KATHMANDU UNIVERSITY

DHULIKHEL, NEPAL

Department of Computer Science & Engineering (DoCSE)

LAB REPORT

COMP-232

Submitted By:

Sambeg Shrestha (45)

Computer Engineering

2nd Year, 2nd Semester

Submitted to:

Mr. Santosh Khanal

15 October, 2020

Acknowledgment

I would like to express my deepest gratitude to all those who provided me with the possibility yo complete this report. A special gratitude to our subject teacher, Mr. Santosh Khanal, whose elucidating lectures on the database system and practical classes in the application of MySQL RDBMS, helped me a lot in understanding and implementation of relational database language MySQL. Furthermore I would like to acknowledge my peers who helped me in understanding the topic and guidance they imparted me throughout implementation of the different laboratory works.

Abstract

Database Management System (DBMS) is a software package designed to *define, manipulate, retrieve* and *manage* data in a database. As computer science students, learning the fundamentals of DBMS is most paramount. The Lab study done under COMP 202 was done in order get acquainted with the fundamentals of one of the most popular Relational DBMS technology: **MySQL**, and to understand the operations done in a database. Using basic MySQL queries we learned the Data *Definition, Manipulation* and *Querying*. In conclusion, data storage and manipulation is the most fundamental requirement of a computing technology and technologies like MySQL are the most structured, systematic and convenient method to manage a database.

Chapter 1 Introduction

1.1 Database

A database is an organized collection of data, generally stored and accessed electronically from a computer system. Complex databases are often developed using formal design and modeling techniques.

1.2 Database Management System (DBMS)

DBMS is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The sum total of the database, the DBMS and the associated applications can be referred to as a "database system". Often the term "database" is also used to loosely refer to any of the DBMS, the database system or an application associated with the database.

1.3 Relational Database

We particularly focused on **Relational** Database in this study. A *relational database* is a type of database. It uses a structure that allows us to identify and access data *in relation* to another piece of data in the database. Often, data in a relational database is organized into tables.

1.4 Relational Database Management System (RDBMS)

An RDBMS is a program that allows you to create, update, and administer a relational database. Most relational database management systems use the SQL language to access the database.

1.5 Structured Query Language (SQL)

SQL (**S**tructured **Q**uery **L**anguage) is a programming language used to communicate with data stored in a relational database management system. SQL syntax is similar to the English language, which makes it relatively easy to write, read, and interpret.

1.6 MySQL

MySQL is the most popular open source SQL database. It is typically used for web application development, and often accessed using PHP.

The main advantages of MySQL are that it is easy to use, inexpensive, reliable (has been around since 1995), and has a large community of developers who can help answer questions.

Some of the disadvantages are that it has been known to suffer from poor performance when scaling, open source development has lagged since Oracle has taken control of MySQL, and it does not include some advanced features that developers may be used to

Chapter 2 Methodology

The Lab Study was performed using the following elucidated chronological structure.

2.1 Installation and Configuration

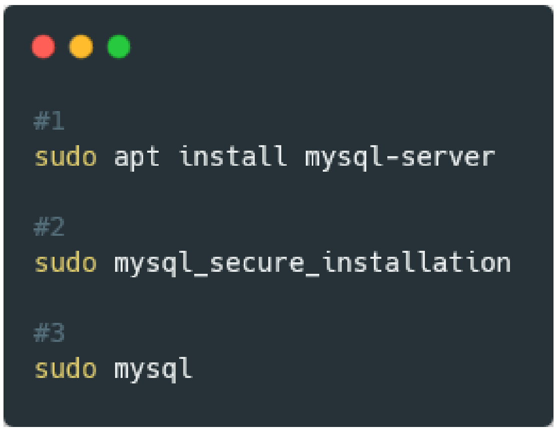
2.1.1 System

Linux (Ubuntu 20.04)

2.1.2 Steps

The bash terminal was used for installation purposes.

- Step 1 — Installing the mysql-server package
- Step 2 — Configuring MySQL. Run security script to set-up root password by following instructoins
- Step 3 — Running MySQL server



```
#1
sudo apt install mysql-server

#2
sudo mysql_secure_installation

#3
sudo mysql
```

As beginners, we dealt mainly with *Data Definition Language (DLL)* and *Data Manipulation Language (DML)*

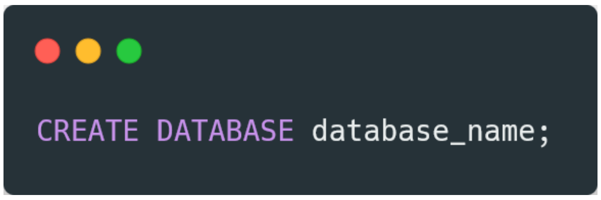
2.2 Data Definition Language

2.2.1 CREATE

The CREATE statement is used to create a database, tables or an index.

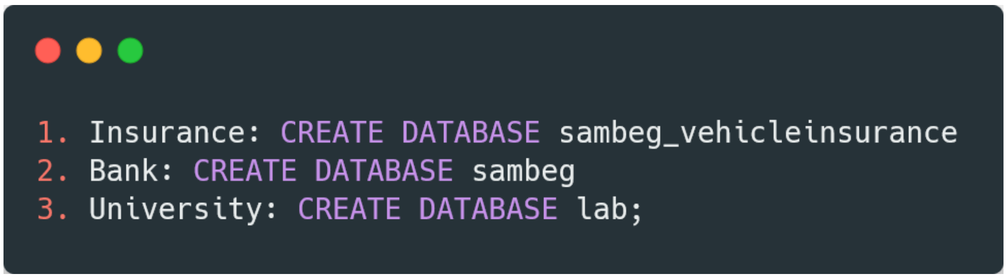
- **Creating a database**

Syntax:



```sql
CREATE DATABASE database_name;
```

Example:



```sql
1. Insurance: CREATE DATABASE sambeg_vehicleinsurance
2. Bank: CREATE DATABASE sambeg
3. University: CREATE DATABASE lab;
```

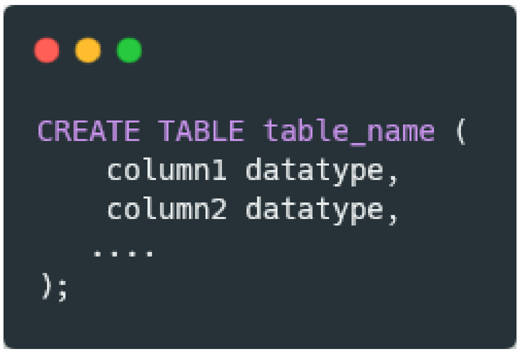
These examples were used to set up Databases in each of our lab sessions.

- **Create a table**

The 'CREATE TABLE' Statement

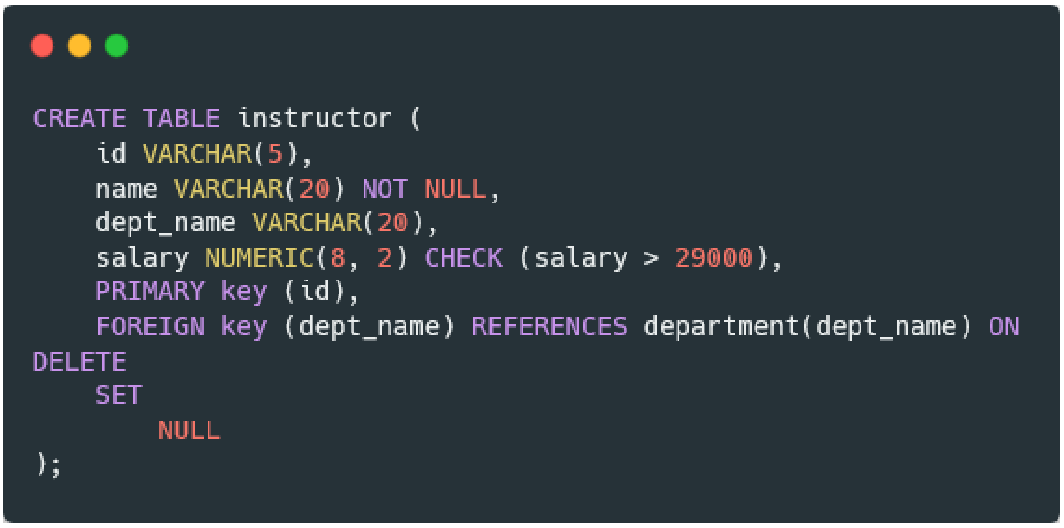
This statement is used to create a new table in a database.

Syntax:



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

Example:



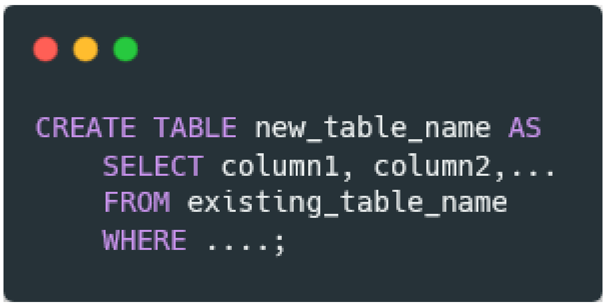
```sql
CREATE TABLE instructor (
    id VARCHAR(5),
    name VARCHAR(20) NOT NULL,
    dept_name VARCHAR(20),
    salary NUMERIC(8, 2) CHECK (salary > 29000),
    PRIMARY key (id),
    FOREIGN key (dept_name) REFERENCES department(dept_name) ON
DELETE
    SET
        NULL
);
```

This example is used to create a Table for instructors in the university database. It consists of attributes: id, name, dept_name, salary. The *Primary key* has been set as *id* and *dept_name* a *Foreign Key* referenced to *dept_name* from *department* Table.

The 'CREATE TABLE AS' Statement

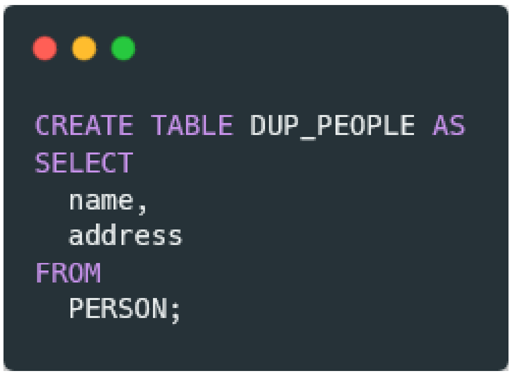
This statement is used to create a new table from an existing table. So, this table gets the same column definitions as that of the existing table.

Syntax:



```sql
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

Example:



```sql
CREATE TABLE DUP_PEOPLE AS
SELECT
  name,
  address
FROM
  PERSON;
```

This was used in the Vehicle Insurance Database in order to create a new table *DUP_PEOPLE* from the *name* and *address* attributes of *PERSON* Table.

2.2.2 Data Types

There are numerous data types available in MySQL. They are-

- **Numeric** – This data type includes integers of various sizes, floating-point(real) of various precisions and formatted numbers.
- **Character-string –** These data types either have a fixed, or a varying number of characters.
- **Bit-string –** These data types are either of a fixed length or varying length of bits.
- **Boolean –**This data type has TRUE or FALSE values. Since SQL, has NULL values, a three-valued logic is used, which is UNKNOWN.
- **Date & Time –**The DATE data type has: YEAR, MONTH, and DAY in the form YYYY-MM-DD. Similarly, the TIME data type has the components HOUR, MINUTE, and SECOND in the form HH:MM: SS. These formats can change based on the requirement.
- **Timestamp & Interval –**The TIMESTAMP data type includes a minimum of six positions, for decimal fractions of seconds and an optional WITH TIME ZONE qualifier in addition to the DATE and TIME fields.

2.2.3 Keys

There are mainly 5 types of Keys, that can be mentioned in the database.

- **Candidate Key –** The minimal set of attributes which can uniquely identify a tuple is known as a candidate key. A relation can hold more than a single candidate key, where the key is either a simple or composite key.
- **Super Key –**The set of attributes which can uniquely identify a tuple is known as Super Key. So, a candidate key is a superkey, but vice-versa isn't true.

- **Primary Key –** A set of attributes that can be used to uniquely identify every tuple is also a primary key. So, if there are 3-4 candidate keys present in a relationship, then out those, one can be chosen as a primary key.

- **Alternate Key –** The candidate key other than the primary key is called as an alternate key**.**

- **Foreign Key –** An attribute that can only take the values present as the values of some other attribute, is the foreign key to the attribute to which it refers.

2.2.4 Constraints

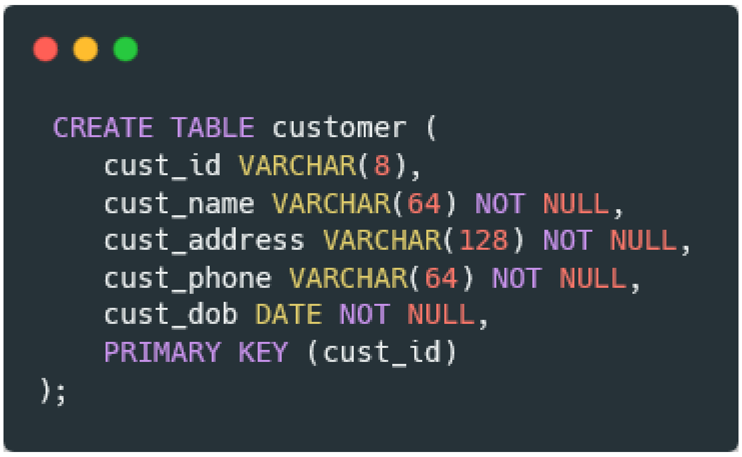
SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

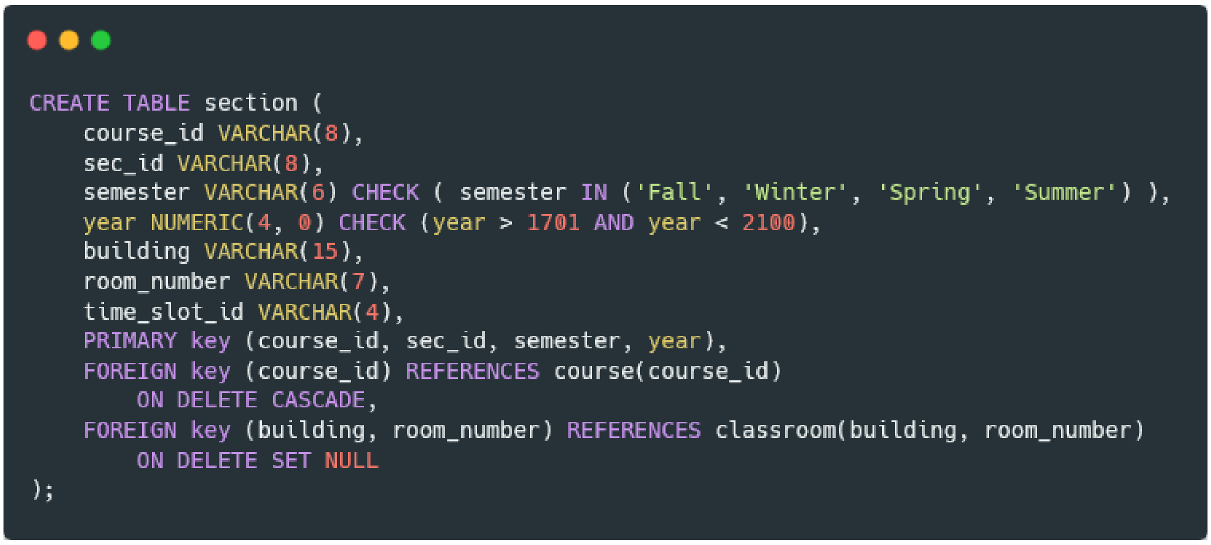
The following constraints are commonly used in SQL:

- NOT NULL – Ensures that a column cannot have a NULL value
- UNIQUE – Ensures that all values in a column are different
- PRIMARY KEY – A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY – Uniquely identifies a row/record in another table
- CHECK – Ensures that all values in a column satisfies a specific condition
- DEFAULT – Sets a default value for a column when no value is specified
- INDEX – Used to create and retrieve data from the database very quickly

Consider the following examples,



```
CREATE TABLE customer (
    cust_id VARCHAR(8),
    cust_name VARCHAR(64) NOT NULL,
    cust_address VARCHAR(128) NOT NULL,
    cust_phone VARCHAR(64) NOT NULL,
    cust_dob DATE NOT NULL,
    PRIMARY KEY (cust_id)
);
```



```
CREATE TABLE section (
    course_id VARCHAR(8),
    sec_id VARCHAR(8),
    semester VARCHAR(6) CHECK ( semester IN ('Fall', 'Winter', 'Spring', 'Summer') ),
    year NUMERIC(4, 0) CHECK (year > 1701 AND year < 2100),
    building VARCHAR(15),
    room_number VARCHAR(7),
    time_slot_id VARCHAR(4),
    PRIMARY key (course_id, sec_id, semester, year),
    FOREIGN key (course_id) REFERENCES course(course_id)
        ON DELETE CASCADE,
    FOREIGN key (building, room_number) REFERENCES classroom(building, room_number)
        ON DELETE SET NULL
);
```

So, in the above examples from Bank and University Databases, we can clearly see different Data Types, Keys as well as Constraints.

Data Types like *VARCHAR, DATE, NUMERIC* have been used to define different attributes. The usage of *Primary keys* and *Foreign keys* have also been made along with constraints like *CHECK, NOT NULL*

2.2.5 ALTER

This statement is used to either add, modify or delete constraints and columns from a table.

The 'ALTER TABLE' Statement

This statement is used to either add, modify or delete constraints and columns from a table.

Syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example:

```
ALTER TABLE
    staff
ADD
    COLUMN designation VARCHAR(32);
```

This example was used to add an attribute named *designation* to the *staff* Table from Bank database named *sambeg*.

2.2.6 DROP

The DROP command is used to delete the database, tables or columns.

Delete a database

The 'DROP DATABASE' Statement

This statement is used to drop the complete database.
Syntax:

```
DROP DATABASE database_name;
```

Example:

```
DROP DATABASE sambeg_vehicleinsurance;
```

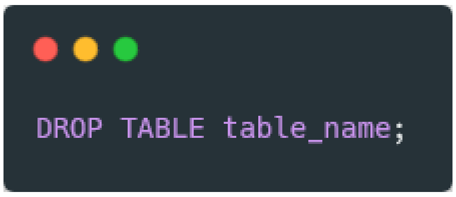
Here, DROP was used to delete the main database from the Insurance Lab.

Delete a table

The 'DROP TABLE' Statement

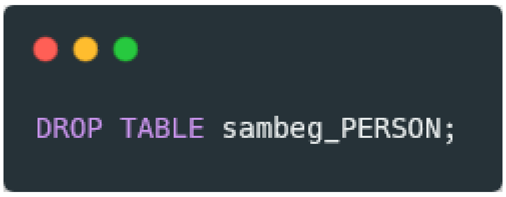
This statement is used to drop the entire table with all its values.

Syntax:



```
DROP TABLE table_name;
```

Example:



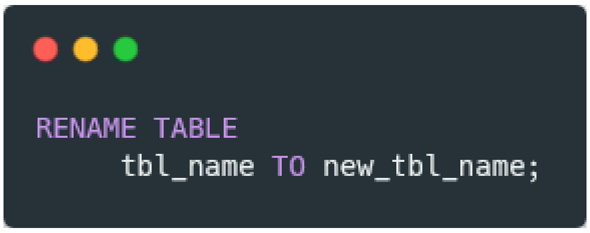
```
DROP TABLE sambeg_PERSON;
```

Here, the table named sambeg_PERSON was dropped from the Insurance database.

2.2.7 RENAME

This statement is used to rename one or more tables.

Syntax:



```
RENAME TABLE
     tbl_name TO new_tbl_name;
```

Example:

```
RENAME TABLE
    PEOPLE TO NEW_PEOPLE;
```

This statement was used to rename the PEOPLE Table in Insurance Lab to NEW_POEPLE

Other DDL commands are:

TRUNCATE: This statement is used to delete the data which is present inside a table, but the table doesn't get deleted.

2.3 Data Manipulation Language (DML)

DML are those commands, by which you can manipulate your database. The commands are: USE, UPDATE, DELETE, SELECT.

2.3.1 USE

The USE statement is used to mention which database has to be used to perform all the operations.

Syntax:

```
USE Database_name;
```

Example:

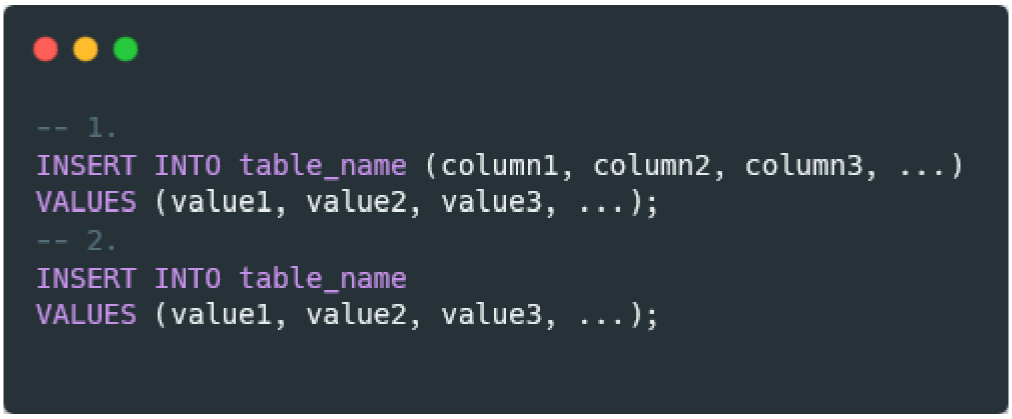
```
USE sambeg_vehicleinsurance;
```

Here, statement is used to mention that sambeg_vehicleinsurance database must be used in Insurance lab.

2.3.2 INSERT

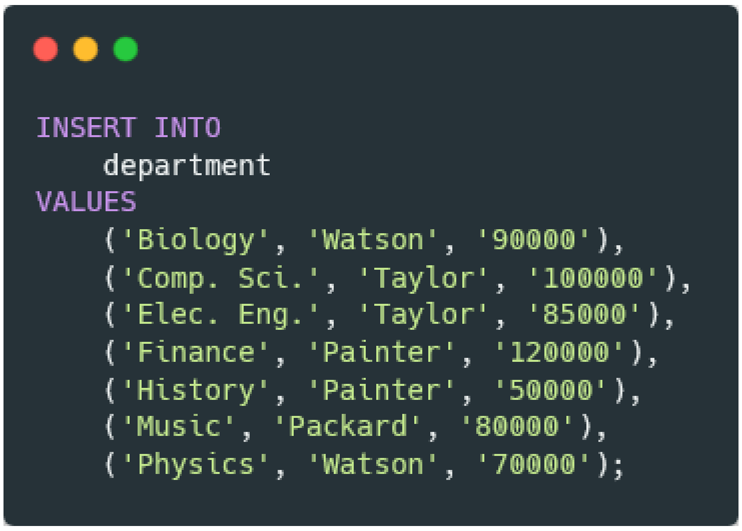
This statement is used to insert new records in a table.

Syntax: INSERT INTO statement can be written in two ways:



```sql
-- 1.
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
-- 2.
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

Example:



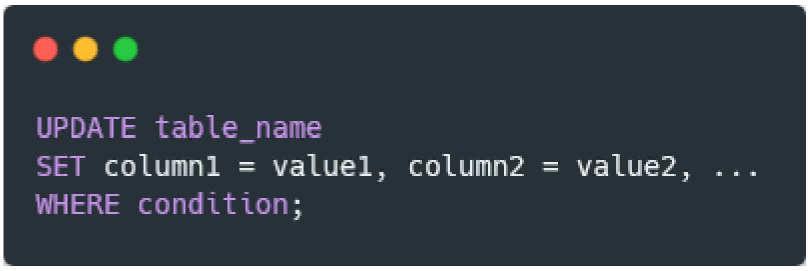
```sql
INSERT INTO
    department
VALUES
    ('Biology', 'Watson', '90000'),
    ('Comp. Sci.', 'Taylor', '100000'),
    ('Elec. Eng.', 'Taylor', '85000'),
    ('Finance', 'Painter', '120000'),
    ('History', 'Painter', '50000'),
    ('Music', 'Packard', '80000'),
    ('Physics', 'Watson', '70000');
```

This is used to populate the values of *department* table of *University*.

2.3.3 UPDATE

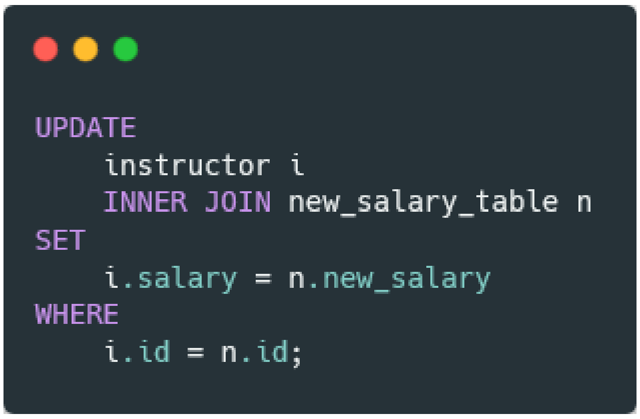
This statement is used to modify the existing records in a table.

Syntax:



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

Example:



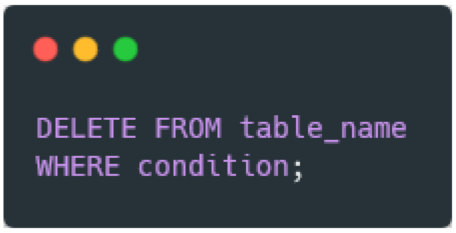
```
UPDATE
    instructor i
    INNER JOIN new_salary_table n
SET
    i.salary = n.new_salary
WHERE
    i.id = n.id;
```

Here, the statement is used to update the salaries of instructors to a new modified one.

2.3.4 DELETE

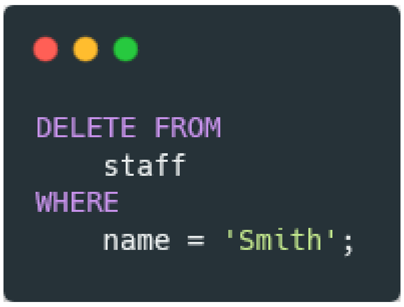
This statement is used to delete existing records in a table.

Syntax:



```
DELETE FROM table_name
WHERE condition;
```

Example:



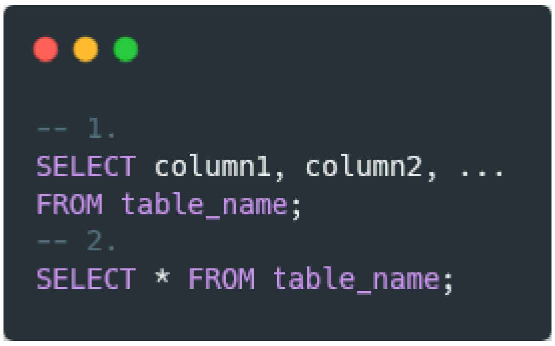
```sql
DELETE FROM
    staff
WHERE
    name = 'Smith';
```

This is used to remoev ‘Smith’ from the table of *staff* in *Bank* database.

2.3.5 SELECT

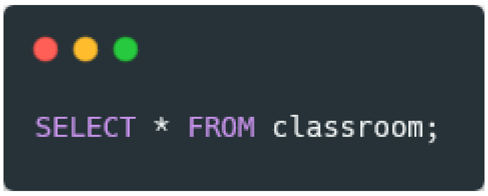
This statement is used to select data from a database and the data returned is stored in a result table, called the result-set.

Syntax:



```sql
-- 1.
SELECT column1, column2, ...
FROM table_name;
-- 2.
SELECT * FROM table_name;
```

Example:



```sql
SELECT * FROM classroom;
```

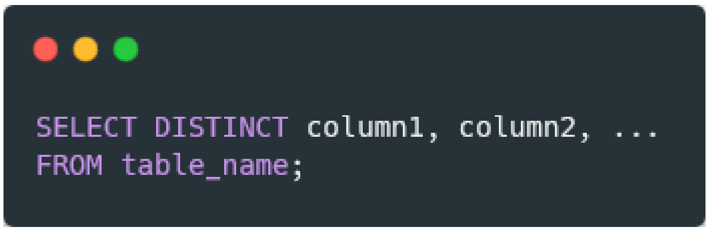
Example is used for selecting all records from *classroom* table of *University*.

Apart from the individual SELECT keyword, some other statements are used with the SELECT keyword: DISTINCT, ORDER BY, GROUP BY, HAVING Clause

The ‘SELECT DISTINCT’ Statement

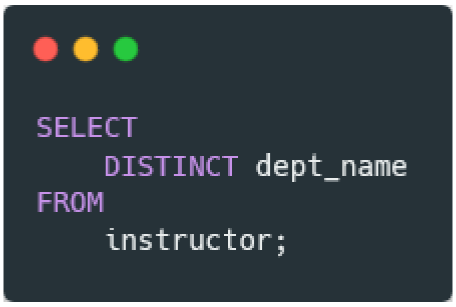
This statement is used to return only distinct or different values. So, if you have a table with duplicate values, then you can use this statement to list distinct values.

Syntax:



```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Example:



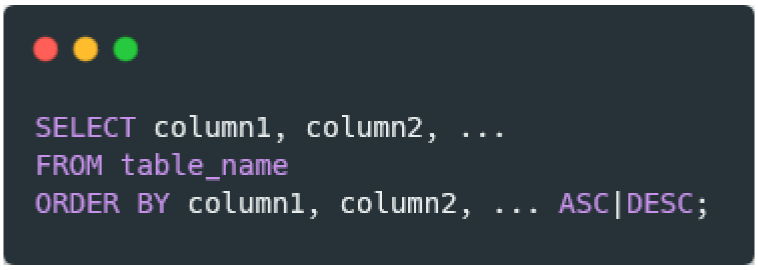
```
SELECT
    DISTINCT dept_name
FROM
    instructor;
```

Used for displaying unique values of *dept_name* from *instructor* table of *University* lab.

The ‘ORDER BY’ Statement

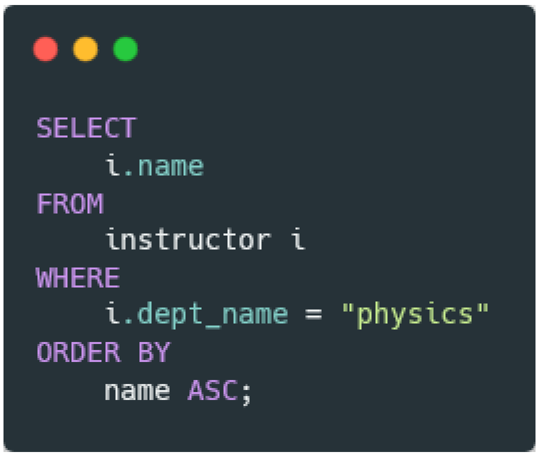
This statement is used to sort the desired results in ascending or descending order.

Syntax:



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

Example:



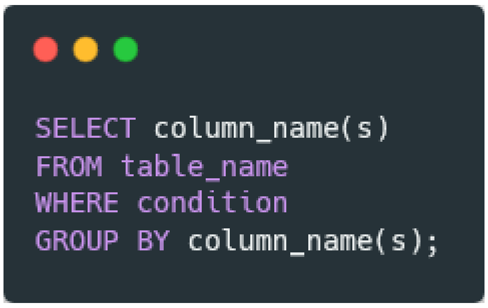
```sql
SELECT
    i.name
FROM
    instructor i
WHERE
    i.dept_name = "physics"
ORDER BY
    name ASC;
```

Used to display the *instructor names* in *ascending* order from *instructor* table with *dept_name* 'Physics'.

The 'GROUP BY' Statement

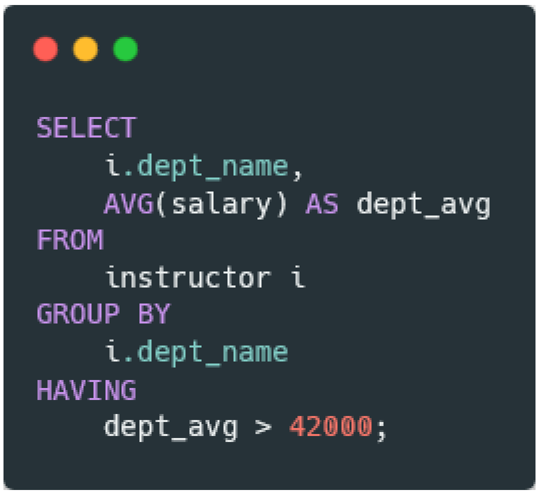
This statement is used with the aggregate functions to group the result-set by one or more columns.

Syntax:



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

Example:



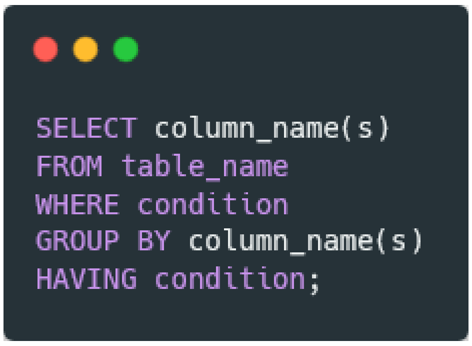
```sql
SELECT
    i.dept_name,
    AVG(salary) AS dept_avg
FROM
    instructor i
GROUP BY
    i.dept_name
HAVING
    dept_avg > 42000;
```

Here, there is Grouping by *dept_name* attribute of *instructor* table.

The 'HAVING' Statement

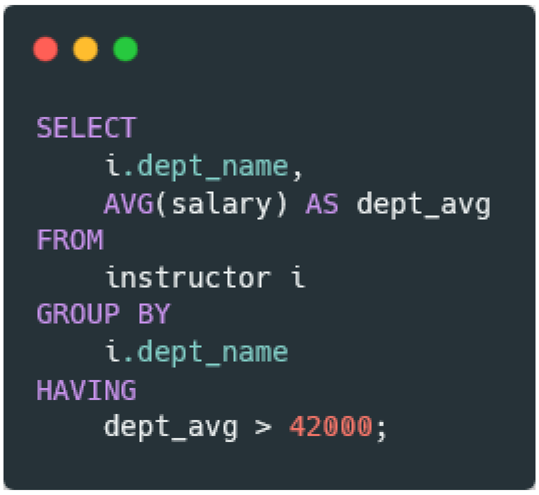
Since the WHERE keyword cannot be used with aggregate functions, the HAVING clause was introduced.

Syntax:



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

Example:



```sql
SELECT
    i.dept_name,
    AVG(salary) AS dept_avg
FROM
    instructor i
GROUP BY
    i.dept_name
HAVING
    dept_avg > 42000;
```

Here having is used to check the *department average salary* is greater than 42000 in *instructor* table.

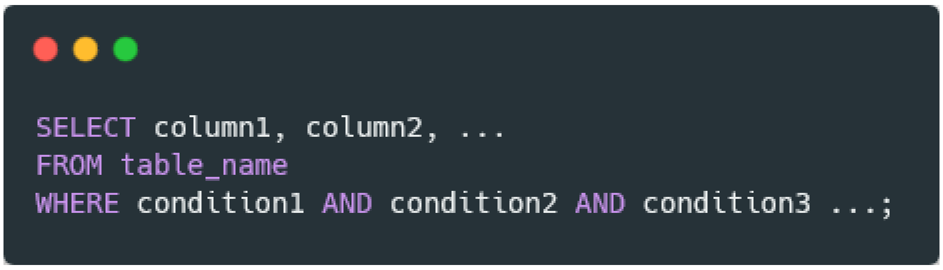
2.3.6 Logical operators

These consists of operators such as AND/OR/NOT.

AND OPERATOR

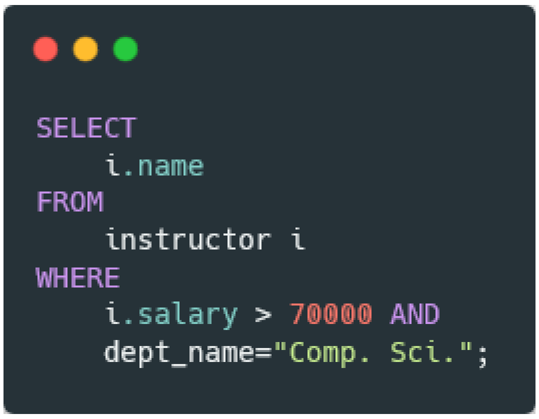
The AND operator is used to filter records that rely on more than one condition. This operator displays the records, which satisfy all the conditions separated by AND, and give the output TRUE.

Syntax:



```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example:



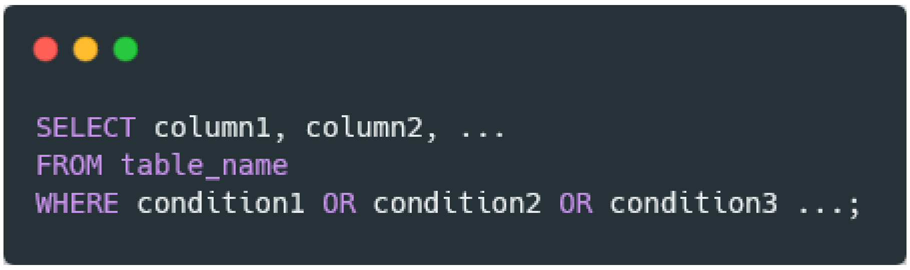
```
SELECT
    i.name
FROM
    instructor i
WHERE
    i.salary > 70000 AND
    dept_name="Comp. Sci.";
```

Here and operator was used to check satisfaction of both conditions: salary greater than 70k and department Comp. Sci.

OR OPERATOR

The OR operator displays those records which satisfy any of the conditions separated by OR and gives the output TRUE.

Syntax:

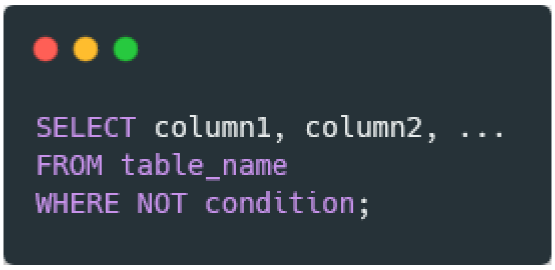


```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT OPERATOR

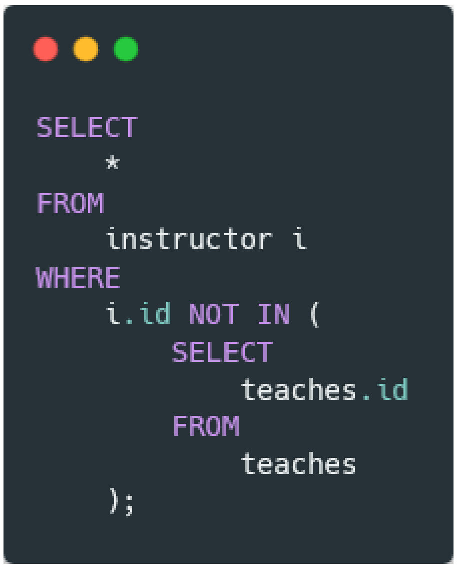
This operator displays a record when the condition (s) is NOT TRUE.

Syntax:



```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Example:



```
SELECT
    *
FROM
    instructor i
WHERE
    i.id NOT IN (
        SELECT
            teaches.id
        FROM
            teaches
    );
```

Here, NOT is used to check *instructor ids* that are not available in *teaches* table of *University*.

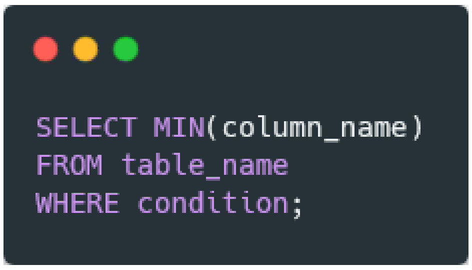
2.3.7 AGGREGATE FUNCTIONS

The aggregate functions performs a calculation on a set of values, and returns a single value. The Aggregate Functions in MySQL are: MIN(), MAX(), COUNT(), AVG(), SUM()

MIN() Function

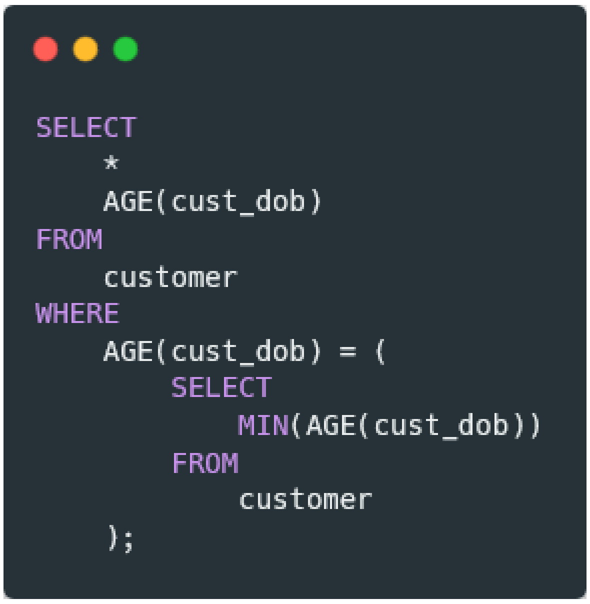
Returns the smallest value of the selected column.

Syntax:



```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

Example:



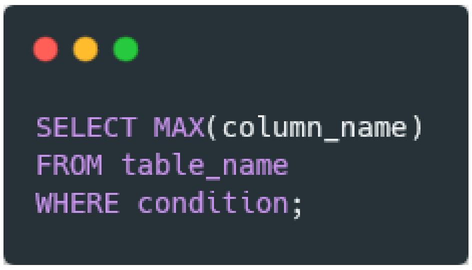
```sql
SELECT
    *
    AGE(cust_dob)
FROM
    customer
WHERE
    AGE(cust_dob) = (
        SELECT
            MIN(AGE(cust_dob))
        FROM
            customer
    );
```

Used to find the youngest customer in the bank.

MAX() Function

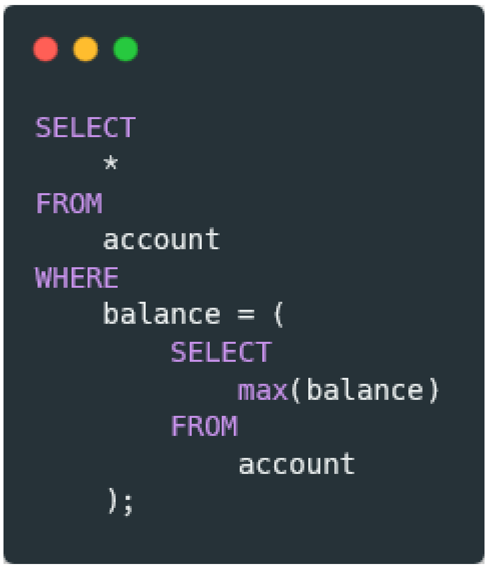
Returns the largest value of the selected column in a table.

Syntax:



```sql
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Example:



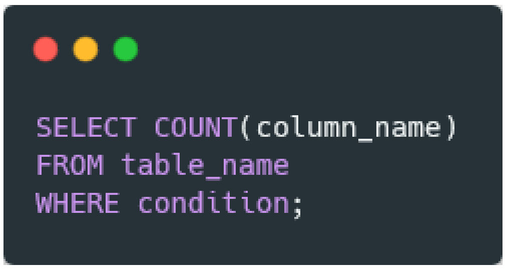
```
SELECT
    *
FROM
    account
WHERE
    balance = (
        SELECT
            max(balance)
        FROM
            account
    );
```

Used to find the account number who has the maximum balance in Bank.

COUNT() Function

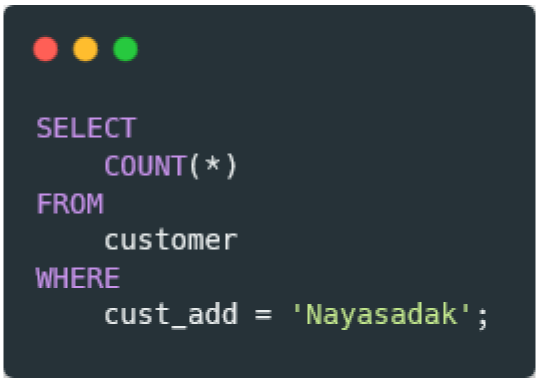
Returns the number of rows that match the specified criteria.

Syntax:



```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

Example:



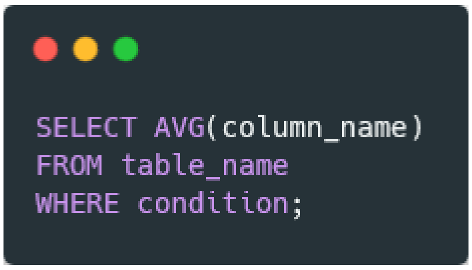
```
SELECT
    COUNT(*)
FROM
    customer
WHERE
    cust_add = 'Nayasadak';
```

Used to list the number of customers from address “Nayasadak” in Bank.

AVG() Function

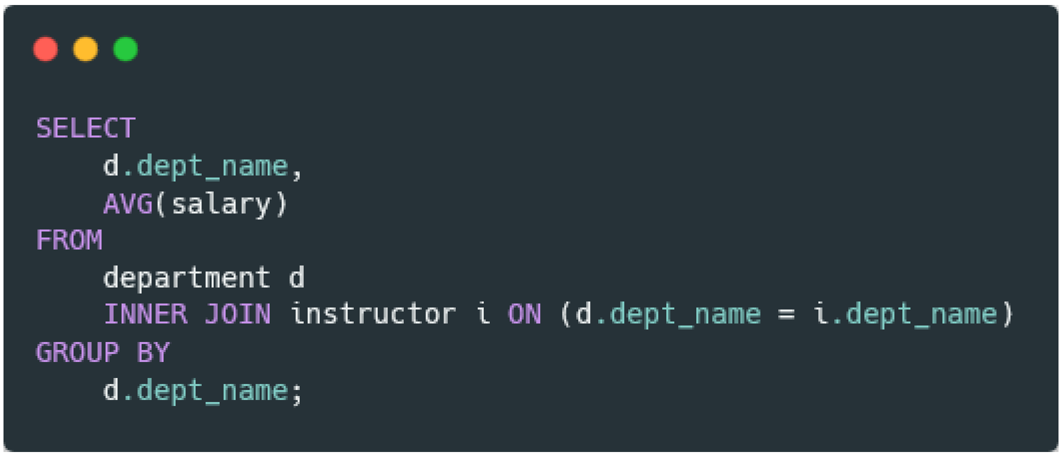
Returns the average value of a numeric column chosen.

Syntax:



```sql
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

Example:



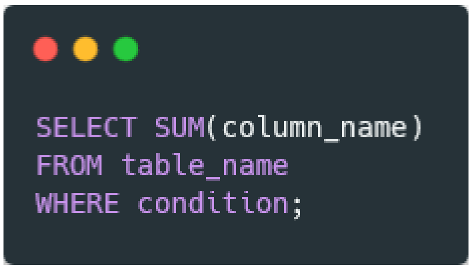
```sql
SELECT
    d.dept_name,
    AVG(salary)
FROM
    department d
    INNER JOIN instructor i ON (d.dept_name = i.dept_name)
GROUP BY
    d.dept_name;
```

Example used to find the average salary in each department in University.

SUM() Function

Returns the total sum of a numeric column tchosen.

Syntax:

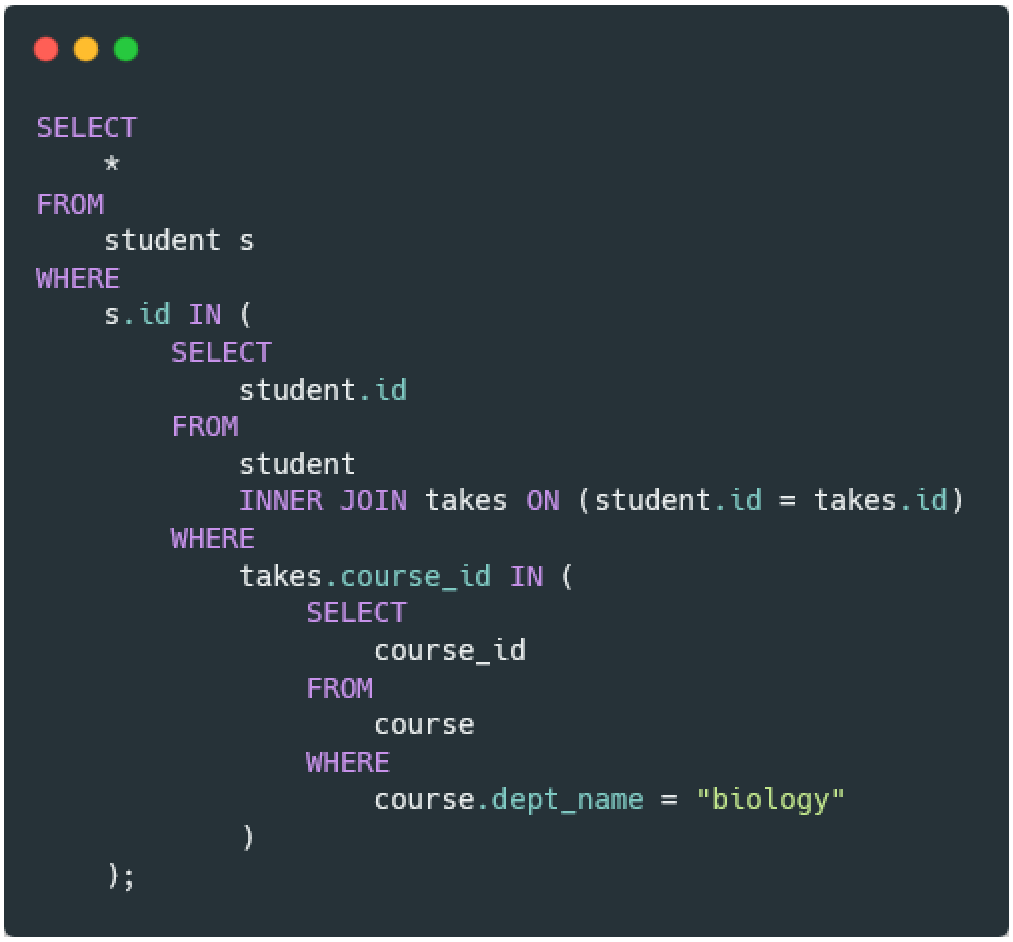


```sql
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

2.3.8 Nested Queries

Nested queries are those queries which have an outer query and inner subquery. So, basically, the subquery is a query which is nested within another query such as SELECT, INSERT, UPDATE or DELETE.

Example:



```sql
SELECT
    *
FROM
    student s
WHERE
    s.id IN (
        SELECT
            student.id
        FROM
            student
            INNER JOIN takes ON (student.id = takes.id)
        WHERE
            takes.course_id IN (
                SELECT
                    course_id
                FROM
                    course
                WHERE
                    course.dept_name = "biology"
            )
    );
```

An example of nesting multiple queries in order to find all students who have taken all courses offered in the Biology department of University.

2.3.9 Joins

JOINS are used to combine rows from two or more tables, based on a related column between those tables. The following are the types of joins:

- **INNER JOIN:** This join returns those records which have matching values in both the tables.

- **FULL JOIN:** This join returns all those records which either have a match in the left or the right table.
- **LEFT JOIN**: This join returns records from the left table, and also those records which satisfy the condition from the right table.
- **RIGHT JOIN**: This join returns records from the right table, and also those records which satisfy the condition from the left table.

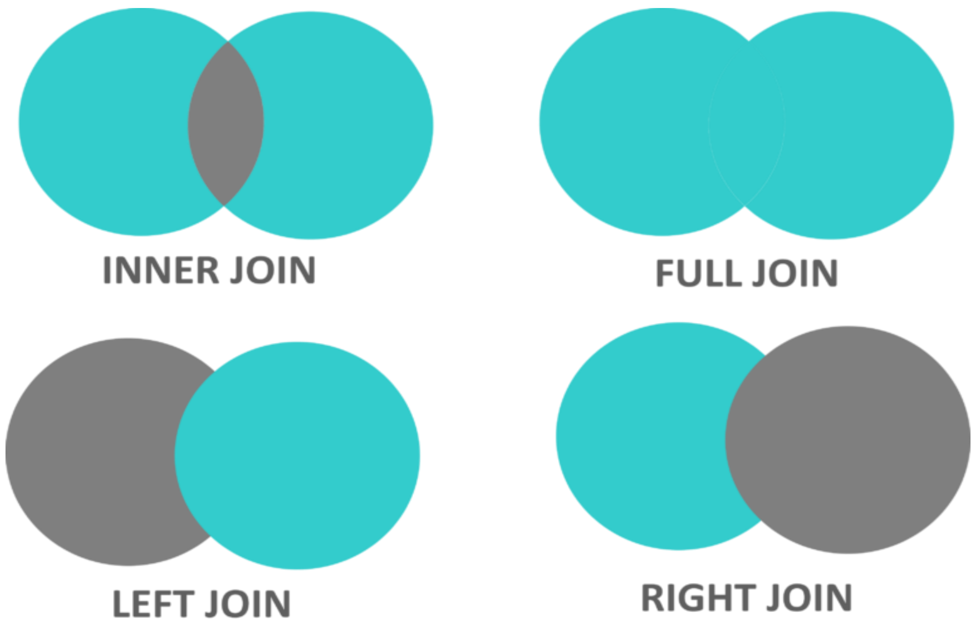
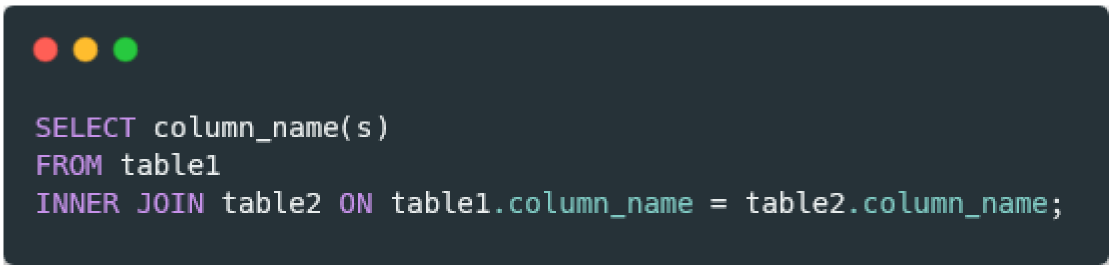


Fig 4: Representation of Joins

- **INNER JOIN**

Syntax:



```
SELECT column_name(s)
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```

;

Example:

```
SELECT
    d.dept_name,
    AVG(salary)
FROM
    department d
    INNER JOIN instructor i ON (d.dept_name = i.dept_name)
GROUP BY
    d.dept_name;
```

Here, used to inner join *department* and *instructor* tables in University to find average salary in each department.

- **FULL JOIN**

Syntax:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;
```

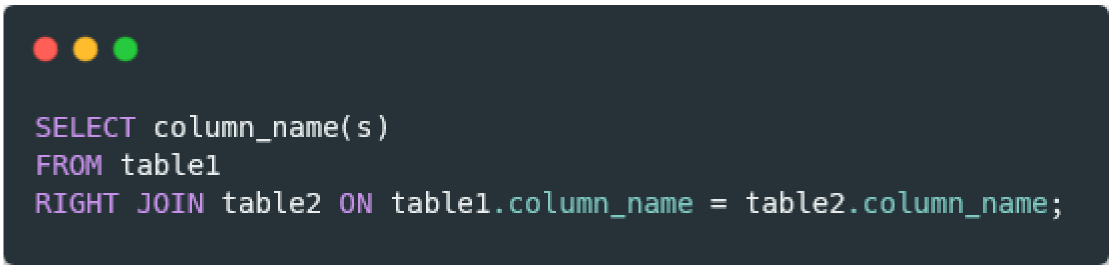
- **LEFT JOIN**

Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

- **RIGHT JOIN**

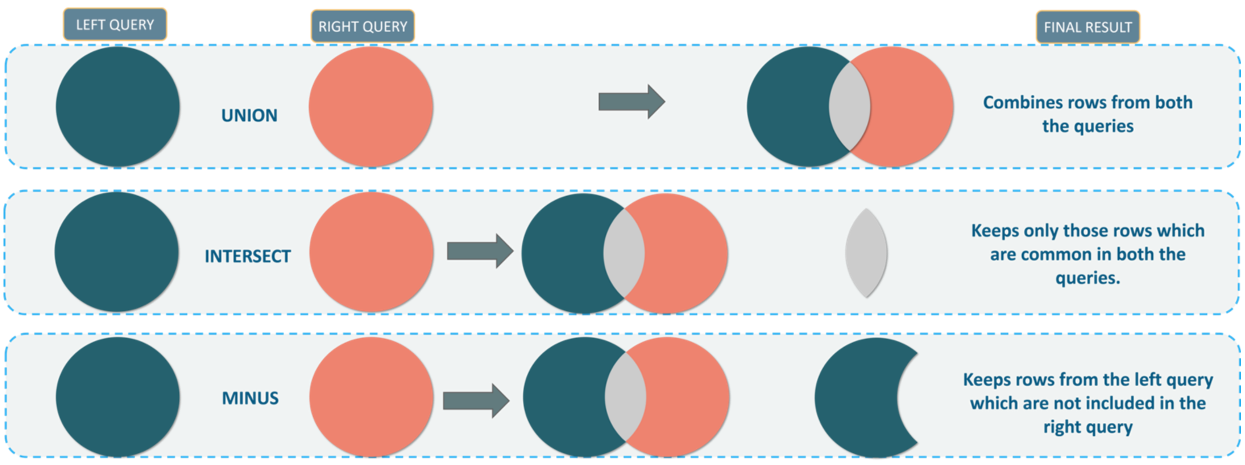
Syntax:



```sql
SELECT column_name(s)
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

2.3.10 Set Operations

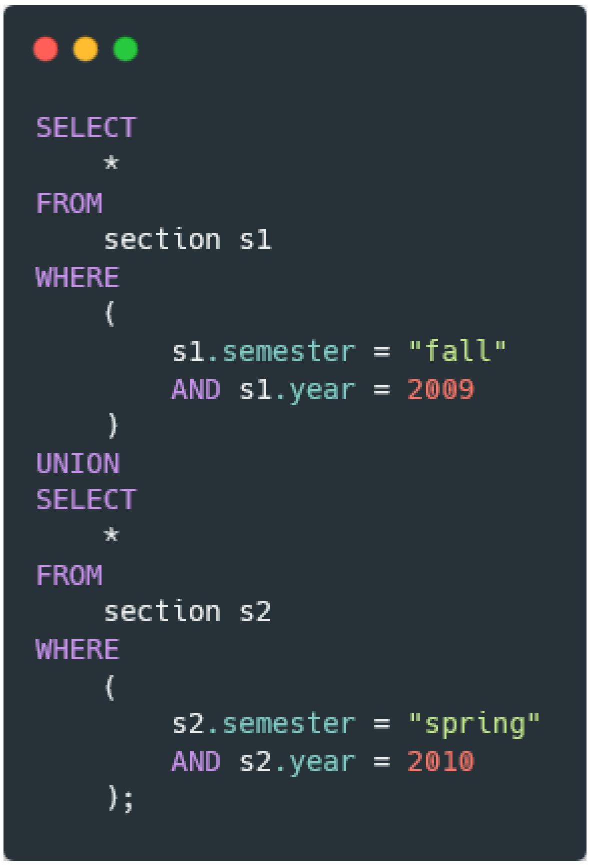
There are mainly three set operations: UNION, INTERSECT, SET DIFFERENCE. You can refer to the image below to understand the set operations in SQL.


LEFT QUERY
RIGHT QUERY
FINAL RESULT
UNION
Combines rows from both the queries
INTERSECT
Keeps only those rows which are common in both the queries.
MINUS
Keeps rows from the left query which are not included in the right query

These are some of the methods used while working with the database.

UNION

Example:



```
SELECT
    *
FROM
    section s1
WHERE
    (
        s1.semester = "fall"
        AND s1.year = 2009
    )
UNION
SELECT
    *
FROM
    section s2
WHERE
    (
        s2.semester = "spring"
        AND s2.year = 2010
    );
```

Used to find the set of all courses taught either in Fall 2009 or in Spring 2010, or both in *University*.

Chapter 3 Conclusion

Hence, MySQL is an excellent query language for database management. It is effective, convenient and very systematic at organizing and managing a database. The clear command syntaxes make the code easy to read.

Therefore, DBMS are an excellent technology and MySQL resides on the top of such technology.

References

Kappagantula, S. (2019, May 22). *MySQL Tutorial – A Beginner's Guide To Learn MySQL.* Edureka. https://www.edureka.co/blog/mysql-tutorial/

What is a Relational Database Management System? (n.d.). Codecademy. Retrieved October 10, 2020, from https://www.codecademy.com/articles/what-is-rdbms-sql