

Title: Create a New Database and Perform Following operations on that Database.

- a) Create table
- b) Alter the table
- c) Rename Table
- d) Drop the table. (Assume your own data).

Theory:

1. CREATE DATABASE query syntax and example.

It is a DDL command used to create new database. Later we have to 'USE' the database

- **Syntax:** CREATE DATABASE database_name;
USE DATABASE database_name;
- **Example:** CREATE DATABASE students;
USE students;

2. CREATE TABLE query syntax and example.

CREATE TABLE is a DDL command used to create a new table.

- **Syntax:** CREATE TABLE table_name (
column1 datatype,
column2 datatype,
column3 datatype,
....
);
- **Example:** CREATE TABLE Persons (
rollNo INT,
lastName VARCHAR(25),
firstName VARCHAR (25),
address VARCHAR (255),
birthDate DATE
);

3. ALTER TABLE query syntax and example.

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

- **Syntax:** ALTER TABLE *table_name*
ADD *column_name datatype*;
- **Example:** ALTER TABLE Persons
ADD email VARCHAR(255);

4. RENAME TABLE query syntax and example.

It is a DDL Command use to rename the table.

- **Syntax:** RENAME TABLE *table_name* TO *new_name*;
- **Example:** RENAME TABLE Persons TO firstYearStudents;

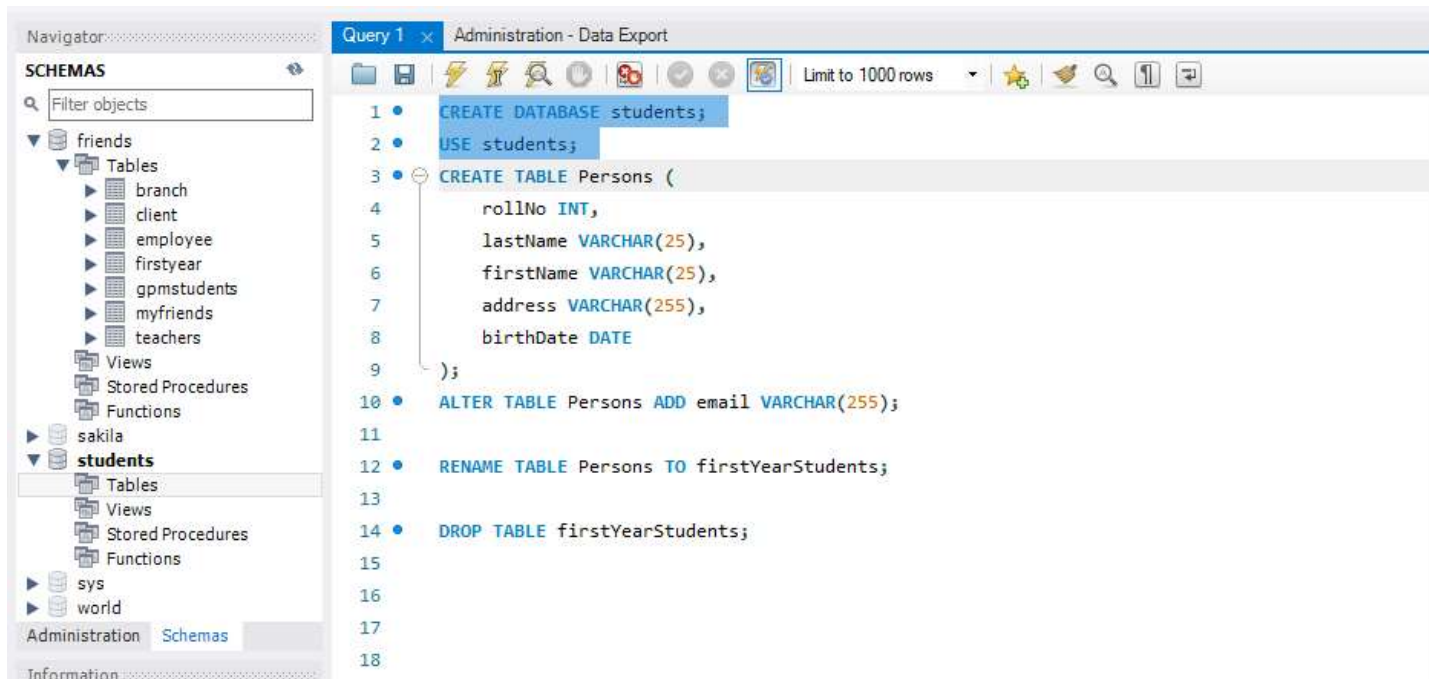
5. DROP TABLE query syntax and example.

The DROP TABLE query is used to drop an existing table in a database along with it's structure.

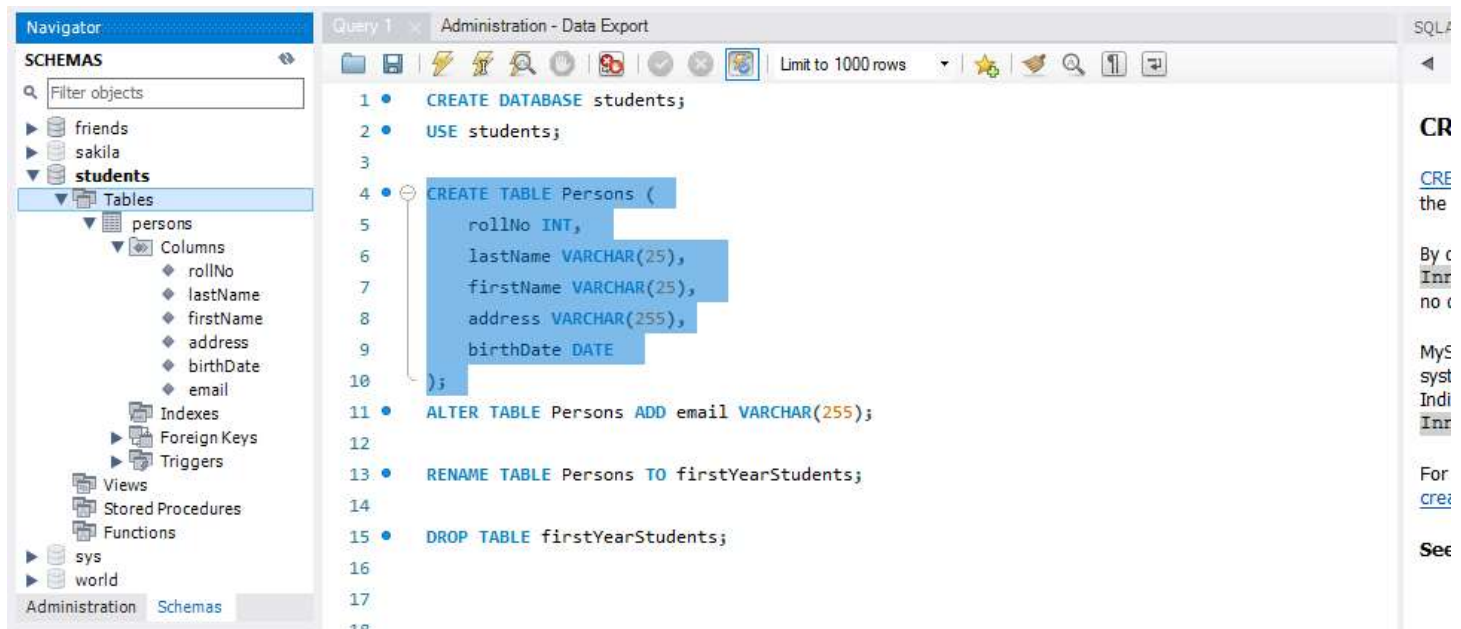
- **Syntax:** DROP TABLE *table_name*;
- **Example:** DROP TABLE firstYearStudents;

Queries and output:

1. CREATE DATABASE query syntax and example.



2. CREATE TABLE query syntax and example.



Navigator: SCHEMAS

Filter objects

- friends
- sakila
- students
 - Tables
 - persons
 - Columns
 - rollNo
 - lastName
 - firstName
 - address
 - birthDate
 - Indexes
 - Foreign Keys
 - Triggers
 - Views
 - Stored Procedures
 - Functions
 - sys
 - world

Administration Schemas

Information

Connection Details

- Local instance
- MySQL80
- localhost
- 3306
- root

Query 1

Limit to 1000 rows

```

6      lastName VARCHAR(25),
7      firstName VARCHAR(25),
8      address VARCHAR(255),
9      birthDate DATE
10     );
11     DESC Persons;
12
13     ALTER TABLE Persons ADD email VARCHAR(255);
14
15     RENAME TABLE Persons TO firstYearStudents;
16
17     DROP TABLE firstYearStudents;
18

```

Result Grid

Field	Type	Null	Key	Default	Extra
rollNo	int	YES		NULL	
lastName	varchar(25)	YES		NULL	
firstName	varchar(25)	YES		NULL	
address	varchar(255)	YES		NULL	
birthDate	date	YES		NULL	

Result Grid

Form Editor

Field

3. ALTER TABLE query syntax and example.

Navigator: SCHEMAS

Filter objects

- friends
- sakila
- students
 - Tables
 - persons
 - Columns
 - rollNo
 - lastName
 - firstName
 - address
 - birthDate
 - email
 - Indexes
 - Foreign Keys
 - Triggers
 - Views
 - Stored Procedures
 - Functions
 - sys
 - world

Administration Schemas

Information

Connection Details

- Local instance
- MySQL80
- localhost
- 3306
- root
- Use root@localhost
- Use root@localhost

Query 1

Limit to 1000 rows

```

1     CREATE DATABASE students;
2     USE students;
3
4     CREATE TABLE Persons (
5         rollNo INT,
6         lastName VARCHAR(25),
7         firstName VARCHAR(25),
8         address VARCHAR(255),
9         birthDate DATE
10    );
11    DESC Persons;
12
13    ALTER TABLE Persons ADD email VARCHAR(255);
14
15    RENAME TABLE Persons TO firstYearStudents;
16
17    DROP TABLE firstYearStudents;
18

```

Output

#	Time	Action	Message
27	11:24:44	CREATE DATABASE students	Error Code: 1007. Can't create database 'students'; database already exists
28	11:24:58	DESC Persons	Error Code: 1146. Table 'students.persons' doesn't exist
29	11:25:08	CREATE TABLE Persons (rollNo INT, lastName VARCHAR(25), firstName VARCHAR(25), address VARCHAR(255), birthDate DATE)	0 row(s) affected
30	11:25:18	DESC Persons	5 row(s) returned
31	11:26:35	ALTER TABLE Persons ADD email VARCHAR(255)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

SQLAdditions

CREATE TABLE

CREATE TABLE creates a new table in the database. The CREATE privilege is required to create a table.

By default, tables are created in the InnoDB storage engine. If no default database is specified, the table is created in the current database.

MySQL has no limit on the number of tables in a database. The system may have a limit on the number of tables in a single database. Individual storage engines may have their own limits. InnoDB permits unlimited tables.

For information about creating tables, see [create-table-files](#).

See also: [Online](#)

Context Help Snippets

4. RENAME TABLE query syntax and example.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'students' expanded. The main query editor contains the following SQL code:

```
6      lastName VARCHAR(25),
7      firstName VARCHAR(25),
8      address VARCHAR(255),
9      birthDate DATE
10   );
11   DESC Persons;
12
13   ALTER TABLE Persons ADD email VARCHAR(255);
14
15   RENAME TABLE Persons TO firstYearStudents;
16
17   DROP TABLE firstYearStudents;
```

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message
28	11:24:58	DESC Persons	Error Code: 1146. Table 'students.persons' doesn't exist
29	11:25:08	CREATE TABLE Persons (rollNo INT, lastName VARCHAR(25), firstName VARCHAR(25...	0 row(s) affected
30	11:25:18	DESC Persons	5 row(s) returned
31	11:26:35	ALTER TABLE Persons ADD email VARCHAR(255)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
32	11:29:56	RENAME TABLE Persons TO firstYearStudents	0 row(s) affected

The right sidebar contains a 'CREATE TABLE' snippet with additional information and a link to 'Online'.

5. DROP TABLE query syntax and example.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'students' expanded. The main query editor contains the following SQL code:

```
6      lastName VARCHAR(25),
7      firstName VARCHAR(25),
8      address VARCHAR(255),
9      birthDate DATE
10   );
11   DESC Persons;
12
13   ALTER TABLE Persons ADD email VARCHAR(255);
14
15   RENAME TABLE Persons TO firstYearStudents;
16
17   DROP TABLE firstYearStudents;
```

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message
29	11:25:08	CREATE TABLE Persons (rollNo INT, lastName VARCHAR(25), firstName VARCHAR(25...	0 row(s) affected
30	11:25:18	DESC Persons	5 row(s) returned
31	11:26:35	ALTER TABLE Persons ADD email VARCHAR(255)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
32	11:29:56	RENAME TABLE Persons TO firstYearStudents	0 row(s) affected
33	11:30:26	DROP TABLE firstYearStudents	0 row(s) affected

The right sidebar contains a 'CREATE TABLE' snippet with additional information and a link to 'Online'.

DBMS Practical no. 2

Title: Create a New Database and Perform Following operations on that Database.

- a) Create a table
- b) Insert values in that table
- c) Update the table
- d) Delete the contents of the table

Theory:

1. INSERT query syntax and example.

The INSERT INTO query is a DML command used to insert new records in a table.

Syntax:

It is possible to write the INSERT INTO statement in two ways.

- a. The first way specifies both the column names and the values to be inserted:

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

- b. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. The INSERT INTO syntax would be as follows:

```
INSERT INTO table_name VALUES (value1, value2, value3, ...);
```

Example:

```
Insert into employee values (1, "abc", "cba");
```

```
Insert into employee (employee_id, f_name) values (1, "xyz");
```

2. UPDATE query syntax and example.

The UPDATE query is a DML command used to modify the existing records in a table.

Syntax: UPDATE *table_name*
SET *column1* = *value1*, *column2* = *value2*, ...
WHERE *condition*;

Example:

Update employee set joining_date = 01-01-2000;

update employee set l_name = "zyx" where employee_id=2;

3. DELETE query syntax and example.

The DELETE query is a DML command used to delete existing records in a table.

Syntax: DELETE FROM *table_name* WHERE *condition*;

Example: DELETE FROM employee where f_name = "abc";

4. Delete all contents of table

The TRUNCATE is a DDL command used to delete all rows of table, while keeping structure intact.

Syntax: TRUNCATE TABLE *database_name*;

Example: TRUNCATE TABLE employee;

OUTPUT:

1. Create a Table

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view with 'company' expanded, containing 'Tables', 'Columns', 'Indexes', 'Foreign Keys', 'Triggers', 'Views', 'Stored Procedures', and 'Functions'. The 'employee' table is listed under 'Tables'. The 'Columns' pane shows 'employeeID' (int), 'firstName' (varchar(20)), and 'lastName' (varchar(20)). The 'Indexes' pane is empty. The 'Foreign Keys' pane is empty. The 'Triggers' pane is empty. The 'Views' pane is empty. The 'Stored Procedures' pane is empty. The 'Functions' pane is empty. The 'Information' pane shows 'Connection Details' for 'MySQL80' on 'localhost' with port '3306', login 'root', and user 'root@localhost'. The 'Object Info' pane shows 'Session' details. The main query editor shows the following SQL code:

```
1 • CREATE DATABASE company;
2 • USE company;
3
4 • CREATE TABLE employee (
5     employeeID INT,
6     firstName VARCHAR(20),
7     lastName VARCHAR(20)
8 );
9 • DESC employee;
10
```

The 'Result Grid' shows the output of the 'DESC employee;' query:

Field	Type	Null	Key	Default	Extra
employeeID	int	YES		NULL	
firstName	varchar(20)	YES		NULL	
lastName	varchar(20)	YES		NULL	

The 'Output' pane shows the 'Action Output' for the query:

#	Time	Action	Message
33	11:30:26	DROP TABLE firstYearStudents	0 row(s) affected
34	13:12:12	CREATE DATABASE company	1 row(s) affected
35	13:12:13	USE company	0 row(s) affected
36	13:16:21	CREATE TABLE employee (employeeID INT, firstName VARCHAR(20), lastName VARC...	0 row(s) affected
37	13:16:24	DESC employee	3 row(s) returned

2. Insert Values in the table

The screenshot shows the MySQL Workbench interface. The main query editor shows the following SQL code:

```
5     employeeID INT,
6     firstName VARCHAR(20),
7     lastName VARCHAR(20)
8 );
9 • DESC employee;
10
11 • INSERT INTO employee VALUES (1, "abc", "cba");
12 • INSERT INTO employee (employeeID, firstName) VALUES (1, "xyz");
13 • SELECT * FROM employee;
14
```

The 'Result Grid' shows the output of the 'SELECT * FROM employee;' query:

employeeID	firstName	lastName
1	abc	cba
1	xyz	NULL

The 'Output' pane shows the 'Action Output' for the query:

#	Time	Action	Message
51	13:18:20	CREATE TABLE employee (employeeID INT, firstName VARCHAR(20), lastName VARC...	0 row(s) affected
52	13:18:21	DESC employee	3 row(s) returned
53	13:18:31	INSERT INTO employee VALUES (1, "abc", "cba")	1 row(s) affected
54	13:18:31	INSERT INTO employee (employeeID, firstName) VALUES (1, "xyz")	1 row(s) affected
55	13:18:31	SELECT * FROM employee LIMIT 0, 1000	2 row(s) returned

3. Update the table

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with the 'company' database selected. The 'employee' table is highlighted under 'Tables'. The right pane shows a query window with the following SQL code:

```
14  
15 ALTER TABLE employee ADD joining_date DATE;  
16  
17 UPDATE employee SET joining_date = "2000-01-01";  
18 UPDATE employee SET lastName = "temp" WHERE firstName="xyz";  
19 SELECT * FROM employee;  
20  
21  
22 DESC employee;
```

The 'Result Grid' shows the output of the query:

employeeID	firstName	lastName	joining_date
1	abc	cba	2000-01-01
1	xyz	temp	2000-01-01

The 'Output' pane shows the execution log:

#	Time	Action	Message
5	16:00:43	SELECT * FROM employee LIMIT 0, 1000	2 row(s) returned
6	16:00:52	ALTER TABLE employee ADD joining_date DATE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
7	16:00:52	UPDATE employee SET joining_date = "2000-01-01"	2 row(s) affected Rows matched: 2 Changed: 2 Warnings: 0
8	16:00:52	UPDATE employee SET lastName = "temp" WHERE firstName="xyz"	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
9	16:00:52	SELECT * FROM employee LIMIT 0, 1000	2 row(s) returned

4. Delete specific row/s

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with the 'company' database selected. The 'employee' table is highlighted under 'Tables'. The right pane shows a query window with the following SQL code:

```
17 UPDATE employee SET joining_date = "2000-01-01";  
18 UPDATE employee SET lastName = "temp" WHERE firstName="xyz";  
19 SELECT * FROM employee;  
20  
21 DELETE FROM employee WHERE firstName = "abc";  
22 SELECT * FROM employee;  
23  
24 TRUNCATE TABLE employee;  
25 SELECT * FROM employee;
```

The 'Result Grid' shows the output of the query:

employeeID	firstName	lastName	joining_date
1	xyz	temp	2000-01-01

The 'Output' pane shows the execution log:

#	Time	Action	Message
7	16:00:52	UPDATE employee SET joining_date = "2000-01-01"	2 row(s) affected Rows matched: 2 Changed: 2 Warnings: 0
8	16:00:52	UPDATE employee SET lastName = "temp" WHERE firstName="xyz"	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
9	16:00:52	SELECT * FROM employee LIMIT 0, 1000	2 row(s) returned
10	16:04:06	DELETE FROM employee WHERE firstName = "abc"	1 row(s) affected
11	16:04:06	SELECT * FROM employee LIMIT 0, 1000	1 row(s) returned

5. Delete all the rows of the table

The screenshot displays a database management interface. On the left, a 'SCHEMAS' tree shows a 'company' database with tables like 'employee'. The main window shows a SQL query editor with the following queries:

```
19 • SELECT * FROM employee;  
20  
21 • DELETE FROM employee WHERE firstName = "abc";  
22 • SELECT * FROM employee;  
23  
24 • TRUNCATE TABLE employee;  
25 • SELECT * FROM employee;  
26  
27
```

Below the editor, a 'Result Grid' shows the columns: employeeID, firstName, lastName, joining_date. The bottom panel shows the 'Output' window with the following execution log:

#	Time	Action	Message
9	16:00:52	SELECT * FROM employee LIMIT 0, 1000	2 row(s) returned
10	16:04:06	DELETE FROM employee WHERE firstName = "abc"	1 row(s) affected
11	16:04:06	SELECT * FROM employee LIMIT 0, 1000	1 row(s) returned
12	16:08:00	TRUNCATE TABLE employee	0 row(s) affected
13	16:08:01	SELECT * FROM employee LIMIT 0, 1000	0 row(s) returned

DBMS Practical no. 3

Title: Create a table and apply following clauses on it: Where , Having ,Group by, Order by clauses.

Theory:

1. Where Clause :

Where clause is used to specify the condition of DDL and DML queries .

Syntax : **SELECT** column1, column2, ...

FROM table_name

WHERE condition;

Example : **SELECT * FROM** employee

WHERE dept="IT";

2. Having Clause :

Having clause is used with aggregate functions instead of Where clause.

Syntax : **SELECT** *column_name(s)* **FROM** *table_name*
 GROUP BY *column_name(s)*
 HAVING *condition*;

Example : SELECT dept, count(emp_id) FROM employee
 GROUP BY dept HAVING COUNT(*)>=2;

3. Group By Clause :

This group by clause groups all rows that have the same values into summary rows. It is often used with aggregate functions to group the result -set by one or more columns.

Syntax: **SELECT** column1, column2 **FROM** table_name **WHERE** condition
 ORDER BY column1, column2... ASC|DESC

Example : SELECT count(*), dept FROM employee GROUP BY dept;

4. Order by clause :

The order by is used to sort result set in ascending or descending order. It sorts record in ascending order by default .to sort record in descending order we used desc keyword .

Example : SELECT emp_name, dept, salary FROM employee ORDER BY salary;
 SELECT emp_name, dept, salary FROM employee ORDER BY salary DESC;

Output:

Table state:

	emp_id	dept	emp_name	salary
▶	1	Marketing	Sam	10000
	2	IT	Clement	30000
	3	Marketing	Rohan	10000
	4	HR	Aman	20000
	5	Marketing	Striver	15000
	6	IT	Harry	50000

1. Example of query using where clause.

The screenshot shows a SQL IDE interface with a Navigator pane on the left, a SQL editor in the center, and a Results pane at the bottom.

Navigator: The 'company' schema is expanded, showing the 'employee' table with columns: emp_id, dept, emp_name, salary. Other schemas like friends, sakila, students, sys, and world are also visible.

SQL Editor: The following SQL script is executed:

```
9 • INSERT INTO employee VALUES(2,"IT","Clement",30000);
10 • INSERT INTO employee VALUES(3,"Marketing","Rohan",10000);
11 • INSERT INTO employee VALUES(4,"HR","Aman",20000);
12 • INSERT INTO employee VALUES(5,"Marketing","Striver",15000);
13 • INSERT INTO employee VALUES(6,"IT","Harry",50000);
14 • SELECT * FROM employee;
15
16 • SELECT * FROM employee WHERE dept="IT";
17
18 • SELECT dept, count(emp_id) FROM employee GROUP BY dept HAVING COUNT(*)>=2;
19
20
```

Results: The 'Result Grid' shows the output of the queries:

emp_id	dept	emp_name	salary
2	IT	Clement	30000
6	IT	Harry	50000

Table: employee

Columns:

emp_id	int
dept	varchar(20)
emp_name	varchar(20)
salary	int

Action Output:

#	Time	Action	Message
✓ 40	12:01:50	SELECT dept, count(emp_id) FROM employee GROUP BY dept HAVING COUNT(*)>=2 L...	2 row(s) returned
✓ 41	12:02:29	SELECT * FROM employee WHERE dept="IT" LIMIT 0, 1000	2 row(s) returned

2. Example of query using having clause

Navigator: SQL File 3*

SCHEMAS

Filter objects

company

Tables

employee

Columns

- emp_id
- dept
- emp_name
- salary

Indexes

Foreign Keys

Triggers

Views

Stored Procedures

Functions

friends

sakila

students

sys

world

Administration Schemas

Information

Table: employee

Columns:

- emp_id int
- dept varchar(20)
- emp_name varchar(20)
- salary int

Object Info Session

SQL File 3*

Limit to 1000 rows

```

9 • INSERT INTO employee VALUES(2,"IT","Clement",30000);
10 • INSERT INTO employee VALUES(3,"Marketing","Rohan",10000);
11 • INSERT INTO employee VALUES(4,"HR","Aman",20000);
12 • INSERT INTO employee VALUES(5,"Marketing","Striver",15000);
13 • INSERT INTO employee VALUES(6,"IT","Harry",50000);
14 • SELECT * FROM employee;
15
16 • SELECT * FROM employee WHERE dept="IT";
17
18 • SELECT dept, count(emp_id) FROM employee GROUP BY dept HAVING COUNT(*)>=2;
19
20

```

Result Grid

dept	count(emp_id)
Marketing	3
IT	2

Result 8 x

Output

Action Output

#	Time	Action	Message
41	12:02:29	SELECT * FROM employee WHERE dept="IT" LIMIT 0, 1000	2 row(s) returned
42	12:03:22	SELECT dept, count(emp_id) FROM employee GROUP BY dept HAVING COUNT(*)>=2 L...	2 row(s) returned

3. Example of query using Group by clause.

Navigator: SQL File 3*

SCHEMAS

Filter objects

company

Tables

employee

Columns

- emp_id
- dept
- emp_name
- salary

Indexes

Foreign Keys

Triggers

Views

Stored Procedures

Functions

friends

sakila

students

sys

world

Administration Schemas

Information

Table: employee

Columns:

- emp_id int
- dept varchar(20)
- emp_name varchar(20)
- salary int

Object Info Session

SQL File 3*

Limit to 1000 rows

```

15
16 • SELECT * FROM employee WHERE dept="IT";
17
18 • SELECT dept, count(emp_id) FROM employee GROUP BY dept HAVING COUNT(*)>=2;
19
20 • SELECT count(*), dept FROM employee GROUP BY dept;
21
22
23
24
25

```

Result Grid

count(*)	dept
3	Marketing
2	IT
1	HR

Result 10 x

Output

Action Output

#	Time	Action	Message
43	12:06:05	SELECT count(*) FROM employee GROUP BY dept LIMIT 0, 1000	3 row(s) returned
44	12:06:14	SELECT count(*), dept FROM employee GROUP BY dept LIMIT 0, 1000	3 row(s) returned

4. Example of query using order by clause.

Navigator

SCHEMAS

Filter objects

company

Tables

employee

Columns

- emp_id
- dept
- emp_name
- salary

Indexes

Foreign Keys

Triggers

Views

Stored Procedures

Functions

friends

sakila

students

sys

world

Administration

Schemas

Information

Table: employee

Columns:

- emp_id int
- dept varchar(20)
- emp_name varchar(20)
- salary int

Object Info

Session

SQL File 3*

Limit to 1000 rows

```
16 SELECT * FROM employee WHERE dept="IT";
17
18 SELECT dept, count(emp_id) FROM employee GROUP BY dept HAVING COUNT(*)>=2;
19
20 SELECT count(*), dept FROM employee GROUP BY dept;
21
22 SELECT emp_name, dept, salary FROM employee ORDER BY salary;
23
24 SELECT emp_name, dept, salary FROM employee ORDER BY salary DESC;
25
26
27
```

Result Grid

Filter Rows:

Exports

Wrap Cell Content:

emp_name	dept	salary
Sam	Marketing	10000
Rohan	Marketing	10000
Striver	Marketing	15000
Aman	HR	20000
Clement	IT	30000
Harry	IT	50000

employee 11 x

Output

Action Output

#	Time	Action	Message
44	12:06:14	SELECT count(*), dept FROM employee GROUP BY dept LIMIT 0, 1000	3 row(s) returned
45	12:09:31	SELECT emp_name, dept, salary FROM employee ORDER BY salary LIMIT 0, 1000	6 row(s) returned

Navigator

SCHEMAS

Filter objects

company

Tables

employee

Columns

- emp_id
- dept
- emp_name
- salary

Indexes

Foreign Keys

Triggers

Views

Stored Procedures

Functions

friends

sakila

students

sys

world

Administration

Schemas

Information

Table: employee

Columns:

- emp_id int
- dept varchar(20)
- emp_name varchar(20)
- salary int

Object Info

Session

SQL File 3*

Limit to 1000 rows

```
15
16 S Execute the selected portion of the script or everything, if there is no selection
17
18 SELECT dept, count(emp_id) FROM employee GROUP BY dept HAVING COUNT(*)>=2;
19
20 SELECT count(*), dept FROM employee GROUP BY dept;
21
22 SELECT emp_name, dept, salary FROM employee ORDER BY salary;
23
24 SELECT emp_name, dept, salary FROM employee ORDER BY salary DESC;
25
26
```

Result Grid

Filter Rows:

Exports

Wrap Cell Content:

emp_name	dept	salary
Harry	IT	50000
Clement	IT	30000
Aman	HR	20000
Striver	Marketing	15000
Sam	Marketing	10000
Rohan	Marketing	10000

employee 12 x

Output

Action Output

#	Time	Action	Message
45	12:09:31	SELECT emp_name, dept, salary FROM employee ORDER BY salary LIMIT 0, 1000	6 row(s) returned
46	12:10:40	SELECT emp_name, dept, salary FROM employee ORDER BY salary DESC LIMIT 0, 1000	6 row(s) returned

DBMS Practical no. 4

Title: Implement the following Functions in SQL a) Date functions b) Time functions c) String functions d) Aggregate functions.

Theory:

A. Date and Time function :

Date Function :

```
SELECT NOW();  
SELECT CURDATE();  
SELECT DATE(SYSDATE());  
SELECT DATE_FORMAT("2020-10-18", "%d/%M/%Y");  
SELECT DATEDIFF("2020-10-18", "2003-03-12");  
SELECT CURTIME();  
SELECT DATE_ADD("2020-10-18", INTERVAL 10 DAY);  
SELECT DATE_SUB("2020-10-18", INTERVAL 10 DAY);  
SELECT TIME(SYSDATE());  
SELECT DAYNAME("2020-10-18");
```

Time Function :

```
SELECT current_time();  
SELECT TIME_TO_SEC("19:30:10");
```

B. String Function :

```
SELECT ASCII("R");  
SELECT CHAR(82);  
SELECT LENGTH(emp_name), emp_name from employee;  
SELECT BIT_LENGTH("Samuel"), emp_name from employee;  
SELECT CONCAT(emp_name, " works in ", dept) from employee;
```

```

SELECT FIELD("s", "h", "a", "r", "p"), emp_name from employee;
SELECT FORMAT(88.38439, 2), emp_name from employee;
SELECT LOWER(emp_name), emp_name from employee;
SELECT UPPER(emp_name), emp_name from employee;
SELECT LEFT(emp_name, 3), emp_name from employee;

```

C. Aggregate functions :

```

SELECT MIN(salary) FROM employee;
SELECT MIN(salary) FROM employee WHERE dept="IT";

SELECT MAX(salary) FROM employee;
SELECT MAX(salary) FROM employee WHERE dept="Marketing";

SELECT AVG(salary) FROM employee;
SELECT AVG(salary) FROM employee WHERE dept IN("HR", "IT");

SELECT SUM(salary) FROM employee;
SELECT SUM(salary) FROM employee WHERE dept="Marketing";

SELECT COUNT(*) FROM employee;
SELECT COUNT(*) FROM employee WHERE dept != "HR";

```

Output:

1. Date and time functions:

Result Grid		Filter Rows:	Result Grid		Filter	Result Grid		Filter Rows
	NOW()			CURDATE()			DATE(SYSDATE())	
▶	2020-10-20 12:59:43		▶	2020-10-20		▶	2020-10-20	

Result Grid	Filter Rows:	DATEDIFF("2020-10-18", "2003-03-12")	CURTIME()
DATE_FORMAT("2020-10-18", "%d/%M/%Y")		6430	12:59:43
18/October/2020			

DATE_ADD("2020-10-18", INTERVAL 10 DAY)	DATE_SUB("2020-10-18", INTERVAL 10 DAY)	TIME(SYSDATE())
2020-10-28	2020-10-08	12:59:44

DAYNAME("2020-10-18")	current_time()	TIME_TO_SEC("19:30:10")
Sunday	13:10:40	70210

2. String functions:

Result Grid	Filter Rows:	Result Grid	Result Grid
ASCII("R")		LENGTH(emp_name) emp_name	
82		3 Sam	
		7 Clement	
		5 Rohan	
		4 Aman	
		7 Striver	

CHAR(82)	
R	

BIT_LENGTH("Samuel") emp_name	
48 Sam	
48 Clement	
48 Rohan	
48 Aman	
48 Striver	

FIELD("s", "h", "a", "r", "p") emp_name	
0 Sam	
0 Clement	
0 Rohan	
0 Aman	
0 Striver	

CONCAT(emp_name, " works in ", dept)	
Sam works in Marketing	
Clement works in IT	
Rohan works in Marketing	
Aman works in HR	
Striver works in Marketing	

FORMAT(88.38439, 2) emp_name	
88.38 Sam	
88.38 Clement	
88.38 Rohan	
88.38 Aman	
88.38 Striver	

LOWER(emp_name) emp_name	
sam Sam	
clement Clement	
rohan Rohan	
aman Aman	
striver Striver	

UPPER(emp_name) emp_name	
SAM Sam	
CLEMENT Clement	
ROHAN Rohan	
AMAN Aman	
STRIVER Striver	

LEFT(emp_name, 3) emp_name	
Sam Sam	
Cle Clement	
Roh Rohan	
Ama Aman	
Str Striver	

3. Aggregate functions:

Employee table:

	emp_id	dept	emp_name	salary
▶	1	Marketing	Sam	10000
	2	IT	Clement	30000
	3	Marketing	Rohan	10000
	4	HR	Aman	20000
	5	Marketing	Striver	15000
	6	IT	Harry	50000

```
7 • SELECT MIN(salary) FROM employee;
8 • SELECT MIN(salary) FROM employee WHERE dept="IT";
9
```

Result Grid Filter Rows: Export: Wrap Cell Content: ☐

MIN(salary)
▶ 10000

```
7 • SELECT MIN(salary) FROM employee;
8 • SELECT MIN(salary) FROM employee WHERE dept="IT";
9
```

Result Grid Filter Rows: Export: Wrap Cell Content: ☐

MIN(salary)
▶ 30000

```
10 • SELECT MAX(salary) FROM employee;
11 • SELECT MAX(salary) FROM employee WHERE dept="Marketing";
```

Result Grid Filter Rows: Export: Wrap Cell Content: ☐

MAX(salary)
▶ 50000

```
11 • SELECT MAX(salary) FROM employee WHERE dept="Marketing";
```

Result Grid Filter Rows: Export: Wrap Cell Content: ☐

MAX(salary)
▶ 15000

```
14 • SELECT AVG(salary) FROM employee WHERE dept IN("HR", "IT");
```

Result Grid Filter Rows: Export: Wrap Cell Content: ☐

AVG(salary)
▶ 22500.0000

```
16 • SELECT SUM(salary) FROM employee;
```

Result Grid Filter Rows: Export:

SUM(salary)
▶ 135000

```
19 • SELECT COUNT(*) FROM employee;
```

Result Grid Filter Rows:

COUNT(*)
▶ 6

```
17 • SELECT SUM(salary) FROM employee WHERE dept="Marketing";
```

Result Grid Filter Rows: Export: Wrap Cell Content: ☐

SUM(salary)
▶ 35000

```
20 • SELECT COUNT(*) FROM employee WHERE dept != "HR";
```

Result Grid Filter Rows: Export: Wrap Cell Content: ☐

COUNT(*)
▶ 5

DBMS Practical no. 5

Title: Implementation of all types of Joins.

Theory:

1. Example of query using INNER JOIN.

The inner join keyword selects records that have same values in both tables.

Syntax : Select column_name() from table_name inner join table_name2
 on table_name.column_name=table_name2.column_name;

Example : SELECT id, loanID, branch, amount FROM customer INNER JOIN loans ON id=loans.cID;

2. Example of query using LEFT OUTER JOIN.

Left join keyword returns all records from the table (table1) , and the matched records from the right table(table2). The results is NULL from the right side if there is no match.

Syntax : Select column_name() from table_name left join table_name2
 on table_name.column_name=table_name2.column_name;

Example : SELECT id, custName, loanID, branch, amount FROM customer LEFT OUTER JOIN loans
 ON customer.ID=loans.cID;

3. Example of query using RIGHT OUTER JOIN.

The right join keyword returns all records from the right table (table2) , and the matched records from the left table(table1). The results is NULL from the left side if there is no match.

Syntax : Select column_name() from table_name1 right join table_name2 on
 table_name1.column_name=table_name2.column_name;

Example : SELECT id, custName, loanID, branch, amount FROM customer RIGHT OUTER JOIN loans
 ON customer.ID=loans.cID;

4. Example of query using FULL OUTER JOIN.

The full outer join keyword returns all records when there ia a match in left(table1) or right(table2) table records.

Syntax : Select column_name(s) from table_name1 full join table_name2

on table_name1.column_name=table_name2.column_name where condition;

Example : SELECT id, custName, loanID, branch, amount FROM customer FULL OUTER JOIN loans
ON customer.ID=loans.cID;

Output:

Customer table:

	id	custName	loanID	balance
▶	1	Max	1	200000
	2	Merry	2	180000
	3	Michal	3	250000
	4	Jimmy	NULL	400000
•	NULL	NULL	NULL	NULL

Loan table:

	loanIDs	cID	branch	amount
▶	1	1	Bangalore	100000
	2	3	Mumbai	150000
	3	2	Banglore	80000
•	NULL	NULL	NULL	NULL

1. Inner join

	id	loanID	branch	amount
▶	1	1	Bangalore	100000
	3	3	Mumbai	150000
	2	2	Banglore	80000

2. Left outer join:

	id	custName	loanID	branch	amount
▶	1	Max	1	Bangalore	100000
	2	Merry	2	Banglore	80000
	3	Michal	3	Mumbai	150000
	4	Jimmy	NULL	NULL	NULL

3. Right outer join

	id	custName	loanID	branch	amount
▶	1	Max	1	Bangalore	100000
	3	Michal	3	Mumbai	150000
	2	Merry	2	Banglore	80000