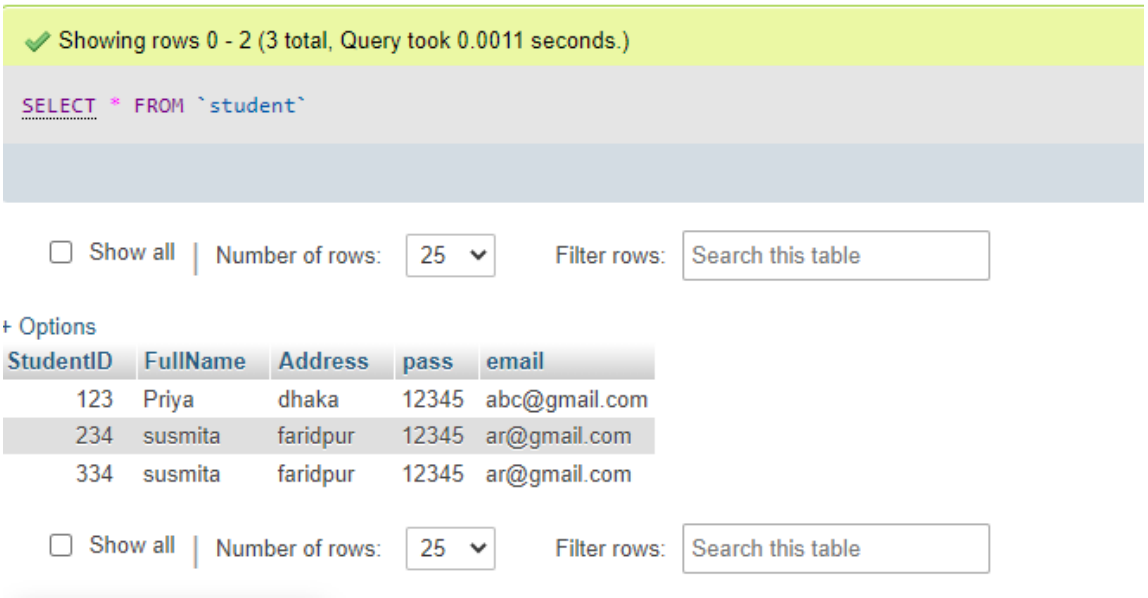
The following SQL statement selects the "StudentId" and "FullName" columns from the "student" table:

```
SELECT StudentId, FullName FROM student;
```



The following SQL statement selects all the columns from the "student" table:

```
SELECT * FROM student;
```

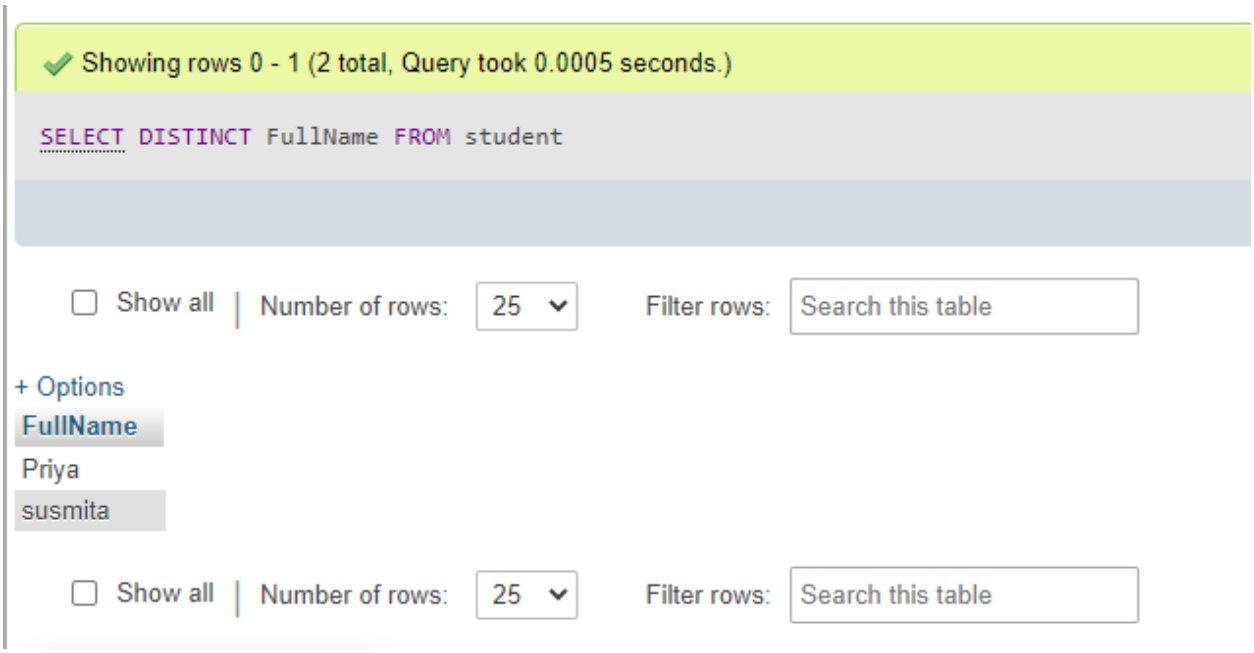


18. SELECT DISTINCT :

The SELECT DISTINCT statement is used to return only distinct (different) values.Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

The following SQL statement selects only the DISTINCT values from the "FullName" column in the "student" table:

```
SELECT DISTINCT FullName FROM student;
```

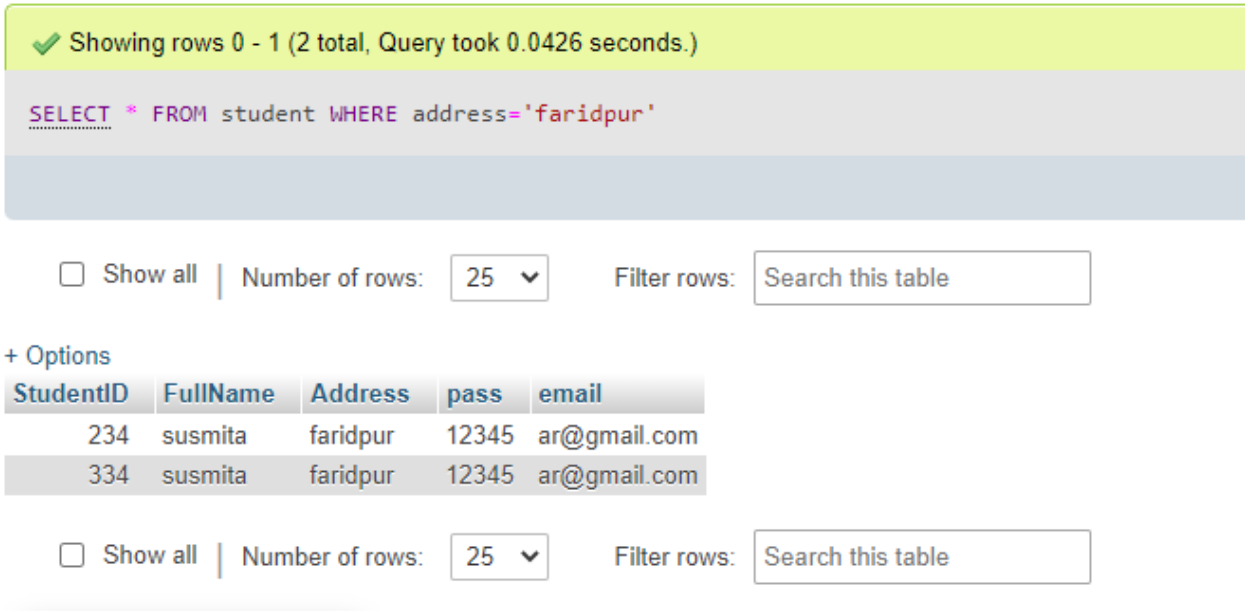


19. WHERE :

The WHERE clause is used to filter records.It is used to extract only those records that fulfill a specified condition.

The following SQL statement selects all the student from "faridpur", in the "student" table:

```
SELECT * FROM student
WHERE address='faridpur';
```



20. AND, OR and NOT:

The WHERE clause can be combined with AND, OR,
and NOT operators.The AND and OR operators are used to filter records based on more than one condition:

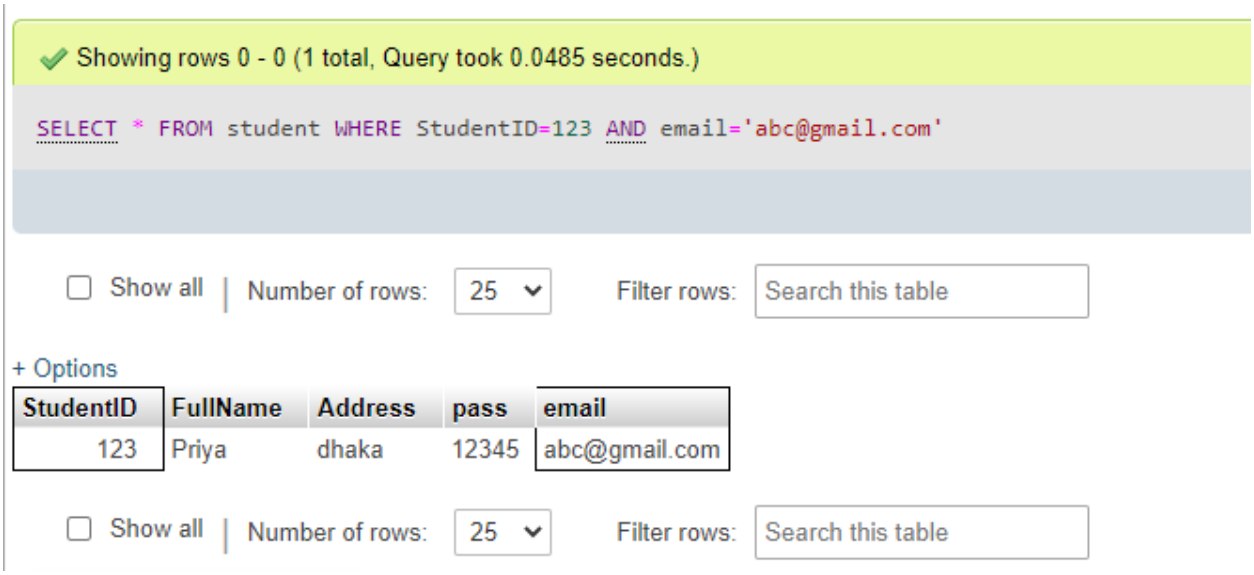
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Example

The following SQL statement selects all fields from "student" where StudentID is "123" AND email is "abc@gmail.com":

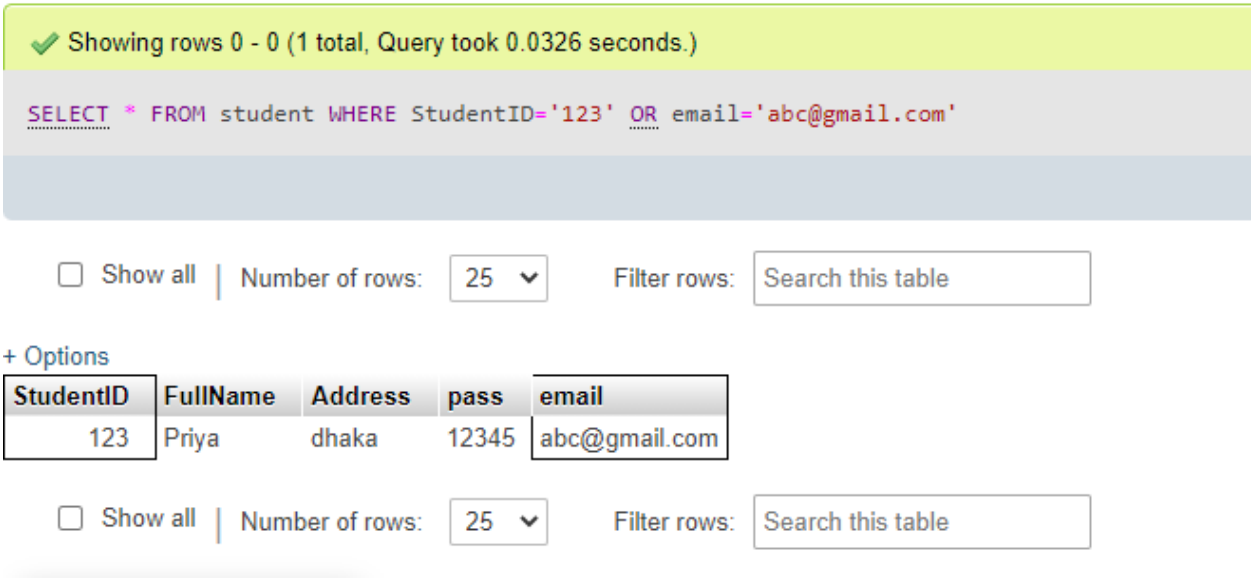
```
SELECT * FROM student
WHERE StudentID='123' AND email='abc@gmail.com';
```



OR Example

The following SQL statement selects all fields from "student" where StudentID is "123" OR email is "abc@gmail.com":

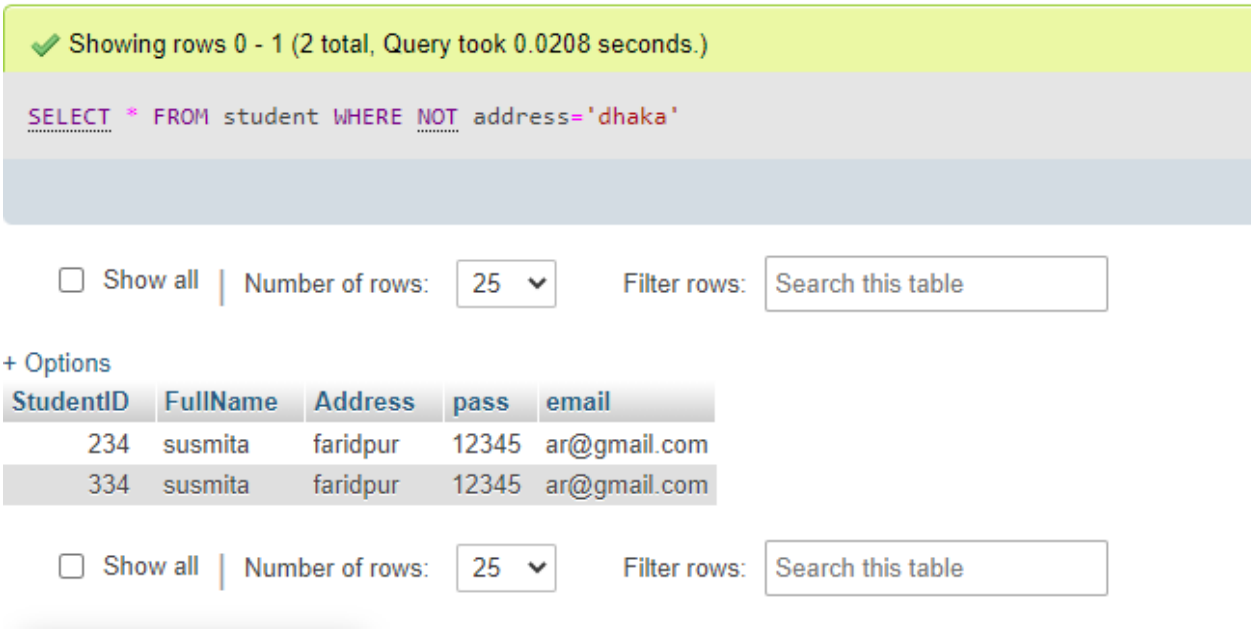
```
SELECT * FROM student
WHERE StudentID='123' OR email='abc@gmail.com';
```



NOT Example

The following SQL statement selects all fields from "student" where address is NOT "dhaka":

```
SELECT * FROM student
  WHERE NOT address='dhaka';
```

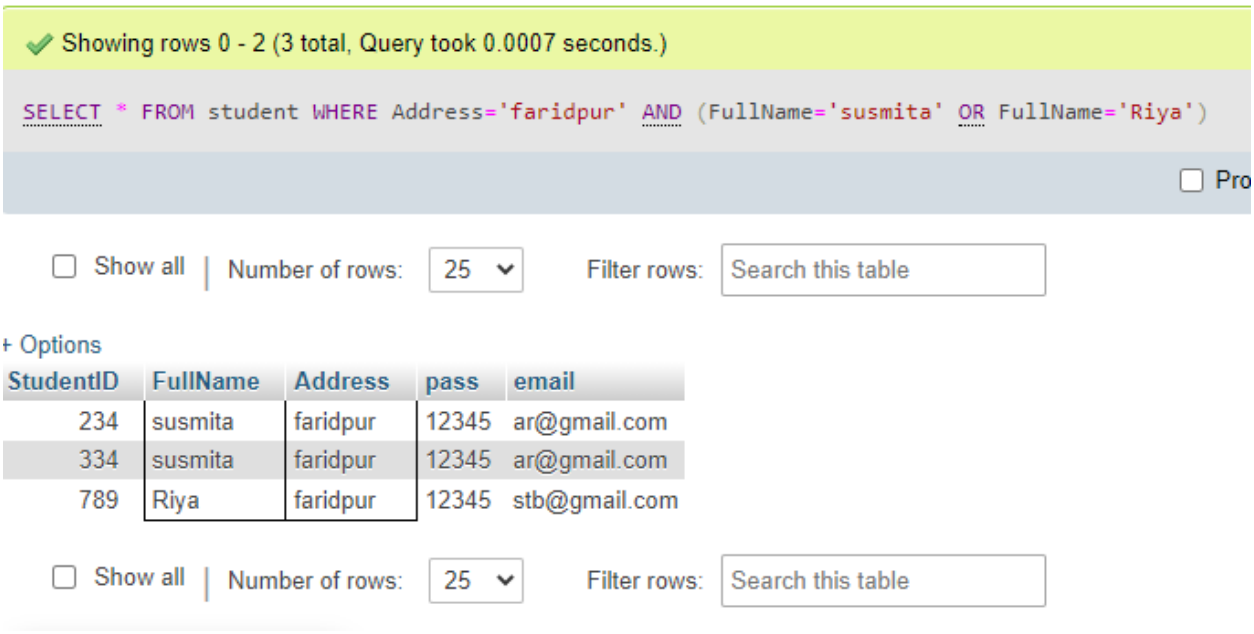


Combining AND, OR and NOT :

You can also combine the AND, OR and NOT operators.

The following SQL statement selects all fields from "student" where Address is "faridpur" AND FullName must be "susmita" OR "riya" (use parenthesis to form complex expressions) :

```
SELECT * FROM student
WHERE Address='faridpur' AND (FullName='susmita' OR FullName='Riya');
```

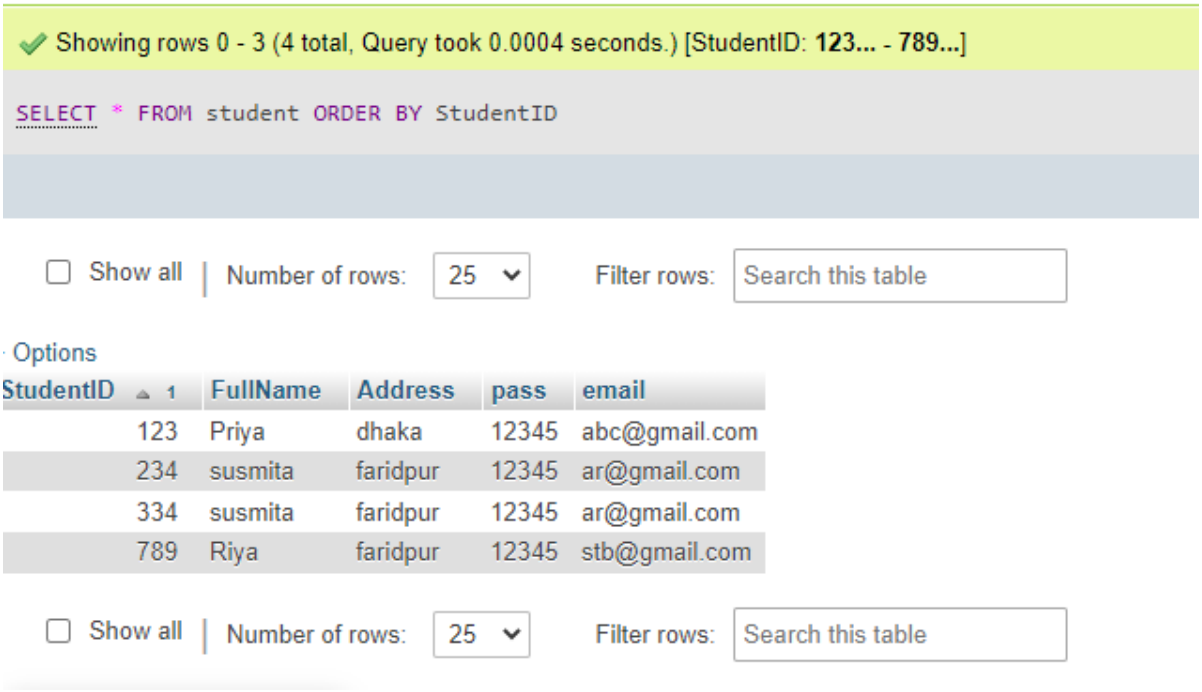


21. ORDER BY :

The ORDER BY keyword is used to sort the result-set in ascending or descending order.The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

The following SQL statement selects all students from the "student" table, sorted by the "StudentID" column:

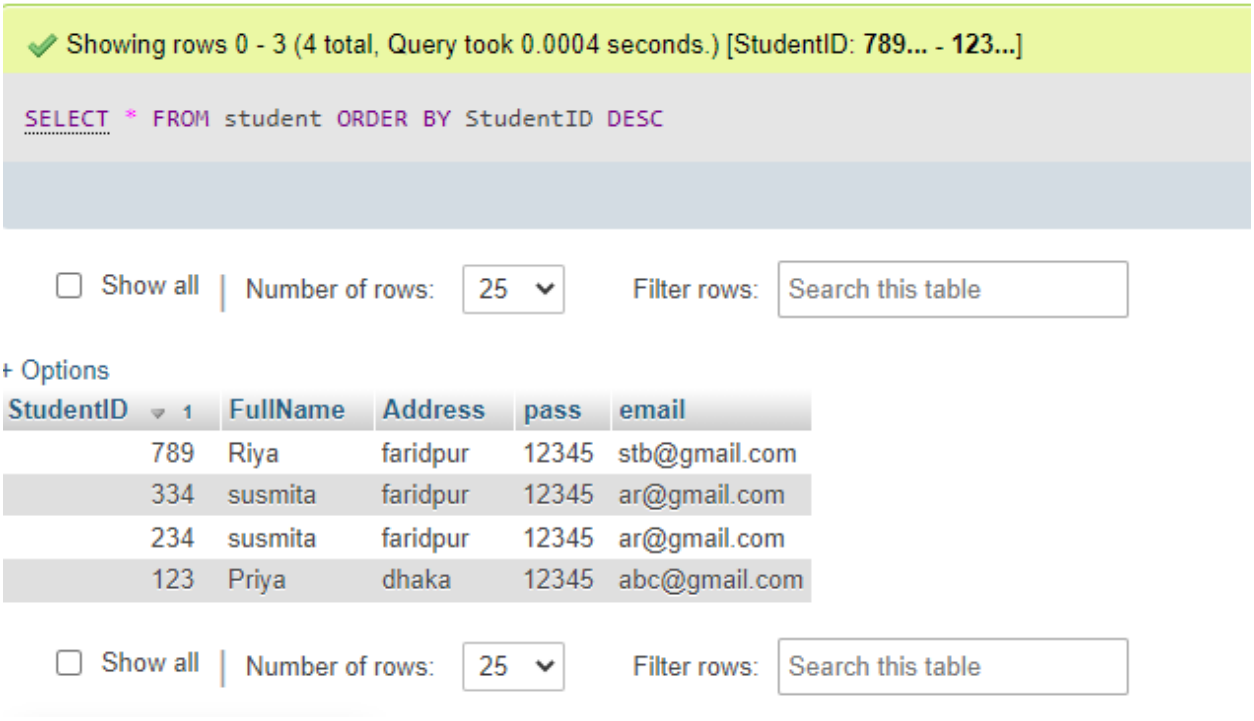
```
SELECT * FROM student
ORDER BY StudentID;
```



ORDER BY DESC Example :

The following SQL statement selects all students from the "student" table, sorted DESCENDING by the "StudentID" column:

```
SELECT * FROM student
 ORDER BY StudentID DESC;
```



Showing rows 0 - 3 (4 total, Query took 0.0004 seconds.) [StudentID: 789... - 123...]

```
SELECT * FROM student ORDER BY StudentID DESC
```

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

StudentID	FullName	Address	pass	email
789	Riya	faridpur	12345	stb@gmail.com
334	susmita	faridpur	12345	ar@gmail.com
234	susmita	faridpur	12345	ar@gmail.com
123	Priya	dhaka	12345	abc@gmail.com

Show all | Number of rows: 25 Filter rows: Search this table

22. NULL Values :

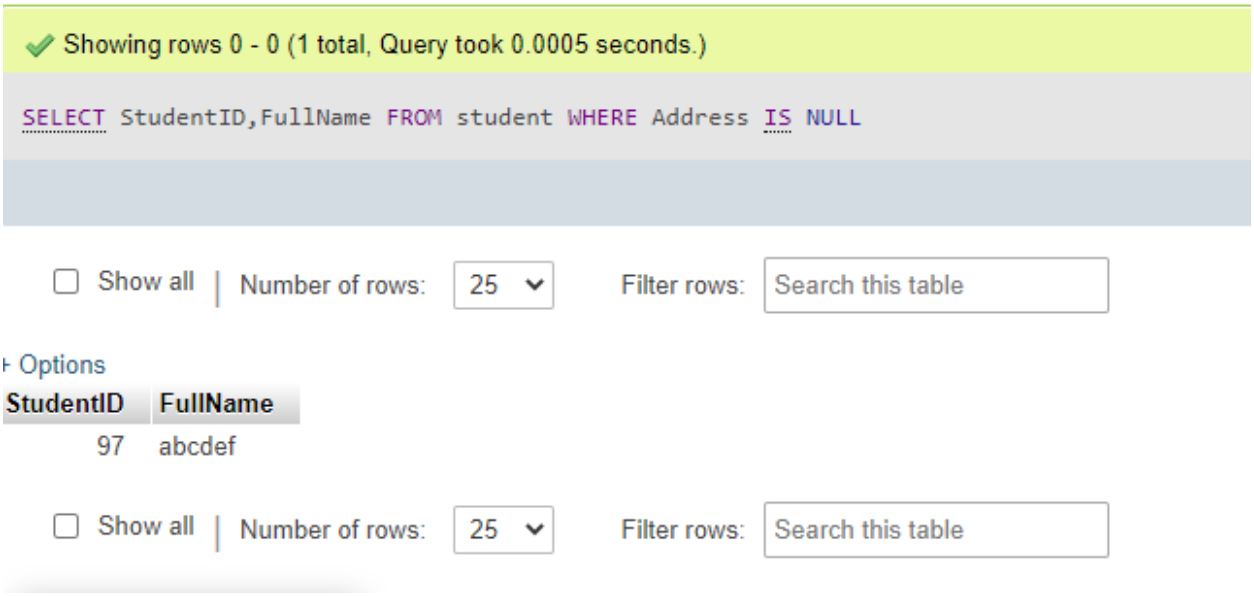
A field with a NULL value is a field with no value.If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

The IS NULL operator is used to test for empty values (NULL values).

The following SQL lists all students with a NULL value in the "Address" field:

```
SELECT StudentID,FullName FROM student
WHERE Address IS NULL;
```

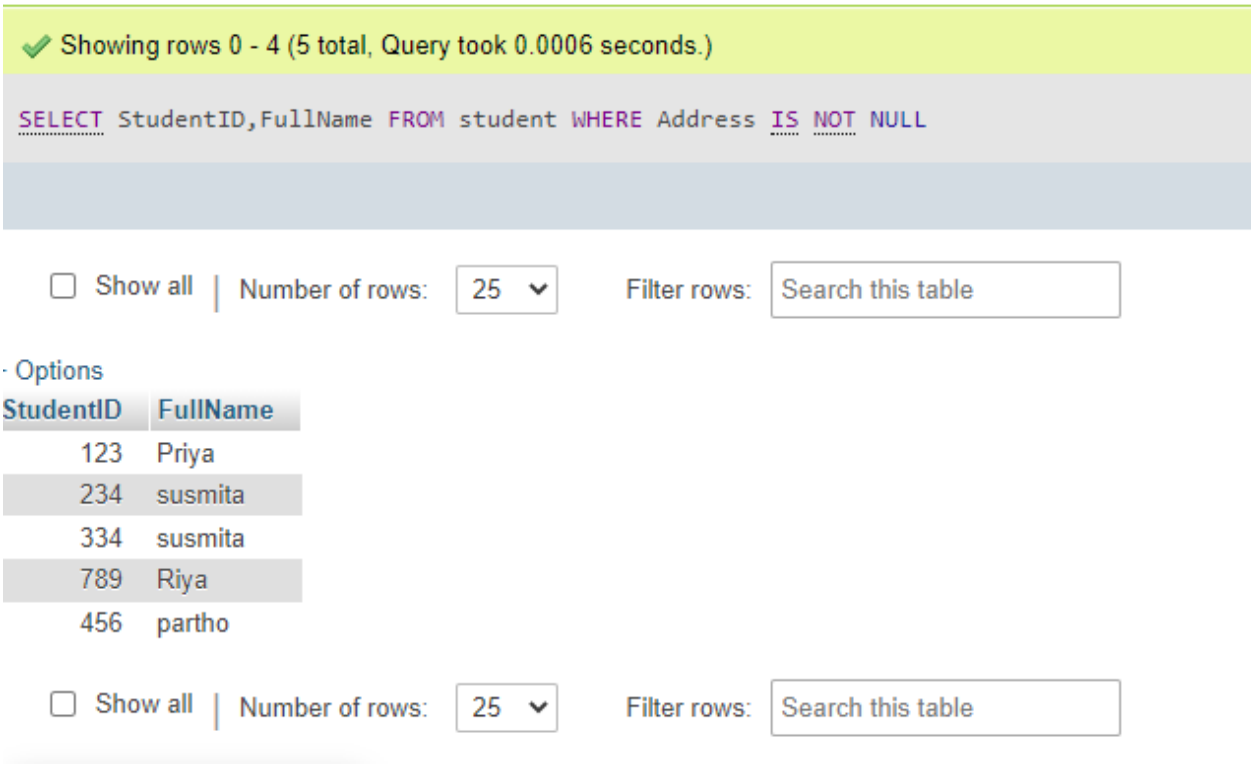


The IS NOT NULL Operator

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).

The following SQL lists all students with a value in the "Address" field:

```
SELECT StudentID,FullName FROM student
WHERE Address IS NOT NULL;
```

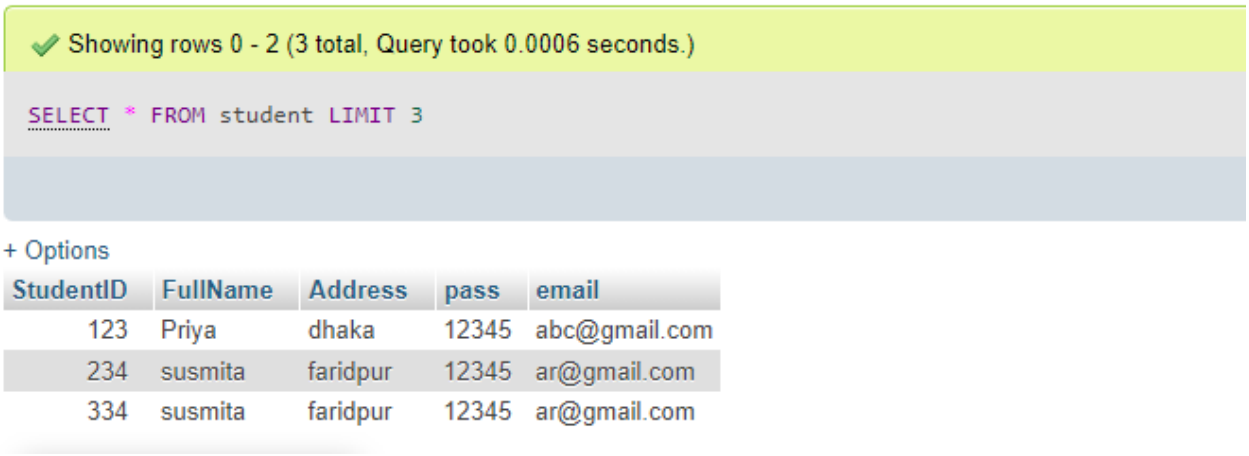


23. SELECT TOP(Limit) :

The SELECT TOP clause is used to specify the number of records to return.The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

The following SQL statement selects the first three records from the "student" table :

```
SELECT * FROM student
LIMIT 3;
```



StudentID	FullName	Address	pass	email
123	Priya	dhaka	12345	abc@gmail.com
234	susmita	faridpur	12345	ar@gmail.com
334	susmita	faridpur	12345	ar@gmail.com

24. MIN() and MAX() Functions :

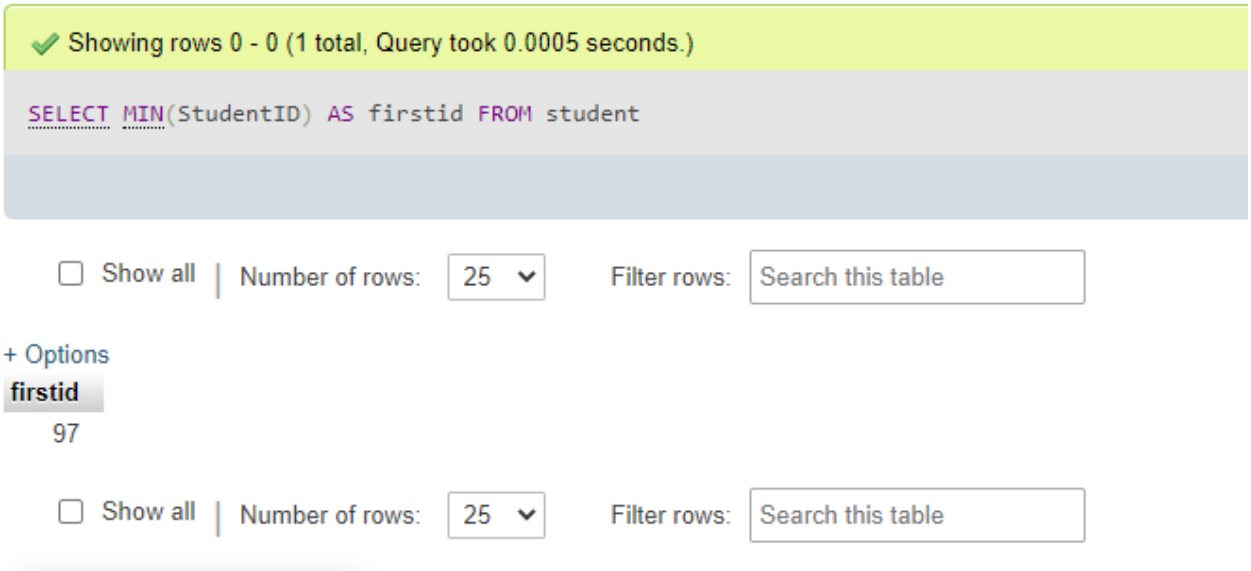
The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Example :

The following SQL statement finds the minimum value of StudentID:

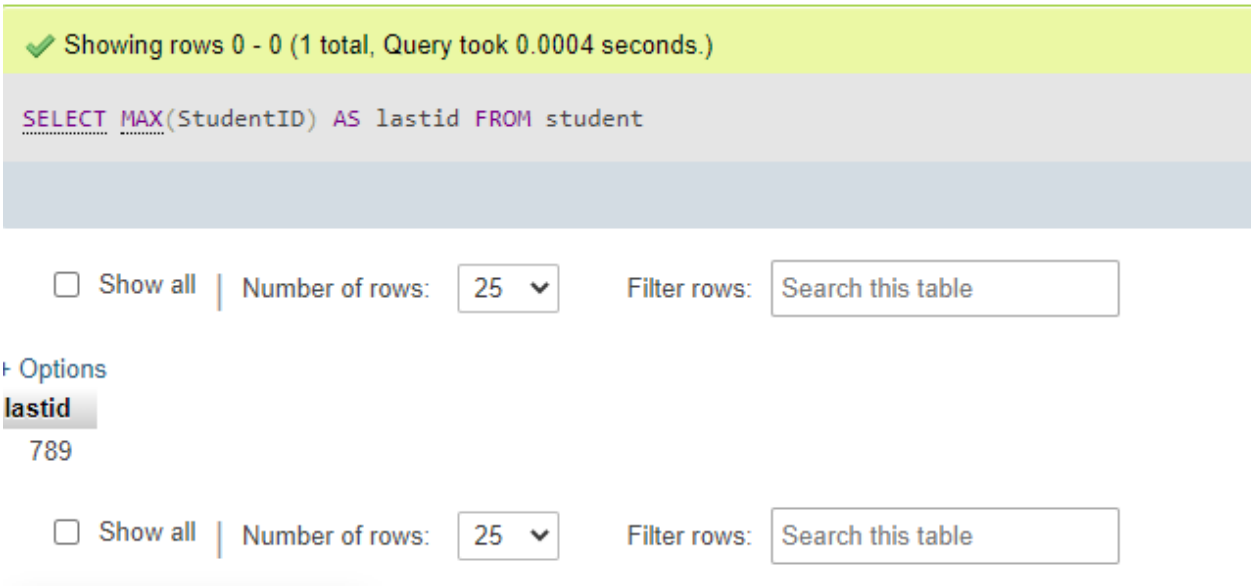
```
SELECT MIN(StudentID) AS firstid
FROM student;
```



MAX() Example

The following SQL statement finds the maximum value of StudentID:

```
SELECT MAX(StudentID) AS lastid
FROM student;
```



25. COUNT(), AVG() and SUM() :

The COUNT() function returns the number of rows that matches a specified criterion.

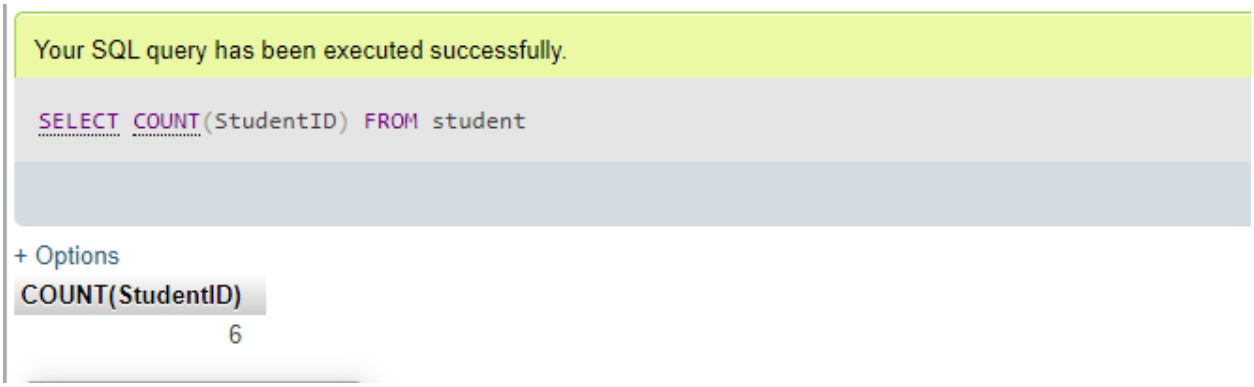
The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.

COUNT() Example

The following SQL statement finds the number of students:

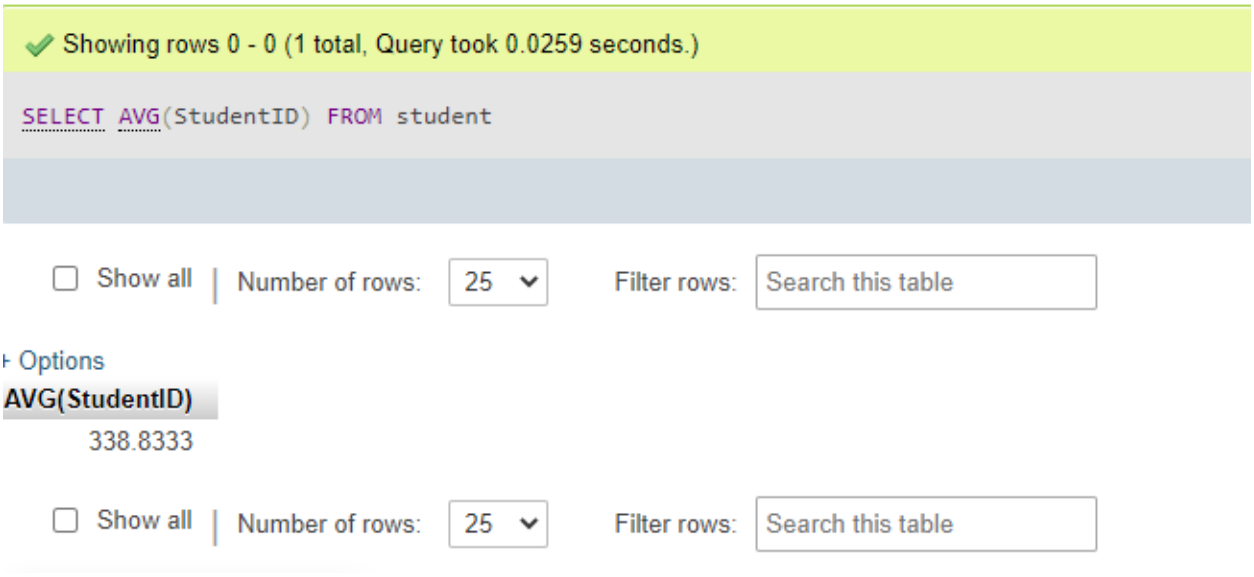
```
SELECT COUNT(StudentID)
FROM student;
```



AVG() Example

The following SQL statement finds the average Studentid of all Studentids:

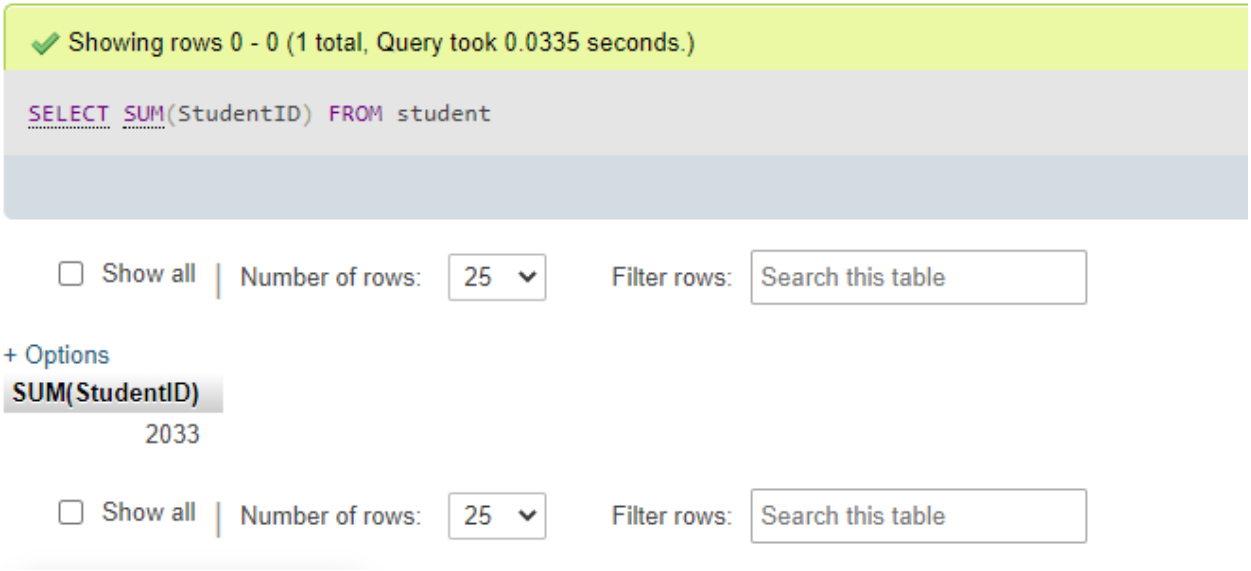
```
SELECT AVG(StudentID)
FROM student;
```



SUM() Example

The following SQL statement finds the sum of the "StudentID" fields in the "student" table:

```
SELECT SUM(StudentID)
FROM student;
```



26. LIKE Operator :

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

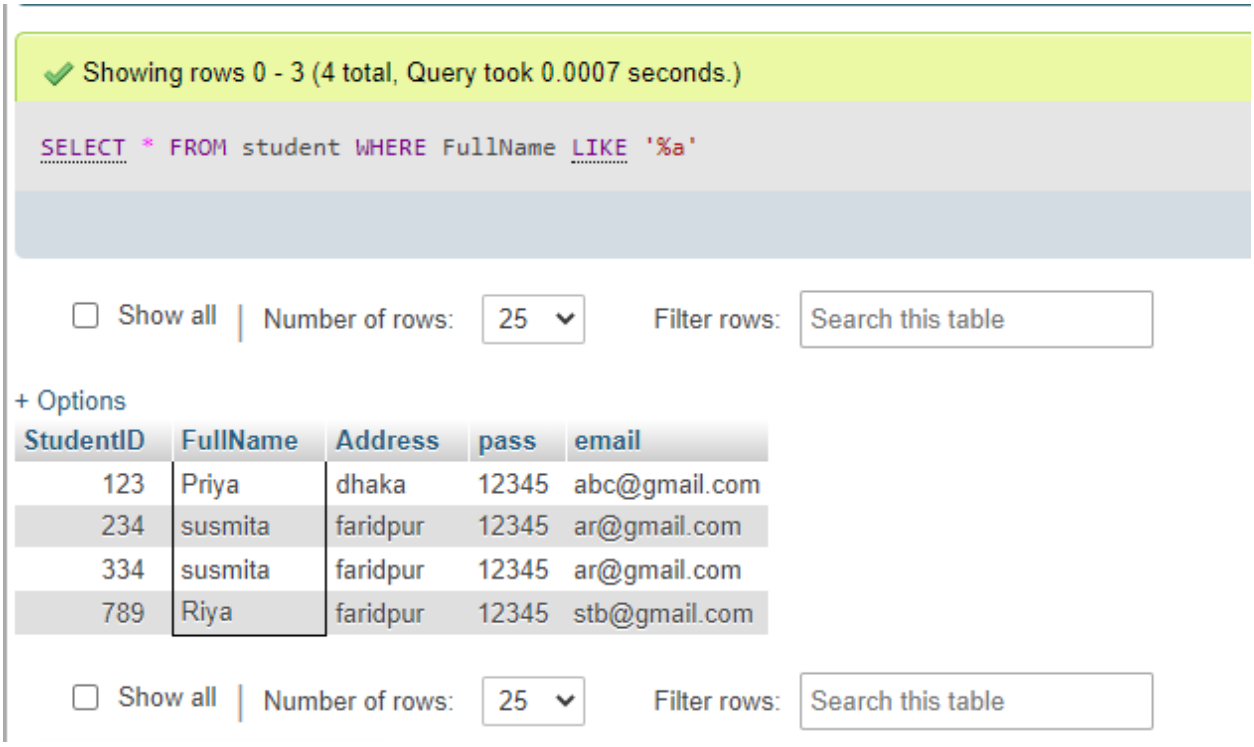
LIKE Operator	Description
WHERE FullName LIKE 'a%'	Finds any values that start with "a"
WHERE FullName LIKE '%a'	Finds any values that end with "a"
WHERE FullName LIKE '%or%'	Finds any values that have "or" in any position
WHERE FullName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE FullName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length

WHERE FullName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE FullName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

For example,

The following SQL statement selects all Students with a FullName ending with "a":

```
SELECT * FROM student
WHERE FullName LIKE '%a';
```



27. Wildcard Characters :

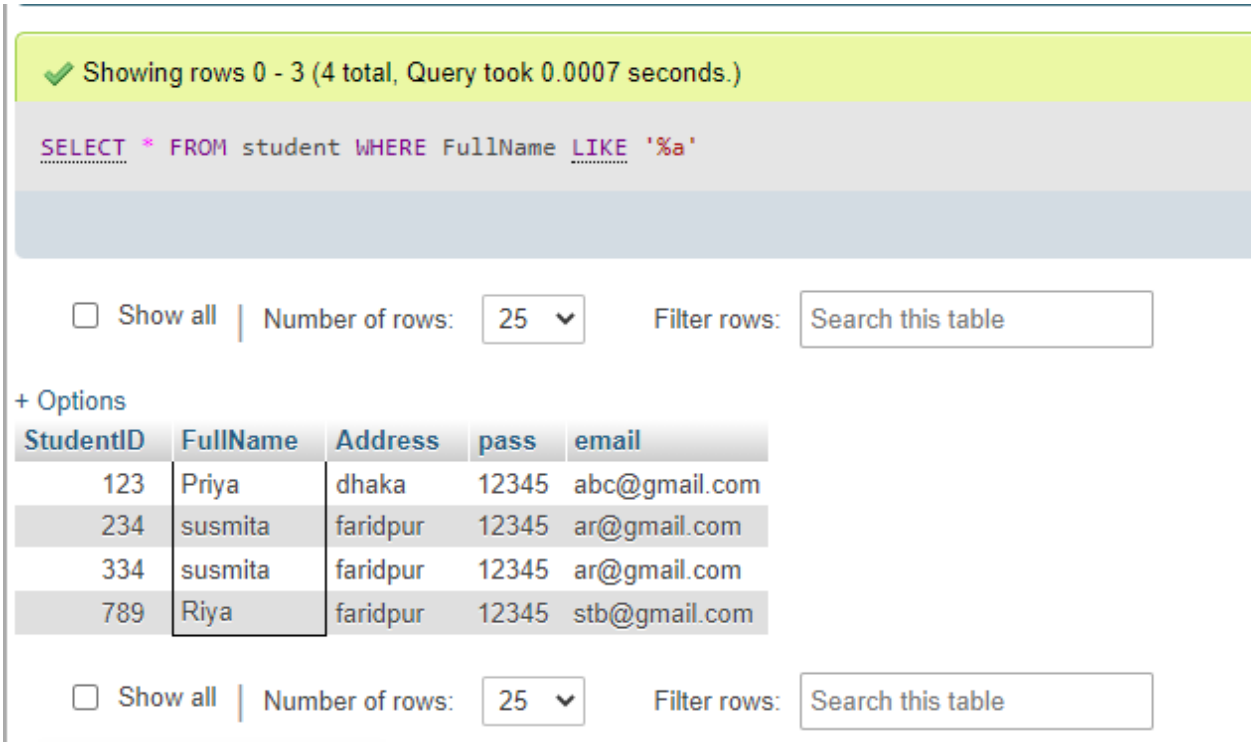
A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not h
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and
-	Represents a range of characters	c[a-b]t finds cat and cbt

The following SQL statement selects all Students with a FullName ending with "a":

```
SELECT * FROM student
WHERE FullName LIKE '%a';
```

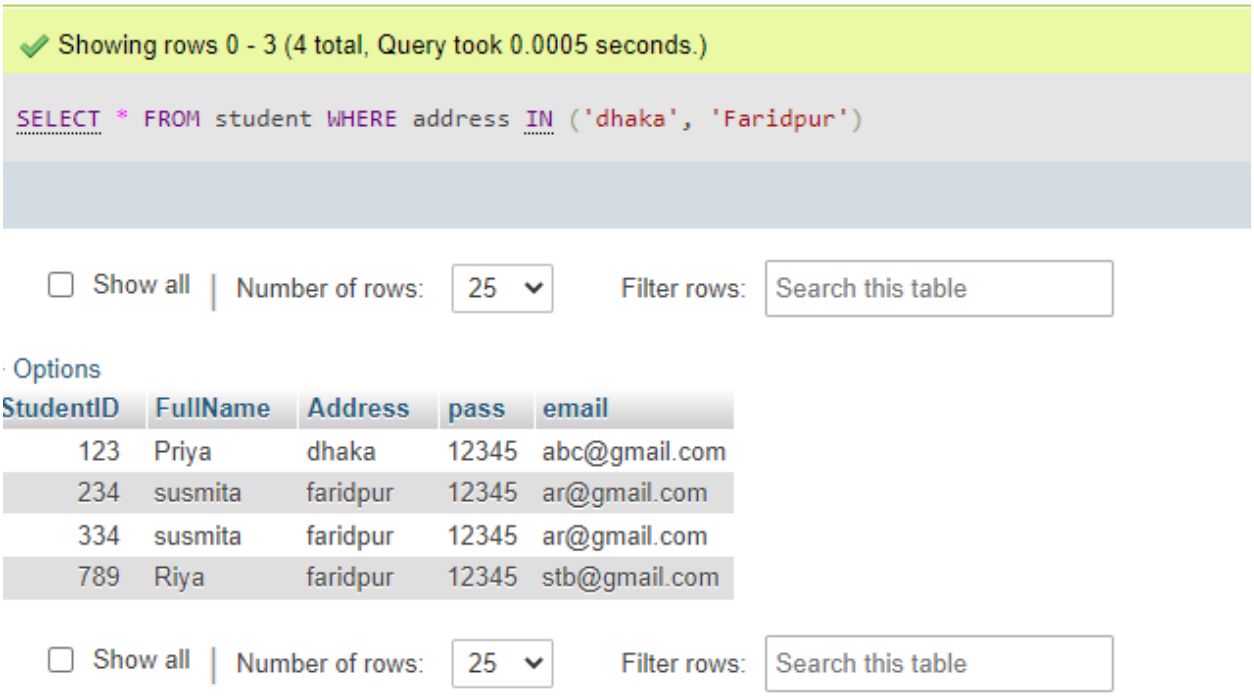


28. IN Operator :

The IN operator allows you to specify multiple values in
a WHERE clause.The IN operator is a shorthand for multiple OR conditions.

The following SQL statement selects all students that are located in "dhaka" or "faridpur":

```
SELECT * FROM student
WHERE address IN ('dhaka', 'Faridpur');
```



Showing rows 0 - 3 (4 total, Query took 0.0005 seconds.)

```
SELECT * FROM student WHERE address IN ('dhaka', 'Faridpur')
```

Show all | Number of rows: 25 Filter rows: Search this table

Options

StudentID	FullName	Address	pass	email
123	Priya	dhaka	12345	abc@gmail.com
234	susmita	faridpur	12345	ar@gmail.com
334	susmita	faridpur	12345	ar@gmail.com
789	Riya	faridpur	12345	stb@gmail.com

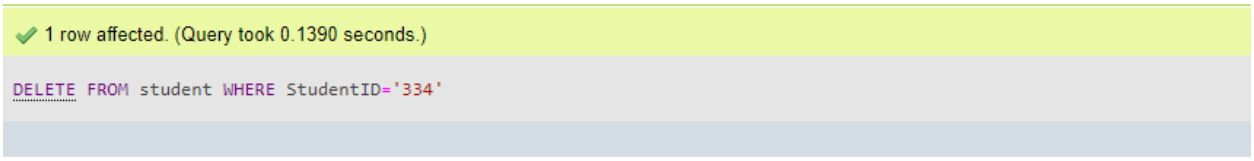
Show all | Number of rows: 25 Filter rows: Search this table

29. DELETE :

The DELETE statement is used to delete existing records in a table.

The following SQL statement deletes the studentid "334" from the "student" table:

```
DELETE FROM student WHERE StudentID='334';
```

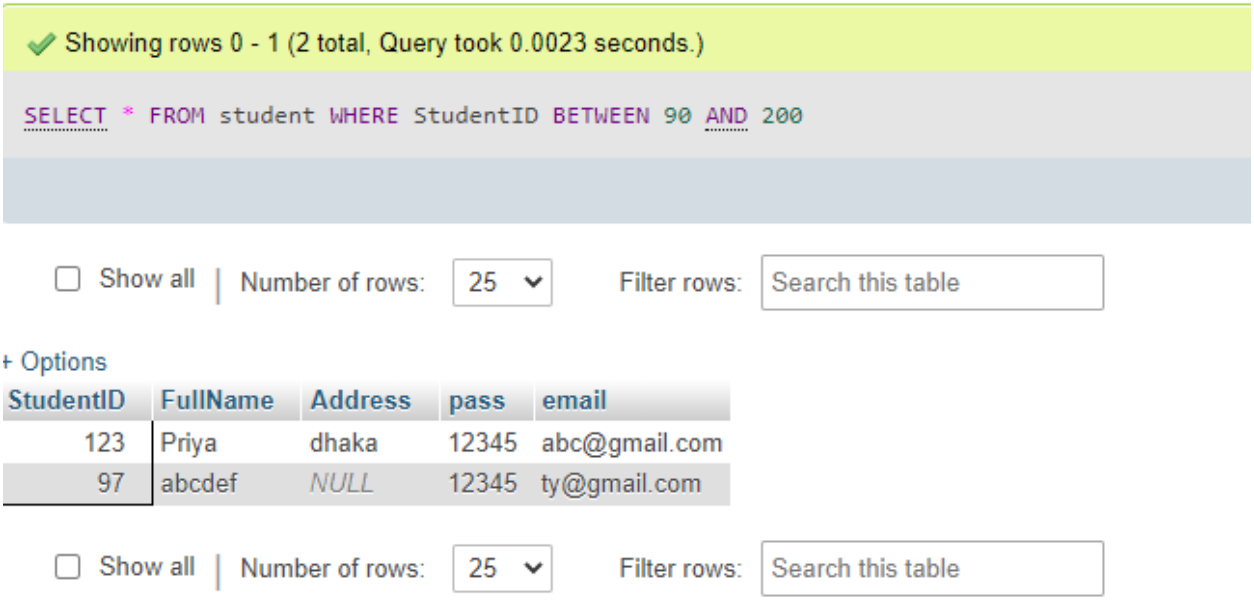


30. BETWEEN :

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.The BETWEEN operator is inclusive: begin and end values are included.

The following SQL statement selects all products with a price between 10 and 20:

```
SELECT * FROM student
WHERE StudentID BETWEEN 90 AND 200;
```



31. Aliases :

SQL aliases are used to give a table, or a column in a table, a temporary name.

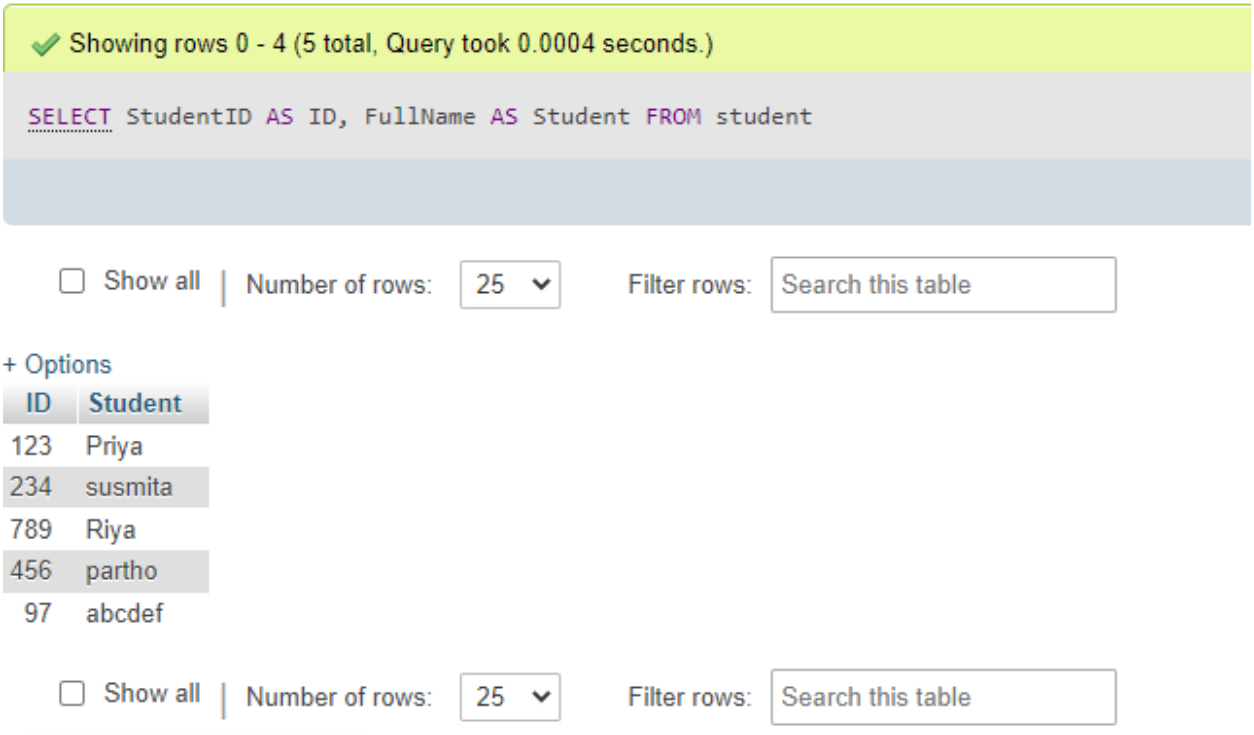
Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

The following SQL statement creates two aliases, one for the StudentID column and one for the FullName column:

```
SELECT StudentID AS ID, FullName AS Student
FROM student;
```



32. JOIN :

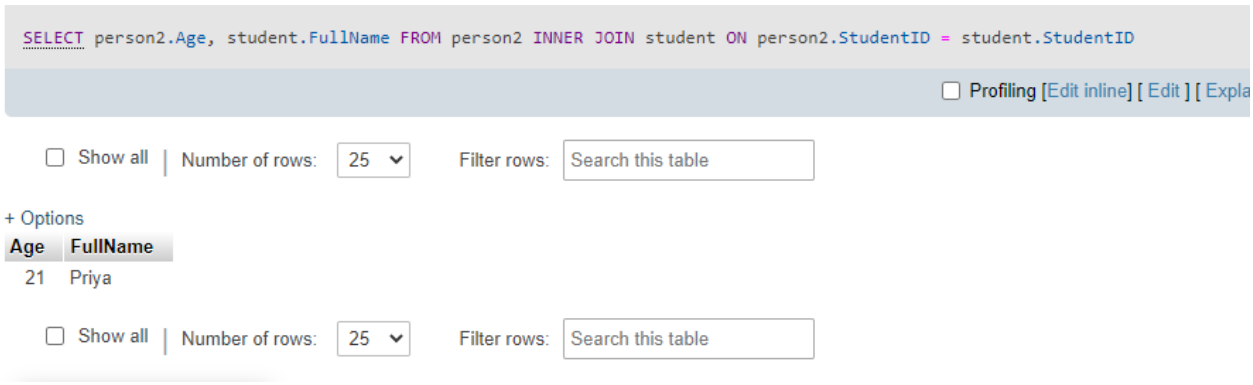
A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

INNER JOIN :

The INNER JOIN keyword selects records that have matching values in both tables.

The following SQL statement selects age with student information:

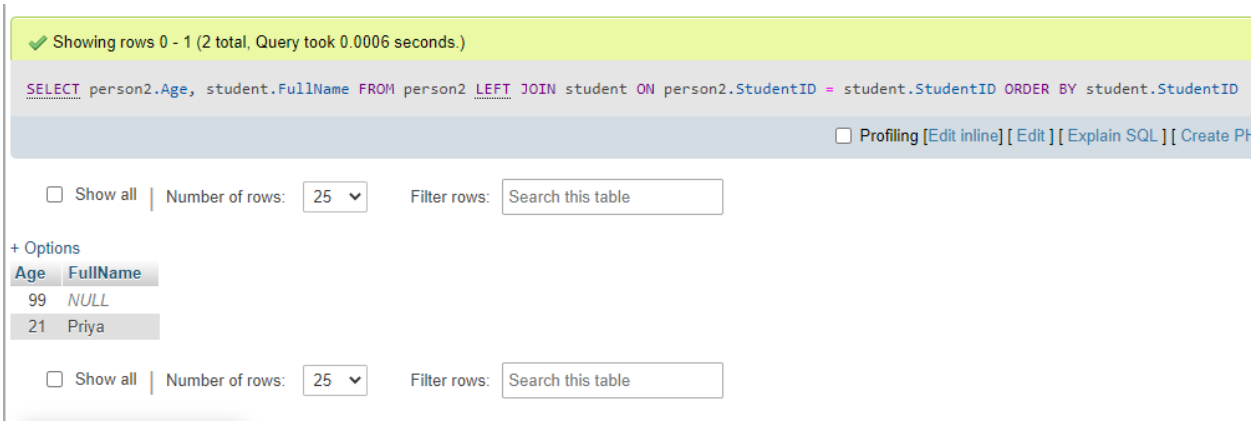
```
SELECT person2.Age, student.FullName
FROM person2
INNER JOIN student ON person2.StudentID = student.StudentID;
```



LEFT JOIN :

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

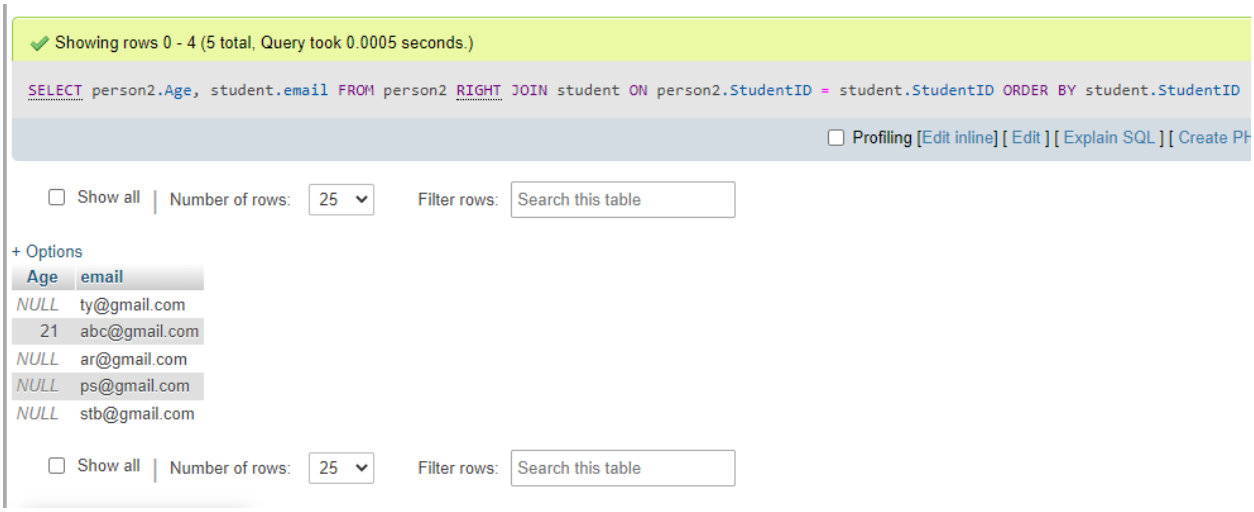
```
SELECT person2.Age, student.FullName
FROM person2
LEFT JOIN student ON person2.StudentID = student.StudentID
ORDER BY student.StudentID;
```



RIGHT JOIN :

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

SELECT person2.Age, student.email
FROM person2
RIGHT JOIN student ON person2.StudentID = student.StudentID
ORDER BY student.StudentID;



FULL OUTER JOIN :

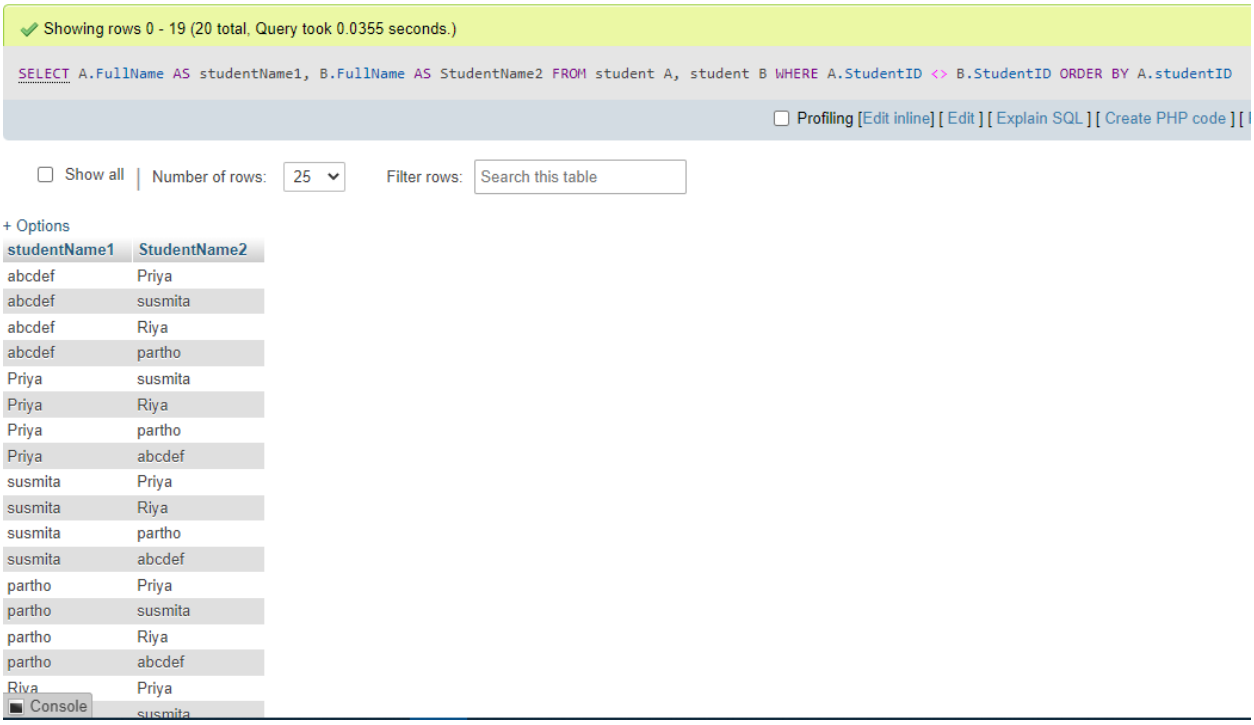
The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

```
SELECT person2.Age, student.FullName
FROM person2
FULL OUTER JOIN student ON person2.StudentID = student.StudentID
ORDER BY student.StudentID;
```

Self Join :

A self join is a regular join, but the table is joined with itself.

```
SELECT A.FullName AS studentName1, B.FullName AS StudentName2
FROM student A, student B
WHERE A.StudentID <> B.StudentID
ORDER BY A.studentID;
```



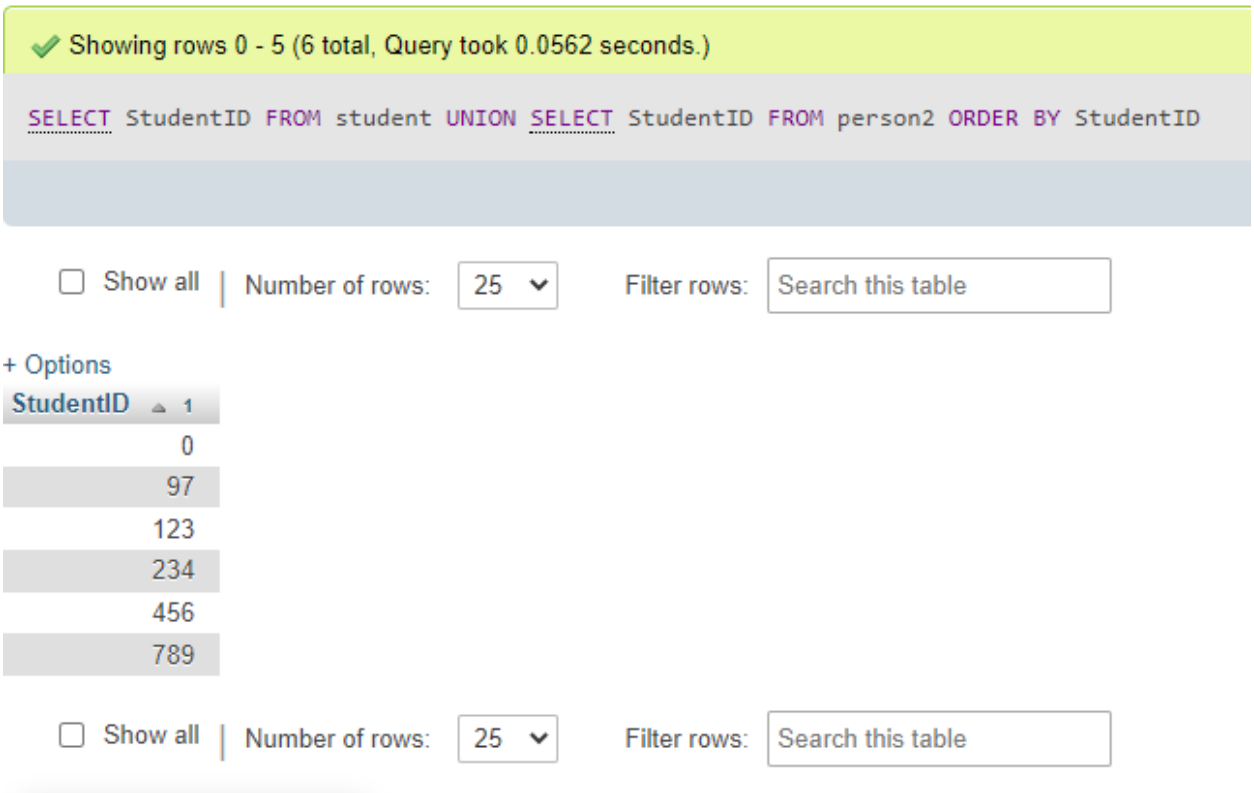
33.UNION :

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

The following SQL statement returns the StyudentID (only distinct values) from both the "student" and the "person2" table:

```
SELECT StudentID FROM student
UNION
SELECT StudentID FROM person2
ORDER BY StudentID;
```



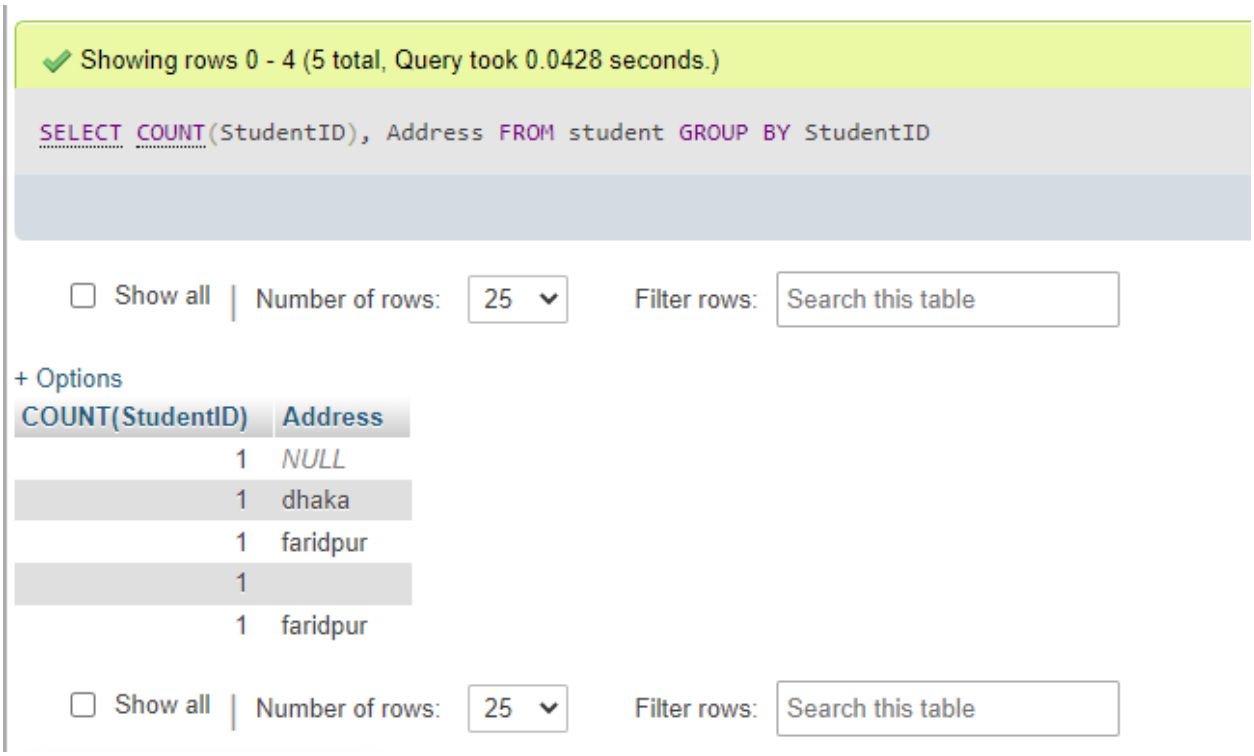
34. GROUP BY :

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

The following SQL statement lists the number of student in each address:

```
SELECT COUNT(StudentID), Address
FROM student
GROUP BY StudentID;
```

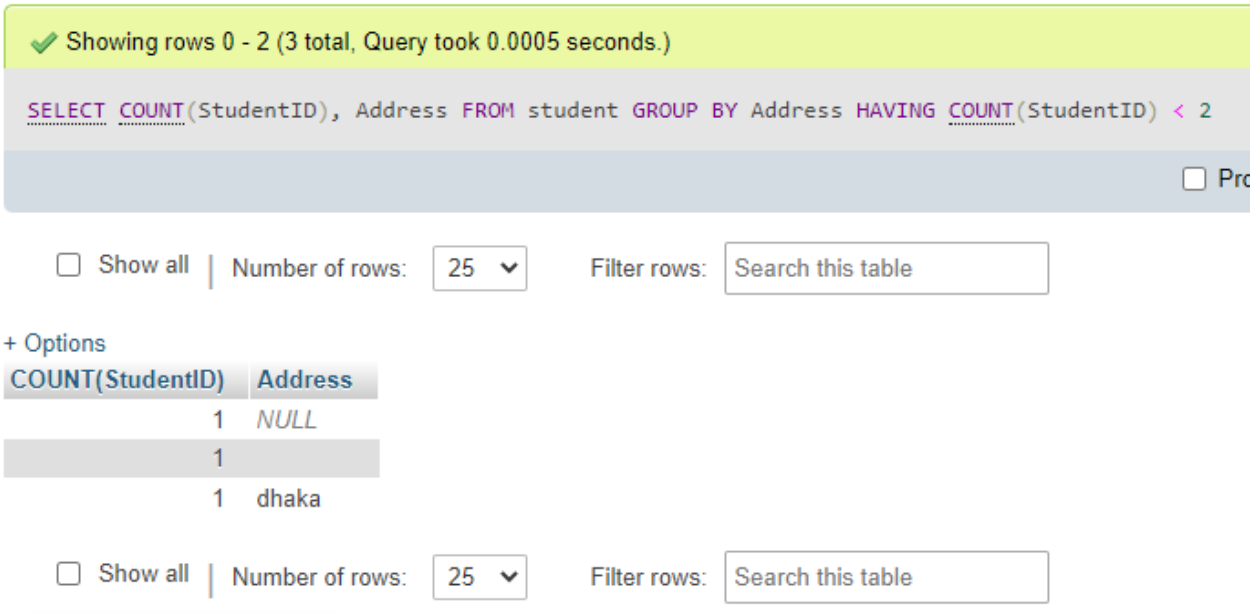


35. HAVING :

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

The following SQL statement lists the number of student in each address. Only include city with less than 2 students:

```
SELECT COUNT(StudentID), Address
FROM student
GROUP BY Address
HAVING COUNT(StudentID) < 2;
```



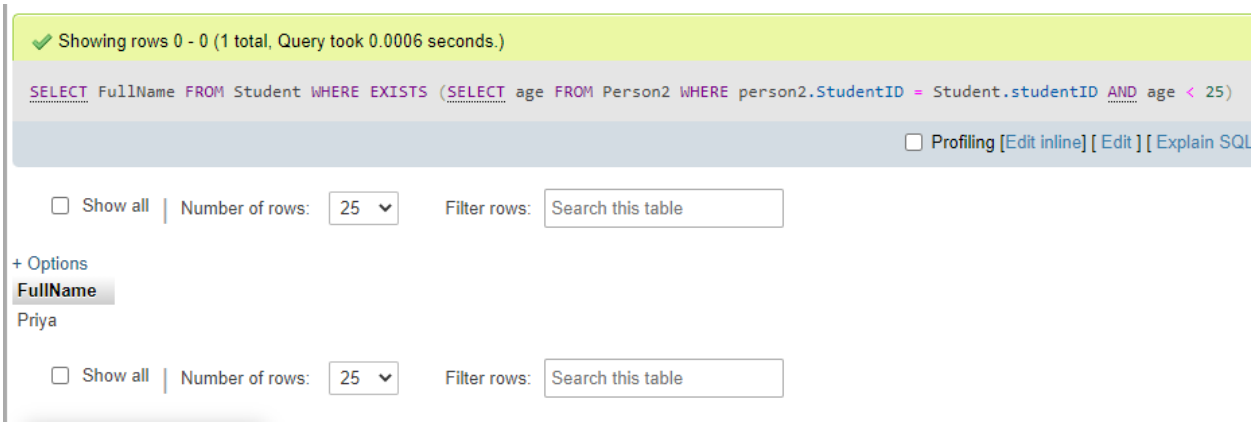
36. EXISTS :

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

The following SQL statement returns TRUE and lists the student with a product price less than 20:

```
SELECT FullName
FROM Student
WHERE EXISTS (SELECT age FROM Person2 WHERE person2.St
udentID = Student.studentID AND age < 25);
```



Thank You