DataBase Management System LAB

Paper Code – ETCS 256

Name: Ankit Goel
Enrollment No.: 00296402719
Semester: 4rd
Group: 4C11

Maharaja Agrasen Institute of Technology, PSP Area,
Sector – 22, Rohini, New Delhi-85

Submitting To:
Ms. Neelam Mam

INDEX

S. No.	Experiment Name
1	Draw E-R diagram and convert entities and relationships to relation table for a given scenario.
2	To study the basic SQL commands.
3	To implement different SOL key constraints & clauses on a given database.
4	Implementation of Arithmetic and String functions on a given database.
5	Implementation of Operators and Views on a given database.
6	Implementation of Joins and Index on a given Database.
7	To study the PL/SQL Database Management language
8	To study the different types of cursors and its implementation
9	Different types of triggers in PL-SQL.
10	Create a program in PL-SQL

EXPERIMENT 1

AIM:

Draw E-R diagram and convert entities and relationships to relation table for a given scenario.

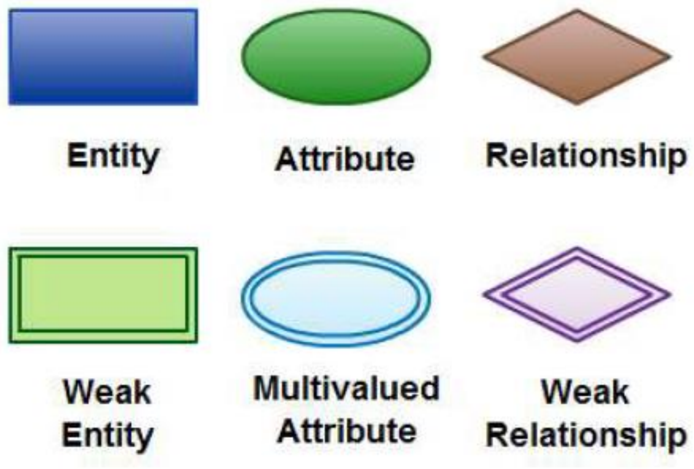
THEORY:

- An Entity Relationship Diagram (ERD) is a visual representation of different entities within a system and how they relate to each other.

- They are widely used to design relational databases. The entities in the ER schema become tables, attributes and converted the database schema. Since they can be used to visualize database tables and their relationships it’s commonly used for database troubleshooting as well.

- **ER Diagram Symbols & Notations:**

  There are three basic elements in an ER Diagram: entity, attribute, relationship. There are more elements which are based on the main elements. They are weak entity, multi-valued attribute, derived attribute, weak relationship, and recursive relationship. Cardinality and ordinality are two other notations used in ER diagrams to further define relationships.



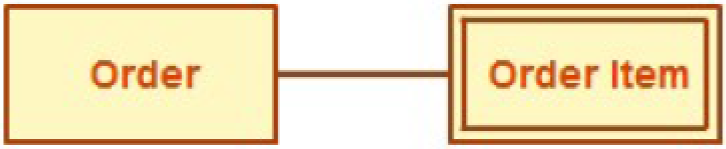
- **Entity:**

An entity can be a person, place, event, or object that is relevant to a given system.

For example, a school system may include students, teachers, major courses, subjects, fees, and other items. Entities are represented in ER diagrams by a rectangle and named using singular nouns.

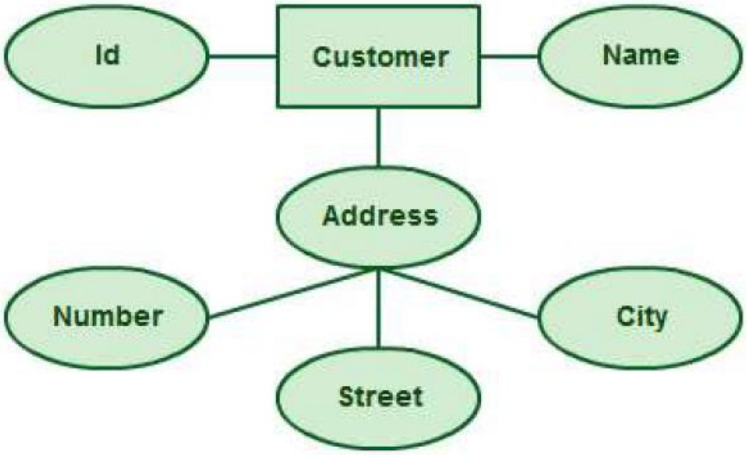
- **Weak Entity**

A weak entity is an entity that depends on the existence of another entity. In more technical terms it can be defined as an entity that cannot be identified by its own attributes. It uses a foreign key combined with its attributed to form the primary key. An entity like order item is a good example for this. The order item will be meaningless without an order so it depends on the existence of the order.



- **Attribute**

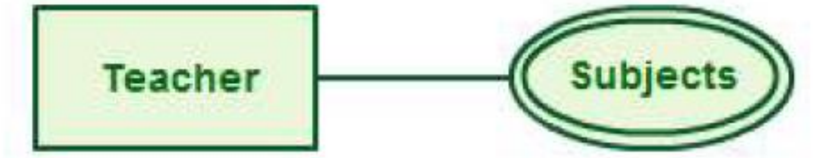
An attribute is a property, trait, or characteristic of an entity, relationship, or another attribute. For example, the attribute Inventory Item Name is an attribute of the entity Inventory Item. An entity can have as many attributes as necessary. Meanwhile, attributes can also have their own specific attributes. For example, the attribute "customer address" can have the attributes number, street, city,



and state. These are called composite attributes. Note that some top level ER diagrams do not show attributes for the sake of simplicity. In those that do, however, attributes are represented by oval shapes.

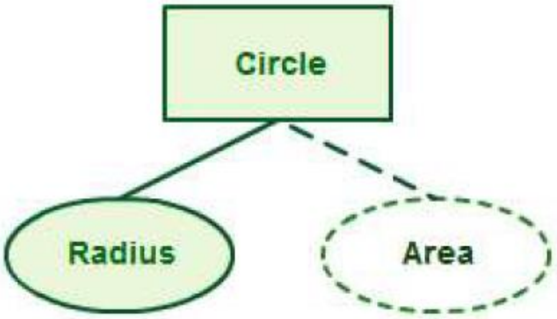
- **Multivalued attribute:**

If an attribute can have more than one value it is called a multi-valued attribute. It is important to note that this is different from an attribute having its own attributes. For example, a teacher entity can have multiple subject values.



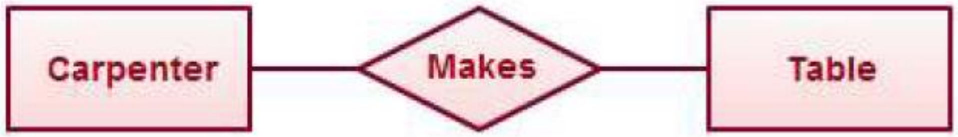
- **Derived Attribute:**

An attribute based on another attribute. This is found rarely in ER diagrams. For example, for a circle, the area can be derived from the radius.



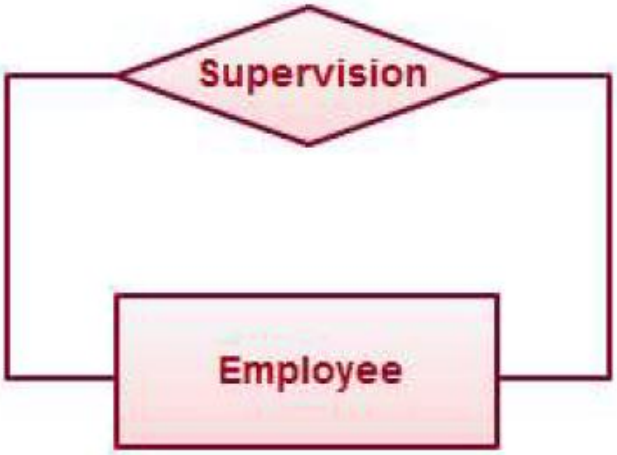
- **Relationship:**

A relationship describes how entities interact. For example, the entity “Carpenter” may be related to the entity “table” by the relationship “builds” or “makes”. Relationships are represented by diamond shapes and are labeled using verbs.



- **Recursive Reltionship:**

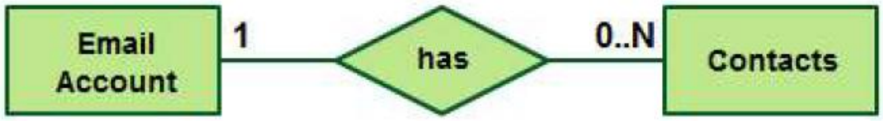
If the same entity participates more than once in a relationship it is known as a recursive relationship. In the below example an employee can be a supervisor and be supervised, so there is a recursive relationship.



- **Cardinality and Ordinality**

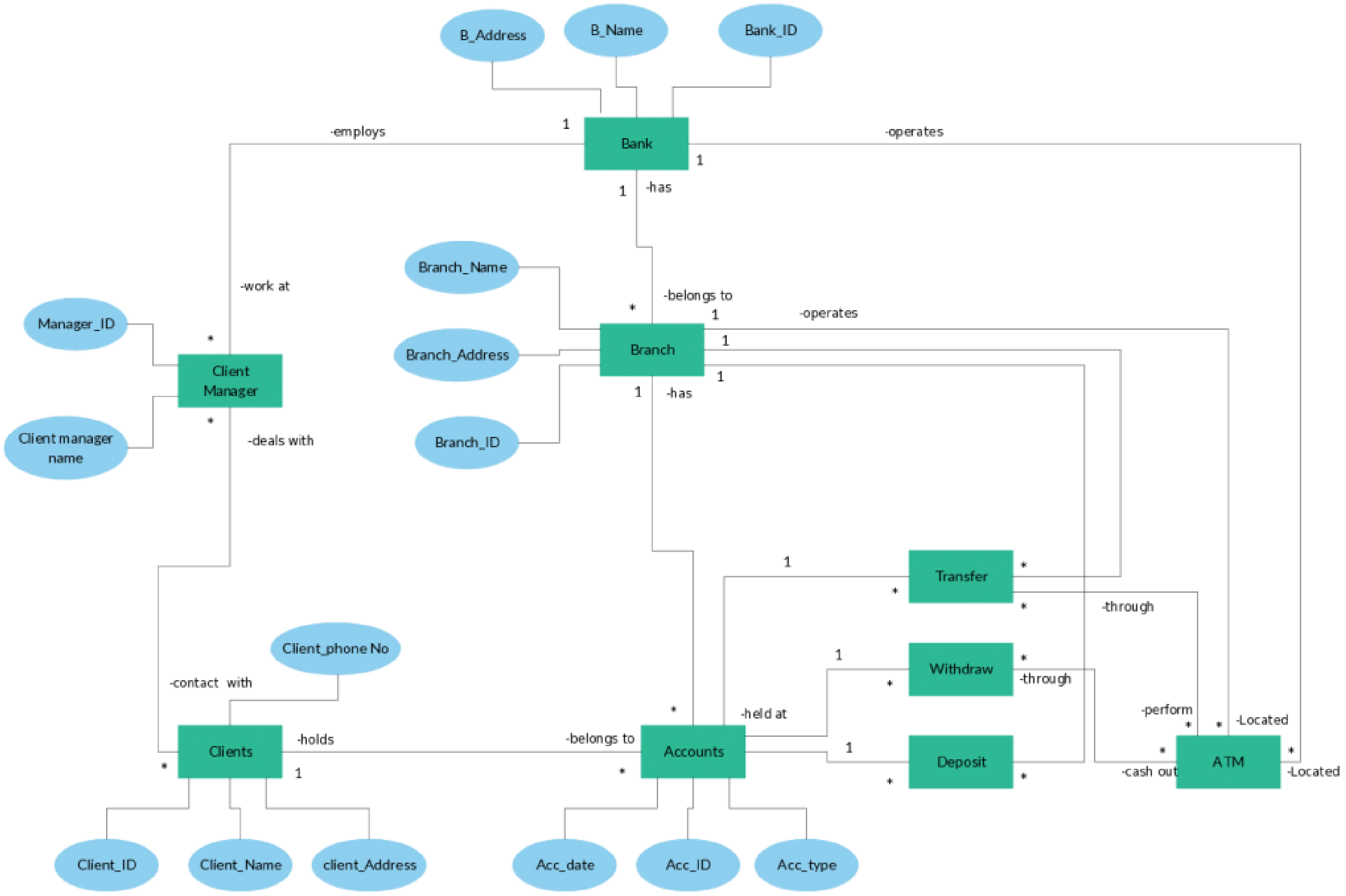
These two further defines relationships between entities by placing the relationship in the context of numbers. In an email system, for example, one account can have multiple contacts. The relationship, in this case, follows a “one to many” model.

There are a number of notations used to present cardinality in ER diagrams. Chen, UML, Crow’s foot, Bachman are some of the popular notations. Creately supports Chen, UML and Crow’s foot notations. The following example uses UML to show cardinality.

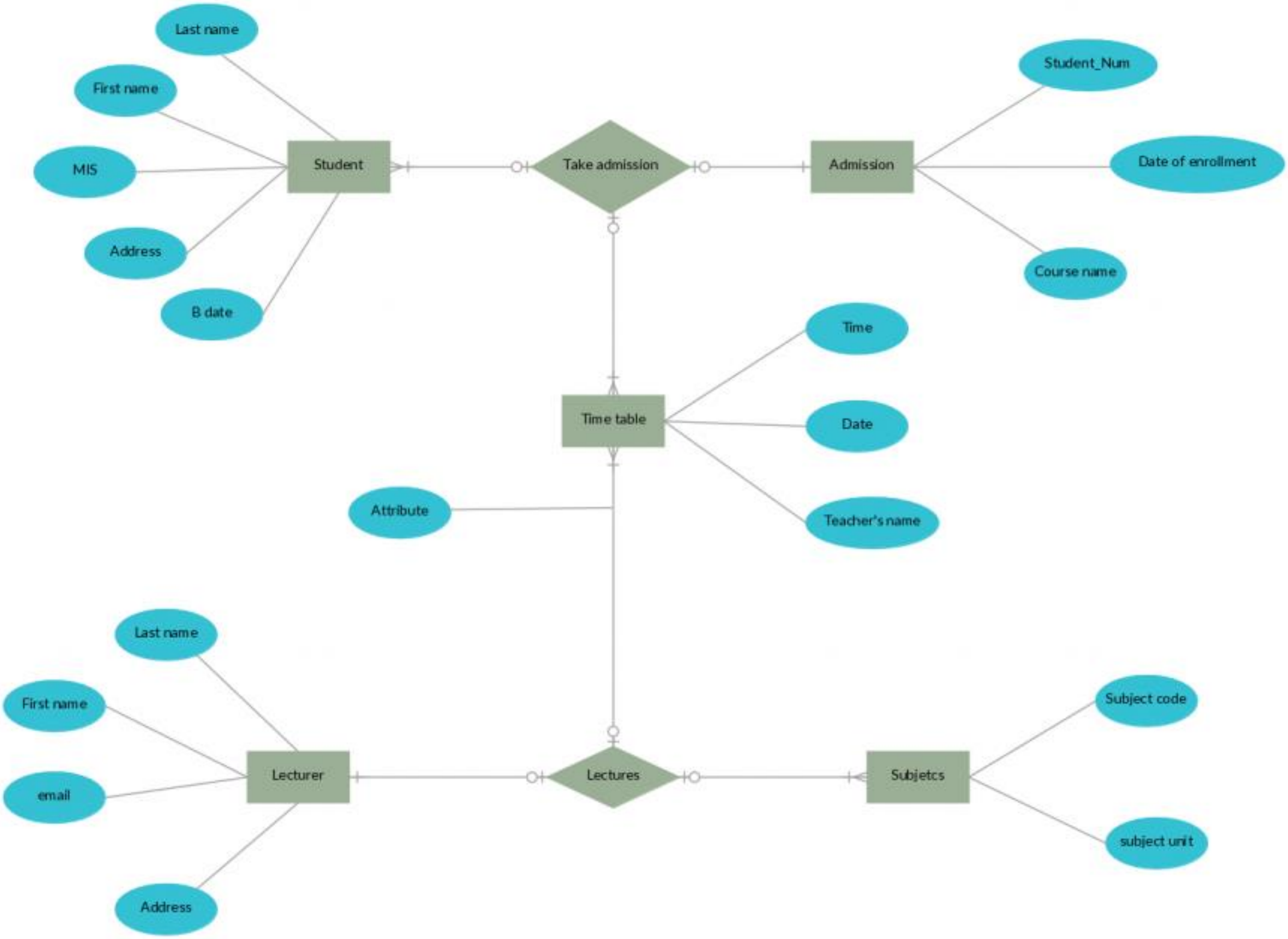


ER Diagram:

1. Bank Management System



2. Course Registration System



EXPERIMENT 2

AIM:

To study the basic SQL commands.

THEORY:

Data Definition Language (DDL)

Data Definition Language changes the structure of the table through creating, reading, altering the table etc.

All the commands & DDL are auto-committed, that means it permanently saw is all the changes in the database.

Here are commands that come under DDL:

a) **CREATE**: It is used to create a new table in the database.

   Syntax:
   ```
   CREATE TABLE table_name (
       column1 datatype,
       column2 datatype,
       column3 datatype,
       ....
   );
   ```

b) **DROP**: It is used to delete both the structure & the records in a table.

   Syntax:
   ```
   DROP TABLE table_name;
   ```

c) **ALTER**: It is used to alter the structure of the database. This change could be either to modify the characteristics q an existing attenbute on to add a new attribute.

Syntax:

1. **To add a new column in the table**
   ```
   ALTER TABLE table_name
   ADD column_name column_definition;
   ```

2. **To modify existing columns as table**
   ```
   ALTER TABLE table_name
   MODIFY (column_name column_definition);
   ```

d) **TRUNCATE**: It is used to delete all the rows from the table & free the space containing the records.

Syntax:

```
TRUNCATE TABLE table_name;
```

Data Manipulation Language (DML)

Data Manipulation Language (DML) commands are used to modify the database. They are responsible for the forms of changes in the database.

DML commands are not auto-committed, which means that it does not permanently save all the changes in the database. They can be rolled back.

Here are commands that come under DDL:

a) **INSERT**: The INSERT command is used to insert data into a now of a table.

Syntax:

```
INSERT INTO table_name VALUES(
    value1,
    value2,
    ....
);
```

b) **UPDATE**: This command is used to update on modify the value of a column in a table.

Syntax:

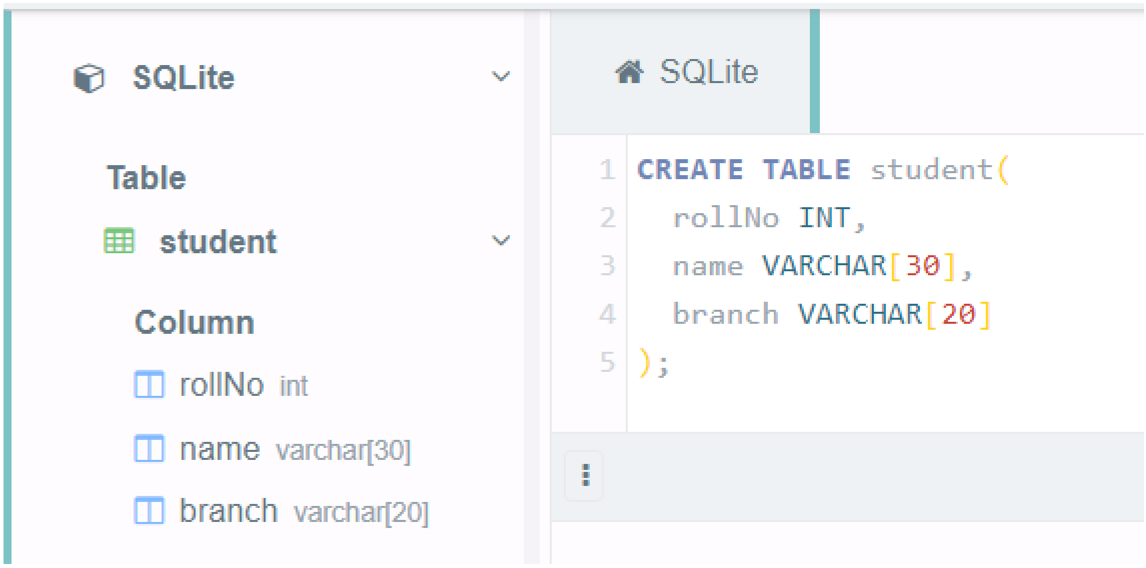
```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

c) **DELETE**: It is used to remove one or more rows from a table.

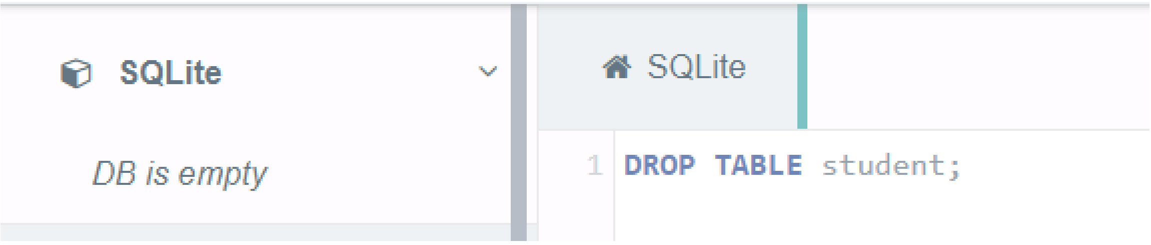
Syntax:

```
DELETE from table_name
WHERE condition;
```

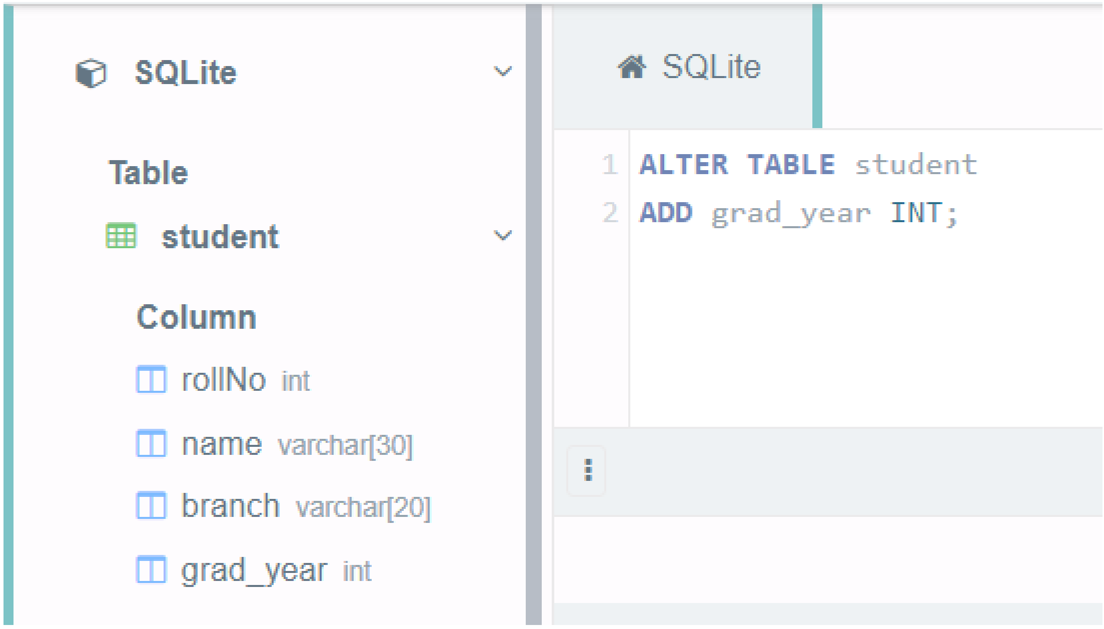
OUTPUT:



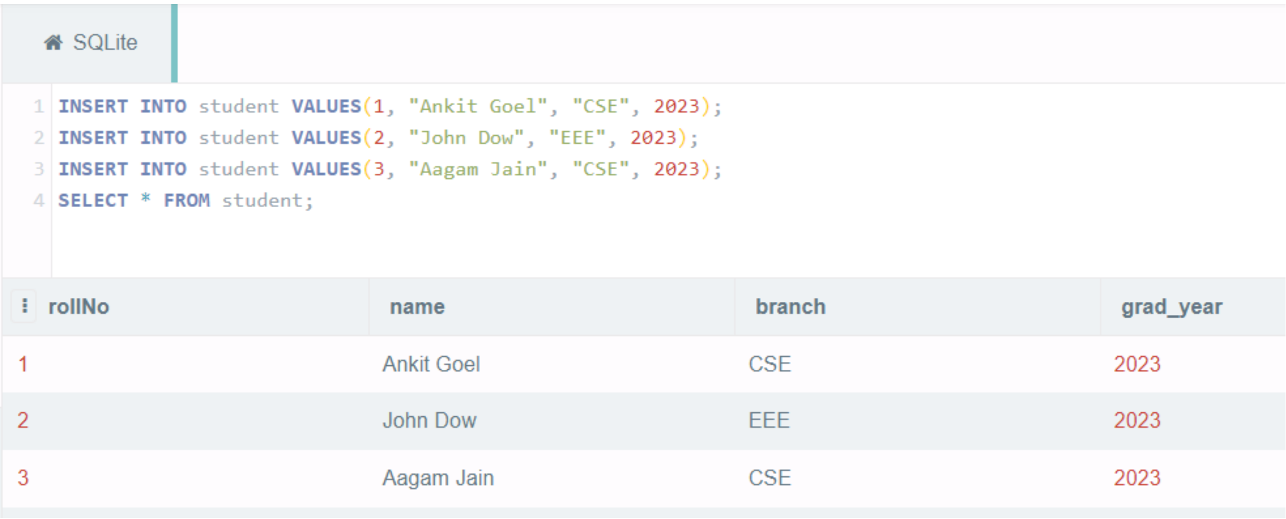
```
CREATE TABLE student(
  rollNo INT,
  name VARCHAR[30],
  branch VARCHAR[20]
);
```



```sql
DROP TABLE student;
```

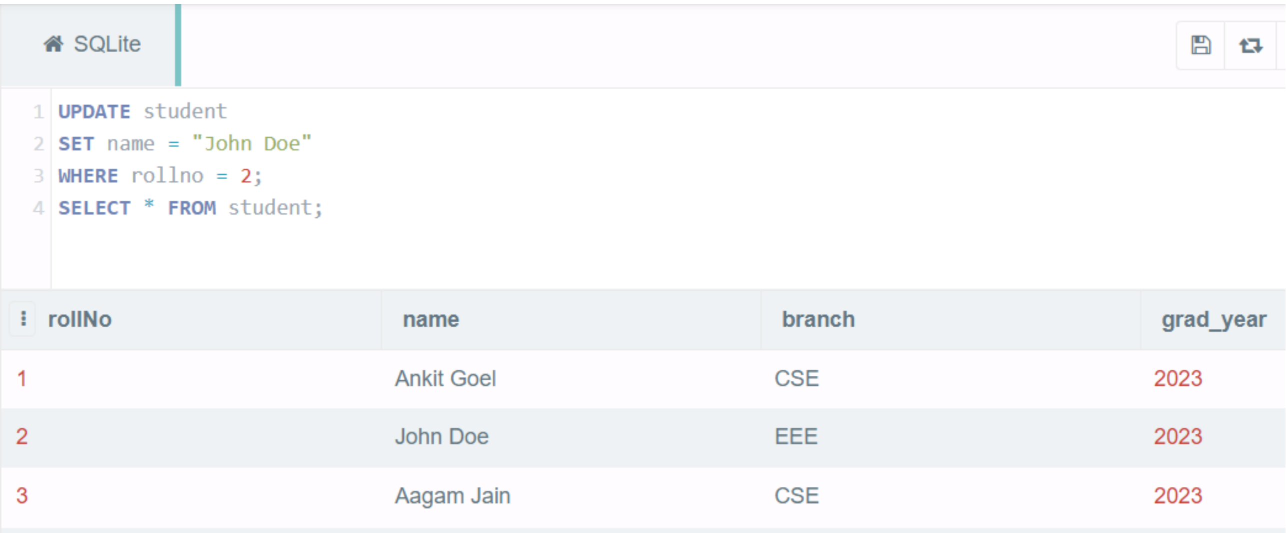


```sql
ALTER TABLE student
ADD grad_year INT;
```



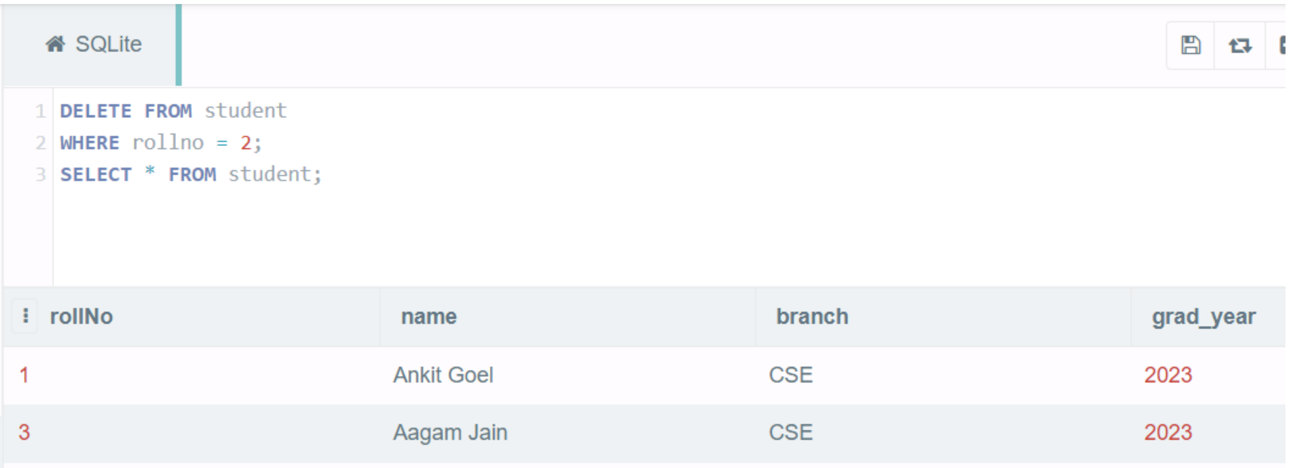
```sql
INSERT INTO student VALUES(1, "Ankit Goel", "CSE", 2023);
INSERT INTO student VALUES(2, "John Dow", "EEE", 2023);
INSERT INTO student VALUES(3, "Aagam Jain", "CSE", 2023);
SELECT * FROM student;
```

rollNo	name	branch	grad_year
1	Ankit Goel	CSE	2023
2	John Dow	EEE	2023
3	Aagam Jain	CSE	2023



```sql
UPDATE student
SET name = "John Doe"
WHERE rollno = 2;
SELECT * FROM student;
```

rollNo	name	branch	grad_year
1	Ankit Goel	CSE	2023
2	John Doe	EEE	2023
3	Aagam Jain	CSE	2023



```sql
DELETE FROM student
WHERE rollno = 2;
SELECT * FROM student;
```

rollNo	name	branch	grad_year
1	Ankit Goel	CSE	2023
3	Aagam Jain	CSE	2023

EXPERIMENT 3

AIM:

To implement different SOL key constraints & clauses on a given database.

THEORY:

SQL Key Constraints

Constraints can be specified when the table is created with CREATE TABLE statement, or after the table is created with ALTER TABLE statement.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

a) **NOT NULL**

   It enforces column to not accept hull values. This ensures that a field always contains a for value, which means that no new record can be inserted without adding this field.

   Syntax:

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

b) **UNIQUE**

   It ensures that all values in a column are different. Both UNIQUE & PRIMARY KEY constraints provide a guarantee. for uniqueness of a column.

   Syntax:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

c) **PRIMARY KEY**

It uniquely identifies each record in a table. A primary key must contain unique values, & cannot contain null values. A table can have only primary key one which can consist of multiple & columns as well.

Syntax:

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

d) **FOREIGN KEY**

It is used to prevent actions that would destroy links between tables. A foreign key is a field in one table, another table. The table with foreign key is known as child table, & the table with primary key is called referenced table.

Syntax:

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

e) **CHECK**

It is used to filter & limit the value range, that can be placed in a column. If we define a CHECK constraint on a column, it will allow only contain types of values in the column.

Syntax:

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

f) **DEFAULT**

It is used to set a default value for a column. It The default value will be added to all new records, if other value is specified.

Syntax:

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

SQL Clauses

a) **WHERE**

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

b) **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".

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

c) **HAVING**

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

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

d) **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.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

EXPERIMENT 4

AIM:

Implementation of Arithmetic and String functions on a given database.

THEORY:

Arithmetic Functions

1. **Power**
   The POWER () function returns the value of a number raised to the power of another number.
   Syntax:
   ```
   Power(a, b)
   ```

2. **Round**
   The ROUND() function rounds a number to a specified number of decimal places.
   Syntax:
   ```
   ROUND(number, decimals, operation)
   ```

3. **Sqrt**
   The SQRT() function returns the square root of a number.
   Syntax:
   ```
   sqrt(number)
   ```

4. **Cos**
   The COS() function returns the cosine of a number.
   Syntax:
   ```
   cos(number)
   ```

5. **Sin**
   The Sin() function returns the sine of a number.
   Syntax:
   ```
   sin(number)
   ```

6. **Tan**
   The TAN() function returns the tangent of a number.

Syntax:

```
tan(number)
```

7. **Abs**

   The ABS() function returns the absolute value of a number.

   Syntax:

   ```
   abs(number)
   ```

8. **Mod**

   SQL MOD() function is used to get the remainder from a division. The SQL DISTINCT command along with the SQL MOD() function is used to retrieve only unique records depending on the specified column or expression.

   Syntax:

   ```
   Mod(dividend , divider);
   ```

9. **Max**

   The MAX() function returns the maximum value in a set of values.

   Syntax:

   ```
   max(number)
   ```

10. **Max**

    The MAX() function returns the maximum value in a set of values.

    Syntax:

    ```
    max(number)
    ```

11. **Min**

    The Min() function returns the minimum value in a set of values.

    Syntax:

    ```
    min(number)
    ```

12. **AVG**

    The AVG() function returns the average value of an expression.

    Syntax

    AVG(expression)

12. **CEILING**

    The CEILING() function returns the smallest integer value that is

larger than or equal to a number.
Syntax
CEILING(number)

13. **FLOOR**

The FLOOR() function returns the largest integer value that is smaller than or equal to a number.
Syntax
FLOOR(number)

STRING FUNCTIONS :

1. **ASCII():** This function is used to find the ASCII value of a character.

Syntax: SELECT ascii('t');
Output: 116

2. CHAR_LENGTH(): Doesn't work for SQL Server. Use LEN() for SQL Server. This function is used to find the length of a word.

Syntax: SELECT char_length('Hello!');
Output: 6

3. CONCAT(): This function is used to add two words or strings.

Syntax: SELECT 'Geeks' || ' ' || 'forGeeks' FROM dual;
Output: 'GeeksforGeeks'

4. INSERT(): This function is used to insert the data into a database.

Syntax: INSERT INTO database (geek_id, geek_name) VALUES (5000, 'abc');
Output: successfully updated

5. LCASE(): This function is used to convert the given string into lower case.

Syntax: LCASE ("GeeksFor Geeks To Learn");
Output: geeksforgeeks to learn

6. LENGTH(): This function is used to find the length of a word.

Syntax: LENGTH('GeeksForGeeks');

```
Output: 13
```

7. LOWER(): This function is used to convert the upper case string into lower case.

```
 Syntax: SELECT LOWER('GEEKSFORGEEKS.ORG');
Output: geeksforgeeks.org
```

8. LPAD(): This function is used to make the given string of the given size by adding the given symbol.

```
 Syntax: LPAD('geeks', 8, '0');
 Output:
000geeks
```

9. LTRIM(): This function is used to cut the given sub string from the original string.

```
Syntax: LTRIM('123123geeks', '123');
 Output: geeks
```

10. REVERSE(): This function is used to reverse a string.

```
Syntax: SELECT REVERSE('geeksforgeeks.org');
Output: ‘gro.skeegrofskeeg’
```

11. RPAD(): This function is used to make the given string as long as the given size by adding the given symbol on the right.

```
Syntax: RPAD('geeks', 8, '0');
   Output: ‘geeks000’
```

12.RTRIM(): This function is used to cut the given sub string from the original string.

```
Syntax: RTRIM('geeksxyxzyyy', 'xyz');
Output: ‘geeks’
```

13. STRCMP(): This function is used to compare 2 strings.

- If string1 and string2 are the same, the STRCMP function will return 0.
- If string1 is smaller than string2, the STRCMP function will return -1.
- If string1 is larger than string2, the STRCMP function will return 1.

```
 Syntax: SELECT STRCMP('google.com', 'geeksforgeeks.com');
Output: -1
```

14. SUBSTRING(): This function is used to find an alphabet from the mentioned size and the given string.

```
Syntax: SELECT SUBSTRING('GeeksForGeeks.org', 9, 1);
```

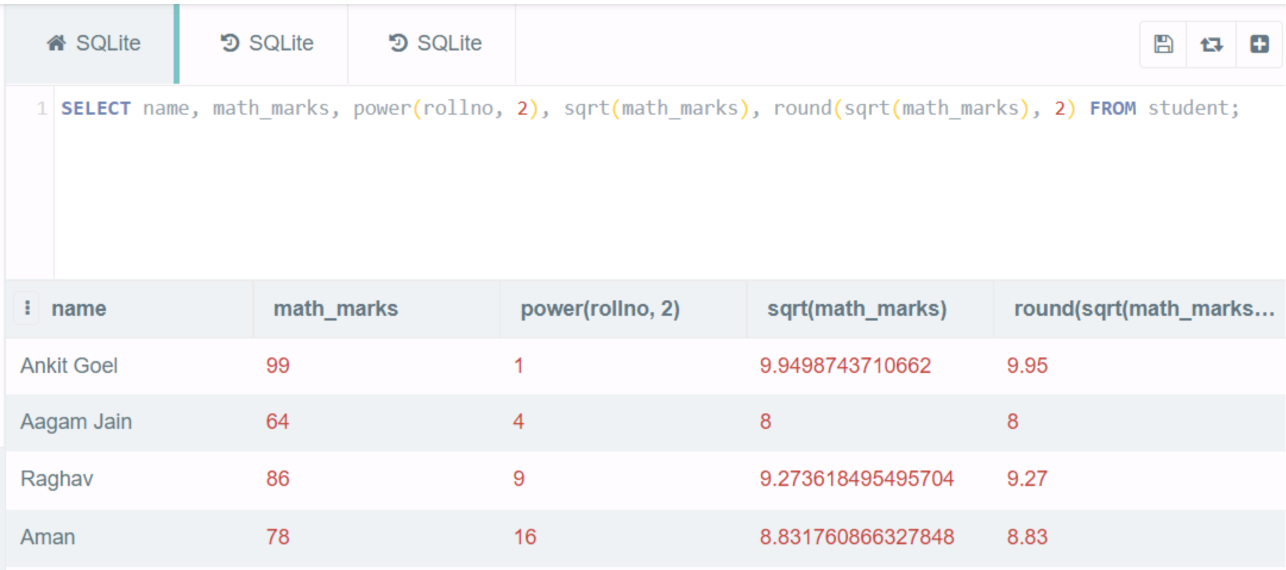
```
Output: ‘G’
```

15.UCASE(): This function is used to make the string in upper case.

```
  Syntax: UCASE ("GeeksForGeeks");
 Output:
GEEKSFORGEEKS
```

OUTPUT:

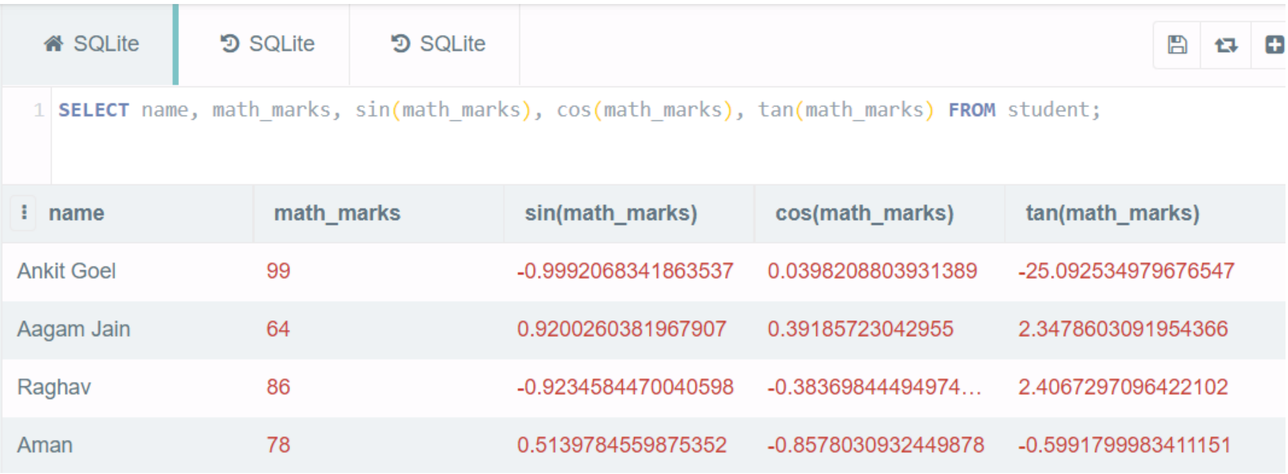
Power, sqrt, round



```
SELECT name, math_marks, power(rollno, 2), sqrt(math_marks), round(sqrt(math_marks), 2) FROM student;
```

name	math_marks	power(rollno, 2)	sqrt(math_marks)	round(sqrt(math_marks...
Ankit Goel	99	1	9.9498743710662	9.95
Aagam Jain	64	4	8	8
Raghav	86	9	9.273618495495704	9.27
Aman	78	16	8.831760866327848	8.83

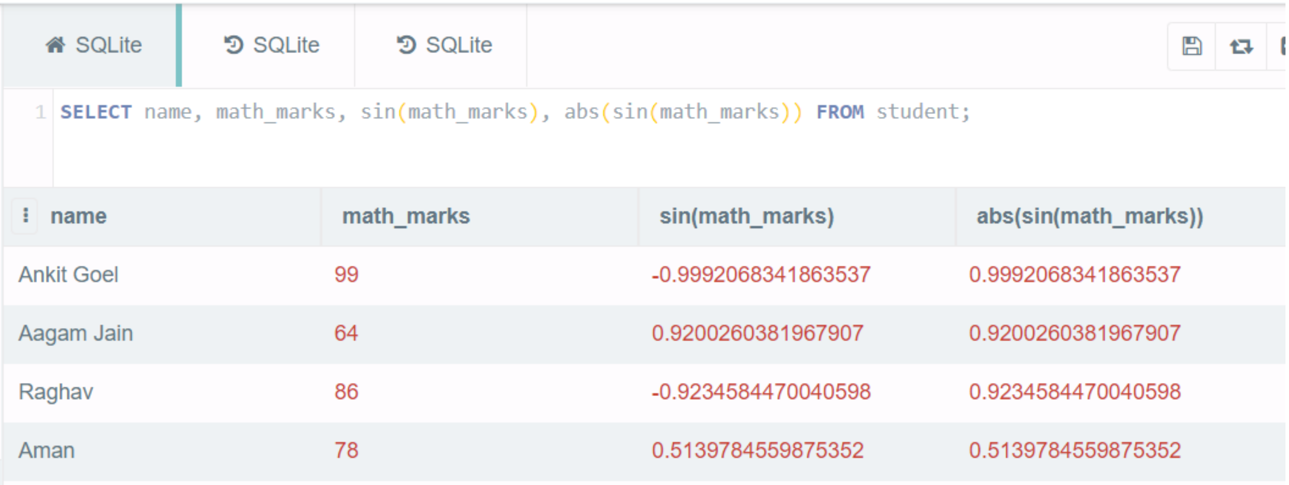
Cos, sin, tan



```
SELECT name, math_marks, sin(math_marks), cos(math_marks), tan(math_marks) FROM student;
```

name	math_marks	sin(math_marks)	cos(math_marks)	tan(math_marks)
Ankit Goel	99	-0.9992068341863537	0.0398208803931389	-25.09253497967654
Aagam Jain	64	0.9200260381967907	0.39185723042955	2.3478603091954366
Raghav	86	-0.9234584470040598	-0.38369844494974...	2.4067297096422102
Aman	78	0.5139784559875352	-0.8578030932449878	-0.5991799983411151

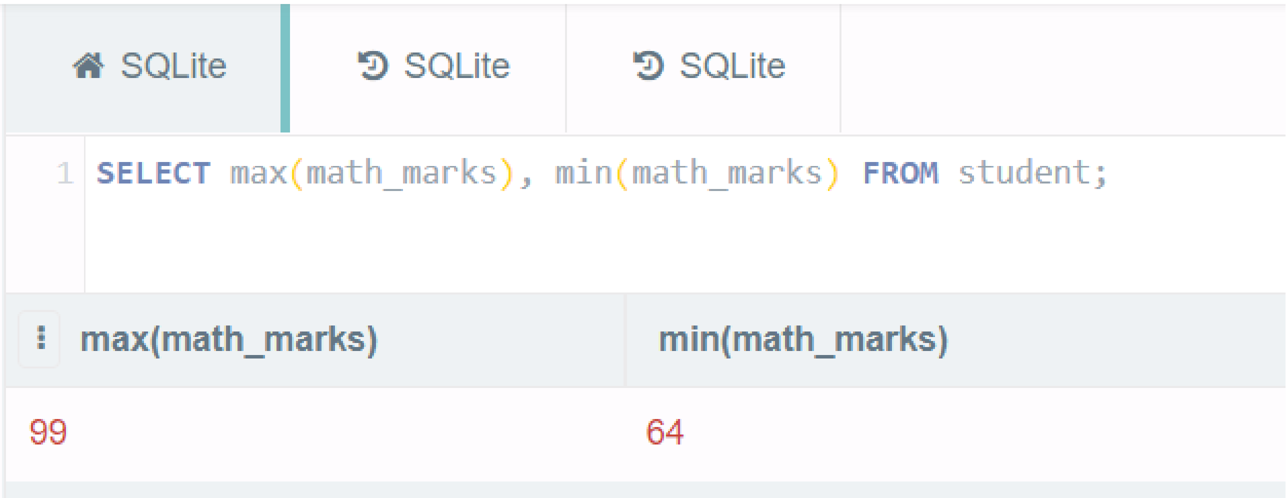
Abs



```sql
SELECT name, math_marks, sin(math_marks), abs(sin(math_marks)) FROM student;
```

name	math_marks	sin(math_marks)	abs(sin(math_marks))
Ankit Goel	99	-0.9992068341863537	0.9992068341863537
Aagam Jain	64	0.9200260381967907	0.9200260381967907
Raghav	86	-0.9234584470040598	0.9234584470040598
Aman	78	0.5139784559875352	0.5139784559875352

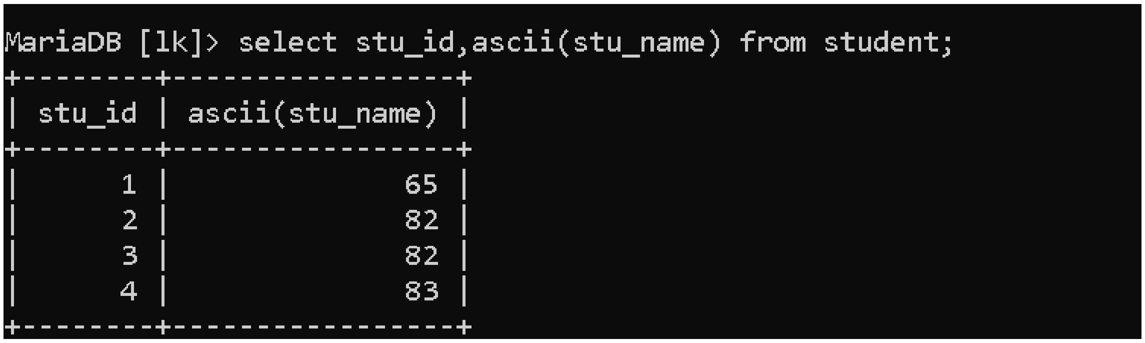
Max, min



```sql
SELECT max(math_marks), min(math_marks) FROM student;
```

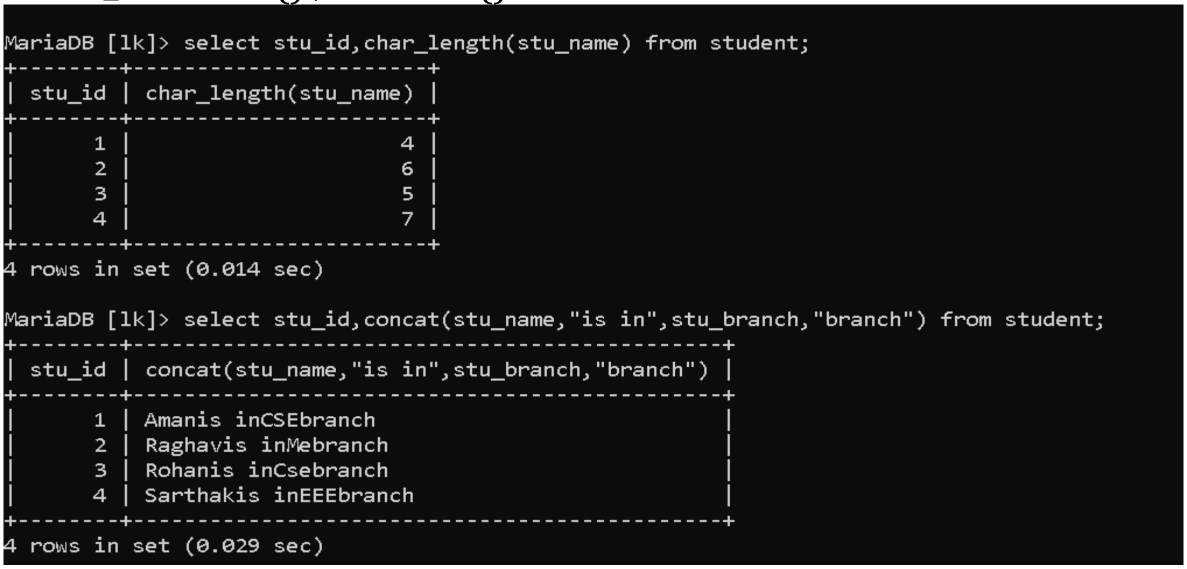
max(math_marks)	min(math_marks)
99	64

Ascii



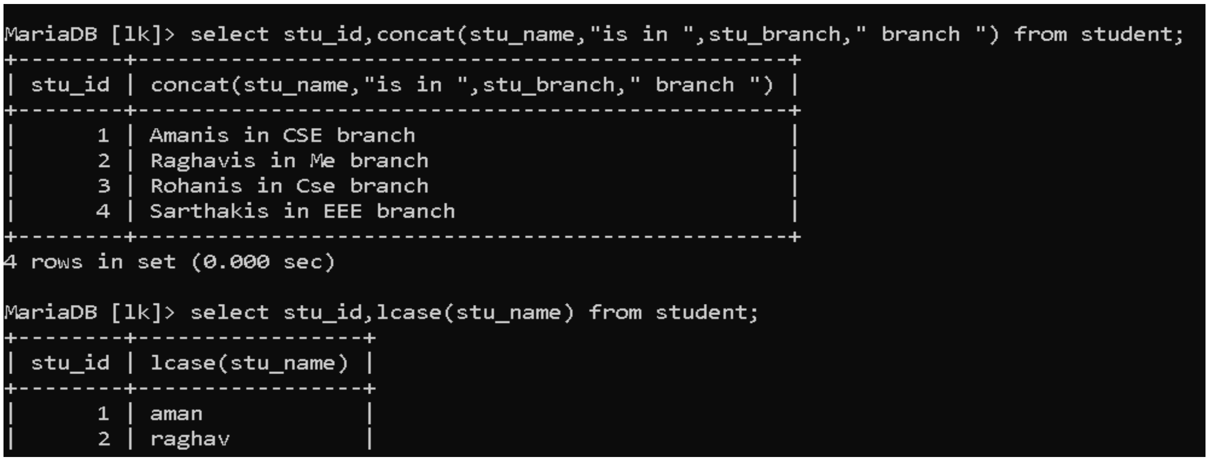
```
MariaDB [lk]> select stu_id,ascii(stu_name) from student;
+--------+-----------------+
| stu_id | ascii(stu_name) |
+--------+-----------------+
|      1 |              65 |
|      2 |              82 |
|      3 |              82 |
|      4 |              83 |
+--------+-----------------+
```

CHAR_LENGTH () , CONCAT()



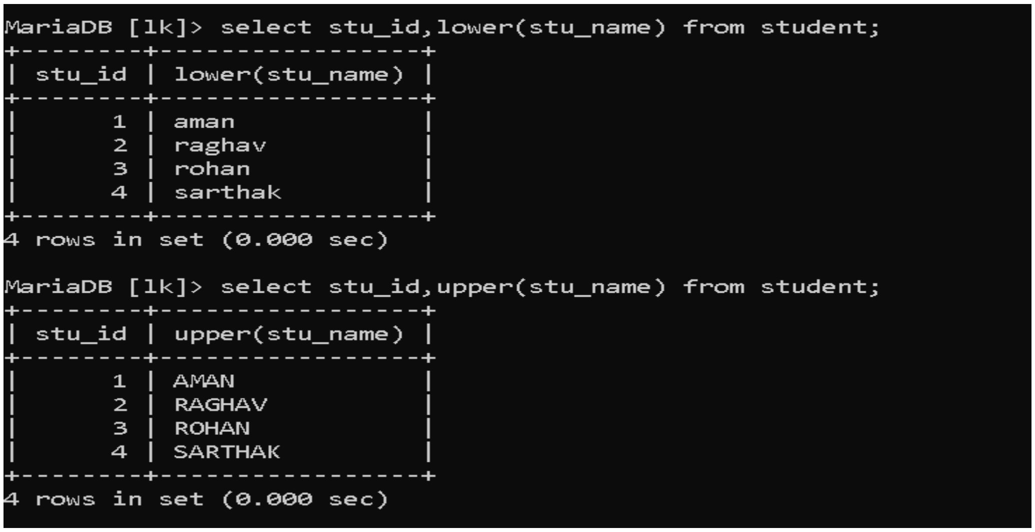
```
MariaDB [lk]> select stu_id,char_length(stu_name) from student;
+--------+-----------------------+
| stu_id | char_length(stu_name) |
+--------+-----------------------+
|      1 |                     4 |
|      2 |                     6 |
|      3 |                     5 |
|      4 |                     7 |
+--------+-----------------------+
4 rows in set (0.014 sec)

MariaDB [lk]> select stu_id,concat(stu_name,"is in",stu_branch,"branch") from student;
+--------+----------------------------------------------+
| stu_id | concat(stu_name,"is in",stu_branch,"branch") |
+--------+----------------------------------------------+
|      1 | Amanis inCSEbranch                           |
|      2 | Raghavis inMebranch                          |
|      3 | Rohanis inCsebranch                          |
|      4 | Sarthakis inEEEbranch                        |
+--------+----------------------------------------------+
4 rows in set (0.029 sec)
```

LCase()



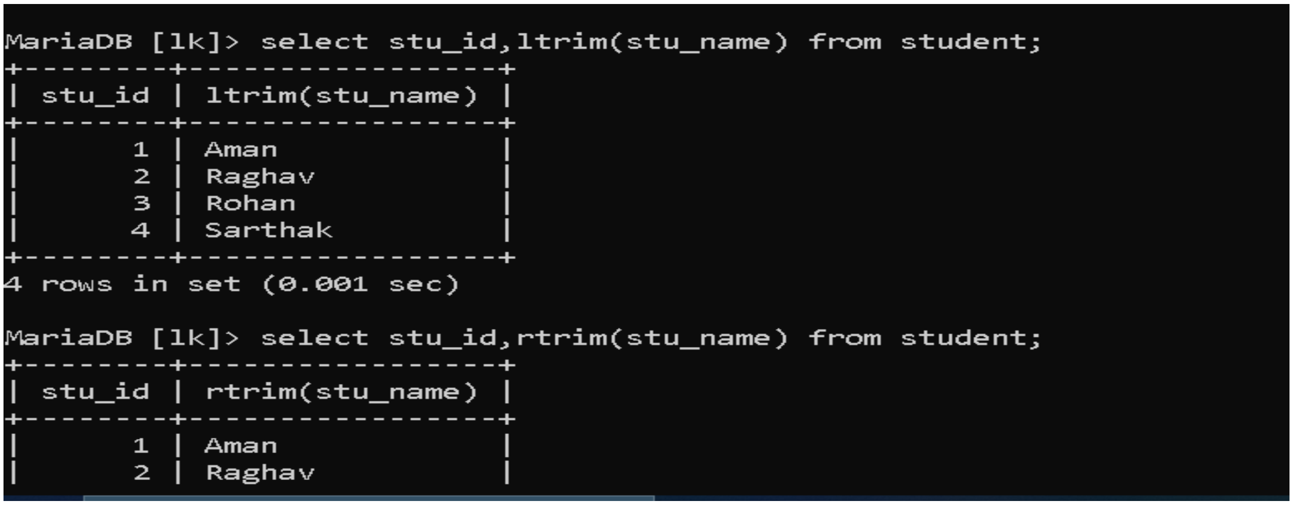
```
MariaDB [lk]> select stu_id,concat(stu_name,"is in ",stu_branch," branch ") from student;
+--------+--------------------------------------------------+
| stu_id | concat(stu_name,"is in ",stu_branch," branch ") |
+--------+--------------------------------------------------+
|      1 | Amanis in CSE branch                             |
|      2 | Raghavis in Me branch                            |
|      3 | Rohanis in Cse branch                            |
|      4 | Sarthakis in EEE branch                          |
+--------+--------------------------------------------------+
4 rows in set (0.000 sec)

MariaDB [lk]> select stu_id,lcase(stu_name) from student;
+--------+-----------------+
| stu_id | lcase(stu_name) |
+--------+-----------------+
|      1 | aman            |
|      2 | raghav          |
```

LOWER() , UPPER() :



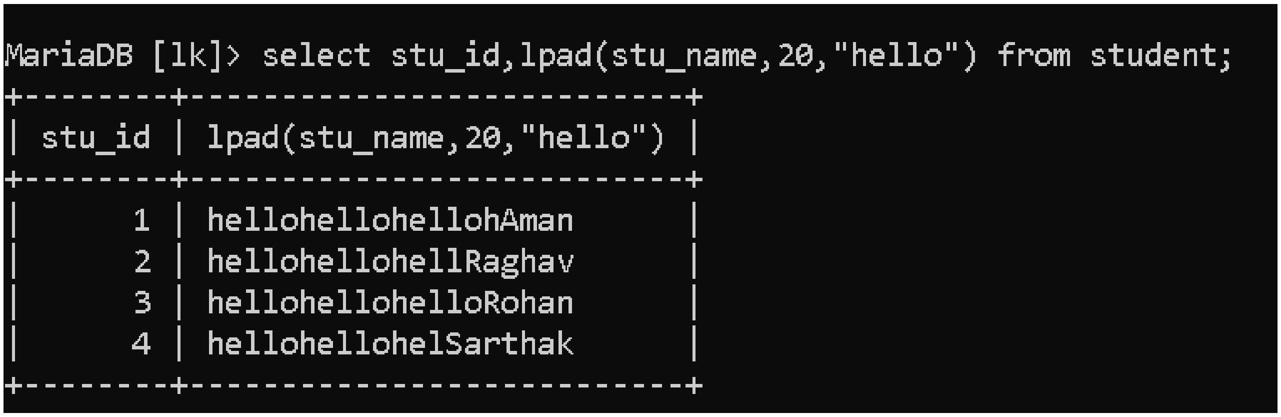
```
MariaDB [lk]> select stu_id,lower(stu_name) from student;
+--------+-----------------+
| stu_id | lower(stu_name) |
+--------+-----------------+
|      1 | aman            |
|      2 | raghav          |
|      3 | rohan           |
|      4 | sarthak         |
+--------+-----------------+
4 rows in set (0.000 sec)

MariaDB [lk]> select stu_id,upper(stu_name) from student;
+--------+-----------------+
| stu_id | upper(stu_name) |
+--------+-----------------+
|      1 | AMAN            |
|      2 | RAGHAV          |
|      3 | ROHAN           |
|      4 | SARTHAK         |
+--------+-----------------+
4 rows in set (0.000 sec)
```

LTRIM() , RTRIM() :



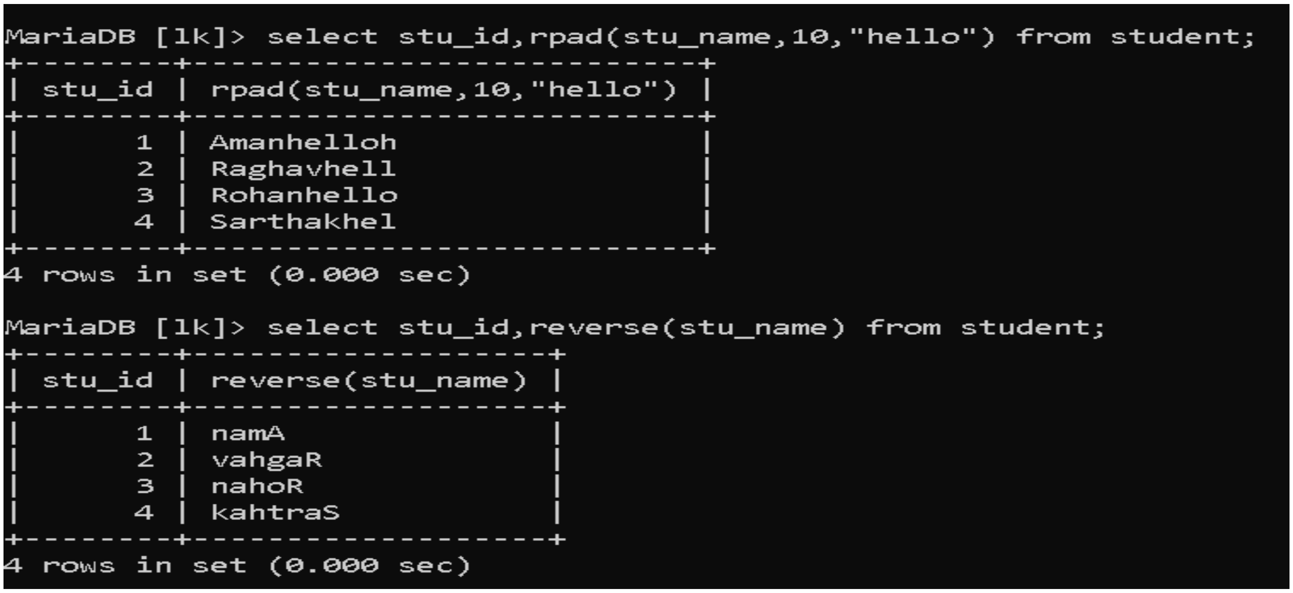
```
MariaDB [lk]> select stu_id,ltrim(stu_name) from student;
+--------+-----------------+
| stu_id | ltrim(stu_name) |
+--------+-----------------+
|      1 | Aman            |
|      2 | Raghav          |
|      3 | Rohan           |
|      4 | Sarthak         |
+--------+-----------------+
4 rows in set (0.001 sec)

MariaDB [lk]> select stu_id,rtrim(stu_name) from student;
+--------+-----------------+
| stu_id | rtrim(stu_name) |
+--------+-----------------+
|      1 | Aman            |
|      2 | Raghav          |
```

LPAD() :



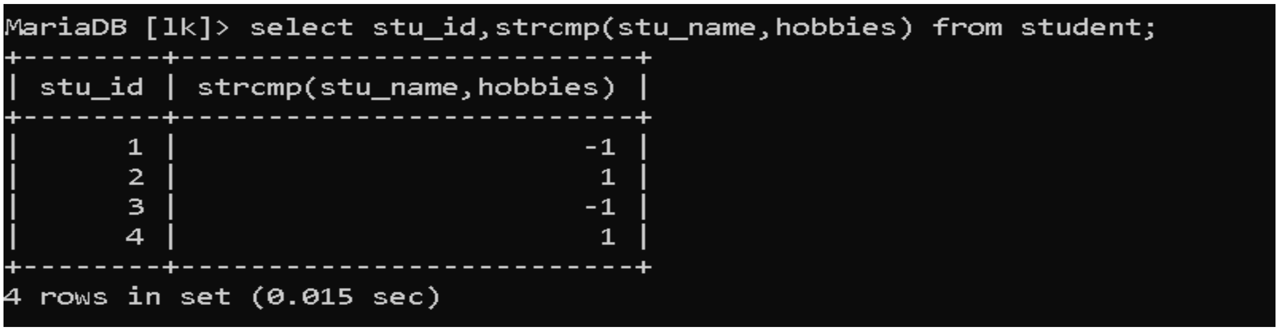
```
MariaDB [lk]> select stu_id,lpad(stu_name,20,"hello") from student;
+--------+---------------------------+
| stu_id | lpad(stu_name,20,"hello") |
+--------+---------------------------+
|      1 | hellohellohellohAman      |
|      2 | hellohellohellRaghav      |
|      3 | hellohellohelloRohan      |
|      4 | hellohellohelSarthak      |
+--------+---------------------------+
```

RPAD() , REVERSE() :



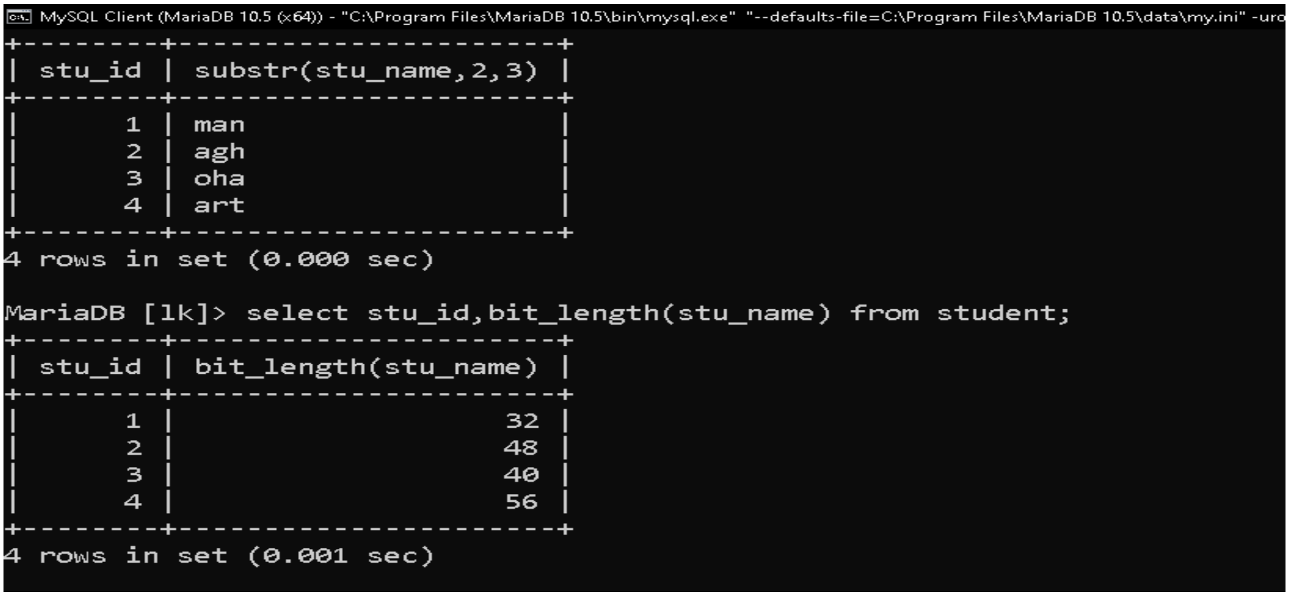
```
MariaDB [lk]> select stu_id,rpad(stu_name,10,"hello") from student;
+--------+---------------------------+
| stu_id | rpad(stu_name,10,"hello") |
+--------+---------------------------+
|      1 | Amanhelloh                |
|      2 | Raghavhell                |
|      3 | Rohanhello                |
|      4 | Sarthakhel                |
+--------+---------------------------+
4 rows in set (0.000 sec)

MariaDB [lk]> select stu_id,reverse(stu_name) from student;
+--------+-------------------+
| stu_id | reverse(stu_name) |
+--------+-------------------+
|      1 | namA              |
|      2 | vahgaR            |
|      3 | nahoR             |
|      4 | kahtraS           |
+--------+-------------------+
4 rows in set (0.000 sec)
```

STRCMP() :



```
MariaDB [lk]> select stu_id,strcmp(stu_name,hobbies) from student;
+--------+--------------------------+
| stu_id | strcmp(stu_name,hobbies) |
+--------+--------------------------+
|      1 |                       -1 |
|      2 |                        1 |
|      3 |                       -1 |
|      4 |                        1 |
+--------+--------------------------+
4 rows in set (0.015 sec)
```

SUBSTRING() , BIT_LENGTH() :



```
MySQL Client (MariaDB 10.5 (x64)) - "C:\Program Files\MariaDB 10.5\bin\mysql.exe"  "--defaults-file=C:\Program Files\MariaDB 10.5\data\my.ini" -uro
+--------+----------------------+
| stu_id | substr(stu_name,2,3) |
+--------+----------------------+
|      1 | man                  |
|      2 | agh                  |
|      3 | oha                  |
|      4 | art                  |
+--------+----------------------+
4 rows in set (0.000 sec)

MariaDB [lk]> select stu_id,bit_length(stu_name) from student;
+--------+----------------------+
| stu_id | bit_length(stu_name) |
+--------+----------------------+
|      1 |                   32 |
|      2 |                   48 |
|      3 |                   40 |
|      4 |                   56 |
+--------+----------------------+
4 rows in set (0.001 sec)
```

EXPERIMENT 5

AIM:

Implementation of Operators and Views on a given database.

THEORY:

SQL Arithmetic Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20,

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

SQL Logical Operators

Sr.No.	Operator & Description
1	**ALL** The ALL operator is used to compare a value to all values in another value set.

2	**AND** The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	**ANY** The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	**BETWEEN** The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	**EXISTS** The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	**IN** The IN operator is used to compare a value to a list of literal values that have been specified.
7	**LIKE** The LIKE operator is used to compare a value to similar values using wildcard operators.
8	**NOT** The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
9	**OR** The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

SQL Comparison Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then **−**

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

Views

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

Just like table query, we can query the view to view the data.

```
SELECT * FROM DetailsView;
```

3. Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
CREATE VIEW MarksView AS
SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

```
DROP VIEW view_name;
```

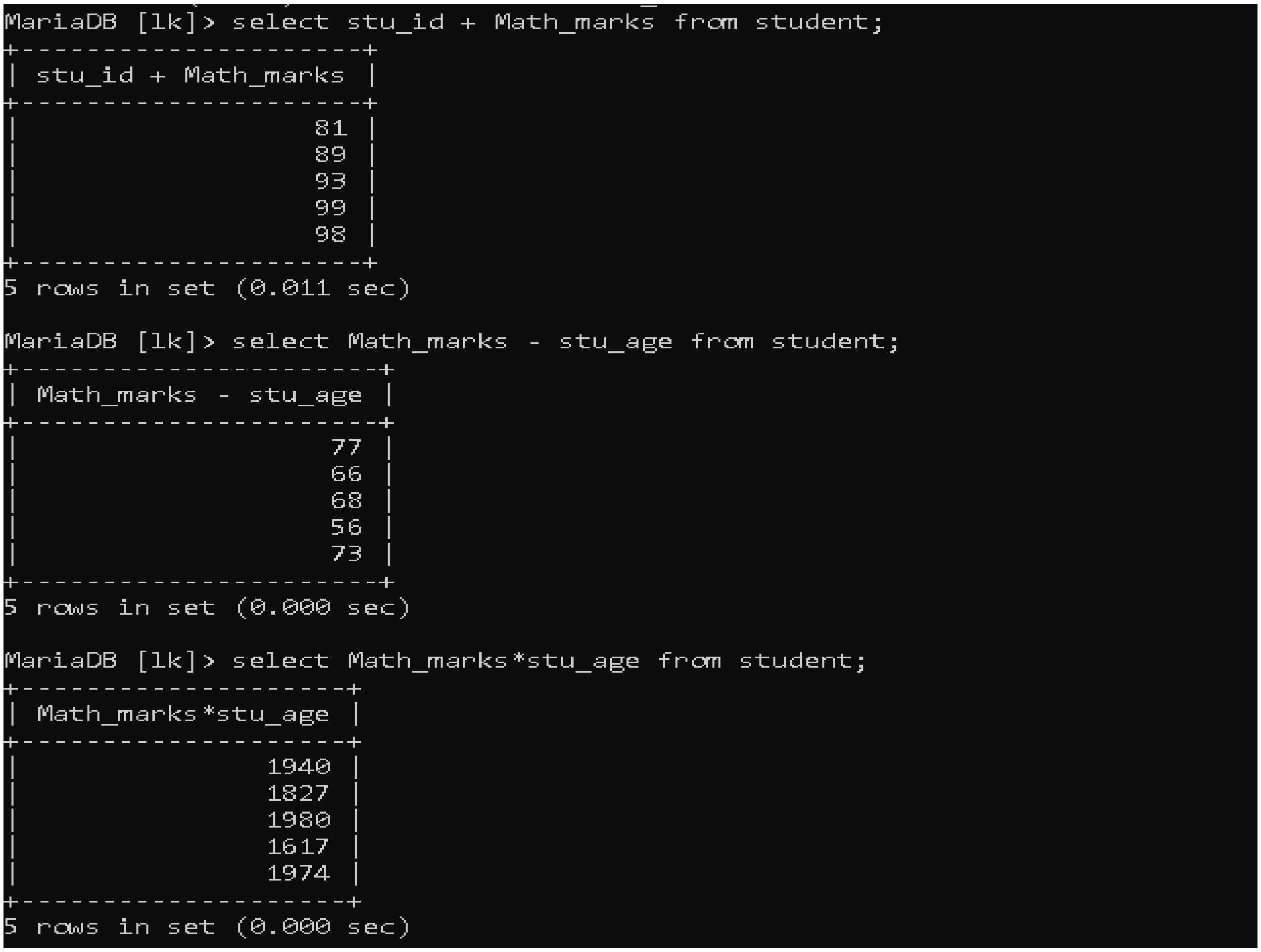
Example:

If we want to delete the View **MarksView**, we can do this as:

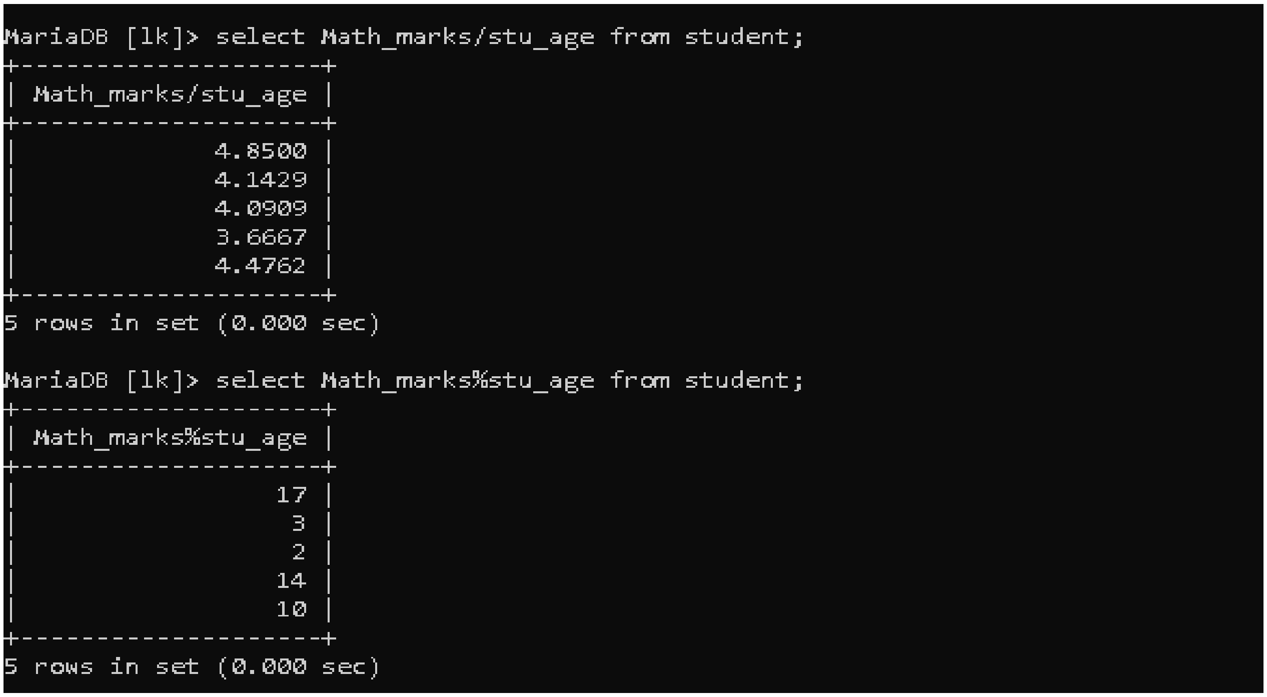
```
DROP VIEW MarksView;
```

OUTPUT:

1. Arithmetic Operators:

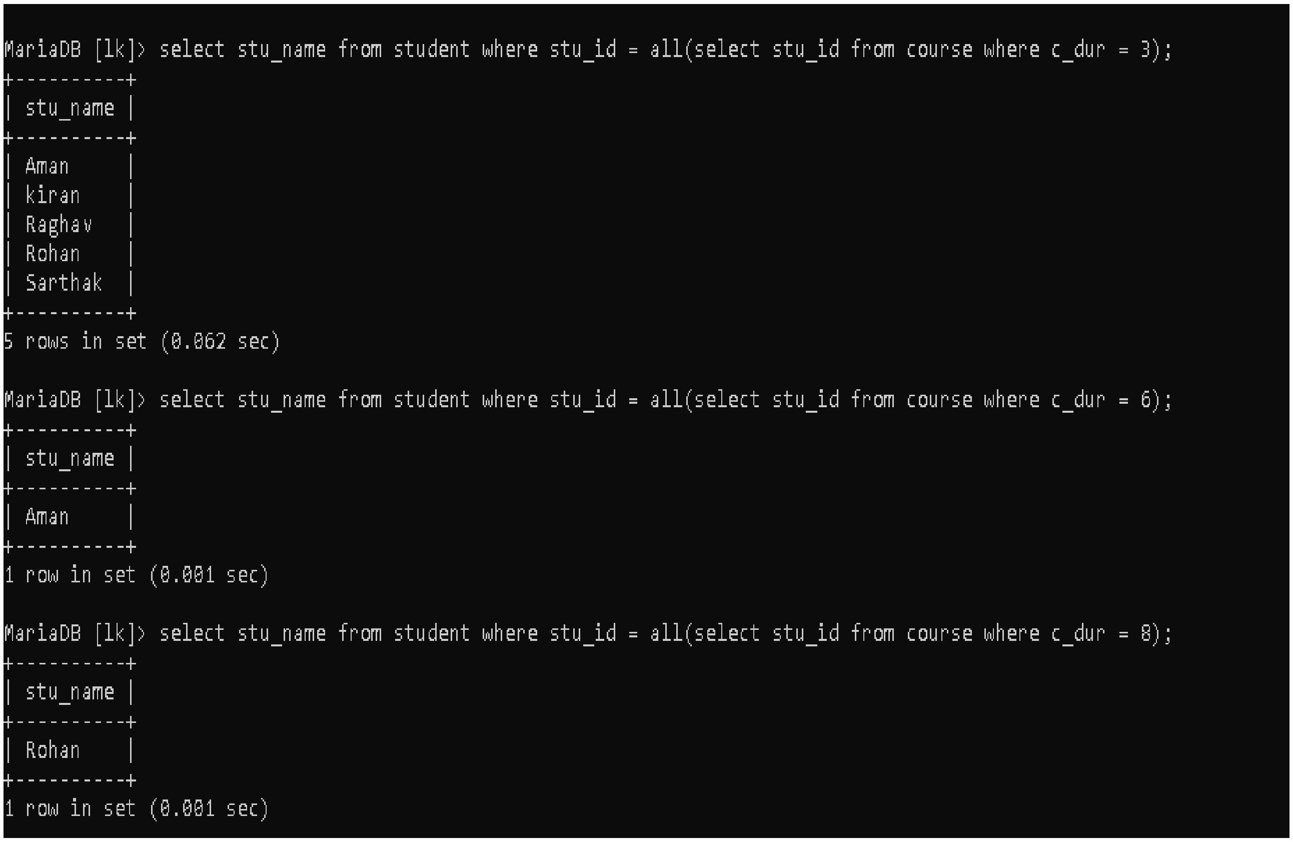


```
MariaDB [lk]> select stu_id + Math_marks from student;
+---------------------+
| stu_id + Math_marks |
+---------------------+
|                  81 |
|                  89 |
|                  93 |
|                  99 |
|                  98 |
+---------------------+
5 rows in set (0.011 sec)

MariaDB [lk]> select Math_marks - stu_age from student;
+----------------------+
| Math_marks - stu_age |
+----------------------+
|                   77 |
|                   66 |
|                   68 |
|                   56 |
|                   73 |
+----------------------+
5 rows in set (0.000 sec)

MariaDB [lk]> select Math_marks*stu_age from student;
+--------------------+
| Math_marks*stu_age |
+--------------------+
|               1940 |
|               1827 |
|               1980 |
|               1617 |
|               1974 |
+--------------------+
5 rows in set (0.000 sec)
```

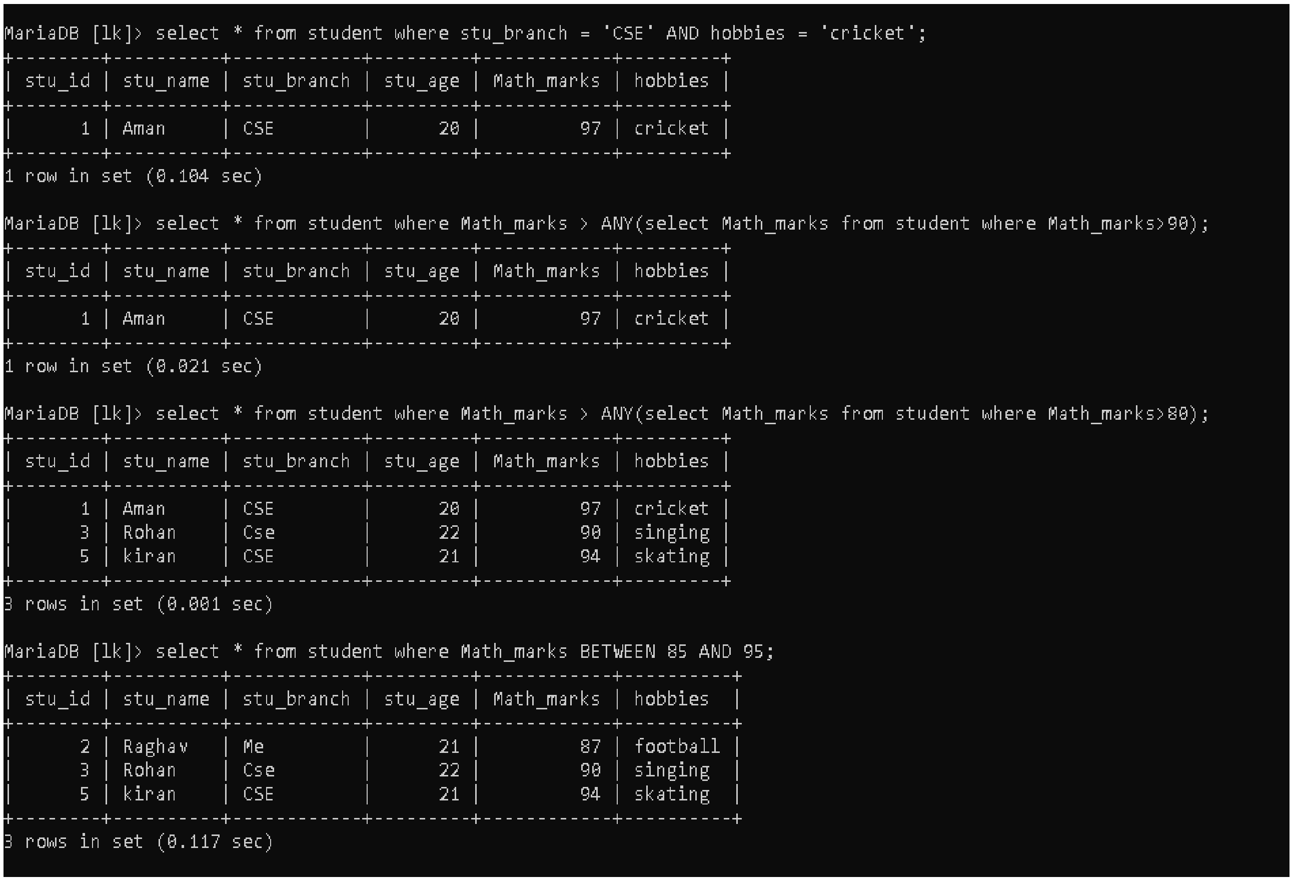


```
MariaDB [lk]> select Math_marks/stu_age from student;
+--------------------+
| Math_marks/stu_age |
+--------------------+
|             4.8500 |
|             4.1429 |
|             4.0909 |
|             3.6667 |
|             4.4762 |
+--------------------+
5 rows in set (0.000 sec)

MariaDB [lk]> select Math_marks%stu_age from student;
+--------------------+
| Math_marks%stu_age |
+--------------------+
|                 17 |
|                  3 |
|                  2 |
|                 14 |
|                 10 |
+--------------------+
5 rows in set (0.000 sec)
```

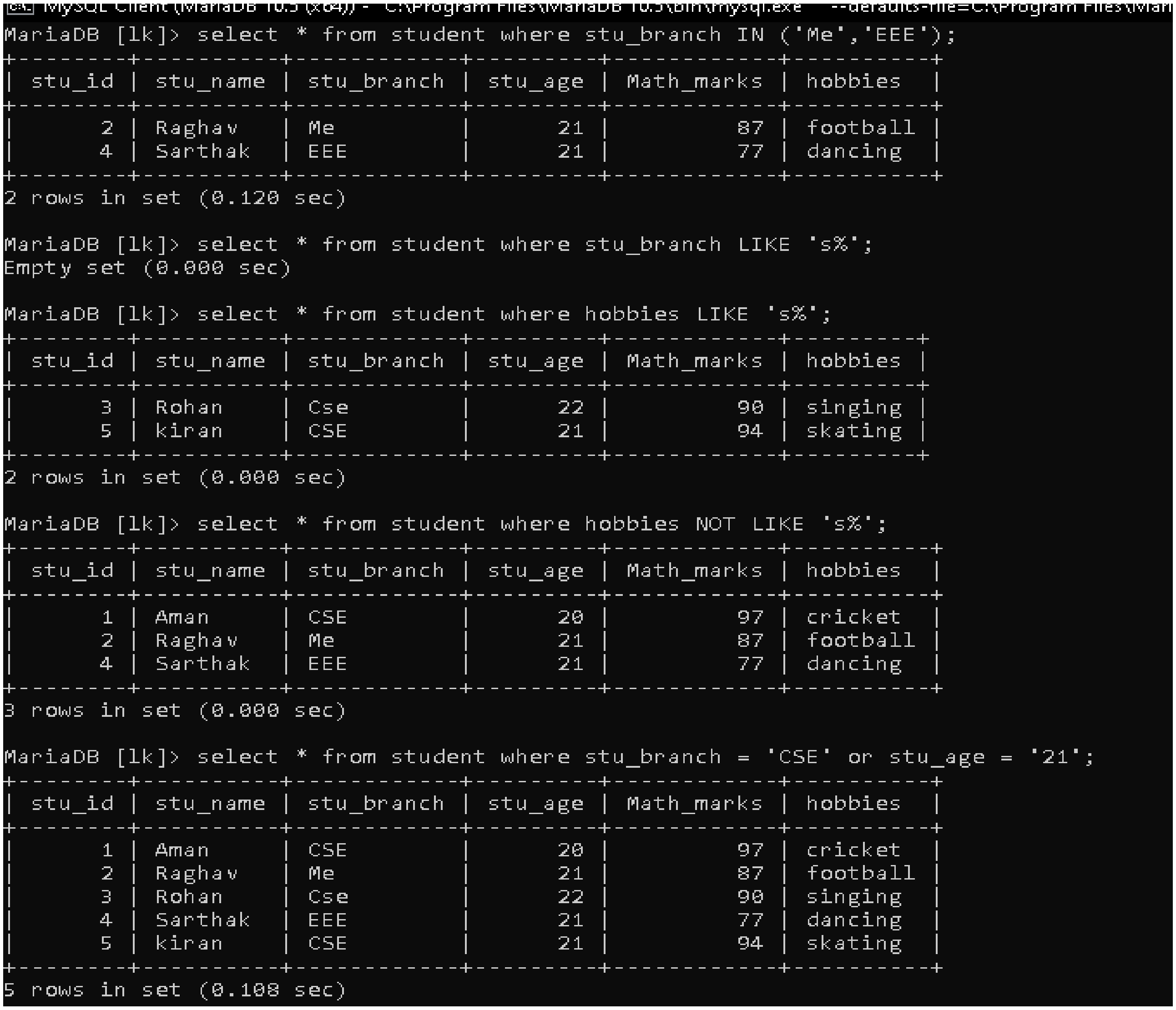
2. Logical Operators :



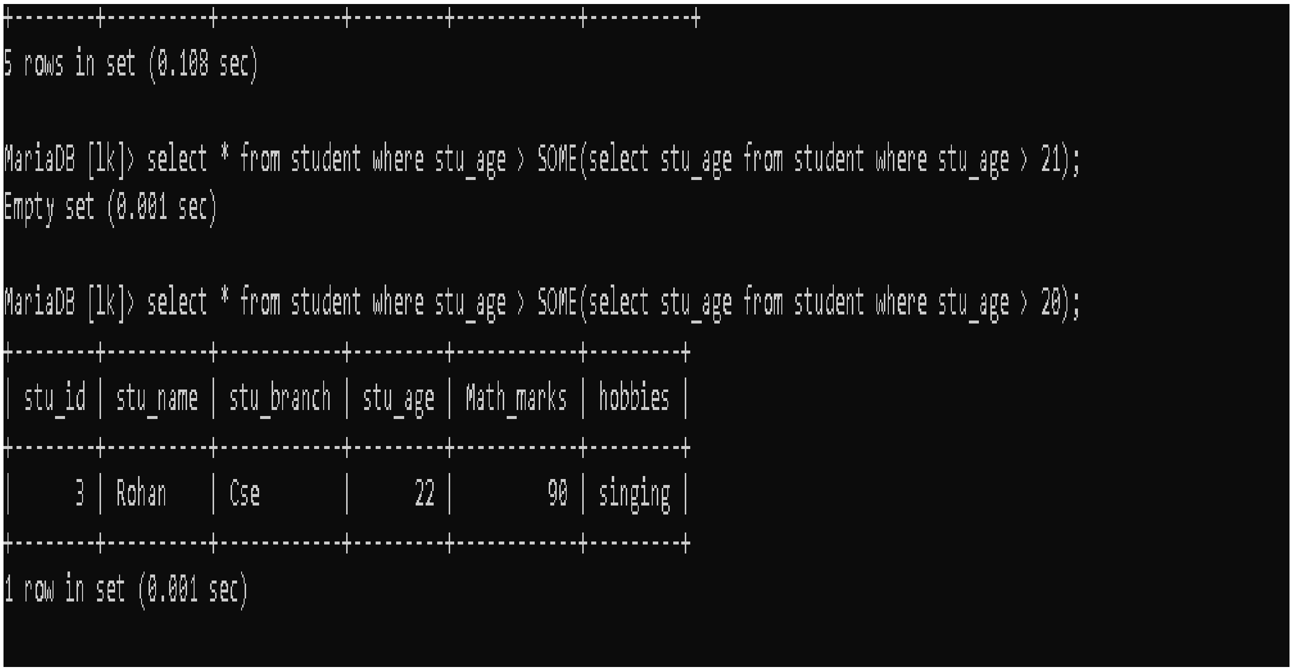
```
MariaDB [lk]> select stu_name from student where stu_id = all(select stu_id from course where c_dur = 3);
+----------+
| stu_name |
+----------+
| Aman     |
| kiran    |
| Raghav   |
| Rohan    |
| Sarthak  |
+----------+
5 rows in set (0.062 sec)

MariaDB [lk]> select stu_name from student where stu_id = all(select stu_id from course where c_dur = 6);
+----------+
| stu_name |
+----------+
| Aman     |
+----------+
1 row in set (0.001 sec)

MariaDB [lk]> select stu_name from student where stu_id = all(select stu_id from course where c_dur = 8);
+----------+
| stu_name |
+----------+
| Rohan    |
+----------+
1 row in set (0.001 sec)
```



```
MariaDB [lk]> select * from student where stu_branch = 'CSE' AND hobbies = 'cricket';
+--------+----------+------------+---------+------------+---------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies |
+--------+----------+------------+---------+------------+---------+
|      1 | Aman     | CSE        |      20 |         97 | cricket |
+--------+----------+------------+---------+------------+---------+
1 row in set (0.104 sec)

MariaDB [lk]> select * from student where Math_marks > ANY(select Math_marks from student where Math_marks>90);
+--------+----------+------------+---------+------------+---------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies |
+--------+----------+------------+---------+------------+---------+
|      1 | Aman     | CSE        |      20 |         97 | cricket |
+--------+----------+------------+---------+------------+---------+
1 row in set (0.021 sec)

MariaDB [lk]> select * from student where Math_marks > ANY(select Math_marks from student where Math_marks>80);
+--------+----------+------------+---------+------------+---------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies |
+--------+----------+------------+---------+------------+---------+
|      1 | Aman     | CSE        |      20 |         97 | cricket |
|      3 | Rohan    | Cse        |      22 |         90 | singing |
|      5 | kiran    | CSE        |      21 |         94 | skating |
+--------+----------+------------+---------+------------+---------+
3 rows in set (0.001 sec)

MariaDB [lk]> select * from student where Math_marks BETWEEN 85 AND 95;
+--------+----------+------------+---------+------------+----------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies  |
+--------+----------+------------+---------+------------+----------+
|      2 | Raghav   | Me         |      21 |         87 | football |
|      3 | Rohan    | Cse        |      22 |         90 | singing  |
|      5 | kiran    | CSE        |      21 |         94 | skating  |
+--------+----------+------------+---------+------------+----------+
3 rows in set (0.117 sec)
```

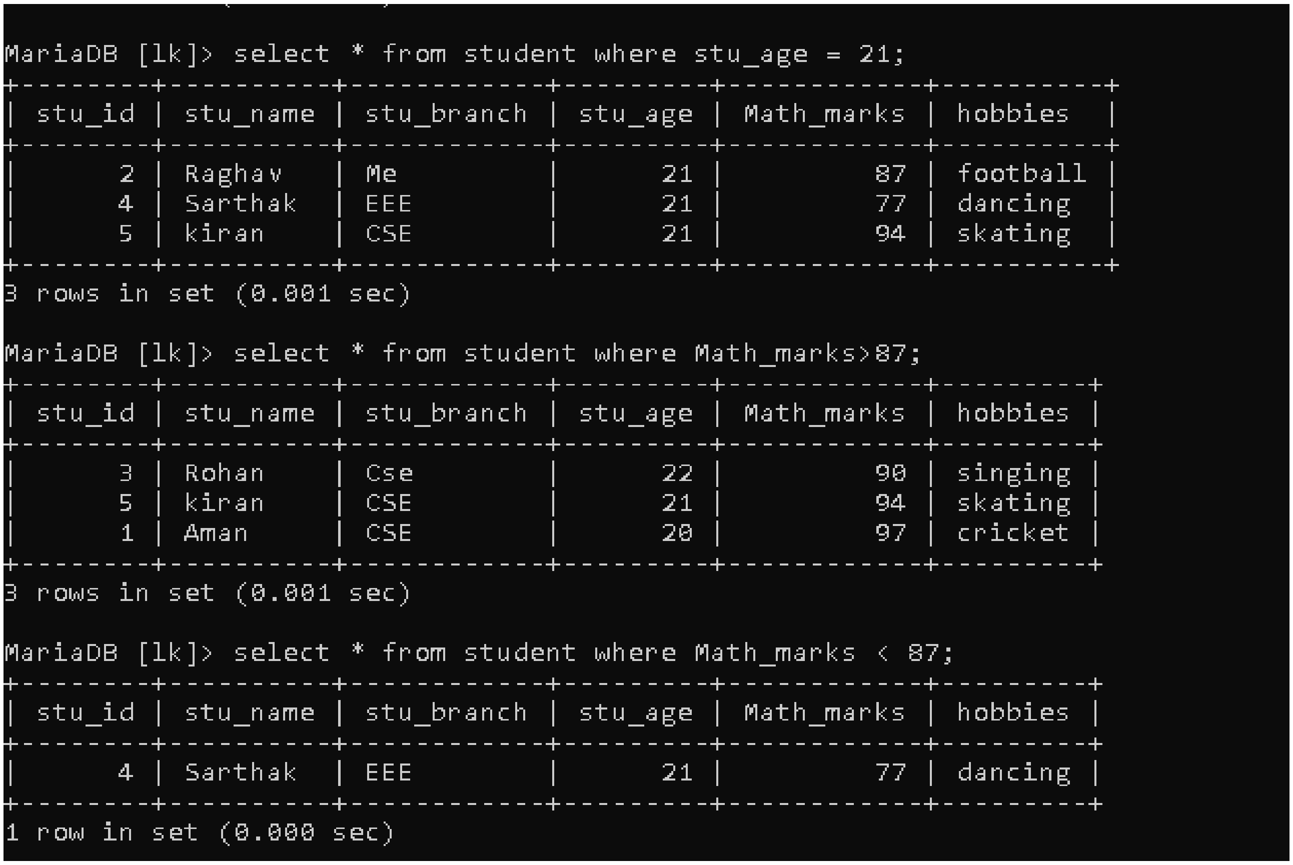

MySQL Client (MariaDB 10.5 (x64)) - C:\Program Files\MariaDB 10.5\bin\mysql.exe --defaults-file=C:\Program Files\Mari

```
MariaDB [lk]> select * from student where stu_branch IN ('Me','EEE');
+--------+----------+------------+---------+------------+----------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies  |
+--------+----------+------------+---------+------------+----------+
|      2 | Raghav   | Me         |      21 |         87 | football |
|      4 | Sarthak  | EEE        |      21 |         77 | dancing  |
+--------+----------+------------+---------+------------+----------+
2 rows in set (0.120 sec)

MariaDB [lk]> select * from student where stu_branch LIKE 's%';
Empty set (0.000 sec)

MariaDB [lk]> select * from student where hobbies LIKE 's%';
+--------+----------+------------+---------+------------+---------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies |
+--------+----------+------------+---------+------------+---------+
|      3 | Rohan    | Cse        |      22 |         90 | singing |
|      5 | kiran    | CSE        |      21 |         94 | skating |
+--------+----------+------------+---------+------------+---------+
2 rows in set (0.000 sec)

MariaDB [lk]> select * from student where hobbies NOT LIKE 's%';
+--------+----------+------------+---------+------------+----------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies  |
+--------+----------+------------+---------+------------+----------+
|      1 | Aman     | CSE        |      20 |         97 | cricket  |
|      2 | Raghav   | Me         |      21 |         87 | football |
|      4 | Sarthak  | EEE        |      21 |         77 | dancing  |
+--------+----------+------------+---------+------------+----------+
3 rows in set (0.000 sec)

MariaDB [lk]> select * from student where stu_branch = 'CSE' or stu_age = '21';
+--------+----------+------------+---------+------------+----------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies  |
+--------+----------+------------+---------+------------+----------+
|      1 | Aman     | CSE        |      20 |         97 | cricket  |
|      2 | Raghav   | Me         |      21 |         87 | football |
|      3 | Rohan    | Cse        |      22 |         90 | singing  |
|      4 | Sarthak  | EEE        |      21 |         77 | dancing  |
|      5 | kiran    | CSE        |      21 |         94 | skating  |
+--------+----------+------------+---------+------------+----------+
5 rows in set (0.108 sec)
```

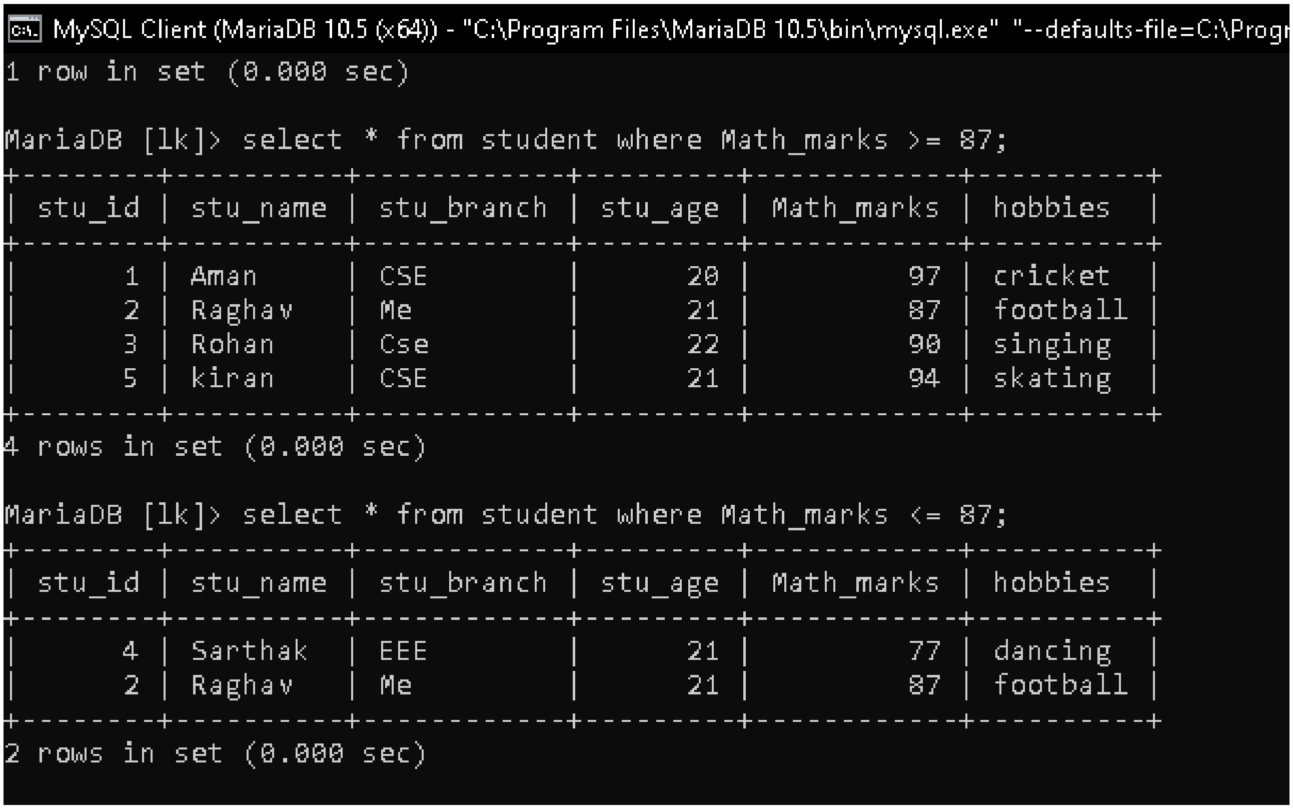


```
+--------+----------+------------+---------+------------+----------+
5 rows in set (0.108 sec)

MariaDB [lk]> select * from student where stu_age > SOME(select stu_age from student where stu_age > 21);
Empty set (0.001 sec)

MariaDB [lk]> select * from student where stu_age > SOME(select stu_age from student where stu_age > 20);
+--------+----------+------------+---------+------------+---------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies |
+--------+----------+------------+---------+------------+---------+
|      3 | Rohan    | Cse        |      22 |         90 | singing |
+--------+----------+------------+---------+------------+---------+
1 row in set (0.001 sec)
```

3. Comparison Operators :



```
MariaDB [lk]> select * from student where stu_age = 21;
+--------+----------+------------+---------+------------+----------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies  |
+--------+----------+------------+---------+------------+----------+
|      2 | Raghav   | Me         |      21 |         87 | football |
|      4 | Sarthak  | EEE        |      21 |         77 | dancing  |
|      5 | kiran    | CSE        |      21 |         94 | skating  |
+--------+----------+------------+---------+------------+----------+
3 rows in set (0.001 sec)

MariaDB [lk]> select * from student where Math_marks>87;
+--------+----------+------------+---------+------------+---------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies |
+--------+----------+------------+---------+------------+---------+
|      3 | Rohan    | Cse        |      22 |         90 | singing |
|      5 | kiran    | CSE        |      21 |         94 | skating |
|      1 | Aman     | CSE        |      20 |         97 | cricket |
+--------+----------+------------+---------+------------+---------+
3 rows in set (0.001 sec)

MariaDB [lk]> select * from student where Math_marks < 87;
+--------+----------+------------+---------+------------+---------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies |
+--------+----------+------------+---------+------------+---------+
|      4 | Sarthak  | EEE        |      21 |         77 | dancing |
+--------+----------+------------+---------+------------+---------+
1 row in set (0.000 sec)
```

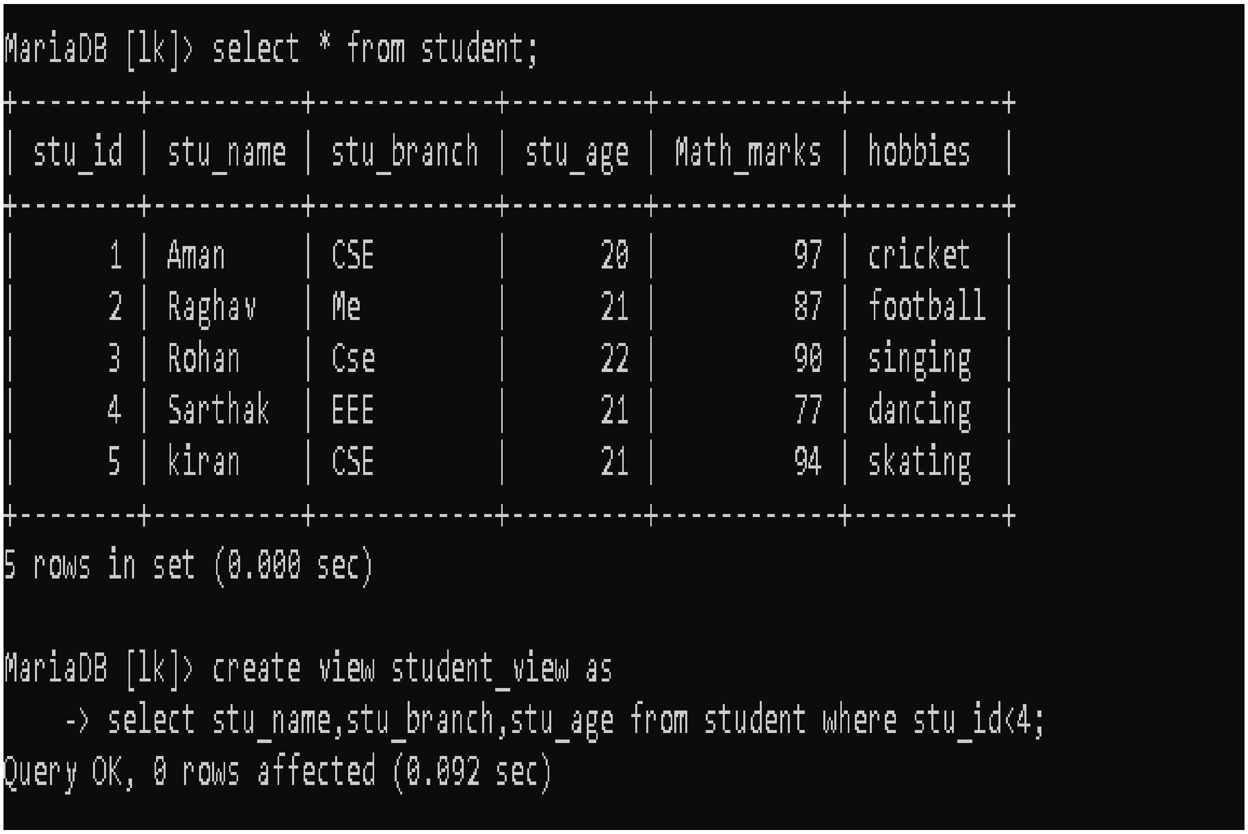


MySQL Client (MariaDB 10.5 (x64)) - "C:\Program Files\MariaDB 10.5\bin\mysql.exe" "--defaults-file=C:\Progr

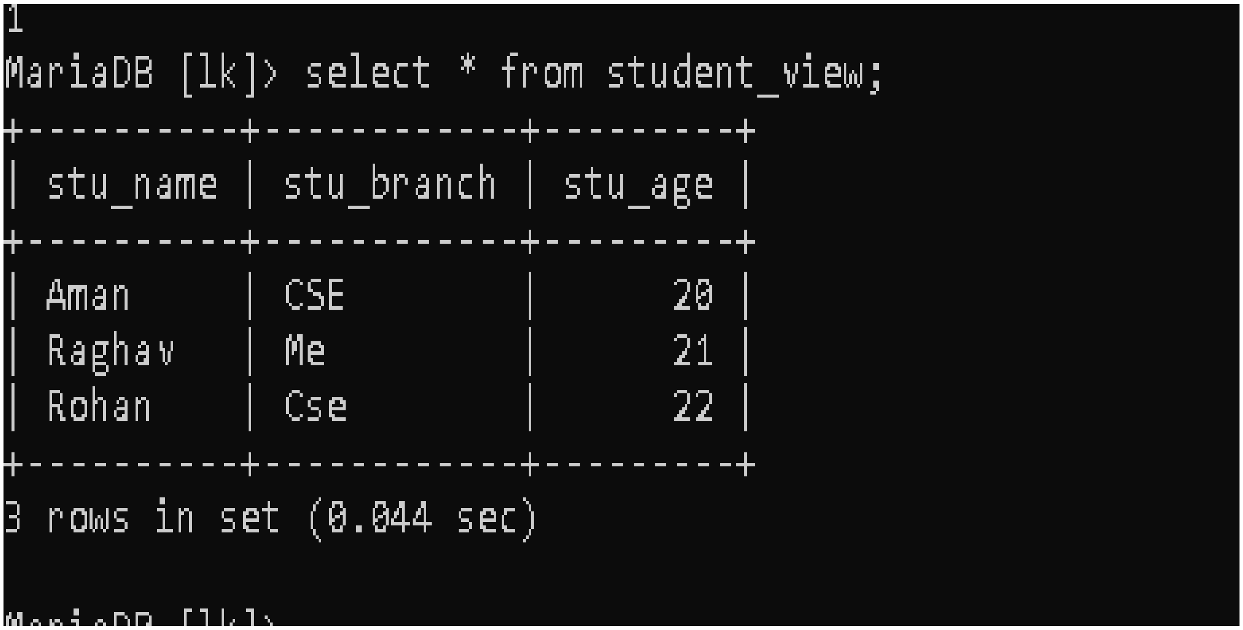
```
1 row in set (0.000 sec)

MariaDB [lk]> select * from student where Math_marks >= 87;
+--------+----------+------------+---------+------------+----------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies  |
+--------+----------+------------+---------+------------+----------+
|      1 | Aman     | CSE        |      20 |         97 | cricket  |
|      2 | Raghav   | Me         |      21 |         87 | football |
|      3 | Rohan    | Cse        |      22 |         90 | singing  |
|      5 | kiran    | CSE        |      21 |         94 | skating  |
+--------+----------+------------+---------+------------+----------+
4 rows in set (0.000 sec)

MariaDB [lk]> select * from student where Math_marks <= 87;
+--------+----------+------------+---------+------------+----------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies  |
+--------+----------+------------+---------+------------+----------+
|      4 | Sarthak  | EEE        |      21 |         77 | dancing  |
|      2 | Raghav   | Me         |      21 |         87 | football |
+--------+----------+------------+---------+------------+----------+
2 rows in set (0.000 sec)
```

VIEWS

1. Creating a view:

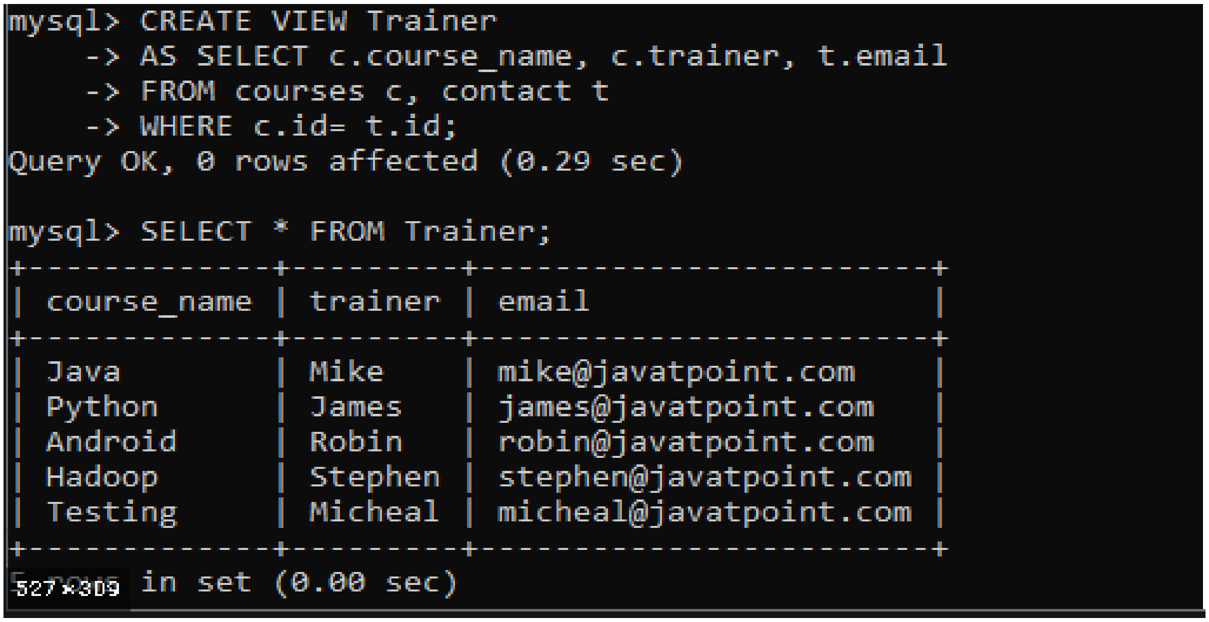


```
MariaDB [lk]> select * from student;
+--------+----------+------------+---------+------------+----------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies  |
+--------+----------+------------+---------+------------+----------+
|      1 | Aman     | CSE        |      20 |         97 | cricket  |
|      2 | Raghav   | Me         |      21 |         87 | football |
|      3 | Rohan    | Cse        |      22 |         90 | singing  |
|      4 | Sarthak  | EEE        |      21 |         77 | dancing  |
|      5 | kiran    | CSE        |      21 |         94 | skating  |
+--------+----------+------------+---------+------------+----------+
5 rows in set (0.000 sec)

MariaDB [lk]> create view student_view as
    -> select stu_name,stu_branch,stu_age from student where stu_id<4;
Query OK, 0 rows affected (0.092 sec)
```



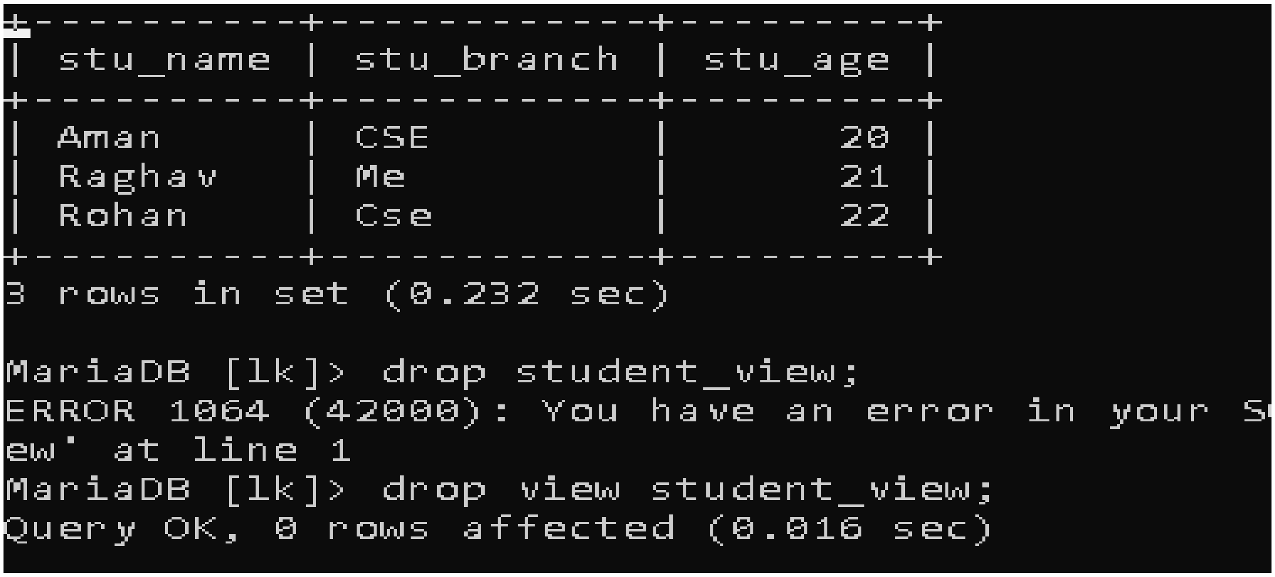
```
1
MariaDB [lk]> select * from student_view;
+----------+------------+---------+
| stu_name | stu_branch | stu_age |
+----------+------------+---------+
| Aman     | CSE        |      20 |
| Raghav   | Me         |      21 |
| Rohan    | Cse        |      22 |
+----------+------------+---------+
3 rows in set (0.044 sec)

MariaDB [lk]>
```

2. **Creating View from multiple tables**



```
mysql> CREATE VIEW Trainer
    -> AS SELECT c.course_name, c.trainer, t.email
    -> FROM courses c, contact t
    -> WHERE c.id= t.id;
Query OK, 0 rows affected (0.29 sec)

mysql> SELECT * FROM Trainer;
+-------------+---------+------------------------+
| course_name | trainer | email                  |
+-------------+---------+------------------------+
| Java        | Mike    | mike@javatpoint.com    |
| Python      | James   | james@javatpoint.com   |
| Android     | Robin   | robin@javatpoint.com   |
| Hadoop      | Stephen | stephen@javatpoint.com |
| Testing     | Micheal | micheal@javatpoint.com |
+-------------+---------+------------------------+
5 rows in set (0.00 sec)
```

3. **Deleting View**



```
+----------+------------+---------+
| stu_name | stu_branch | stu_age |
+----------+------------+---------+
| Aman     | CSE        |      20 |
| Raghav   | Me         |      21 |
| Rohan    | Cse        |      22 |
+----------+------------+---------+
3 rows in set (0.232 sec)

MariaDB [lk]> drop student_view;
ERROR 1064 (42000): You have an error in your S
ew' at line 1
MariaDB [lk]> drop view student_view;
Query OK, 0 rows affected (0.016 sec)
```

EXPERIMENT 6

AIM:

Implementation of Joins and Index on a given Database.

THEORY:

SQL JOIN

As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".

In SQL, JOIN clause is used to combine the records from two or more tables in a database.

Types of SQL JOIN

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN

1. INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

2. LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
```

```
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

3. RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

4. FULL JOIN

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

SQL Index

- Indexes are special lookup tables. It is used to retrieve data from the database very fast.
- An Index is used to speed up select queries and where clauses. But it shows down the data input with insert and update statements. Indexes can be created or dropped without affecting the data.
- An index in a database is just like an index in the back of a book.

- **For example:** When you reference all pages in a book that discusses a certain topic, you first have to refer to the index, which alphabetically lists all the topics and then referred to one or more specific page numbers.

1. Create Index statement

It is used to create an index on a table. It allows duplicate value.

Syntax

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

2. Unique Index statement

It is used to create a unique index on a table. It does not allow duplicate value.

Syntax

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

3. Drop Index Statement

It is used to delete an index in a table.

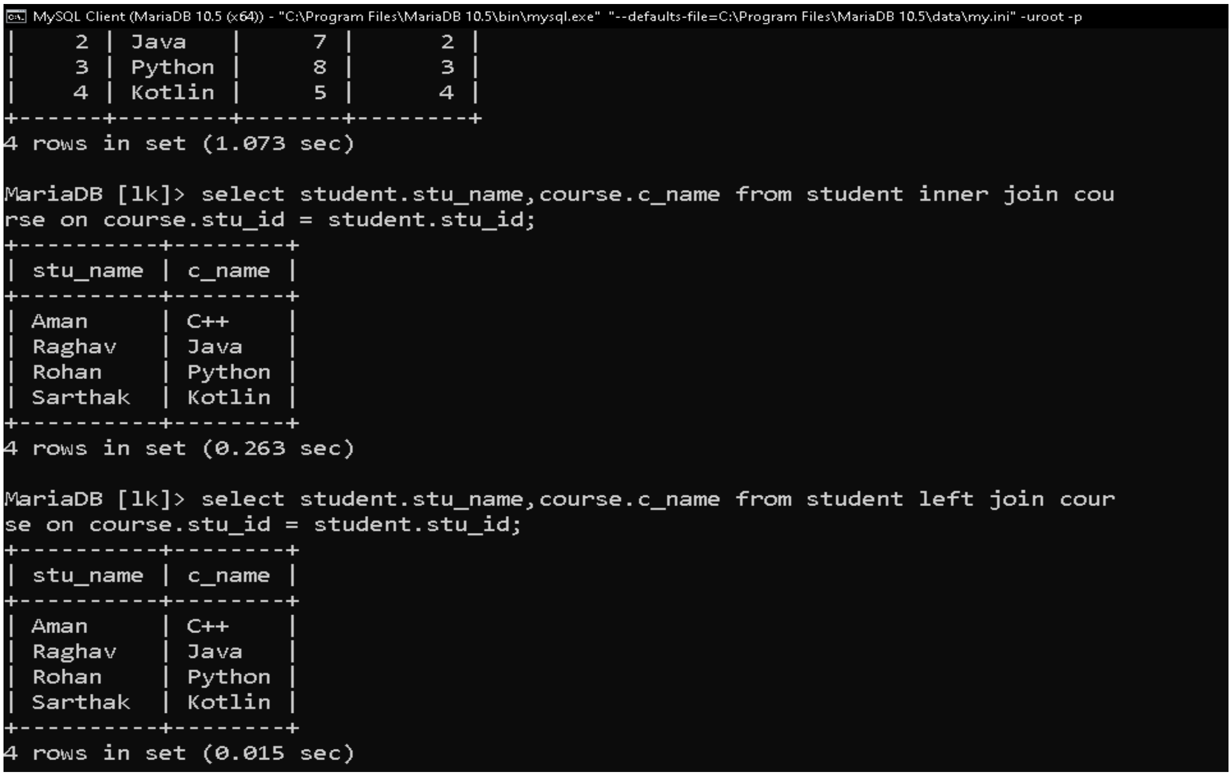
Syntax

```
    DROP INDEX index_name;
```

OUTPUT:

JOINS

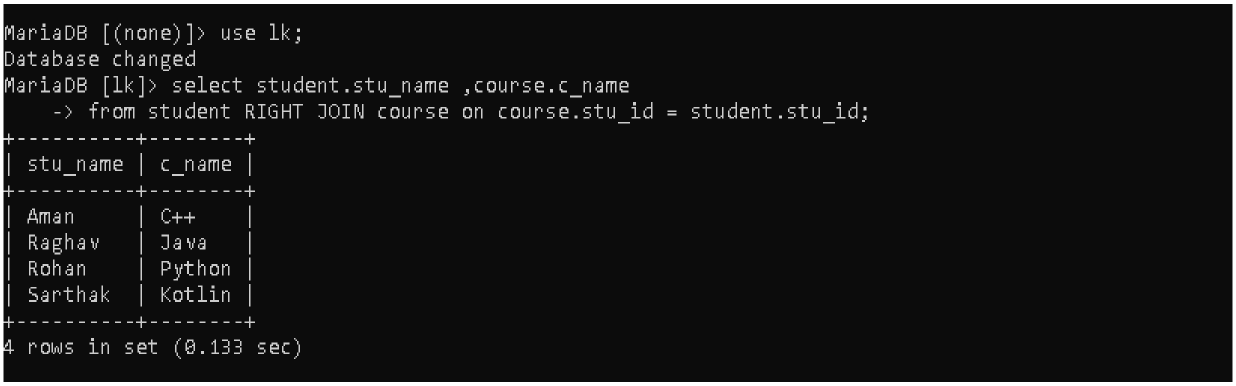
1. INNER JOIN & LEFT JOIN



MySQL Client (MariaDB 10.5 (x64)) - "C:\Program Files\MariaDB 10.5\bin\mysql.exe" "--defaults-file=C:\Program Files\MariaDB 10.5\data\my.ini" -uroot -p

```
|    2 | Java   |     7 |      2 |
|    3 | Python |     8 |      3 |
|    4 | Kotlin |     5 |      4 |
+------+--------+------+--------+
4 rows in set (1.073 sec)

MariaDB [lk]> select student.stu_name,course.c_name from student inner join cou
rse on course.stu_id = student.stu_id;
+----------+--------+
| stu_name | c_name |
+----------+--------+
| Aman     | C++    |
| Raghav   | Java   |
| Rohan    | Python |
| Sarthak  | Kotlin |
+----------+--------+
4 rows in set (0.263 sec)

MariaDB [lk]> select student.stu_name,course.c_name from student left join cour
se on course.stu_id = student.stu_id;
+----------+--------+
| stu_name | c_name |
+----------+--------+
| Aman     | C++    |
| Raghav   | Java   |
| Rohan    | Python |
| Sarthak  | Kotlin |
+----------+--------+
4 rows in set (0.015 sec)
```

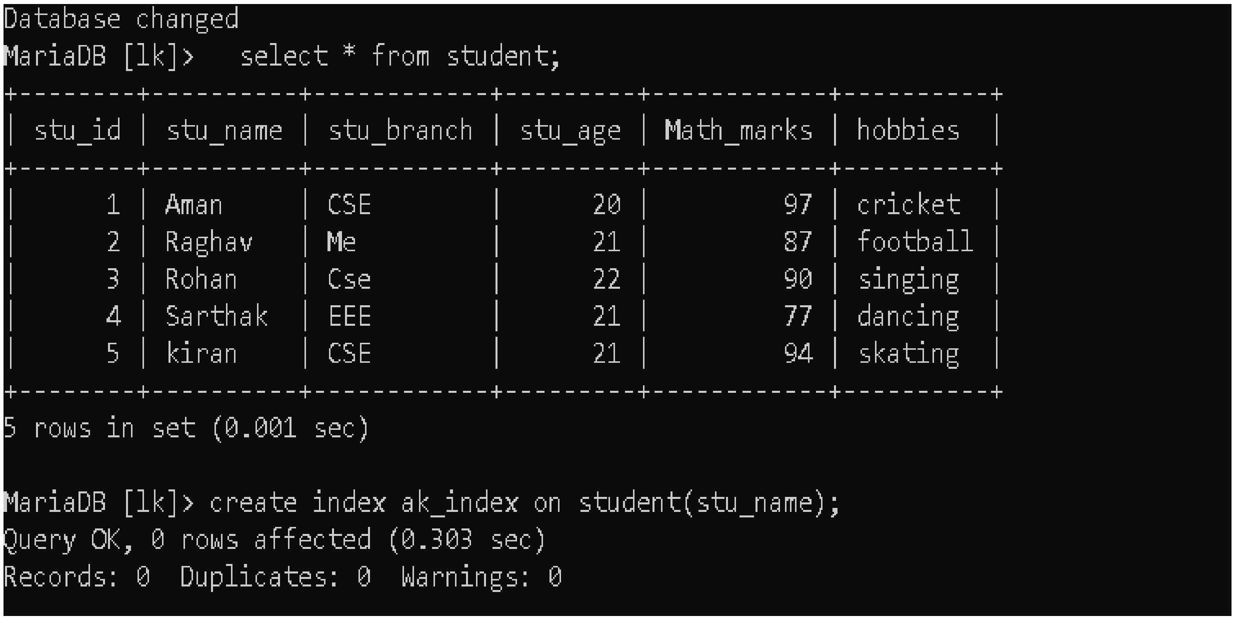
2. RIGHT JOIN:



```
MariaDB [(none)]> use lk;
Database changed
MariaDB [lk]> select student.stu_name ,course.c_name
    -> from student RIGHT JOIN course on course.stu_id = student.stu_id;
+----------+--------+
| stu_name | c_name |
+----------+--------+
| Aman     | C++    |
| Raghav   | Java   |
| Rohan    | Python |
| Sarthak  | Kotlin |
+----------+--------+
4 rows in set (0.133 sec)
```

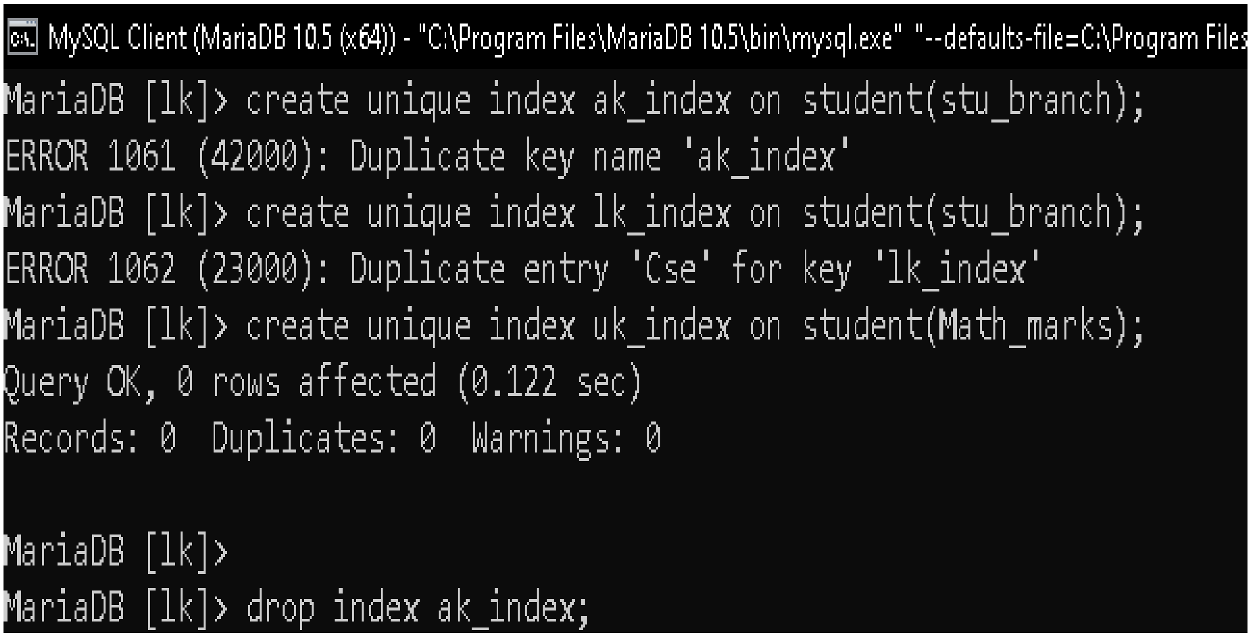
INDEX

1. Create Index



```
Database changed
MariaDB [lk]>   select * from student;
+--------+----------+------------+---------+------------+----------+
| stu_id | stu_name | stu_branch | stu_age | Math_marks | hobbies  |
+--------+----------+------------+---------+------------+----------+
|      1 | Aman     | CSE        |      20 |         97 | cricket  |
|      2 | Raghav   | Me         |      21 |         87 | football |
|      3 | Rohan    | Cse        |      22 |         90 | singing  |
|      4 | Sarthak  | EEE        |      21 |         77 | dancing  |
|      5 | kiran    | CSE        |      21 |         94 | skating  |
+--------+----------+------------+---------+------------+----------+
5 rows in set (0.001 sec)

MariaDB [lk]> create index ak_index on student(stu_name);
Query OK, 0 rows affected (0.303 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

2. Create Unique Index & Drop Index


MySQL Client (MariaDB 10.5 (x64)) - "C:\Program Files\MariaDB 10.5\bin\mysql.exe" "--defaults-file=C:\Program Files

```
MariaDB [lk]> create unique index ak_index on student(stu_branch);
ERROR 1061 (42000): Duplicate key name 'ak_index'
MariaDB [lk]> create unique index lk_index on student(stu_branch);
ERROR 1062 (23000): Duplicate entry 'Cse' for key 'lk_index'
MariaDB [lk]> create unique index uk_index on student(Math_marks);
Query OK, 0 rows affected (0.122 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [lk]>
MariaDB [lk]> drop index ak_index;
```

EXPERIMENT 7

AIM:

To study the PL/SQL Database Management language.

THEORY:

PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL. PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.

Features of PL/SQL

- completely portable, high-performance transaction-processing language
- provides a built-in, interpreted and OS independent programming environment
- direct call can also be made from external programming language calls to database
- available in Times Ten in-memory database and IBM DB2

Advantages of PL/SQL

- PL/SQL provides high security level
- PL/SQL provides access to predefined SQL packages
- PL/SQL provides support for Object-Oriented Programming
- PL/SQL provides support for developing Web Applications and Server Pages.

PL/SQL Identifiers

PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifiers consist of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.

By default, identifiers are not case-sensitive.

PL/SQL Delimiters

A delimiter is a symbol with a special meaning. Following is the list of delimiters in PL/SQL –

+, -, *, /	Addition, subtraction/negation, multiplication, division
%	Attribute indicator
'	Character string delimiter
.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator
"	Quoted identifier delimiter
=	Relational operator
@	Remote access indicator
;	Statement terminator
:=	Assignment operator
=>	Association operator
||	Concatenation operator

**	Exponentiation operator
<<, >>	Label delimiter (begin and end)
/*, */	Multi-line comment delimiter (begin and end)
--	Single-line comment indicator
..	Range operator
<, >, <=, >=	Relational operators
<>, '=, ~=, ^=	Different versions of NOT EQUAL

PL/SQL Scalar Data Types

PL/SQL Scalar Data Types and Subtypes come under the following categories –

S.No	Date Type & Description
1	Numeric Numeric values on which arithmetic operations are performed.
2	Character

	Alphanumeric values that represent single characters or strings of characters.
3	Boolean Logical values on which logical operations are performed.
4	Datetime Dates and times.

PL/SQL Code:

```
DECLARE
   num1 INTEGER;
   num2 REAL;
   num3 DOUBLE PRECISION;
BEGIN
   null;
END;
/
```

OUTPUT:

```
PL/SQL procedure successfully completed
```

EXPERIMENT 8

AIM:

Implementation of Operators and Views on a given database.

THEORY:

Cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors.

Implicit Cursors: Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.

Explicit Cursors: Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

Implicit Cursors Attributes

S.No	Attribute & Description
1	%FOUND Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

2	%NOTFOUND The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	%ISOPEN Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	%ROWCOUNT Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Creating Explicit Cursors

1. **Creating cursor object**
   Syntax: DECLARE cursor_name CURSOR FOR SELECT * FROM table_name;

2. **Open Cursor Connection**
   Syntax: OPEN cursor_connection;

3. **Fetching Data from Cursors**
   There are a total of 6 methods to access data from the cursor. They are as follows:

   FIRST is used to fetch only the first row from the cursor table.
   LAST is used to fetch only the last row from the cursor table.

NEXT is used to fetch data in forward direction from the cursor table.
PRIOR is used to fetch data in backward direction from the cursor table.
ABSOLUTE n is used to fetch the exact nth row from the cursor table.
RELATIVE n is used to fetch the data in incremental as well as decremental ways.

Syntax:
FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor_name

4. **Close Cursor Connection**
   Syntax: CLOSE cursor_name

5. **Deallocate cursor memory.**
   Syntax: DEALLOCATE cursor_name

Code:

```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers
   SET salary = salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('no customers selected');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' customers
selected ');
   END IF;
END;
/
```

OUTPUT:

```
6 customers selected

PL/SQL procedure successfully completed.
```

EXPERIMENT 9

AIM:

Different types of triggers in PL-SQL.

THEORY:

A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server. DML triggers run when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Advantages of Triggers:

These are the following advantages of Triggers:

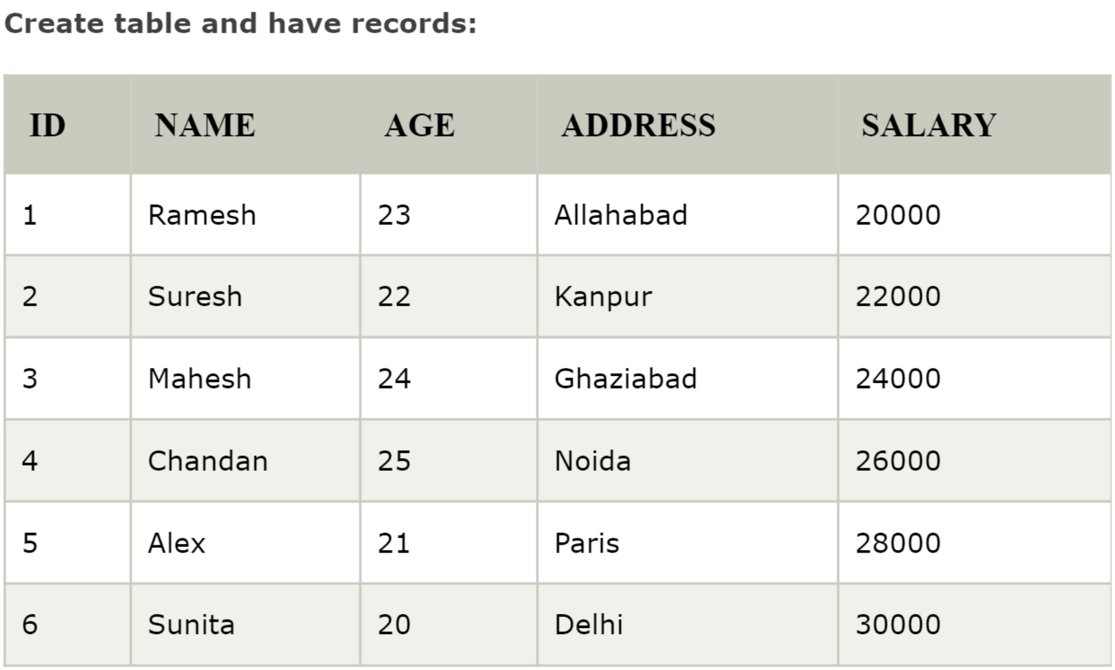
- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger

Syntax for creating trigger:

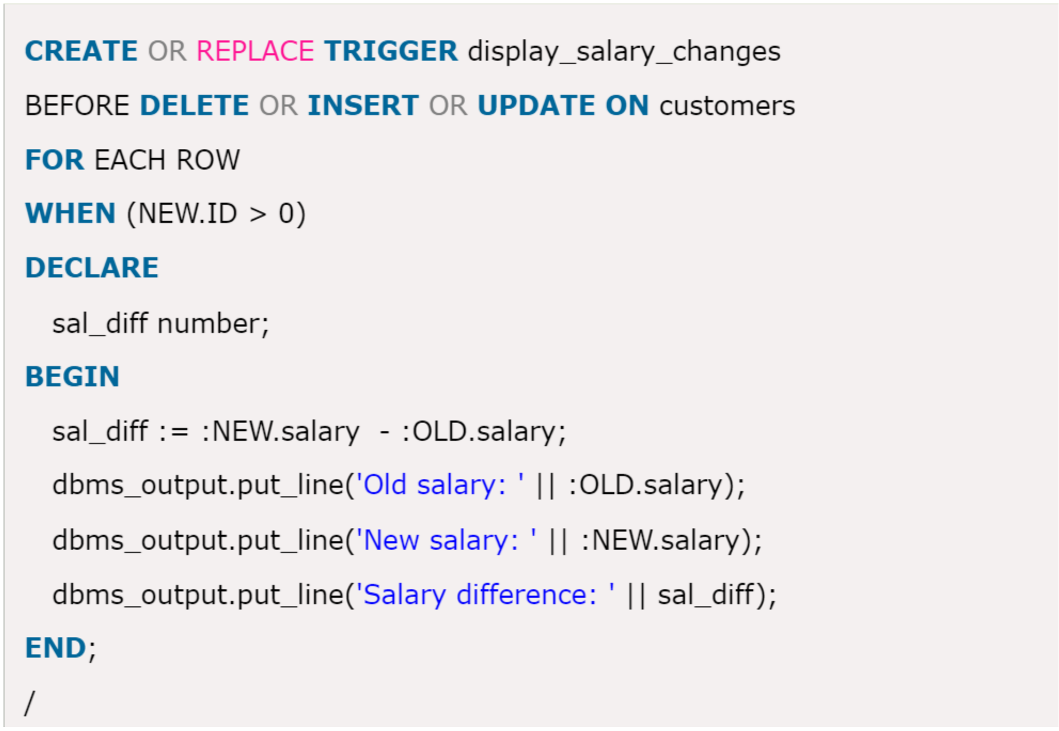
```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
      Declaration-statements
BEGIN
      Executable-statements
      EXCEPTION
      Exception-handling-statements
END;
```

OUTPUT:

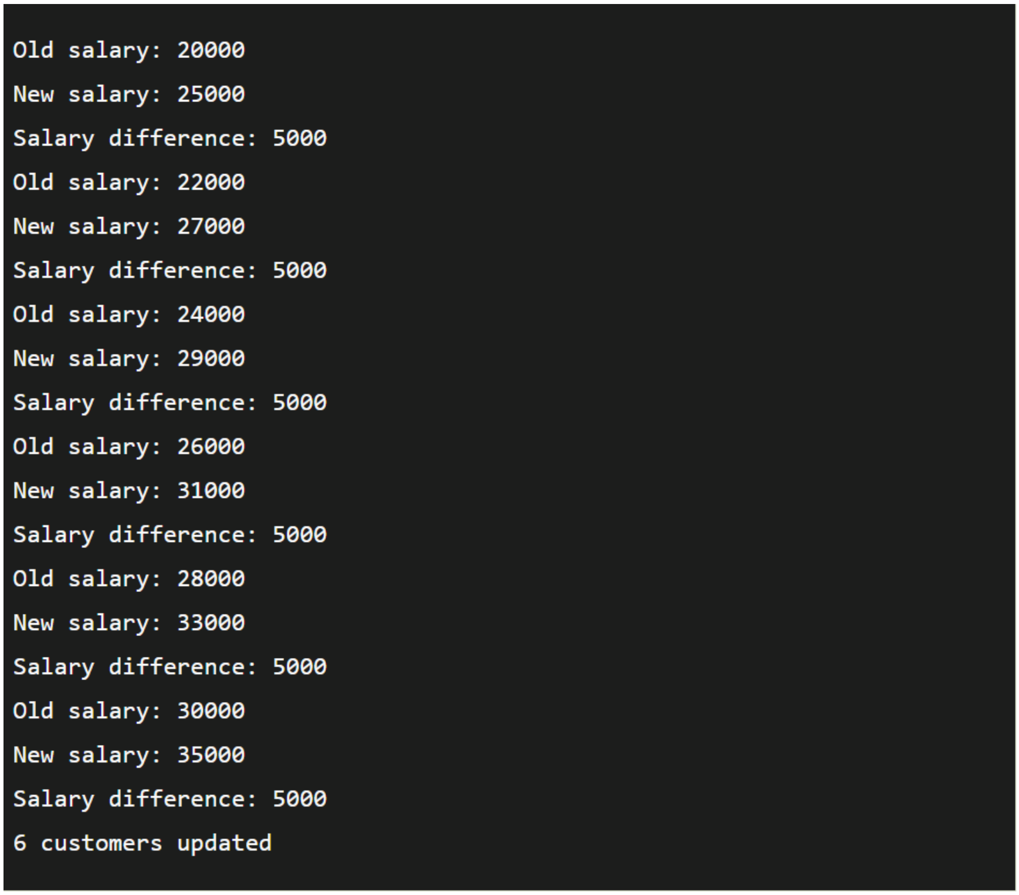


Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000



```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```



```
Old salary: 20000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
Old salary: 26000
New salary: 31000
Salary difference: 5000
Old salary: 28000
New salary: 33000
Salary difference: 5000
Old salary: 30000
New salary: 35000
Salary difference: 5000
6 customers updated
```

EXPERIMENT 10

AIM:

Create a program in PL-SQL

THEORY:

PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL. PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java. This tutorial will give you great understanding on PL/SQL to proceed with Oracle database and other advanced RDBMS concepts.

Overview of PL-SQL

PL/SQL is a block structured language that enables developers to combine the power of SQL with procedural statements.All the statements of a block are passed to oracle engine all at once which increases processing speed and decreases the traffic.

Disadvantages of SQL:

- SQL doesn’t provide the programmers with a technique of condition checking, looping and branching.
- SQL statements are passed to Oracle engine one at a time which increases traffic and decreases speed.
- SQL has no facility of error checking during manipulation of data.

Features of PL/SQL:

1. PL/SQL is basically a procedural language, which provides the functionality of decision making, iteration and many more features of procedural programming languages.
2. PL/SQL can execute a number of queries in one block using single command.
3. One can create a PL/SQL unit such as procedures, functions, packages, triggers, and types, which are stored in the database for reuse by applications.
4. PL/SQL provides a feature to handle the exception which occurs in PL/SQL block known as exception handling block.
5. Applications written in PL/SQL are portable to computer hardware or operating system where Oracle is operational.
6. PL/SQL Offers extensive error checking.

Advantages of PL-SQL

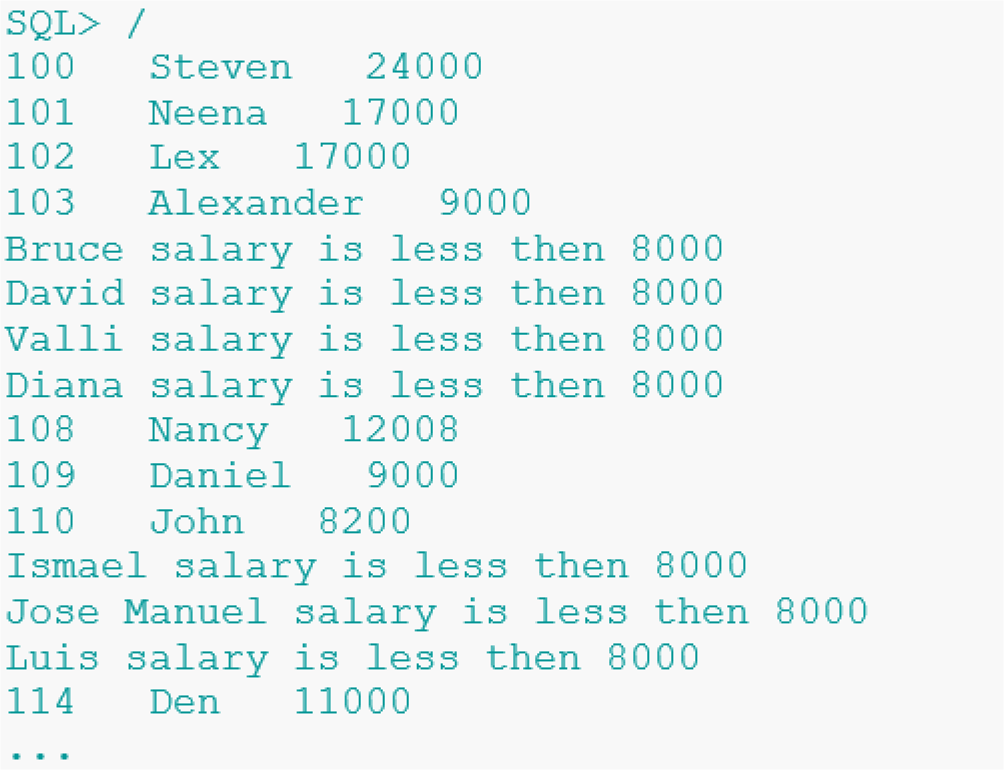
PL/SQL offers the following advantages over any other procedural language:

- support for SQL,
- closer integration with Oracle leading to better performance, and
- support for object-oriented programming.

Code:

```
DECLARE
  z_empid employees.employee_id%TYPE;
  z_empname employees.first_name%TYPE;
  z_salary employees.salary%TYPE;
  CURSOR employee_cursor IS  -- declaring a cursor
    SELECT employee_id,
           first_name,
           salary
    FROM   employees;

BEGIN
  OPEN employee_cursor;    -- opening the cursor
  LOOP
    FETCH employee_cursor  -- fetching records from the cursor
    INTO  z_empid,
          z_empname,
          z_salary;
    EXIT
  WHEN employee_cursor%NOTFOUND;
    IF (z_salary > 8000) THEN
      dbms_output.Put_line(z_empid
      || '   '
      || z_empname
      || '   '
      || z_salary);
    ELSE
      dbms_output.Put_line(z_empname
      || ' salary is less then 8000');
    END IF;
  END LOOP;
  CLOSE employee_cursor;  --closing the cursor
END;
/
```

OUTPUT:



```
SQL> /
100   Steven   24000
101   Neena   17000
102   Lex   17000
103   Alexander   9000
Bruce salary is less then 8000
David salary is less then 8000
Valli salary is less then 8000
Diana salary is less then 8000
108   Nancy   12008
109   Daniel   9000
110   John   8200
Ismael salary is less then 8000
Jose Manuel salary is less then 8000
Luis salary is less then 8000
114   Den   11000
...
```