CodeQuotient

Department of Computer Science & Engineering

LAB MANUAL

22CS007-Database Management System - 3rdSem-2023

Student Name	**SHIVDEV JOSHI**
Email	**shivdev2448.be23@chitkara.edu.in**
Course Name	**22CS007-Database Management System - 3rdSem-2023**

Faculty Incharge

_____

Table of Contents

Aim: DDL Command - CREATE TABLE

SQL, or Structured Query Language, is a language designed to allow both technical and non-technical users query, manipulate, and transform data from a relational database. And due to its simplicity, SQL databases provide safe and scalable storage for millions of websites and mobile applications.

There are many popular SQL databases including SQLite, MySQL, Postgres, Oracle and Microsoft SQL Server. All of them support the common SQL language standard, which is what this site will be teaching, but each implementation can differ in the additional features and storage types it supports.

SQL Table

SQL Table is a collection of data that is organized in terms of rows and columns. In DBMS, the table is known as relation and row as a tuple. Table is a simple form of data storage. A table is also considered as a convenient representation of relations.

Create Table

When you have new entities and relationships to store in your database, you can create a new database table using the CREATE TABLE statement.

```
CREATE TABLE IF NOT EXISTS table_name(
  column1 DataType TableConstraint DEFAULT default_value,
  column2 DataType TableConstraint DEFAULT default_value,
  column3 DataType TableConstraint DEFAULT default_value,
  ....
  );
```

The structure of the new table is defined by its *table schema*, which defines a series of columns. Each column has a name, the type of data allowed in that column, an *optional* table constraint on values being inserted, and an optional default value.

If there already exists a table with the same name, the SQL implementation will usually throw an error, so to suppress the error and skip creating a table if one exists, you can use the **IF NOT EXISTS** clause.

Here's an example schema for the Employee table

```
CREATE TABLE employee (
    empId INT ,
    empName VARCHAR(50),
    joining DATE,
    salary INT);
```

Task : Create a student table with the following columns:

```
rollNo INT,
studentName VARCHAR(50),
branch VARCHAR(10),
semester INT
```

Solution:

```
CREATE TABLE student (
 rollNo INT,
studentName VARCHAR(50),
branch VARCHAR(10),
semester INT
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DDL Command - ALTER TABLE

As your data changes over time, SQL provides a way for you to update your corresponding tables and database schemas by using the ALTER TABLE statement to add, remove, or modify columns and table constraints.

Adding columns

You need to specify the data type of the column along with any potential table constraints and default values to be applied to both existing and new rows.

```
ALTER TABLE mytable
ADD column DataType OptionalTableConstraint
    DEFAULT default_value;
```

The following command adds a department column in the employee table

```
ALTER TABLE employee ADD dept VARCHAR(30);
```

Task: Add a column named **gender** with a **VARCHAR(1)** data type to a predefined **student** table.

Solution:

```
/*CREATE TABLE student (
   rollNo INT,
);
Table is already created
*/
ALTER TABLE student ADD gender VARCHAR(1);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DDL Command - ALTER TABLE - 2

ALTER command is also used to change the structure of a table.It can also change the data type of a particular column,delete the column from table structure or rename a column.
MODIFY command is used to change the column definition of the table.
Syntax as follows:
ALTER TABLE table_name MODIFY column_name column_definition;
To modify the emp_name column of the Employee table, the following query will be written:
ALTER TABLE EMPLOYEE MODIFY EmpName VARCHAR(20);
And for deleting a column from the table DROP COLUMN command is used.It's syntax is given as follows:
ALTER TABLE table_name DROP COLUMN column_name;
To delete the salary column of the Employee table, the following query will be written:
ALTER TABLE EMPLOYEE DROP COLUMN salary;
To rename a column , the RENAME command is used. Syntax as follows:
ALTER TABLE table_name RENAME COLUMN old_column_name TO new_column_name;
To rename the column phone to mobile_no, the following query will be executed:
ALTER TABLE EMPLOYEE RENAME COLUMN phone TO mobile_no;

Task

Given a predefined table **'STUDENT',**perform the following tasks on that table:
Rename the column named 'phone' to 'contact'Change the data-type of the column 'stud_name' to 'VARCHAR(30)'Delete the column named 'marks' from the table.

Solution:

```
ALTER TABLE students RENAME COLUMN phone TO contact;
ALTER TABLE students MODIFY stud_name VARCHAR(30);
ALTER TABLE students DROP COLUMN marks;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DDL Command - RENAME TABLE

Because business requirements change, we need to rename the current table to a new one to better reflect the new situation. SQL provides us with a very useful statement that changes the name of one or more tables.
To change one or more tables, we use the RENAME statement as follows:
ALTER TABLE existing_table RENAME TO new_table;
The **existing_table** (current table name) must exist, and the new_table (new table name) must not. If the new table does exist, the statement will fail.
Here's an example to RENAME the table

```
/* Create a table employee */
þÿCREATE TABLE employee (
   empId INT ,
   empName VARCHAR(50),
   joining DATE,
   salary INT);
ALTER TABLE  employee RENAME TO emp_data;
```

Task: Rename the predefined student table to student_data

Solution:

```
/*CREATE TABLE student (
);
Table is already created
*/
alter table student rename to student_data;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DDL Command - DROP TABLE

You may want to remove an entire table including all of its data and metadata and to do so, you can use the DROP TABLE statement.
DROP TABLE IF EXISTS mytable;
Like the CREATE TABLE statement, the database may throw an error if the specified table does not exist, and to suppress that error, you can use the IF EXISTS clause.
In addition, if you have another table that is dependent on columns in table you are removing then you will have to either update all dependent tables first to remove the dependent rows or to remove those tables entirely.
The following command removes the **employee** table
DROP TABLE IF EXISTS employee;

þÿTask: Drop the predefined **student** table from the database.

Solution:

```
/*CREATE TABLE student (
);
Table is already created
*/
drop table if exists student;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - INSERT

When inserting data into a database, we need to use an INSERT statement, which declares which table to write into, the columns of data that we are filling, and one or more rows of data to insert. In general, each row of data you insert should contain values for every corresponding column in the table. You can insert multiple rows at a time by just listing them sequentially.

```
/*Insert statement with values for all columns*/
INSERT INTO table_name VALUES(value1,value2,..valueN);
```

Here's an example to insert in the **employee** table

```
CREATE TABLE employee (
   empId INT ,
   empName VARCHAR(50),
   joining DATE,
   salary INT);
/* Insert single row*/
INSERT INTO employee VALUES (1,"David",'14-09-18',12000);
/*Insert multiple rows*/
INSERT INTO employee VALUES (2,"Burns",'14-09-16',14000),
(3,"Rohit",'14-06-20',10000);
```

In some cases, if you have incomplete data and the table contains columns that support default values, you can insert rows with only the columns of data you have by specifying them explicitly.

```
/*Insert statement with specific columns*/
INSERT INTO mytable
(column, another_column, …)
VALUES (value_or_expr, another_value_or_expr, …),
     (value_or_expr_2, another_value_or_expr_2, …),
     …;
```

Insert only id & name of the employee

```
INSERT INTO employee
   (empId , empName)
   VALUES
   (4,"Rohit"),
   (5,"David");
```

Refer the below schema & complete the following task.

```
CREATE TABLE IF NOT EXISTS student (
  rollNo INT,
 studentName VARCHAR(50),
 branch VARCHAR(10),
 semester INT
);
```

Task1.: Insert the following data in the predefined student table.

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

rollNo	studentName	branch	semester
111	Ashmit	CSE	4
124	Paras	ECE	6
547	Ravi	CSE	
654	Sumit	CSE	

Solution:

```
/*CREATE TABLE IF NOT EXISTS student (
    rollNo INT PRIMARY KEY,
   studentName VARCHAR(50),
   branch VARCHAR(10),
   semester INT
);
Table is already created
*/
insert into student values(111,'Ashmit','CSE',4);
insert into student values(124,'Paras','ECE',6);
insert into student values(547,'Ravi','CSE',NULL);
insert into student values(654,'Sumit','CSE',NULL);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - DELETE

When you need to delete data from a table in the database, you can use
a DELETE statement, which describes the table to act on, and the rows of the table to delete through the WHERE clause.
/*Delete statement with condition*/
DELETE FROM mytable
WHERE condition;
If you decide to leave out the WHERE constraint, then all rows are removed, which is a quick and easy way to clear out a table completely.
Refer to the below Table & delete the rows where salary is not present.

empId	empName	joining	salary
1	David	14-09-2018	12000
2	Burns	14-09-2016	14000
3	Rohit	14-06-2020	10000
4	Rohit		
5	David		

DELETE FROM employee WHERE salary is NULL;
/*All rows where salary is not exist, removed from the table*/

empId	empName	joining	salary
1	David	14-09-2018	12000
2	Burns	14-09-2016	14000
3	Rohit	14-06-2020	10000

The predefined student table needs to be cleaned up a little bit, so try and delete a few rows in the tasks below.

Task 1: WHERE branch = 'ECE'

Task 2: WHERE semester is NULL

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Solution:

```
/*CREATE TABLE IF NOT EXISTS student (
   rollNo INT PRIMARY KEY,
    studentName VARCHAR(50),
    branch VARCHAR(10),
    semester INT
);
Table is already created
*/
delete from student where branch ="ECE";
delete from student where semester is NULL;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - UPDATE

In addition to adding new data, a common task is to update existing data, which can be done using an UPDATE statement. Similar to the INSERT statement, you have to specify exactly which table, columns, and rows to update. In addition, the data you are updating has to match the data type of the columns in the table schema.

```
/*Update statement with values*/
UPDATE mytable
SET column = value_or_expr,
    other_column = another_value_or_expr,
    …
WHERE condition;
```

The statement works by taking multiple column/value pairs, and applying those changes to each and every row that satisfies the constraint in the WHERE clause.

Refer to the below Table & update the column salary where salary is not present.

empId	empName	joining	salary
1	David	14-09-2018	12000
2	Burns	14-09-2016	14000
3	Rohit	14-06-2020	10000
4	Rohit		
5	David		

UPDATE employee SET SALARY = 8000 WHERE salary is NULL;

empId	empName	joining	salary
1	David	14-09-2018	12000
2	Burns	14-09-2016	14000
3	Rohit	14-06-2020	10000
4	Rohit		8000
5	David		8000

Refer the below schema & complete the following task.

```
CREATE TABLE IF NOT EXISTS student (
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

```
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT
);
```

Task 1: UPDATE the **studentName** of **rollNo** 654 to "**Sumit Sharma**"
Task 2: UPDATE the column **semester** to 4 where the **semester** is not present.

Solution:

```
/*CREATE TABLE IF NOT EXISTS student (
   rollNo INT PRIMARY KEY,
    studentName VARCHAR(50),
    branch VARCHAR(10),
    semester INT
);
Table is already created
*/
update student set studentName = "Sumit Sharma" where rollNo=654;
update student set semester =4 where semester is NULL;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - SELECT

To retrieve data from a SQL database, we need to write SELECT statements, which are often colloquially referred to as queries. A query in itself is just a statement which declares what data we are looking for, where to find it in the database, and optionally, how to transform it before it is returned.
Select query for a specific columns
SELECT column, another_column, …
FROM mytable;
The result of this query will be a two-dimensional set of rows and columns, effectively a copy of the table, but only with the columns that we requested.
If we want to retrieve absolutely all the columns of data from a table, we can then use the asterisk (*****) shorthand in place of listing all the column names individually.

```
/*Select query for all columns*/
SELECT *
FROM mytable;
```

This query, in particular, is really useful because it's a simple way to inspect a table by dumping all the data at once.

Here's an example to select in the employee table

```
/* Create a table employee*/
þÿCREATE TABLE employee (
   empId INT ,
   empName VARCHAR(50),
   joining DATE,
   salary INT);

/* Insert some data in table*/
INSERT INTO employee VALUES
   (1,"David",'14-09-18',12000),
   (2,"Burns",'14-09-16',14000),
   (3,"Rohit",'14-06-20',10000);
```

empId	empName	joining	salary
1	David	14-09-2018	12000
2	Burns	14-09-2016	14000
3	Rohit	14-06-2020	10000


```
/*select the empName column from the table */
SELECT empName FROM employee ; /* It'll return 3 rows with single column empName  */
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

David
Burns
Rohit

/* select empName & salary columns from the table*/
SELECT empName, salary FROM employee ;þÿ

David	12000
Burns	14000
Rohit	10000

/* select all columns from the table*/
SELECT * FROM employee ;þÿ

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

1	David	14-09-2018	12000
2	Burns	14-09-2016	14000
3	Rohit	14-06-2020	10000

Refer the below schema & complete the following task.

```
CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT
);
```

Task 1: Get all the **student** Names from the **student** table.
Task 2: Get **branch & rollNo** from the **student** table.
Task 3: Get all the columns from the **student** table.
Note: All the tasks should be performed in sequence.

Solution:

```
/*CREATE TABLE IF NOT EXISTS student (
   rollNo INT PRIMARY KEY,
    studentName VARCHAR(50),
    branch VARCHAR(10),
    semester INT
);
Table is already created
*/
select studentName from student;
select branch,rollNo from student;
select * from student;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - SELECT with constraints

Now we know how to select for specific columns of data from a table, but if you had a table with a hundred million rows of data, reading through all the rows would be inefficient and perhaps even impossible.
In order to filter certain results from being returned, we need to use a **WHERE** clause in the query. The clause is applied to each row of data by checking specific column values to determine whether it should be included in the results or not.

```
/*Select query with constraints*/
SELECT column, another_column, …
FROM mytable
WHERE condition
```

Refer to the below Table & select the column salary where salary is not present.

empId	empName	joining	salary
1	David	14-09-2018	12000
2	Burns	14-09-2016	14000
3	Rohit	14-06-2020	10000
4	Rohit		
5	David		

SELECT * FROM employee WHERE salary is NULL;

4	Rohit
5	David

Refer the below schema & complete the following task.

```
CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

```
  semester INT
);
```

Task 1: Get all the columns from the student table WHERE **studentName** is not present.
Task 2: Get branch & rollNo from the student table WHERE **studentName** is not present.
Task 3: Get all the columns from the student table WHERE branch is not present..

Solution:

```
/*CREATE TABLE IF NOT EXISTS student (
   rollNo INT PRIMARY KEY,
    studentName VARCHAR(50),
    branch VARCHAR(10),
    semester INT
);
Table is already created
*/
select * from student where studentName is null;
select branch, rollNo from student where studentName is null;
select * from student where branch is null;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - SELECT with Relational Operators

In SQL, relational or comparison operators (e.g., >, <, =) are used to compare values and value expressions and return a logical value, which is either **true(1)** or **false(0)**.

The following are the different types of comparison or relational operators available in SQL. Here for example, variables x, y, are holding values like **x = 10**, **y = 20**.
< (Less than) - It will check if the left operand value less than right-hand operand value or not.
(x<y) = 0
<= (Less than equal to) - It will check if the left operand value less than or equal to right-hand operand value or not.
(x<=y) = 1
> (Greater than) - It will check if the left operand value greater than right-hand operand value or not.
(x>y) = 0
>= (Greater than equal to) - It will check if the left operand value greater than or equal to right-hand operand value or not.
(x>=y) = 0
=, == (Equal to) - It will check if both operand values equal or not
(x=y) = 0,
(x==y) = 0
<> ,!= (Not Equal to) – It will check if two operand values are equal or not. In case if the values not equal then it will return true
(x<>y) = 1,
(x!=y) = 1
Refer to the below **employee** Table & select the rows where **empID** is greater than 5..

empId	empName
1	Paul
2	Allen
3	Teddy
4	Mark
5	David
6	Davidson
7	Markson

Query:
select * from employee where empID >= 6;
Output:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

6	Davidson
7	Markson

Query:
select * from employee where EmpID != 5;
Output:

1	Paul
2	Allen
3	Teddy
4	Mark
6	Davidson
7	Markson

Refer the below schema & complete the following task.

```
CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT
);
```

Task 1: Get all the columns from the student table WHERE **studentName** is "**Ashmit**"

Task 2: Get all the columns from the student table **except studentName**, "**Ravi**"

.

Task 3: Get all the columns from the student table with **branch "CSE**"

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Task 4: Get rollNo from the student table WHERE rollNo greater than 653

Task 5: Get rollNo from the student table WHERE rollNo less than 654

Solution:

```
/*CREATE TABLE IF NOT EXISTS student (
   rollNo INT PRIMARY KEY,
    studentName VARCHAR(50),
    branch VARCHAR(10),
    semester INT
);
Table is already created
*/
CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT
);
select * from student where studentName='Ashmit';
select * from student where studentName!='Ravi';
select * from student where branch='CSE';
select rollNo from student where rollNo>653;
select rollNo from student where rollNo<654;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - SELECT with Logical Operators

In SQL, **Logical operators** (AND, OR, NOT, IN, Between, Exists, etc.) are used to define multiple conditions in SQLite statements to return rows or records from a statement based on the defined conditions.
The following are the different types of logical operators

AND : Multiple conditions can be defined in SQL statements and it will return records only when all the defined conditions are true.
OR : Multiple conditions can be defined in SQL statements and it will return records if any one of defined conditions is true.
IN : By using IN operator it can be checked if a given value exists in the list of values or in the subquery result set.
NOT IN : It's just opposite to the IN operator. By using the NOT IN operator it can be checked if a given value exists in the list of values or in the subquery result set.
BETWEEN : We can return values which exist between a defined range of values.
EXISTS : By using EXISTS we can check whether the required value exists in the result set or not
Refer to the below employee Table & select the rows where **age** is greater than 30 & **salary** less than equal to 20000.

empId	empName	age	salary
1	Paul	24	10000
2	Allen	34	20000
3	Teddy	20	25000
4	Mark	34	40000
5	David	45	35000
6	Davidson	22	10000
7	Markson	33	27500

Query 1:
select * from employee where age > 30 AND salary <= 20000;
Output:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

2	Allen	34	20000

Query 2:
select * from employee where age > 40 OR salary <= 20000;Output:
Output:

1	Paul	24	10000
2	Allen	34	20000
5	David	45	35000
6	Davidson	22	10000

Query 3:
select * from employee where empName IN ("Teddy" , "Covid", "David");
Output:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

3	Teddy	20	25000
5	David	45	35000

Query 4:
select * from employee where age BETWEEN 30 AND 40;
Output:

2	Allen	34	20000
4	Mark	34	40000
7	Markson	33	27500

Refer the below schema & complete the following task.

```
CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT,
  marks INT
þÿ  );
```

Task 1 : Get rollNo & studentName,from student table who are from semester 4 and scored greater than 75 marks.
Task 2: Get roll No & student Name,from student table who are from branch "**ECE**" or **semester 6.**
Task 3: Get **rollNo** from table **student** who are from branch "**CSE" or scored** less than **40**

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

marks**.**

Solution:

```
/*CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT,
  marks INT
);
Table is already created */
select rollNo,studentName from student where semester=4 and marks>75;
select rollNo, studentName from student where branch="ECE" or semester =6;
select rollNo from student where branch="CSE" or marks<40;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - SELECT with Logical Operators 2

Refer the below schema & complete the following task.

```
CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT,
  marks INT
þÿ  );
```

Task 1: Get rollNo of the students who scored marks between 55 & 65.
Task 2: Get all the data of the students table who are from branch IT , ECE & ME.
Task 3: Get all the data of the students table who doesn't belongs to following branches
i) IT
ii) ECE
iii) ME

Solution:

```
/*CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT,
  marks INT
);
Table is already created */
CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT,
  marks INT
);
select rollNo from student where marks between 55 and 65;
select * from student where branch in("IT","ECE","ME");
select * from student where branch NOT in("IT","ECE","ME");
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - SELECT with Arithmetic Operators

In SQL,Arithmetic operators are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc. in statements.
The following are the different types of comparison or relational operators available in SQL.
Here for example, variables x, y, are holding values like **x = 10**, **y = 20**.
+ (Addition) - It will perform the addition operation.
x+y = 30
- (Subtraction) – It will perform the subtraction operation
x-y = -10
***(Multiplication)** – It will perform the multiplication operation
x*y = 200
/ (Division) – It will perform the division operation.(Returns an integer value)
x/y = 0
% (Modulus) – It will perform modulus operation and return remainder value
x%y = 10
Refer to the below employee Table & select those employee who's age is an even number.

empId	empName	age	salary
1	Paul	24	10000
2	Allen	34	20000
3	Teddy	20	25000
4	Mark	34	40000
6	Davidson	22	10000

Query 1:
select * from employee where age%2 = 0;
Output:

1	Paul	24	10000
2	Allen	34	20000
3	Teddy	20	25000
4	Mark	34	40000
6	Davidson	22	10000

Refer the below schema & complete the following task.
CREATE TABLE IF NOT EXISTS student (
rollNo INT,
studentName VARCHAR(50),
branch VARCHAR(10),
semester INT,
marks INT
þÿ);
Task 1 : Get all details of the students,If we cut their marks by half, it must be more than 40

Solution:

/*CREATE TABLE IF NOT EXISTS student (

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

```
 rollNo INT,
 studentName VARCHAR(50),
 branch VARCHAR(10),
 semester INT,
 marks INT
);
Table is already created */
select * from student where marks/2>40;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: DML Command - SELECT with Like Operators

In SQL, LIKE operator is used to check whether the given string value matches with specific pattern or not. In case if string value matches with pattern value then it will return true.
In SQL, LIKE operator is case insensitive, so 'a' LIKE 'A' is true and all non-NULL parameter expressions will be converted to text values.
SELECT column1,column2..columnN FROM table_name WHERE expression LIKE pattern;
In above SQL LIKE syntax,
expression is the character expression such as a column or field.
pattern is the character expression that contains pattern matching.
In SQL,LIKE operator pattern syntax supports two wildcards.
% allows you to match any string of any length (including zero-length).
_ allows you to match on a single character.
The following details give an idea to use Like operator with wildcard characters %, _ to get records from tables.

Where name LIKE 'a%' - It returns all the records whose name starts with 'a'
Where name LIKE '%b' - It returns all the records whose name ends with 'b'
Where name LIKE '%c%' - It returns all the records whose name contains 'c' character
Where name LIKE 'Dav_' - It returns records whose name starts with 'Dav' and contains 4 characters
Where name LIKE '_a%' - It returns records whose name contains the second character as 'a' and it should contain a minimum of 2 characters
Refer to the below employee Table & select the rows where **empName** start with **Mark**.

empId	empName	age	salary
1	Paul	24	10000
2	Allen	34	20000
3	Teddy	20	25000
4	Mark	34	40000
5	David	45	35000
6	Davidson	22	10000
7	Markson	33	27500

Query 1:
select * from employee where empName LIKE "Mark%";
Output:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

4	Mark	34	40000
7	Markson	33	27500

Refer the below schema & complete the following task.

```
CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT,
  marks INT
þÿ  );
```

Task 1 : Get the details of the student whose name end with "it".

Solution:

```
/*CREATE TABLE IF NOT EXISTS student (
   rollNo INT,
  studentName VARCHAR(50),
  branch VARCHAR(10),
  semester INT,
  marks INT
);
Table is already created */
select * from student where studentName like "%it";
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Fetch Unique Values from Table

Write An SQL Query To fetch unique values of Department from Worker table.

Solution:

```
SELECT DISTINCT Department
FROM Worker;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print records in which field value lies in between a range

Write An SQL Query To Print Details Of The Workers Whose SALARY Lies Between 10000 And 50000.

Solution:

```
/*Print Details Of The Workers Whose SALARY Lies Between 10000 And 50000.
 Table 'Worker' is already created.*/
 select * from worker where salary between 10000 and 50000;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Fetch First Name using the Alias Name

Aliases are used in SQL to give a table or a column of table a temporary name.Below given is the syntax to give an alias:
SELECT column_name AS alias_name FROM table_name;
To select employee name as 'name' from the employee table, the following query will be written:
SELECT emp_name AS name from employee;

Task:

A company maintains the details of all its workers in the Worker Table. Write An SQL Query to fetch "FirstName" from worker table using the Alias name as "FName".
Structure of the Worker Table is as below:
1. **WorkerID** INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT ,
2. **FirstName** CHAR(25),
3. **LastName** CHAR(25),
4. **Salary** INT(15),
5. **JoiningDate** DATETIME,
6. **Department** CHAR(25)

Solution:

```
/* Fetch "FirstName" from worker table using the Alias name as "FName".
  Table 'Worker' already created. */
  select FirstName as FName from Worker ;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print records in which field ends with a specific alphabet - 1

Write An SQL Query To Print Details Of The Workers Whose FirstName Ends With 't'.

Solution:

```
/* Print Details Of The Workers Whose FirstName Ends With 't'
 Table 'Worker' is already created */
 select * from worker where FirstName like "%t";
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print records in which field ends with a specific alphabet - 2

Write An SQL Query to print all the details of the Workers whose FirstName ends with 'v' And contains exactly 6 alphabets.

Solution:

```
/* Print all the details of the Workers whose FirstName ends with 'v' And contains exactly 6 alphabets.
 Table 'Worker' is already created.*/
 /* Print all the details of the Workers whose FirstName ends with 'v' And contains exactly 6 alphabets.
 Table 'Worker' is already created.*/
 select * from worker where firstname like "%_____v";
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print records in which field contains a specific alphabet

Write An SQL Query To Print Details Of The Workers Whose FirstName Contains 't'.

Solution:

```
/* Print Details Of The Workers Whose FirstName Contains 't'
Table 'Worker' is already created  */
SELECT *
FROM Worker
WHERE FirstName LIKE '%t%';
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print records of a specific department

Write An SQL query to print all the details of workers who are working in the department "Admin".

Solution:

```
/* Print all the details of workers who are working in the department "Admin".
 Table 'Worker' is already created */
 SELECT *
FROM Worker
WHERE Department = 'Admin';
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print records by excluding some values

Write an SQL Query to print details of workers excluding first names - "Lahsiv" and "Amitabh" from worker table.

Solution:

```
/* Print details of workers excluding first names - "Lahsiv" and "Amitabh"
 Table 'Worker' is already created */
 SELECT *
FROM Worker
WHERE FirstName NOT IN ('Lahsiv', 'Amitabh');
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print records in a range of values

Write an SQL Query to print details for Workers with the FirstName as “Amitabh” And “Vivek” from worker table.

Solution:

```
/* Print details for Workers with the FirstName as "Amitabh" And "Vivek"
 Table 'Worker' is already created */
 SELECT *
FROM Worker
WHERE FirstName IN ('Amitabh', 'Vivek');
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Check Constraint

CHECK Constraint is used to define specific conditions on a column to validate those conditions while inserting or updating the data in table columns.
Create a table book where the price of the book should be higher than 49.
CREATE TABLE books(Book_id INT,bookName VARCHAR(50) ,PRICE INT CHECK (PRICE > 49));
If you insert a lower price than 50 it gives the error.
INSERT INTO books VALUES(3,'Operating Systems',0);
Output:
Error: CHECK constraint failed: books

Task 1: Create a table **students** with column id (INT), name (VARCHAR 50) & age (INT). Apply a check on column age, which should be higher than 15.

Solution:

```
/* create your table students*/CREATE TABLE students (
    id INT,
    name VARCHAR(50),
    age INT CHECK (age > 15)
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Default Constraint

In SQL, Default Constraint is used to define default values for a column. Generally, in SQL, default constraint will insert default value in a column in case if the column value is null or empty.
Create a table book where the default price of the book is 500.
CREATE TABLE books(Book_id INT, bookName VARCHAR(50) , PRICE INT DEFAULT 500);
Query:
INSERT INTO books(Book_id,bookName) VALUES(1,'DBMS');
Output:

1	DBMS	500

Task 1: Create a table students with column id (INT), name (VARCHAR 50), age (INT) and default branch (VARCHAR 10) "CSE".

Solution:

```
/* create your table students*/
CREATE TABLE Students (
    id INT,
    name VARCHAR(50),
    age INT,
    branch VARCHAR(10) DEFAULT 'CSE'
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Primary Key Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table. There can be more UNIQUE columns, but only one primary key in a table. Primary keys are important when designing database tables. Primary keys are unique IDs. They are used to refer to table rows.
Create a table book where the Book_id is the PRIMARY KEY.
CREATE TABLE books(Book_id INTEGER PRIMARY KEY,bookName VARCHAR(50));
One exception to the typelessness of SQLite is a column whose type is INTEGER PRIMARY KEY. (And you must use "INTEGER" not "INT". A column of type INT PRIMARY KEY is typeless just like any other.) INTEGER PRIMARY KEY columns must contain a 32-bit signed integer. Any attempt to insert non-integer data will result in an error.
INTEGER PRIMARY KEY columns can be used to implement the equivalent of AUTOINCREMENT. If you try to insert a NULL into an INTEGER PRIMARY KEY column, the column will actually be filled with an integer that is one greater than the largest key already in the table
Query:

```
INSERT INTO books VALUES(1,'DBMS'),
   (2,'Programming Using C'),
   (2,'Operating Systems');
```

Output :
Error: UNIQUE constraint failed: books.Book_id
Task 1: Create a table students with column id (INTEGER), name (VARCHAR 50) & age (INT). Apply PRIMARY KEY constraint on id.

Solution:

```
/* create your table students*/
CREATE TABLE Students (
   id INTEGER PRIMARY KEY,
   name VARCHAR(50),
   age INT
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Not Null Constraint

A column with a NOT NULL constraint cannot have NULL values.
By default, columns will allow NULL values in case if you have requirements like not to allow NULL values in column means then we need to add NOT NULL constraint on column.
Create a table books & apply NOT NULL check on book's name
CREATE TABLE books(Book_id INTEGER PRIMARY KEY,bookName VARCHAR(50) NOT NULL);
Query:
INSERT INTO books(Book_id) VALUES(1);
Output:
Error: NOT NULL constraint failed: books.bookName
Task1: Create a table students with column id (INTEGER), name (VARCHAR 50) & age (INT). The name of the student shouldn't be empty.

Solution:

```
/* create your table students*/
CREATE TABLE Students (
    id INTEGER PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age INT
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Unique Constraint

Unique constraint is used to set restriction on a column to allow only unique values in the column.
The SQL Unique Constraint same as the Primary Key constraint except that Unique Constraint will accept single Null value but the Primary Key will not accept any Null values and we can create a number of Unique Constraints on the table but we can create only one Primary Key constraint on a table.

Create a table **books** & apply UNIQUE constraint on Book's ID.
CREATE TABLE books(BookId INT UNIQUE,bookName VARCHAR(50));
Query:
INSERT INTO books VALUES(1,'DBMS');
INSERT INTO books VALUES(1,'Programming Using C');
Output:
Error: UNIQUE constraint failed: books.BookId

Task 1: Create a table ‘students’ with columns **id** (Integer), **name** varchar(50) and **age**(int). And apply the UNIQUE constraint on the id of the students table.

Solution:

```
/* create your table students*/
CREATE TABLE students (
   id INTEGER UNIQUE,
   name VARCHAR(50),
   age INT
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Foreign Key Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table. It is a referential constraint between two tables. The foreign key identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table.

Create a Department table with columns id as PRIMARY KEY & dept. name. & insert the data
CREATE TABLE dept(ID INT PRIMARY KEY,Deptname VARCHAR(50));
INSERT INTO dept VALUES(1,'ADMIN');
INSERT INTO dept VALUES(2,'HR');

Create a table employee with columns emp's ID, name, & dept.id as foreign key.
CREATE TABLE employee(EmpID INT,Empname VARCHAR(50),DeptID INT,FOREIGN KEY(DeptId) REFERENCES dept(ID));
Query:
INSERT INTO employee VALUES(1,'Rohit',3);
Output:
Error: FOREIGN KEY constraint failed
Explanation
When we insert 3 as dept.id, it's not present in the **dept** table. So it gives an error.

Task 1: Create a table **dept** with columns **dept_id**(INT),**dept_name**(Varchar(50)). Make the **dept_id** a primary key in the table.
Task 2: Create a table Students with columns roll_no(INT),student_name(varchar(50)) ,age(INT) and dept_id(INT). Make the dept_id a foreign key in the students table

Solution:

```
/* create your table students & dapartment*/
CREATE TABLE dept (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(50)
);
CREATE TABLE Students (
    roll_no INT,
    student_name VARCHAR(50),
    age INT,
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES dept(dept_id)
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Group By Clause

The SQL GROUP BY clause is used with the SELECT statement to collaborate the same identical elements into groups.
The GROUP BY clause is used with WHERE clause in the SELECT statement and precedes the ORDER BY clause
SELECT column-list FROM table_name WHERE [conditions] GROUP BY column1, column2....columnN;
Refer to the below **employee** table & count the number of employees in each department

EmpID	EmpName	Dept	Age	SALARY
1	Paul	Engineering	24	10000
2	Allen	Admin	34	20000
3	Teddy	Accounts	20	25000
4	Mark	HR	34	40000
5	David	Engineering	45	35000
6	Davidson	HR	22	10000
7	Markson	Accounts	33	27500

Query:
select Dept,COUNT(EmpID) from employee group by dept;
In the above query, we use the COUNT function to count the number of employees. The group by clause groups all the records together that are in the same department and the count function returns the number of EmpID’s that are present in the particular group.
Output as follows:
[image:https://lh3.googleusercontent.com/cfzL_GUnmm1d2LYfAaFnFlJRXb50PvS0RuorYdMEyCw9IBWVGcwnQghrZ5qTT8-twr99YeSeJKk7vyv_4q51z-KkYNWBVq-9sIbklGT3_jbXg0IaTk9-6iQuqHr1bIYGqQy1FObr]

A table students is already created with the following schema

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Table Name : Students

Column Name	Data type
roll_no	INTEGER PRIMARY KEY
stud_name	VARCHAR(50)
stud_age	INT
stud_dept	VARCHAR(50)

Task: Write a query to display all the student's department (**stud_dept**) followed by the count of students in that particular department.

Solution:

```
/*create table students (
  roll_no INTEGER PRIMARY KEY,
  stud_name varchar(50),
  stud_age int,
  stud_dept varchar(50)
);
Table already created.
*/
SELECT stud_dept, COUNT(*)
FROM students
GROUP BY stud_dept;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Having Clause

The SQL HAVING clause is used to specify conditions that filter which group results appear in the final results. The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause. HAVING clause is often used with the GROUP BY clause. The GROUP BY clause groups a set of rows into a set of summary rows or groups. Then the HAVING clause filters groups based on a specified condition.
While using the HAVING clause, GROUP BY clause must be included; otherwise, the following error will pop-up;
Error: a GROUP BY clause is required before HAVING
Syntax:
SELECT column1_list FROM table_name WHERE [conditions] GROUP BY column_name HAVING [conditions];
Refer to the below employee table & count the number of employees in '**HR**' department

EmpID	EmpName	Dept	Age	SALARY
1	Paul	Engineering	24	10000
2	Allen	Admin	34	20000
3	Teddy	Accounts	20	25000
4	Mark	HR	34	40000
5	David	Engineering	45	35000
6	Davidson	HR	22	10000
7	Markson	Accounts	33	27500

Query:
select Dept,COUNT(EmpID) from employee group by dept having dept='HR';
The group by clause groups all the records together that are in the same department and the count function returns the number of EmpID's that are present in the particular group.Then the having clause displays the records which are having 'HR' department.
Output as follows:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

[image:https://lh3.googleusercontent.com/bAsqgnNkw3tJUB01vSyvFaATVpAO-HGcU7OSr1ZOifijA8QNIuPdi85BRp5hD5CqYfAvyzSk0eR1LuMNLQnBQHhP7jokGzKy1E7yqbJlsJ4m57ONiWUy-WldumsD1Q846syqf9YD]

Note: We can use logical operators in having clause just the same way it is used in the where clause.

A table students is already created with the following schema

Table Name : Students

Column Name	Data type
roll_no	INTEGER PRIMARY KEY
stud_name	VARCHAR(50)
stud_age	INT
stud_dept	VARCHAR(50)

Task: Write a query to display the stud_dept followed by the count of the students of the ‘CSE’ and ‘Mechanical’ department.
þÿ

Solution:

```
SELECT stud_dept, COUNT(*)
FROM students
GROUP BY stud_dept
HAVING stud_dept = 'CSE' OR stud_dept = 'Mechanical';
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Order By Clause

The SQL ORDER BY clause is used to sort the fetched data in ascending or descending order, based on one or more columns.
The ORDER BY clause comes after the FROM clause. It allows you to sort the result set based on one or more columns in ascending or descending order.
In the below syntax, place the column name which is to be sorted after the ORDER BY clause followed by the ASC or DESC keyword.
The ASC keyword means ascending.And the DESC keyword means descending.
If it isn't specified through the ASC or DESC keyword, SQL sorts the result set using the ASC option. In other words, it sorts the result set in the ascending order by default.
To sort the result set by multiple columns, use a comma (,) to separate two columns. The ORDER BY clause sorts rows using columns or expressions from left to right. In other words, the ORDER BY clause sorts the rows using the first column in the list. Then, it sorts the sorted rows using the second column, and so on.
SELECT column-list FROM table_name [WHERE condition] [ORDER BY column1,column2,..columnN] [ASC | DESC];
Refer to the below employee table & sort the rows by EmpName.

EmpID	EmpName	Dept	Age	SALARY
1	Paul	Engineering	24	10000
2	Allen	Admin	34	20000
3	Teddy	Accounts	20	25000
4	Mark	HR	34	40000
5	David	Engineering	45	35000
6	Davidson	HR	22	10000
7	Markson	Accounts	33	27500

Query:
select * from employee order by EmpName;
Output:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

[image:https://lh6.googleusercontent.com/KJ63qMrngDH4MnPUtMs3cHlPdytzxpIOcW4aRqwrONM_0SUTxKCVNf5hHqT3qESO4DcTEaa3dVko7G6SzG3RsvhSPomtxT4-iQ6jxwQW72sKQK4jEVQ7OlUcCcnfGCvRMfcYlEOK]

A table students is already created with the following schema

Table Name : Students

Column Name	Data type
roll_no	INTEGER PRIMARY KEY
stud_name	VARCHAR(50)
stud_age	INT
stud_dept	VARCHAR(50)

Task: Display all the records of the table in ascending order of the student's names.

Solution:

```
SELECT *
FROM students
ORDER BY stud_name ASC;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: MAX / MIN Aggregate Functions

The **min()** aggregate function returns the minimum non-NULL value of all values in the group. The minimum value is the first non-NULL value that would appear in an ORDER BY of the column. The function returns NULL if and only if there are no non-NULL values in the group.
Select MIN(expression) from table_name where [Conditions];

The **max()** aggregate function returns the maximum value of all values in the group. The maximum value is the value that would be returned last in an ORDER BY on the same column. The function returns NULL if and only if there are no non-NULL values in the group.
Select MAX(expression) from table_name where [Conditions];
Where expression can be a column_name or a valid SQL expression.
These functions can also be used while using the group by and having clauses.

Refer to the below employee table & get the maximum and minimum salaries of the employee.

EmpID	EmpName	Dept	Age	SALARY
1	Paul	Engineering	24	10000
2	Allen	Admin	34	20000
3	Teddy	Accounts	20	25000
4	Mark	HR	34	40000
5	David	Engineering	45	35000
6	Davidson	HR	22	10000
7	Markson	Accounts	33	27500

Query:
select MAX(SALARY),MIN(SALARY) from employee;
Output:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

MAX(SALARY)	MIN(SALARY)
40000	10000

A table students with the following schema is already created
[image:https://lh4.googleusercontent.com/3XgkyWyIudmgtiwlX9KSq13lyDSK9wohDhAwOQcHTRkkoksX_E9aIgo2-Uu73q5lW41nZb42tfRSaYORfkoNVaAhfJfuc4-HmbT8xvG8UZFds5_6EE0wTw1fupcRn_WA6oi1jjau]

Task: Find the highest-scoring student’s marks and the lowest scoring student's marks.

Solution:

```
SELECT MAX(stud_marks), MIN(stud_marks)
FROM students;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: COUNT Function

In SQL **Count()** Function is an aggregate function that is used to return the total number of rows in a table based on the specified expression or conditions.

Generally, the SQL **Count()** function will work with non-NULL numeric values to return a number of times non-NULL values exist in columns based on defined expressions.

In **Count()** function, if we define the expression as an asterisk character (*) then the COUNT function will return the number of rows in the table based on the defined aggregate group or condition.

We can use the where clause while using the COUNT function also.

Refer to the below employee table & count no. of employees present in the table.

EmpID	EmpName	Dept	Age	SALARY
1	Paul	Engineering	24	10000
2	Allen	Admin	34	20000
3	Teddy	Accounts	20	25000
4	Mark	HR	34	40000
5	David	Engineering	45	35000
6	Davidson	HR	22	10000
7	Markson	Accounts	33	27500

Query:

select COUNT(*) from employee;

Count(*)
7

A table students with the following schema is already created

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

[image:https://lh4.googleusercontent.com/3XgkyWyIudmgtiwlX9KSq13lyDSK9wohDhAwOQcHTRkkoksX_E9aIgo2-Uu73q5lW41nZb42tfRSaYORfkoNVaAhfJfuc4-HmbT8xvG8UZFds5_6EE0wTw1fupcRn_WA6oi1jjau]

Task: Display the count of students who have scored more than 50 marks.

Solution:

```
SELECT COUNT(*)
FROM students
WHERE stud_marks > 50;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Sum & Avg functions

The SUM function is an aggregate function that returns the sum of the non-NULL values or only the distinct values in a group.
The result of the SUM function is an integer if all input non-NULL values are integers. If any input value is neither an integer nor a NULL value, the result of the SUM function is a floating-point value.
The result of the SUM function is NULL if and only if all input values are NULL.

The AVG function is an aggregate function that calculates the average value of all non-NULL values within a group.
The value of the AVG function is always a floating-point value or a NULL value. The AVG function only returns a NULL value if and only if all values in the group are NULL values.

Refer to the below employee table & get the count of employees, total salary, and the average salary given to employees.

EmpID	EmpName	Dept	Age	SALARY
1	Paul	Engineering	24	10000
2	Allen	Admin	34	20000
3	Teddy	Accounts	20	25000
4	Mark	HR	34	40000
5	David	Engineering	45	35000
6	Davidson	HR	22	10000
7	Markson	Accounts	33	27500

Query:
select COUNT(*),SUM(SALARY),AVG(SALARY) from employee;
Output:

Count(*)	SUM(SALARY)	AVG(SALARY)
7	167500	23928.57143

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

A table students with the following schema is already created
[image:https://lh4.googleusercontent.com/3XgkyWyIudmgtiwlX9KSq13lyDSK9wohDhAwOQcHTRkkoksX_E9aIgo2-Uu73q5lW41nZb42tfRSaYORfkoNVaAhfJfuc4-HmbT8xvG8UZFds5_6EE0wTw1fupcRn_WA6oi1jjau]

Task 1: Display the sum of all the marks of students who belong to the ECE department.
Task 2: Display the average of all the marks of the students who have scored more than 70.

Note: Display all the tasks using different select statements

Solution:

```
/*create table students (
  roll_no INTEGER PRIMARY KEY,
  stud_name varchar(50),
  stud_age int,
  stud_dept varchar(50),
  stud_marks INT
);
Table already created.
*/SELECT SUM(stud_marks)
FROM students
WHERE stud_dept = 'ECE';
SELECT AVG(stud_marks)
FROM students
WHERE stud_marks > 70;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: UPPER,LOWER & LENGTH Functions

SQL lower() converts all the characters in a string to lowercase characters.
The lower(str) function returns a copy of string str with all ASCII characters converted to lowercase. The default built-in lower() function works for ASCII characters only.

SQL upper() converts all the characters in a string to uppercase characters.
The upper(X) function returns a copy of input string X in which all lower-case ASCII characters are converted to their upper-case equivalent

For a string value str, the length(str) function returns the number of characters (not bytes) in str prior to the first NULL character.
Since SQL strings do not normally contain NULL characters, the length(str) function will usually return the total number of characters in the string str.If str is NULL then length(str) is NULL. If str is numeric then length(str) returns the length of a string representation of str.

Refer to the below employee table & display the name in upper case, department name in lower case and find the character's length of the name.

EmpID	EmpName	Dept	Age	SALARY
1	Paul	Engineering	24	10000
2	Allen	Admin	34	20000
3	Teddy	Accounts	20	25000
4	Mark	HR	34	40000
5	David	Engineering	45	35000
6	Davidson	HR	22	10000
7	Markson	Accounts	33	27500

Query:
select UPPER(EmpName),Lower(Dept),LENGTH(EmpName) from employee;
Output:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

[image:https://lh3.googleusercontent.com/q7hQb3nkBxN0zfypGJQ-dXJPIwyeG6Fnt9iTDFWYI-N-SMHReMMdMkGrUB7ymspcMd47XUhOmy730PFUzpEmU2tjGUETYm5ti1M7XFhfjOVznpHNDI4HoSKVMkl96AcRHsAVV85b]

A table students with the following schema is already created

Table Name : Students	
Column Name	Data type
roll_no	INTEGER PRIMARY KEY
stud_fname	VARCHAR(50)
stud_lname	VARCHAR(50)
stud_full_name	VARCHAR(50)
stud_age	INT
stud_dept	VARCHAR(50)
stud_marks	INT

Task 1: Display the student first names in uppercase.
Task 2: Display the student last names in lowercase.
Task 3: Display the length of the student's full name.
Note: Display all the tasks using different select statements.

Solution:

```
SELECT UPPER(stud_fname) AS UppercaseFirstName
FROM students;
SELECT LOWER(stud_lname) AS LowercaseLastName
FROM students;
SELECT LENGTH(stud_fname || ' ' || stud_lname) AS FullNameLength
FROM students;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Number Functions

These functions accept numeric input and return numeric values. Some of the functions under this category are ABS,ROUND, TRUNCATE, and MOD.
ROUND and **TRUNCATE** functions are used to round and truncate the number value.
Syntax:
ROUND(number,number_of_decimals_to_be_rounded);
TRUNCATE(number, decimals);
Example:
select ROUND(135.375, 2),TRUNCATE(123.45678, 2); -- Outputs 135.38 123.45

MOD is used to return the remainder of the division operation between two numbers
Syntax:
MOD(number1, number2);
Example:
select MOD(5, 3); -- Outputs 2

ABS is used to return the absolute value of the number passed in the functions
Syntax:
ABS(number);
Example:
select ABS(-5); -- Outputs 5
select ABS(2); -- Outputs 2

Task:
Given a predefined ‘students’ table with columns **‘roll_no’**(Integer Primary Key), **‘fname’** (VARCHAR(20)), **‘lname’** (VARCHAR(20)) and **‘CGPA’**(Double).Perform the given tasks on the given table:
Show the rounded CGPA of each student to 2 decimal places.Use '**Rounded**' as an alias name to show the rounded CGPA.Show the truncated CGPA of each student to 1 decimal place.Use '**Truncated**' as an alias name to show the truncated CGPA.
Note: Use different select statements for each task.

Solution:

```
/* Perform the given tasks using appropriate number functions
  Table 'students' already created*/
  SELECT ROUND(CGPA, 2) AS Rounded
FROM students;
SELECT TRUNCATE(CGPA, 1) AS Truncated
FROM students;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Date Functions

These functions help in date arithmetic operations and return date or numeric values. Some of the functions under this category are DATEDIFF,ADDDATE,DAY/MONTH/YEAR, DAYNAME/MONTHNAME ,SYSDATE.

DATEDIFF function returns the difference of days between the two dates.Returns NULL if any of the date is invalid.
Syntax:
DATEDIFF(date1,date2);
Example:
select DATEDIFF("2021-02-25", "2021-02-15"); -- Outputs 10

ADDDATE function adds interval passed into it to the date provided.
Syntax:
ADDDATE("2021-02-15", interval_value);
Example:
SELECT ADDDATE("2021-02-15", INTERVAL 10 DAY); -- Outputs 2021-02-25

DAY/MONTH/YEAR extracts the day/month/year from Date passed in the function.
Syntax:
[DAY | MONTH | YEAR](date);
Example:
select DAY("2021-02-25"); -- Outputs 25

DAYNAME/MONTHNAME returns the name of the day/month from the date passes in the functions
Syntax
[DAYNAME | MONTHNAME](date);
Example
select DAYNAME("2021-2-15"); -- Outputs Monday

SYSDATE function returns the current date and time in YYYY-MM-DD HH:MM:SS format .The date and time returned is according UTC.
Syntax:
SYSDATE();
Example:
select SYSDATE(); -- Outputs 2021-03-11 10:11:14

Task:
Given a predefined ‘students’ table with columns **‘roll_no’**(Integer Primary Key), **‘fname’** (VARCHAR(20)), **‘lname’** (VARCHAR(20)) and **‘birthdate’**(Date).Perform the given tasks on the given table:
Display the month number from the birthdate of students. Name the column as '**MonthNumber**' using alias. Display the day’s name of the student’s birthday.Name the column as '**Name**' using aliasDisplay the difference of student’s birthdate and the date ‘2021-03-11’.Display the absolute value of difference. Name the column as '**DateDifference**' using alias.
Note: Use different select statements to perform the tasks.

Solution:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

```
/* Perform the given tasks using appropriate date functions
  Table 'students' already created*/
SELECT MONTH(birthdate) AS MonthNumber
FROM students;
SELECT DAYNAME(birthdate) AS Name
FROM students;
SELECT ABS(DATEDIFF('2021-03-11', birthdate)) AS DateDifference
FROM students;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Character Functions

These functions accept a string and return the specified value. Functions under the category are CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD, TRIM and REPLACE.
CONCAT function concatenates the strings passed in it.
Syntax
CONCAT(string1,string2,....);
Example:
select CONCAT('Hello',' ','CodeQuotient!'); -- Outputs Hello CodeQuotient!

SUBSTR function extracts a string of length 'len' from the string passed starting from the startingIndex.
Syntax
SUBSTR(string1, startingIndex, no_of_characters_to_be_extracted);
Example
select SUBSTR("HELLO",1,2); -- Outputs HE

The **INSTR** function returns the position of a character or a string in a given string.
Syntax
INSTR(string_in_which_index_is_to_be_found, string_whose_index_is_to_be_found);
Example
select INSTR ("CodeQuotient","Code"); -- Outputs 1

LPAD and **RPAD** functions pad the given string upto a specific length with a given character.
Syntax:
[LPAD | RPAD](string, max_length_till_padding, string_to_be_padded_with);
Example:
select LPAD("Hello", 20, "#"); -- Outputs ###############Hello

LTRIM or **RTRIM** function trims the string input from the start or end.**TRIM** function trims the string input from both start and end.
Syntax:
[LTRIM|RTRIM|TRIM](string)
Example:
select RTRIM("Hello "); -- Outputs Hello

REPLACE function replaces characters from the input string with a given character.
Syntax
REPLACE(string, string_to_be_replaced, string_to_be_replaced_with)
Example
select REPLACE("Welcome to CodeQuotient!!","o","a"); -- Outputs Welcame ta CadeQuatient!!

Using Functions with Tables
To use functions with a table, use the table column names instead of strings. Below is an example of how to do that:
-- Table employee already created
select CONCAT(emp_fname,' ',emp_lname) from employee;
þÿTask:
Given a predefined ‘students’ table with columns **‘roll_no’**(Integer Primary Key), **‘fname’** (VARCHAR(20)), **‘lname’** (VARCHAR(20)).Perform the given tasks on the given table:
Show the name of the student as ‘fname lname’.Display the name as '**FullName**' using

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

alias.There should be a space present between fname and lname.Display the student’s first name by replacing all occurrences of the alphabet ‘a’ by the alphabet ‘i’.Display the updated name as '**UpdatedFirstName**' using aliasDisplay the First 5 characters of the student's first name.Display the output as '**Initials**' using alias
Note: Use different select statements for each tasks.

Solution:

```
/* Perform the given tasks using appropriate character functions
  Table 'students' already created */
SELECT CONCAT(fname, ' ', lname) AS FullName
FROM students;
SELECT REPLACE(fname, 'a', 'i') AS UpdatedFirstName
FROM students;
SELECT SUBSTR(fname, 1, 5) AS Initials
FROM students;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print the count of records

Write An SQL Query to fetch the count of employees working in the department 'Admin'.

Solution:

```
/*Fetch the count of employees working in the department 'Admin'.
Table 'Worker' is already created*/
SELECT COUNT(*)
FROM worker
WHERE Department = 'Admin';
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print records ordered by two fields

Write An SQL Query to print all worker details from the Worker Table by ordering FirstName in ascending order and department in descending order.

Solution:

```
/* Print all worker details from the Worker Table by ordering FirstName in ascending order and department in descending order
 Table 'Worker' is already created*/SELECT *
FROM Worker
ORDER BY FirstName ASC, Department DESC;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print records in descending order

Write an SQL Query to print all worker details from the worker table in descending order of their FirstName.

Solution:

```
/*Print all worker details from the worker table in descending order of their FirstName
 Table 'Worker' is already created */SELECT *
FROM Worker
ORDER BY FirstName DESC;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Fetch Unique value

Write An SQL Query That Fetches The Unique Values Of DEPARTMENT From Worker Table And Prints Its Length.

Solution:

```
select distinct(length(department)) from worker;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Replace a Alphabet

Write An SQL Query To Print The 'FIRSTNAME' From Worker Table After Replacing 'a' With 'l'.

Solution:

```
/* Print The FIRSTNAME From Worker Table After Replacing 'a' With 'l'.
TABLE Worker already created*/SELECT REPLACE(FIRSTNAME, 'a', 'l') AS
UpdatedFirstName
FROM Worker;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Remove spaces from the field - 1

Write An SQL Query To Print The **FirstName** From **Worker** Table After Removing White Spaces From The Left Side.
Note: Only trim the spaces from left side.

Solution:

```
/*To Print The FirstName From Worker Table After Removing White Spaces From The Left Side.
Table 'Worker' is already created */SELECT LTRIM(FIRSTNAME) AS TrimmedFirstName FROM Worker;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Remove spaces from the field - 2

Write An SQL Query to print the FirstName from Worker table after removing white spaces from the right side.

Note: Only trim the spaces from the right side.

Solution:

```
/* Print The FirstName From Worker Table After Removing White Spaces From The Right Side.
 Table 'Worker' is already Created */SELECT RTRIM(FIRSTNAME) AS TrimmedFirstName FROM Worker;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Print first 3 characters of a field

Write An SQL Query to print the first 3 characters of FirstName from Worker table.

Solution:

```
SELECT SUBSTR(FIRSTNAME, 1, 3) AS FirstThreeCharacters
FROM Worker;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Fetch First Name in Upper Case

Write An SQL Query to fetch "**FirstName**" from Worker table in Uppercase.

Solution:

```
/* Fetch "FirstName" From Worker Table In Upper Case.
 Table 'Worker' is already created */SELECT UPPER(FIRSTNAME)
FROM Worker;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Combine two fields and print

Write An SQL Query to combine the 'FirstName' And 'LastName' from Worker Table and show it as 'FullName'. The 'FirstName' and 'LastName' should be separated by a space.

Solution:

```
SELECT CONCAT(FIRSTNAME, ' ', LASTNAME) AS FullName
FROM Worker;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Cross Join

When the two tables have no relation in any way, SELECT would produce a more fundamental kind of join , which is called a cross join or Cartesian product. The Cartesian join is one of the fundamental relational operations. It is a kind of join that results in the combination of all rows from the first table with all rows in the second.
CROSS JOIN is used to get the Cartesian product of rows by matching each row of the first table with every row of the second table.
Cross join resultant table will contain multiplication number of rows in input tables. Suppose if **table1** contains **10** rows and **table2** contains **5** rows then if we apply **Cross Join** on these two tables we will get a resultant table that contains **50** rows.
SELECT [* | column_list] FROM table_name_1 CROSS JOIN table_name_2 [WHERE conditions];

Refer to the below employee & department. And display the cartesian product of the tables.
employee table

emp_id	emp_name	emp_dept
1001	Rohan	101
1002	Vishal	102

department table

emp_dept	dept_name	dept_loc
101	HR	Chandigarh
102	Accounts	Mumbai
103	Admin	Banglore

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Query:
select * from employee cross join department;
Output:

emp_id	emp_name	emp_dept	dept_id	dept_name	dept_loc
1001	Rohan	101	101	HR	Chandigarh
1001	Rohan	101	102	Accounts	Mumbai
1001	Rohan	101	103	Admin	Banglore
1002	Vishal	102	101	HR	Chandigarh
1002	Vishal	102	102	Accounts	Mumbai
1002	Vishal	102	103	Admin	Banglore

Two tables ‘students’ and ‘stud_marks’ are already created with schema given

Table ‘students’:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

[image:https://lh5.googleusercontent.com/Va9-NG3n1Z5DSd4NpJV4Fpoys41y3zL2sFGyiLY-6N6FnoNiEJr5bdvByI0YdRHJT7iTF7NINGPd4IASe02WPSqlt7aXAI-8AWaSM9-PCOP35axjVFmfwilAZ8ghzz9Uz-Td4w21]

And ‘student_marks:’
[image:https://lh5.googleusercontent.com/GIKQnQ_qq6m0z5yz9AeMuZDdQuV_PoYBYorGiRVGMYlXgILZCce8B1smG9hqh2GXrLoeeMxtGHSZkMHL-mMwDtgD4Dcxu474ByIZECeUQL2jBL6T3Dy1php_WwDclUOUXWkD0-z6]

Task: Display the cartesian product of the two tables.
Note: For every row of ‘students’ table, all the rows of the ‘stud_marks’ should be displayed.

Solution:

```
SELECT *
FROM students
CROSS JOIN stud_marks;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Inner Join

The INNER JOIN returns only the rows that match the join condition and eliminate all other rows that don't match the join condition.
The INNER JOIN clause returns rows from the first table that has the corresponding row in the second table. The same logic is applied when more than one table is joined.
Its Syntax is a follows:
SELECT [* | column_list] FROM table_name_1 INNER JOIN table_name_2 ON [table_name_1.column_name=table_name_2.column_name and ...] [WHERE conditions];

Refer to the below employee & department. And see how actually an Inner Join Works:

employee table

emp_id	emp_name	emp_dept
1001	Rohan	101
1002	Vishal	102
1003	Kapil	
1004	Anshul	104

department table

dept_id	dept_name	dept_loc
101	HR	Chandigarh
102	Accounts	Mumbai
103	Admin	Banglore
104	Engineering	Pune
105	Sales	Delhi

Query:
select * from employee inner join department on employee.emp_dept=department.dept_id;

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Output:
The INNER JOIN works as following:
The INNER JOIN clause is written after the first table is referenced with the "From" clause. The join condition is specified with ‘ON’.The INNER word is optional, you can just write JOIN.In short, Inner Join is the default keyword for Join and both can be used interchangeably.The INNER JOIN produces the records from both - the employee’s and the department's tables that match the condition which is “employee.emp_dept = department.dept_Id ". The unmatched rows will be ignored and not included in the result. Notice that the employee with ‘emp_id’ 1003 isn’t present in the output table because the ‘emp_dept’ value of the employee was empty or we can say NULL.

emp_id	emp_name	emp_dept	dept_id	dept_name	dept_loc
1001	Rohan	101	101	HR	Chandigarh
1002	Vishal	102	102	Accounts	Mumbai
1004	Anshul	104	104	Engineering	Pune

Two tables ‘students’ and ‘stud_marks’ are already created with schema given
Table ‘students’:
[image:https://lh5.googleusercontent.com/Va9-NG3n1Z5DSd4NpJV4Fpoys41y3zL2sFGyiLY-6N6FnoNiEJr5bdvByI0YdRHJT7iTF7NINGPd4IASe02WPSqlt7aXAI-8AWaSM9-PCOP35axjVFmfwilAZ8ghzz9Uz-Td4w21]

Table '**student_marks':**
[image:https://lh5.googleusercontent.com/GIKQnQ_qq6m0z5yz9AeMuZDdQuV_PoYBYorGiRVGMYlXgILZCce8B1smG9hqh2GXrLoeeMxtGHSZkMHL-mMwDtgD4Dcxu474ByIZECeUQL2jBL6T3Dy1php_WwDclUOUXWkD0-z6]

Task: Display all the data of students, using Inner Join, who have scored more than 75.

Solution:

SELECT *

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

```
FROM students
INNER JOIN stud_marks
ON students.roll_no = stud_marks.stud_roll
WHERE stud_marks.marks > 75;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Left Outer Join

By using Inner Join we can get only matching rows from multiple tables based on the conditions defined in select statements.SQL Outer Join will select matching rows from multiple tables same as Inner Join and some other rows outside of the relationship.
The SQL LEFT Outer JOIN joins two tables and fetches rows based on a condition, which is matching in both the tables and the unmatched rows from the ‘left’ will also be available from the table written before the JOIN clause.
So, in the case of LEFT JOIN or LEFT OUTER JOIN, SQL -
Takes all selected values from the left tableCombines them with the column names (specified in the condition) from the right tableRetrieve the matching rows from both the associated tables.Sets the value of every column from the right table to NULL which is unmatched with the left table.
Here is the basic syntax.
SELECT [* | column_list] FROM table_name_1 LEFT [OUTER] JOIN table_name_2 ON table_name_1.column = table_name_2.column [WHERE CONDITIONS]
A LEFT JOIN will preserve the records of the "left" table. SQL starts with the left table. For each row from the alias, an SQL scans the table if the condition is satisfied and returns the record.For unmatched rows, it returns null.

Refer to the below employee & department. And see how actually a Left Outer Join Works:

employee table

emp_id	emp_name	emp_dept
1001	Rohan	101
1002	Vishal	102
1003	Kapil	
1004	Anshul	104

department table

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

dept_id	dept_name	dept_loc
101	HR	Chandigarh
102	Accounts	Mumbai
103	Admin	Banglore
104	Engineering	Pune
105	Sales	Delhi

Query:
select * from employee left outer join department on employee.emp_dept=department.dept_id;
Output:

emp_id	emp_name	emp_dept	dept_id	dept_name	dept_loc
1001	Rohan	101	101	HR	Chandigarh
1002	Vishal	102	102	Accounts	Mumbai
1003	Kapil				
1004	Anshul	104	104	Engineering	Pune

Two tables ‘students’ and ‘stud_marks’ are already created with schema given
Table ‘students’:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

[image:https://lh5.googleusercontent.com/Va9-NG3n1Z5DSd4NpJV4Fpoys41y3zL2sFGyiLY-6N6FnoNiEJr5bdvByI0YdRHJT7iTF7NINGPd4IASe02WPSqlt7aXAI-8AWaSM9-PCOP35axjVFmfwilAZ8ghzz9Uz-Td4w21]

And 'student_marks:'
[image:https://lh5.googleusercontent.com/GIKQnQ_qq6m0z5yz9AeMuZDdQuV_PoYBYorGiRVGMYlXgILZCce8B1smG9hqh2GXrLoeeMxtGHSZkMHL-mMwDtgD4Dcxu474ByIZECeUQL2jBL6T3Dy1php_WwDclUOUXWkD0-z6]

Task: Display all the data of students and their marks irrespective of the fact that their marks are present or not in the 'stud_marks' table.

Solution:

```
SELECT *
FROM students
LEFT JOIN stud_marks
ON students.roll_no = stud_marks.stud_roll;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Natural Join

A NATURAL JOIN is very similar to an Inner JOIN, the difference is that it automatically tests for equality between the values of every column that exists in both tables.

The difference between INNER JOIN and a NATURAL JOIN:

In INNER JOIN,a join condition is specified through which the inner join uses to join the two tables. Whereas in the natural join,no join condition is written.Write the two tables' names without any condition.Then the natural join will automatically test for equality between the values for every column in both tables.In the NATURAL JOIN, all the columns from both tables with the same name will be matched against each other. For example, if there are two tables with two column names in common(the two columns exist with the same name in the two tables), then the natural join will join the two tables by comparing the values of both columns and not just from one column.The NATURAL JOIN is structured in such a way that columns with the same name of associate tables will appear once only,unlike INNER JOIN which prints the same column twice.

Natural Join: Guidelines

To implement a natural join,there are guidelines one must follow:

The associated tables have one or more pairs of identically named columns.The columns must be of the same data type.‘ON’ clause is never used in a NATURAL JOIN.

SELECT [* | column_list] FROM table_name_1 NATURAL JOIN table_name_2;

For understanding natural joins let us consider 2 tables, employee and department

Employee:

emp_id	emp_name	emp_dept
1001	Rohan	101
1002	Vishal	102
1003	Kapil	
1004	Anshul	104

Department:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

emp_dept	dept_name	dept_loc
101	HR	Chandigarh
102	Accounts	Mumbai
103	Admin	Banglore
104	Engineering	Pune
105	Sales	Delhi

Query:
select * from employee natural join department;

emp_id	emp_name	emp_dept	dept_id	dept_name	dept_loc
1001	Rohan	101	101	HR	Chandigarh
1002	Vishal	102	102	Accounts	Mumbai
1004	Anshul	104	104	Engineering	Pune

Note: If any columns aren't matched, then the Natural Join clause will return the cross join of the tables.

Two tables 'students' and 'stud_marks' are already created with schema given

students
roll_no INTEGER PRIMARY KEY,
marks INTEGER
stud_marks
roll_no INTEGER PRIMARY KEY,
name VARCHAR(50)

Task: Display all the data of students and their marks using natural join where marks >= 45.

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Solution:

```
/*
create table stud_marks(
  roll_no INTEGER PRIMARY KEY,
    marks INTEGER
);
create table students(
  roll_no INTEGER PRIMARY KEY,
    name VARCHAR(50)
);
Tables are already created
*/SELECT *
FROM students
NATURAL JOIN stud_marks
WHERE marks >= 45;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Nested Queries 1

We have learned about DBMS joins and how to use them. Sometimes when we have a lot of data available in the database. By using joins we can select the data that we actually want, but it can reduce the performance of the query processor as it will first join the tables and then select the data based on the where clause. To overcome that problem SQL has something called 'Subqueries'. Let's learn about subqueries in detail!
A subquery, also known as a nested query or subselect, is a SELECT query embedded within the WHERE or HAVING clause of another SQL query. The data returned by the subquery is used by the outer statement in the same way a literal value would be used.
A correct subquery follows the given rules:
A subquery must always appear within parentheses.A subquery must return only one column. This means you cannot use SELECT * in a subquery unless the table that is being referred to has only one column. A subquery may return multiple columns if the purpose is row comparison.Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.BETWEEN operator cannot be used with a subquery; however, BETWEEN can be used within the subquery.Grouping clauses like GROUP BY cannot be used in a subquery. The same is the case with the ORDER BY clause. Subqueries are most frequently used with the SELECT statement, however, you can use them within an INSERT, UPDATE, or DELETE statement as well, or inside another subquery.
SELECT [* | column_list] from table_name where CONDITION [IN | NOT IN]
(expression);
Let's consider the following tables:
Employee:

emp_id	emp_name	emp_dept
1001	Rohan	101
1002	Vishal	102
1003	Kapil	
1004	Anshul	104

Department:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

dept_id	dept_name	dept_loc
101	HR	Chandigarh
102	Accounts	Mumbai
103	Admin	Banglore
104	Engineering	Pune
105	Sales	Delhi

To display the employee details who work in the '**Accounts**' department we will write the following query:
select * from employee where emp_dept = (select dept_id from department where dept_name='Accounts');
Output:

1002	Vishal	102

Three tables '**students**', '**class**' and '**teacher**' are already created with schema given

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

[image:https://lh3.googleusercontent.com/KH_99-pZpezDQPgkIoWBhN53KpjALGjHcjzrtwaKRiVbqfpMSD7SyKfjQfaN2EWIj66yOPRKD2X0qWpGkfS2lRsPNAIFsqh-B3iN4hyWqO1ghXXES29L6Hrch3aUWN26Xf1ML_79]

[image:https://lh4.googleusercontent.com/15b2H07gUnC2yNbea2iRacDXBqVhx3jRAOfFfovRjQX5zYyA1L9_8Ga1Tt3oibdSGDvjiPQQSAmSvmSM0FPUcI6NiU146wjVPPc_srcUoKgpGMKhXxzwWAT4vctGJq3JIhkRuXqT]

[image:https://lh6.googleusercontent.com/nXMnjCAb4c8JsNrxJisS1FX3589NYKC0F0bAeDb_zQYGcSG8LH5giB9_O6iheMN-5j-8E7aBSzaW09tzjZG6l0zbAsycIHw2Jyly0r8vQmSuhk_ztNYGvj4gBhof3irY-rQwW-B2]

Task 1: Find the names of teachers who teach classes that have more than 35 students.

Solution:

```
SELECT teacher.teacher_name
FROM teacher
WHERE teacher.teacher_id IN (
   SELECT class.teacher_id
   FROM class
   WHERE class.no_of_students > 35
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Nested Queries 2

Three tables '**students**', '**class**' and '**teacher**' are already created with schema given

[image:https://lh3.googleusercontent.com/KH_99-pZpezDQPgkIoWBhN53KpjALGjHcjzrtwaKRiVbqfpMSD7SyKfjQfaN2EWIj66yOPRKD2X0qWpGkfS2lRsPNAIFsqh-B3iN4hyWqO1ghXXES29L6Hrch3aUWN26Xf1ML_79]

[image:https://lh4.googleusercontent.com/15b2H07gUnC2yNbea2iRacDXBqVhx3jRAOfFfovRjQX5zYyA1L9_8Ga1Tt3oibdSGDvjiPQQSAmSvmSM0FPUcI6NiU146wjVPPc_srcUoKgpGMKhXxzwWAT4vctGJq3JIhkRuXqT]

[image:https://lh6.googleusercontent.com/nXMnjCAb4c8JsNrxJisS1FX3589NYKC0F0bAeDb_zQYGcSG8LH5giB9_O6iheMN-5j-8E7aBSzaW09tzjZG6l0zbAsycIHw2Jyly0r8vQmSuhk_ztNYGvj4gBhof3irY-rQwW-B2]

Task: Find the names of students who have more than 30 classmates.

Solution:

```
/*
create table students(
  roll_no INTEGER PRIMARY KEY,
    stud_name VARCHAR(50),
    marks INT,
    class_id INTEGER
 );
create table class(
  class_id INTEGER PRIMARY KEY,
    teacher_id INTEGER,
    no_of_students INTEGER
);
create table teacher(
    teacher_id INTEGER PRIMARY KEY,
    teacher_name varchar(50),
    class_id INTEGER
);
Tables already created
*/SELECT students.stud_name
FROM students
WHERE students.class_id IN (
    SELECT class.class_id
    FROM class
    WHERE class.no_of_students > 30
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Nested Queries 3

Three tables ‘**students**’, '**class'** and ‘**teacher**’ are already created with schema given

[image:https://lh3.googleusercontent.com/KH_99-pZpezDQPgkIoWBhN53KpjALGjHcjzrtwaKRiVbqfpMSD7SyKfjQfaN2EWIj66yOPRKD2X0qWpGkfS2lRsPNAIFsqh-B3iN4hyWqO1ghXXES29L6Hrch3aUWN26Xf1ML_79]

[image:https://lh4.googleusercontent.com/15b2H07gUnC2yNbea2iRacDXBqVhx3jRAOfFfovRjQX5zYyA1L9_8Ga1Tt3oibdSGDvjiPQQSAmSvmSM0FPUcI6NiU146wjVPPc_srcUoKgpGMKhXxzwWAT4vctGJq3JIhkRuXqT]

[image:https://lh6.googleusercontent.com/nXMnjCAb4c8JsNrxJisS1FX3589NYKC0F0bAeDb_zQYGcSG8LH5giB9_O6iheMN-5j-8E7aBSzaW09tzjZG6l0zbAsycIHw2Jyly0r8vQmSuhk_ztNYGvj4gBhof3irY-rQwW-B2]

þÿTask: Find the name of the teacher who teaches the student's name 'Rohit'.

Solution:

```
SELECT teacher.teacher_name
FROM students
JOIN class ON students.class_id = class.class_id
JOIN teacher ON class.teacher_id = teacher.teacher_id
WHERE students.stud_name = 'Rohit';
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Correlated Queries

All the queries we have learned so far can be executed as independent queries. In other words, they don't depend on the outer query for execution.
The correlated subquery is a subquery that uses the values from the outer query. Unlike an ordinal subquery, a correlated subquery cannot be executed independently.

The correlated subquery may or may not be efficient depending on the number of records retrieved by the outer query. But in most cases correlated are inefficient because of the fact that it evaluates the inner query for each row processed by the outer query.
SELECT [* | column_list] FROM table1 as outer WHERE CONDITIONS operator (SELECT column_list FROM table2 WHERE expr1 = outer.expr2);
Note that we have used 'As' as an alias so that the outer table can be referenced from the inner query.
Refer the below **employee** table & display the details of employee who have more salary than the average salary in their departments

emp_id	emp_name	salary	emp_dept
1001	Rohan	30000	101
1002	Vishal	10000	102
1003	Kapil	20000	103
1004	Anshul	15000	104
1005	Ravi	30000	102
1006	Rohit	40000	101
1007	Ajay	20000	105

Query:
select * from employee as emp where salary >= (select AVG(salary) from employee where emp_dept=emp.emp_dept);
Output:

1003	Kapil	20000	103
1004	Anshul	15000	104
1005	Ravi	30000	102
1006	Rohit	40000	101
1007	Ajay	20000	105

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

In the above example we have given the outer table an alias of ‘emp’ so to use the reference of the outer table in the inner query.
How the query works.
For each row processed in the outer query, the correlated subquery calculates the average of the salary.The condition in the WHERE clause calculates the average salary of only those whose department is the same as the department of the row for which the inner query is running.
A table students with the following schema is already created
[image:https://lh4.googleusercontent.com/3XgkyWyIudmgtiwlX9KSq13lyDSK9wohDhAwOQcHTRkkoksX_E9aIgo2-Uu73q5lW41nZb42tfRSaYORfkoNVaAhfJfuc4-HmbT8xvG8UZFds5_6EE0wTw1fupcRn_WA6oi1jjau]

Task: Display the details of students who scored the maximum number of marks from each department.

Solution:

```
/*
create table students(
   roll_no INTEGER PRIMARY KEY,
   stud_name VARCHAR(50),
   stud_age integer,
   stud_dept integer,
   stud_marks INTEGER);
Table already created
*/
SELECT *
FROM students AS s1
WHERE stud_marks = (
   SELECT MAX(stud_marks)
   FROM students AS s2
   WHERE s1.stud_dept = s2.stud_dept
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Correlated Queries 2

A table students with the following schema is already created
[image:https://lh4.googleusercontent.com/3XgkyWyIudmgtiwlX9KSq13lyDSK9wohDhAwOQcHTRkkoksX_E9aIgo2-Uu73q5lW41nZb42tfRSaYORfkoNVaAhfJfuc4-HmbT8xvG8UZFds5_6EE0wTw1fupcRn_WA6oi1jjau]

Task: Display the details of the eldest student of each department.

Solution:

```
/*
create table students(
   roll_no INTEGER PRIMARY KEY,
   stud_name VARCHAR(50),
   stud_age integer,
   stud_dept integer,
   stud_marks INTEGER);
Table already created
*/
SELECT *
FROM students AS s1
WHERE stud_age = (
   SELECT MAX(stud_age)
   FROM students AS s2
   WHERE s1.stud_dept = s2.stud_dept
);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Trigger

Trigger is a named database object that is executed automatically when an *INSERT*, *UPDATE* or *DELETE* statement is issued against the associated table.
Triggers are database callback functions, which are automatically performed/invoked when a specified database event occurs.
Create Trigger
Below given is the basic syntax for creating a trigger:

```
CREATE TRIGGER [IF NOT EXISTS] trigger_name
  [BEFORE|AFTER|INSTEAD OF] [INSERT|UPDATE|DELETE]
  ON table_name
  [WHEN condition]
BEGIN
statements;
END;
```

In this syntax, it can be noticed that:
A trigger can be created with CREATE TRIGGER statementAfter that determine when the trigger will be executed, BEFORE, AFTER, or INSTEAD OF. BEFORE will execute the trigger before the statement next specified will execute, AFTER will execute after the statement will be executed, and INSTEAD OF will execute the trigger instead of the statement.After determining the time of execution of the trigger, determine the statement on which the trigger will execute, INSERT, UPDATE or DELETE.After that, indicate the table to which the trigger belongs.Finally, place the trigger logic in the BEGIN END block, which can be any valid SQL statements.

If we want to create a trigger on an **employee** table that runs after every insert statement, we would write a query as follows:

```
CREATE TRIGGER emp_trigger
  AFTER INSERT
  ON employee

BEGIN
  select null; /*Just for creating a empty valid trigger*/
END;
```

The above example creates a trigger and whenever data is inserted into the table that trigger will execute.

Task: A table **students** is already created. Create an empty trigger named '**stud_trigger**' on the table students.

Solution:

```
CREATE TRIGGER stud_trigger
  AFTER INSERT
  ON students
BEGIN
  SELECT NULL; /*Empty trigger logic*/
END;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Trigger on Insert Statement

To maintain records of changes occurring in the database, we use triggers. Triggers also can be used for validating input such that if there is a valid input it will insert at that time only.For this we use the WHEN clause in the trigger. Sometimes after inserting the data we have to calculate. Also for that, we can use triggers.

General Syntax for using triggers with insert statement is :

```
CREATE TRIGGER [IF NOT EXISTS] trigger_name
  [BEFORE|AFTER] INSERT
  ON table_name
  [WHEN condition]
BEGIN
statements;
END;
```

Let's look at how to use a trigger on table **employee**

Table : employee
emp_id INTEGER PRIMARY KEY
emp_name VARCHAR(50)
emp_dept INTEGER
salary INT
increment INT
total_sal INT

```
CREATE TRIGGER emp_trigger
 AFTER INSERT
 ON employee
BEGIN
  update employee set total_sal = NEW.salary + (NEW.salary*NEW.increment)/100 where
emp_id=NEW.emp_id;
END;
```

In the above example, we can see that we created a trigger named 'emp_trigger' that works after the data is inserted into the table and increments the salary given to the employee by the percent given in the data.

We can see something called 'New' in the trigger body.

Well, there are two extensions to triggers 'OLD' and 'NEW'. OLD and NEW are not case sensitive.

Within the trigger body, the OLD and NEW keywords enable you to access columns in the rows affected by a triggerOLD is used to get the old value of the column before it gets updated or deleted. Therefore OLD is used only with delete or update statements triggers NEW is used to get the value that is updated or just inserted in the database. Therefore NEW

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

is used with the only update and insert statements triggers

Insert some rows in table employee.

```
INSERT INTO employee(emp_id,emp_name,emp_dept,salary,increment)
VALUES(1001,"Rohan",101,30000,5);
INSERT INTO employee(emp_id,emp_name,emp_dept,salary,increment)
VALUES(1002,"Vishal",102,10000,2);
INSERT INTO employee(emp_id,emp_name,emp_dept,salary,increment)
VALUES(1003,"Kapil",103,20000,8);
```

Query:

```
select * from employee;
```

Output:

emp_id	emp_name	emp_dept	salary	increment	total_sal
1001	Rohan	101	30000	5	31500
1002	Vishal	102	10000	2	10200
1003	Kapil	103	20000	8	21600

A table named '**Students**' is already created.

[image:https://lh6.googleusercontent.com/K4kTingj9pfFrEL27UBvpNNUdrs2v7t9yc2Yy8rmo-qYphWmBpZEar8m8fheVn3RK1ZP2USD5pOrXUlVM91v6Oehx_FhNeJxeEW8Jhb69BL3NA8MErCQju2k4Kw_OeS90-HtgPmk]

Task: Create a trigger named '**calc_marks**' which calculates the total marks of the student and updates the column '**total_marks**'.

Solution:

```
/*
create table students(
  roll_no INTEGER PRIMARY KEY,
  stud_name VARCHAR(50),
  sub1_marks INT,
  sub2_marks INT,
  sub3_marks INT,
  total_marks INT
 );
```

Table already created

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

```
*/CREATE TRIGGER calc_marks
  AFTER INSERT
  ON students
BEGIN
  UPDATE students
  SET total_marks = NEW.sub1_marks + NEW.sub2_marks + NEW.sub3_marks
  WHERE roll_no = NEW.roll_no;
END;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Trigger on Update and Delete Statement

Just like triggers on insert statements, triggers also work on update and delete statements. Mainly triggers on update and delete statements are used for logging into tables at what time the database was updated.

```
CREATE TRIGGER [IF NOT EXISTS] trigger_name
  [BEFORE|AFTER] [INSERT|DELETE]
  ON table_name
  [WHEN condition]
BEGIN
statements;
END;
```

Let's look at how to use a trigger on table **employee**

emp_id	emp_name	emp_dept	salary
1001	Rohan	101	30000
1002	Vishal	102	10000
1003	Kapil	103	20000

```
CREATE TRIGGER emp_trigger
 AFTER UPDATE
 ON employee
BEGIN
   insert into emp_log values(OLD.emp_id,datetime('now'));
END;
```

In the above example, whenever an update statement is executed the trigger inserts a row into the emp_log table telling which emp_id was updated and at what time it was updated.

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Table : emp_log
emp_id INT
time DATETIME

Update some rows in table employee.
update employee set salary=35000 where emp_id=1002;
update employee set emp_dept=105 where emp_id=1001;
Query:
select * from emp_log;
Output:

emp_id	time
1001	17-02-2021 13:12
1002	17-02-2021 13:12

Two tables ‘students’ and ‘stud_logs’ are already created. Below are the schemas of the two tables:

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

[image:https://lh3.googleusercontent.com/Z8Q3ult_4B8mmNhXUxJDoJ0cJj2bc9YnTePW3a5wxT9lhPEMOZM18kkkwfctu0JsWmqDIlidO-v5g41JgXXpBt4JlQeox-NvBeU5V5xYQhwQ5Vb95-wGfftIRP8mAA_kBROKJwhL]

[image:https://lh4.googleusercontent.com/gr5lQKeFBBYEFDaAK7x0wvleUGYpOURCYRSsC2QT_gJeCMCIF0_0xWWlSS75h5ZDJHX3ZkbPJg0zMpT2VYFypWJpZ8GtPl8JpzGWRw64LOYPnA3xnFoMhX7QBmv1dtccJPK6u2Qq]

Task: Create 2 triggers named ‘stud_update’ and ‘stud_delete’. Whenever a student is updated or deleted the respective triggers add the student’s roll number and adds the status ‘updated’ or ‘deleted’ based on the operation performed.

Solution:

```
CREATE TRIGGER stud_update
  AFTER UPDATE
  ON students
BEGIN
  INSERT INTO stud_logs (roll_no, status)
  VALUES (OLD.roll_no, 'updated');
END;
CREATE TRIGGER stud_delete
  AFTER DELETE
  ON students
BEGIN
  INSERT INTO stud_logs (roll_no, status)
  VALUES (OLD.roll_no, 'deleted');
END;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Drop Trigger

To destroy or delete a trigger use the 'DROP TRIGGER' command.
Following is the syntax to delete or drop or remove triggers in SQL.

```
DROP TRIGGER [IF EXISTS] trigger_name.
```

In the above syntax IF EXISTS will help us to prevent throwing errors in the case when we try to delete triggers that do not exist in the database.

Task: Given a table students and a trigger named 'stud_trigger' that we have already created. Your task is to delete the trigger on the students table.

Solution:

```
/*
Drop trigger stud_trigger
*/DROP TRIGGER IF EXISTS stud_trigger;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Introduction to Indexes

In relational databases, a table is a list of rows. At the same time, each row has the same column structure that consists of cells. Each row also has a consecutive **rowid** sequence number used to identify the row. Therefore, you can consider a table as a list of pairs: (rowid, row).
Unlike a table, an index has an opposite relationship: (row, rowid). An index is an additional data structure that helps improve the performance of a query.
Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.
An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.
SQL uses B-tree for organizing indexes. Where B stands for balanced, B-tree is a balanced tree, not a binary tree.
The B-tree keeps the amount of data at both sides of the tree balanced so that the number of levels that must be traversed to locate a row is always in the same approximate number. In addition, querying using equality (=) and ranges (>, >=, <,<=) on the B-tree indexes are very efficient.
So how does an index work?
Each index must be associated with a specific table. An index consists of one or more columns, but all columns of an index must be in the same table. A table may have multiple indexes.
Whenever an index is created, SQL creates a B-tree structure to hold the index data.
The index contains data from the columns that you specify in the index and the corresponding rowid value. This helps SQL quickly locate the row based on the values of the indexed columns.
Imagine an index in the database like an index of a book. By looking at the index, one can quickly identify page numbers based on the keywords.
Creating an index
To create an index, use the CREATE INDEX statement with the following syntax:
CREATE [UNIQUE] INDEX index_name
ON table_name(column_list);
While creating an index,information specified to the command is :
The name of the index after the CREATE INDEX keywords.The name of the table to the index belongs.A list of columns of the index.It can be a single column index or a composite column index(when multiple columns are supplied in the CREATE INDEX command).
To make sure that values in one or more columns are unique like email and phone, use the UNIQUE option in the CREATE INDEX statement. The CREATE UNIQUE INDEX creates a new unique index.
Below given is an example to create an index on **emp_id** column of the **employee** table
create index emp_index on employee(emp_id);
Task:
A table student is already created with columns '**roll_no**' (Integer Primary Key), '**stud_name** '(Varchar(50)) and '**marks**'(Integer).
Your task is to make data retrieval from the database faster using indexes.Create an index named 'stud_index' on the table students on the 'roll_no' column.

Solution:

```
/* Create an index 'stud_index' on table 'students'
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

```
 Table 'students' already created */
CREATE INDEX stud_index ON students(roll_no);
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Dropping an Index

To remove an index from a database, use the DROP INDEX statement as follows:
DROP INDEX [IF EXISTS] index_name;
In this syntax,specify the name of the index that is to drop after the DROP INDEX keywords.
The IF EXISTS option removes an index only if it exists.
For example, the following statement is used to remove the ‘emp_index’ index:
DROP INDEX emp_index;
The ‘emp_index’ index is removed completely from the database.
Task:
A table student is already created with columns ‘**roll_no**’ (Integer Primary Key), ‘**stud_name** ’(Varchar(50)) and ‘**marks**’(Integer).
Your task is to completely remove the index named ‘stud_index’ from the database.

Solution:

/* Delete the index 'stud_index' from the database */DROP INDEX IF EXISTS stud_index;

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Views-Introduction

In SQL view is nothing but a virtual table which is created by SQL statements.
Generally, SQL views will return rows and columns same as a normal table but those are the result set of joining multiple SQL statements.By default, **Views are read-only** tables that means it doesn't allow to perform any operations like INSERT, UPDATE, and DELETE.
Views which are kind of virtual tables, allow the users to "
Structure data in a way that users or classes of users find natural or intuitive.Restrict access to the data such that a user can only see limited data instead of a complete table.Summarize data from various tables, which can be used to generate reports.
The data of the underlying tables can be accessed through a view. The tables that the query in the view definition refers to are called base tables.
A view is useful in some cases:
Views provide an abstraction layer over tables.Views can be used to encapsulate complex queries with joins to simplify the data access.
Given below is the syntax of creating a view:
CREATE [OR REPLACE] VIEW view_name AS SELECT [* | column_list] FROM table_name WHERE [condition];
First, specify a name for the view. The OR REPLACE option creates a new view if it doesn't exist. If the view already exists, it replaces the previous view definition with the new one.
Second, specify a SELECT statement for the view. By default, the columns of the view derive from the result set of the SELECT statement.Aliases can also be used in the select statement if one doesn't want to use the column names provided in the table.
In the given below example,a view named 'emp_view' is created on the table employee where records having salary > 20000 are shown.Also emp_id is named as id and emp_name is named as name.
CREATE OR REPLACE VIEW emp_view AS SELECT emp_id as id,emp_name as name, salary FROM employee WHERE salary > 20000;
Task:
A table 'students' is already created with columns '**roll_no**' (Integer Primary Key), '**stud_name**'(Varchar(50)) and '**marks**'(Integer).
Your task is to create a view named 'student_view' in which show the roll number of students as **'id'** and **'marks'** of students who have scored more than 75.

Solution:

```
/* Create a view 'student_view' according to the given task
 Table students already created*/CREATE OR REPLACE VIEW student_view AS
SELECT roll_no AS id, marks
FROM students
WHERE marks > 75;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Altering Views

Just like a table is altered , Views can also be altered.ALTER VIEW statement changes the definition of an existing view.Syntax of altering a view is as follows:
ALTER
VIEW view_name
AS select_statement;
To alter the 'emp_view' by changing the records who are having salary > 15000,the following query will be written:
ALTER VIEW emp_view AS SELECT emp_id,emp_name ,salary from employee where salary > 15000;
Task:
A table student is already created with columns '**roll_no**' (Integer Primary Key), '**stud_name** '(Varchar(50)) and '**marks**'(Integer).
A view named 'student_view' is already created which shows the roll number of students as **'id'** and '**marks**' of students who have scored more than 75.
Your task is to alter the view and select roll_no as id,stud_name as name and marks of students who scored more than 50.

Solution:

```
/* Alter the view as specified */ALTER VIEW student_view AS
SELECT roll_no AS id, stud_name AS name, marks
FROM students
WHERE marks > 50;
```


SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Renaming Views

To rename a view, RENAME TABLE statement can be used to change the name of the view. Because views and tables share the same namespace, the RENAME TABLE statement is used to change the name of a view.
Here is the basic syntax of the RENAME TABLE for renaming a view:
RENAME TABLE original_view_name TO new_view_name;
In the example given below emp_view is renamed using RENAME TABLE statement:
RENAME TABLE emp_view TO employee_view;
Task:
A table '**students'** is already created with columns '**roll_no**' (Integer Primary Key), ' **stud_name**'(Varchar(50)) and '**marks**'(Integer).
A view named '**stud_view**' is already created.
Your task is to rename the view to '**student_view**'.

Solution:

/* Rename the already created 'stud_view' to 'student_view' */RENAME TABLE stud_view TO student_view;

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in

Aim: Deleting a View

The **DROP VIEW** statement deletes a view from the database schema. Here is the basic syntax of the DROP VIEW statement:
DROP VIEW [IF EXISTS] view_name;
In this syntax:
First, specify the name of the view that is to be removed after the DROP VIEW keywords. Second, specify the schema of the view that you want to delete.Third, use the IF EXISTS option to remove a view only if it exists. If the view does not exist, the DROP VIEW IF EXISTS statement does nothing. However, trying to drop a non-existing view without the IF EXISTS option will result in an error.
Given below is an example to drop a view from the database.
DROP VIEW emp_view;
Note: DROP VIEW statement only removes the view object from the database schema. It does not remove the data of the base tables.
Task:
A table student is already created with columns '**roll_no**' (Integer Primary Key), '**stud_name** '(Varchar(50)) and '**marks**'(Integer).
Your task is to delete an already defined view named 'student_view' entirely from the database.

Solution:

/* Delete the view named 'student_view' */DROP VIEW IF EXISTS student_view;

SHIVDEV JOSHI
shivdev2448.be23@chitkara.edu.in