SQL CHAPTER LIST

EXPT. No.	CHAPTERS DESCRIPTION
1	Introduction to SQL.
2	To study and implement DDL, DML commands.
3	To study and implement ORDER BY, GROUP BY, HAVING CLAUSE.
4	To implement SQL commands using aggregate functions.
5	To study and implement DCL and TCL commands.
6	To implement displaying data from multiple tables. JOINS
7	To study and implement VIEWS.
8	To study SQL queries using union, union all, intersect and minus.
9	To study integrity constraints.

Chapter No. 1

Aim: Introduction to SQL

Learning outcome: At the end of this Chapter, students will be able to understand basic concept of SQL which consist of history of SQL, what is SQL, SQL command categories and data types in SQL.

Resources:

Software Required: MySQL/Oracle, Windows 7 or any other compatible operating system.

Hardware Required:

- Processor: Pentium dual core or any other similar or higher configuration recommended.
- Ram: Minimum 1 GB recommended.
- Hard-disk: 500 GB or higher recommended.

Theory:

(1) Database management system:

A database is a collection of related data that lives for a long time. A database management system (DBMS) is a system (software) that provides an interface to database for information storage and retrieval. DBMS has capacity to store huge amount of data, it has easy interface, efficient retrieval, security management and multiuser support.

(2) History of SQL:

SQL was developed by IBM to provide an interface to the relational database. In a very short period of time, SQL has become the standard language to access most relational database. Because of this, scripts of SQL statements are more or less portable between operating system & different relation databases. In 1970 − Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases. In 1974 − Structured Query Language appeared. In 1978 − IBM worked to develop Codd's ideas and released a product named System. In 1986 − IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

(3) What is SQL?

SQL is more than a query Language notwithstanding its name. Within the SQL language there are two major components:

a) DDL (Data definition language)
b) DML (Data manipulation language)

SQL is a programming language but it is not a procedural language (white loops, if statements etc) Generally SQL works on sets of data tables restricted by the constraint included in the statement. A single SQL statement can retrieve a single item or thousands of items from the database depending on what is the requests in the SQL statement.

(4) Categories of Language:

The database system is an intermediate between physical database, operator, operating system & user. In order to provide various facilities to different types of uses, DBMS provides a specialized programming language called database language which are as follows,

a) Data definition language:

Data definition language is used to describe the details of data. The DBMS will have a compiler whose function is to process DDL statement in order to identify the description of scheme (summary). The storage structure and access methods used by the database system are specified by of DDL called a data storage and data definition language. The implementation details of the database schemas details are usually hidden from the user. This category consist of create, alter, truncate, drop and rename command.

b) Data Manipulation Language:

Once the database scheme (summary) is compiled user does manipulation of the database, and manipulation means,

1) Retrieval of information stored in database
2) Insertion of new information
3) Deletion of information
4) Modification of information

This language enables users to access or manipulate data that's why it is called data manipulation language. It contains a basic select statement to retrieve the data from the tables. Commands like select, update, delete & inserts are used in DML.

(5) Data types:

Following are some of the data types in SQL,
a) Char(n):- It is a fixed length character string with uses specified length. The full from character can be used instead.
b) Varchar(n):- It is a variable length character string, with user specified maximum l
c) Int(n):- It is an integer data type which is used to store integer value.
d) Number:- (p, s) It is a fixed point number with user specified precision. The number consist of precision and scale. Thus, number (3, 1) allows 44.5 to be store exactly, but neither 444.5 nor 0.45 can be stored exactly in a field of this type.
e) Date:- It is a calendar date containing a (four digit) year, month and day of month.

Result: In this way, the basic concepts of SQL like history, command categories and data types are studied successfully.

Suggested questions:

1. Define database and database management system?
2. What is SQL?
3. Explain types of command categories in SQL?
4. What are the various data types in SQL?

Chapter No. 2

Aim: To study and implement DDL, DML commands.

Learning outcome: At the end of this Chapter, students will be able to implement DML and DDL commands such as create, insert, delete, update, alter etc.

Resources:

Software Required: MySQL/Oracle, Windows 7 or any other compatible operating system.

Hardware Required:

- Processor: Pentium dual core or any other similar or higher configuration recommended.
- Ram: Minimum 1 GB recommended.
- Hard-disk: 500 GB or higher recommended.

Theory:

(a) Data definition language commands:

DDL (Data Definition Language) statements are used to create, delete, or change the objects of a database. Typically a database administrator is responsible for using DDL statements or production databases in a large database system.
Create - It is used to create a table.
Alter - This command is used to add a new column, modify the existing column definition and to include or drop integrity constraint.
Drop - It will delete the table structure. Provided table will be empty.
Truncate - If there is no further use of records stored in a table and the structure has to be retained, and then the records alone can be deleted with this command.
Desc - This is used to view the structure of the table.

1) CREATE:

Purpose : Used to create a table.
Syntax : create table table_name (column_name data type);
Example : create table employee(first varchar(15), last varchar(20), age number(3));

2) ALTER:

Purpose : To add, modify or drop a column.

i) Adding column to table:

Syntax : alter table table_name add column_name data type;
Example : alter table student add roomno char [10];

ii) Modifying the column:

Syntax : alter table table_name modify column_name data type;
Example : alter table student modify roomno char [20];"

iii) Dropping the column:

Syntax : alter table table_name drop column_name;
Example : alter table student drop roomno;

3) DROP:

Purpose : It is used to delete all the rows and table structure.
Syntax : drop table table_name;
Example : drop table student;

- All data and structure in the table is deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- We cannot rollback the drop table statement.

4) TRUNCATE:

Purpose : It is used to delete all the table contents.
Syntax : truncate table table_name;
Example : truncate table student;

- Removes all rows from a table.
- Release the storage space used by that table.
- We cannot rollback row removal when using truncate.

5) RENAME:

Purpose : It is used rename the name of a table, view, sequence etc.
Syntax : alter table table_name rename to new_name;
Example : alter table student rename to goodstudent;

(b) Data manipulation language commands:

Data manipulation language is used for inserting, selecting, updating & deleting the data in the table in a database. DML commands are the most frequently used SQL commands and are used to query and manipulate the existing database objects. Some of the commands are,

1. Insert
2. Select
3. Update
4. Delete

1) SELECT:

Purpose : It is used to select particular column or entire table. We can either display all columns in a table or only specify column from the table.

Retrieval of all columns from a table:
Syntax : select * from table_name;
Example : select * from student;

Retrieval of specific columns from a table:
Syntax **:** select column_name1, …..,column_name2 from table_name;
Example : select firstname, lastname from student;

2) **INSERT:**

Purpose : This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes.
Syntax : insert into table_name values(fieldvalue-1,fieldvalue-2,…,fieldvalue-n);
Example : insert into student values(1,'abc','shegaon');

3) UPDATE:

Purpose : Used to change or modify the existing column values.
Syntax : update table_name set column_name=new value where col_name=value;
Example : update student set address =’kknagar’ where firstname=’Tanmay’;

4) DELETE:

Purpose : Delete the corresponding row.
Syntax : delete from table_name where column_name=value;
Example : delete from student where firstname=’Tanmay’;

Result: In this way, DDL and DML commands are studied and implemented successfully.

Suggested questions:

1. Define DDL and DML?
2. Explain various commands from DDL category?
3. Explain various commands from DML category?

Chapter No. 3

Aim: To study and implement ORDER BY, GROUP BY and HAVING clause.

Learning outcome: At the end of this Chapter, students will be able perform various operations on table by using clauses which consist of order by, group by and having clause.

Resources:

Software Required: MySQL/Oracle, Windows 7 or any other compatible operating system.

Hardware Required:

- Processor: Pentium dual core or any other similar or higher configuration recommended.
- Ram: Minimum 1 GB recommended.
- Hard-disk: 500 GB or higher recommended.

Theory:

1) ORDER BY clause:

The SQL order by clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

Syntax: select * from table_name order by column_name; // to get in ascending order
select * from table_name order by column_name desc; // to get in descending order

You can use more than one column in the order by clause. Make sure whatever column you are using to sort, that column should be in column-list. Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   | MP        |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
+----+----------+-----+-----------+----------+
```


Following is an example, which would sort the result in ascending order by NAME:

```
SQL> SELECT * FROM CUSTOMERS
     ORDER BY NAME;
```

This would produce the following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|6   |Komal     |22   | MP        |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
+----+----------+-----+-----------+----------+
```

Following is an example, which would sort the result in descending order by NAME:

```
SQL> SELECT * FROM CUSTOMERS
     ORDER BY NAME DESC;
```

This would produce the following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
|6   |Komal     |22   | MP        |4500.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
+----+----------+-----+-----------+----------+
```

(b) GROUP BY clause:

The SQL group by clause is used in collaboration with the SELECT statement to arrange identical data into groups.
The group by clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax: select column1, column2 from table_name where [condition] group by column1;

The basic syntax of GROUP BY clause is given above. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

Example:

Consider the CUSTOMERS table is having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   | MP        |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
+----+----------+-----+-----------+----------+
```

If you want to get some specific columns from customer table, then GROUP BY query would be as follows:

```
SQL> SELECT NAME,SUM FROM CUSTOMERS
     GROUP BY NAME;
```

This would produce the following result:

```
+----------+--------+
| NAME     |SUM     |
+----------+--------+
|Chaitali  |6500.00 |
|Hardik    |8500.00 |
|kaushik   |2000.00 |
|Khilan    |1500.00 |
|Komal     |4500.00 |
|Muffy     |10000.00|
|Ramesh    |2000.00 |
+----------+--------+
```

(b) HAVING clause:

The HAVING clause enables you 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.

Syntax: select from table_name where group by having order by;

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

Example: Now, let us have following table where CUSTOMERS table with following records;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   | MP        |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
```

Following is the example, which would display record for entries having salary greater than or equal to 8000.

```
SQL > SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
GROUP BY age
HAVING SALARY>=8000;
```

This would produce the following result:

```
+----+--------+-----+---------+---------+
| ID | NAME   | AGE | ADDRESS |SALARY   |
+----+--------+-----+---------+---------+
|5   |Hardik  |27   |Bhopal   |8500.00  |
|7   |Muffy   |24   |Indore   |10000.00 |
+----+--------+-----+---------+---------+
```

Result: Hence we have successfully executed SQL queries using order by, group by and having clause.

Suggested questions:

1) What are various clauses?
2) Explain working of ORDER BY clause and GROUP BY clause?
3) Explain HAVING clause?

Chapter No. 4

Aim: To implement SQL commands using aggregate functions.

Learning outcome: At the end of this Chapter, students will be able to execute different aggregate functions such as count, min, max, avg, distinct and sum on given table.

Resources:

Software Required: MySQL/Oracle, Windows 7 or any other compatible operating system.

Hardware Required:

- Processor: Pentium dual core or any other similar or higher configuration recommended.
- Ram: Minimum 1 GB recommended.
- Hard-disk: 500 GB or higher recommended.

Theory:

Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group. These functions are: COUNT, MAX, MIN, AVG, SUM, DISTINCT**.**

1. COUNT():

This function returns the number of rows in the table that satisfies the condition specified in the WHERE condition. If the WHERE condition is not specified, then the query returns the total number of rows in the table.
Syntax: select count(*) from table_name where [condition];
Example: Consider the following table CUSTOMER,

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   |Mumbai     |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
+----+----------+-----+-----------+----------+
```


If you want to get entries from above table having address is Mumbai then query would be as follows,

select count (*) from customer where address = 'Mumbai';

And output would be ‘2’ rows as follows,

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|6   |Komal     |22   |Mumbai     |4500.00   |
+----+----------+-----+-----------+----------+
```

2. DISTINCT():

This function is used to select the distinct (unique) rows.
Syntax: select distinct address from employee;
Example: Consider the following table CUSTOMER,

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   |Mumbai     |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
+----+----------+-----+-----------+----------+
```

If you want to select all distinct address from customer table, the query would be:
select distinct (address) from customer;
And the output would be as follows,

```
+-----------+
| ADDRESS   |
+-----------+
|Ahmedabad  |
|Delhi      |
|Kota       |
|Mumbai     |
|Bhopal     |
|Indore     |
+-----------+
```


3. MAX():

This function is used to get the maximum value from a column.
Syntax: select max(column_name) from table_name;
Example: Consider the following table CUSTOMER,

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   |Mumbai     |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
+----+----------+-----+-----------+----------+
```

To get the maximum salary drawn by an employee, the query would be:

select max(salary) from customer;

And the output would be as follows,

```
+----------+
|  SALARY  |
+----------+
|10000.00  |
+----------+
```

4. MIN():

This function is used to get the minimum value from a column.
Syntax: select min(column_name) from table_name;
Example: Consider the following table CUSTOMER,

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   |Mumbai     |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
+----+----------+-----+-----------+----------+
```


To get the minimum salary drawn by an employee, the query would be:

select min(salary) from customer;

And the output would be as follows,

```
+----------+
|  SALARY  |
+----------+
|1500      |
+----------+
```

5. AVG():

This function is used to get the average value from a numeric column.
Syntax: select avg(column_name) from table_name;
Example: Consider the following table CUSTOMER,

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   |Mumbai     |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
+----+----------+-----+-----------+----------+
```

To get the average of salary column, the query would be:

select avg(salary) from customer;

And the output would be as follows,

```
+----------+
|  SALARY  |
+----------+
| 5000     |
+----------+
```

6. SUM():

This function is used to get the sum of a numeric column.
Syntax: select sum(column_name) from table_name;
Example: Consider the following table CUSTOMER,

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   |Mumbai     |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
+----+----------+-----+-----------+----------+
```

To get the sum of salary column, the query would be:

select sum(salary) from customer;

And the output would be as follows,

```
+----------+
|  SALARY  |
+----------+
|35000      |
+----------+
```

Result: Hence, we have successfully executed SQL queries by using various aggregate functions.

Suggested questions:

1) What are various aggregate functions?
2) Explain how each aggregate function works?

Chapter No. 5

Aim: To study and implement DCL and TCL commands.

Learning outcome: At the end of this Chapter, students will be able to elaborate DCL and TCL commands and implementation of same commands.

Resources:

Software Required: MySQL/Oracle, Windows 7 or any other compatible operating system.

Hardware Required:

- Processor: Pentium dual core or any other similar or higher configuration recommended.
- Ram: Minimum 1 GB recommended.
- Hard-disk: 500 GB or higher recommended.

Theory:

(1) Data control language commands:

Data control language provides users with privilege commands.

a) GRANT:

It is used to give permission to specific user with privilege on database object.
Syntax: grant privilege on <db_object_name> to <user_name>;
Example: grant select on employee to ravi;

b) REVOKE:

Used to withdraw the privilege that has been granted to the user.
Syntax: revoke on <db_object_name> from <user_name>;
Example: revoke select on employee from ravi;

Precautions:

1. Select username and password very precisely to create user.
2. Allocate privileges to authorized user only.

(2) Transaction control language commands:

TCL commands are used to manage transactions in the database. These commands are used to manage changes made in the table by statements.

a) COMMIT:

Commit command is used to save any transaction permanently into database. Once we use command then we can't rollback. When we use DML command then changes made by these commands are not permanent until the session is closed and these changes can be rolled back.

Syntax: commit;

Example: commit;

b) SAVEPOINT:

Savepoint command is used to save a transaction temporarily so that we can rollback to that transaction when required.

Syntax: savapoint savepoint_name;

Example: savepoint a;

c) ROLLBACK:

This command restores the database to the last committed state. This command is use with savepoint command to go to a savepoint in an ongoing transaction.

Syntax: rollback to savepoint_name:

Example: rollback to a;

Example: Use of commit, savepoint and rollback command as follows,

Consider the following table sample;

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from sample;
+------+------+
| id   | name |
+------+------+
|    1 | abc  |
|    2 | mnp  |
|    3 | xyz  |
+------+------+
3 rows in set (0.00 sec)

mysql> insert into sample values(4,'psd');
Query OK, 1 row affected (0.00 sec)

mysql> savepoint a;
Query OK, 0 rows affected (0.00 sec)
```

Here we add one entry and savepoint a.

```
mysql> select *from sample;
+------+------+
| id   | name |
+------+------+
|    1 | abc  |
|    2 | mnp  |
|    3 | xyz  |
|    4 | psd  |
+------+------+
4 rows in set (0.00 sec)

mysql> insert into sample values(5,'der');
Query OK, 1 row affected (0.02 sec)

mysql> select *from sample;
+------+------+
| id   | name |
+------+------+
|    1 | abc  |
|    2 | mnp  |
|    3 | xyz  |
|    4 | psd  |
|    5 | der  |
+------+------+
5 rows in set (0.00 sec)
```

Now as shown above, we add one more entry into the given table to check rollback command working.

```
mysql> rollback to a;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from sample;
+------+------+
| id   | name |
+------+------+
|    1 | abc  |
|    2 | mnp  |
|    3 | xyz  |
|    4 | psd  |
+------+------+
4 rows in set (0.00 sec)

mysql>
```

As shown above, we used rollback command and we get the output till savepoint that we have added earlier.

```
mysql> select *from sample;
+------+------+
| id   | name |
+------+------+
|    1 | abc  |
|    2 | mnp  |
|    3 | xyz  |
|    4 | psd  |
+------+------+
4 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

As shown above, we used commit command.

Result: Hence DCL and TCL commands are studied and implemented successfully.

Suggested questions:

1) Name various DCL and TCL commands?
2) What is the use of COMMIT command?
3) Explain working of SAVEPOINT and ROLLBACK?
4) Explain GRANT and REVOKE command?

Chapter No. 6

Aim: To implement displaying data from multiple tables.

Learning outcome: At the end of this Chapter, students will be able to understand concept of JOINS and implementation of displaying data from multiple tables by using JOINS.

Resources:

Software Required: MySQL/Oracle, Windows 7 or any other compatible operating system.

Hardware Required:

- Processor: Pentium dual core or any other similar or higher configuration recommended.
- Ram: Minimum 1 GB recommended.
- Hard-disk: 500 GB or higher recommended.

Theory:

Data can be displayed by using various types of joins, Cartesian product. Joins can be expressed as a Cartesian product followed by a selection. The SQL Join clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. Consider the following two tables;

(a) CUSTOMERS table is as follows:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1   |Ramesh    |32   |Ahmedabad  |2000.00   |
|2   |Khilan    |25   |Delhi      |1500.00   |
|3   |kaushik   |23   |Kota       |2000.00   |
|4   |Chaitali  |25   |Mumbai     |6500.00   |
|5   |Hardik    |27   |Bhopal     |8500.00   |
|6   |Komal     |22   | MP        |4500.00   |
|7   |Muffy     |24   |Indore     |10000.00  |
+----+----------+-----+-----------+----------+
```


(b) Another table is ORDERS as follows:

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
|102  |2009-10-0800:00:00   |3            |3000    |
|100  |2009-10-0800:00:00   |3            |1500    |
|101  |2009-11-2000:00:00   |2            |1560    |
|103  |2008-05-2000:00:00   |4            |2060    |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables in our SELECT statement as follows:

```
SQL> SELECT ID, NAME, AGE, AMOUNT
        FROM CUSTOMERS, ORDERS
WHERE CUSTOMERS.ID= ORDERS.CUSTOMER_ID;
```

This would produce the following result:

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|3   |kaushik   |23   |3000    |
|3   |kaushik   |23   |1500    |
|2   |Khilan    |25   |1560    |
|4   |Chaitali  |25   |2060    |
+----+----------+-----+--------+
```

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=,!=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

There are different types of joins available in SQL:
INNER JOIN: returns rows when there is a match in both tables.
LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
FULL JOIN: returns rows when there is a match in one of the tables.
SELF JOIN: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined tables.

1) INNER JOIN:
The most frequently used and important of the join is the INNER JOIN. They are also referred to as an EQUIJOIN. The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the

join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax:

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

Example:

Consider table fast and slow as follows,

```
mysql> select *from fast;
+------+------+----------+
| id   | name | city     |
+------+------+----------+
|    1 | abc  | shegaon  |
|    2 | mnp  | khamgaon |
|    3 | jhy  | buldhana |
+------+------+----------+
3 rows in set (0.00 sec)

mysql> select *from slow;
+------+------+------------+
| no   | name | mob        |
+------+------+------------+
|    6 | abc  | 9865321474 |
|    5 | mnp  | 9965321474 |
|    4 | jhy  | 9765321474 |
+------+------+------------+
3 rows in set (0.00 sec)
```

The query for inner join would be as follows,

select fast.id,slow.mob,fast.city from fast inner join slow on fast.name=slow.name;

And output for above query would be as follows,

```
mysql> select fast.id,slow.mob,fast.city from fast inner join slow on fast.name=
slow.name;
+------+------------+----------+
| id   | mob        | city     |
+------+------------+----------+
|    1 | 9865321474 | shegaon  |
|    2 | 9965321474 | khamgaon |
|    3 | 9765321474 | buldhana |
+------+------------+----------+
3 rows in set (0.00 sec)

mysql>
```


2) LEFT JOIN:

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax:

```
SELECT table1.column1, table2.column2...
FROM table1
LEFT JOIN table2
ON table1.common_field = table2.common_field;
```

Example:

Consider table fast and slow as follows,

```
mysql> select *from fast;
+------+------+-----------+
| id   | name | city      |
+------+------+-----------+
|    1 | abc  | shegaon   |
|    2 | mnp  | khamgaon  |
|    3 | jhy  | buldhana  |
+------+------+-----------+
3 rows in set (0.00 sec)

mysql> select *from slow;
+------+------+------------+
| no   | name | mob        |
+------+------+------------+
|    6 | abc  | 9865321474 |
|    5 | mnp  | 9965321474 |
|    4 | jhy  | 9765321474 |
+------+------+------------+
3 rows in set (0.00 sec)
```

Query for left join would be as follows,

select fast.id,fast.city,slow.mob from fast left join slow on fast.name=slow.name;

And output for above mention query would be as follows,

```
mysql> select fast.id,fast.city,slow.mob from fast left join slow on fast.name=s
low.name;
+------+----------+------------+
| id   | city     | mob        |
+------+----------+------------+
|    1 | shegaon  | 9865321474 |
|    2 | khamgaon | 9965321474 |
|    3 | buldhana | 9765321474 |
+------+----------+------------+
3 rows in set (0.00 sec)

mysql>
```

3) RIGHT JOIN:

The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.
This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax:

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

Example:

Consider table fast and slow as follows,

```
mysql> select *from fast;
+------+------+----------+
| id   | name | city     |
+------+------+----------+
|    1 | abc  | shegaon  |
|    2 | mnp  | khamgaon |
|    3 | jhy  | buldhana |
+------+------+----------+
3 rows in set (0.00 sec)

mysql> select *from slow;
+------+------+------------+
| no   | name | mob        |
+------+------+------------+
|    6 | abc  | 9865321474 |
|    5 | mnp  | 9965321474 |
|    4 | jhy  | 9765321474 |
+------+------+------------+
3 rows in set (0.00 sec)
```

Query for right join would be as follows,

select fast.id,fast.city,slow.mob from fast right join slow on fast.name=slow.name;

And output for above mention query would be as follows,

```
mysql> select fast.id,fast.city,slow.mob from fast right join slow on fast.name=
slow.name;
+------+----------+------------+
| id   | city     | mob        |
+------+----------+------------+
|    1 | shegaon  | 9865321474 |
|    2 | khamgaon | 9965321474 |
|    3 | buldhana | 9765321474 |
+------+----------+------------+
3 rows in set (0.00 sec)

mysql>
```

4) FULL JOIN:

The SQL FULL JOIN combines the results of both left and right outer joins.
The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

Syntax:

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

Example: Consider table customers and demand as follows,

```
mysql> select *from customers;
+------+--------+------+----------+--------+
| id   | name   | age  | address  | salary |
+------+--------+------+----------+--------+
|    1 | ram    |   25 | shegaon  |  25766 |
|    2 | shyam  |   27 | akola    |  52766 |
|    3 | ganesh |   29 | buldhana |  87766 |
|    4 | akash  |   23 | pune     |  78766 |
|    5 | samir  |   24 | mumbai   |  99766 |
+------+--------+------+----------+--------+
5 rows in set (0.30 sec)

mysql> select *from demand;
+------+------+------+
| oid  | amt  | cid  |
+------+------+------+
|   10 | 2500 |    1 |
|   20 | 3500 |    2 |
|   30 | 4500 |    4 |
+------+------+------+
3 rows in set (0.05 sec)
```

Query for full join would be as follows,

select id,name,amt from customers left join demand on customers.id=demand.cid union all select id,name,amt from customers right join demand on customers.id=demand.cid;

Note: As MySQL does not support Full join so we used union all to execute query.

And output for above mention query would be as follows,

```
mysql> select id,name,amt from customers left join demand on customers.id=demand
.cid union all select id,name,amt from customers right join demand on customers.
id=demand.cid;
+------+--------+------+
| id   | name   | amt  |
+------+--------+------+
|    1 | ram    | 2500 |
|    2 | shyam  | 3500 |
|    3 | ganesh | NULL |
|    4 | akash  | 4500 |
|    5 | samir  | NULL |
|    1 | ram    | 2500 |
|    2 | shyam  | 3500 |
|    4 | akash  | 4500 |
+------+--------+------+
8 rows in set (0.09 sec)
```

5) SELF JOIN:

The SQL SELF JOIN is used to join a table to itself as if the table were two tables; temporarily renaming at least one table in the SQL statement.

Syntax:

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_field = b.common_field;
```

Example: CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
```

Now, let us join this table using SELF JOIN as follows,

```
SQL> SELECT a.ID, b.NAME, a.SALARY FROM CUSTOMERS a,CUSTOMERS b WHERE a.SALARY <
b.SALARY;
```


This would produce the following result,

```
+----+----------+---------+
| ID | NAME     | SALARY  |
+----+----------+---------+
|  2 | Ramesh   | 1500.00 |
|  2 | kaushik  | 1500.00 |
|  1 | Chaitali | 2000.00 |
|  2 | Chaitali | 1500.00 |
|  3 | Chaitali | 2000.00 |
|  6 | Chaitali | 4500.00 |
|  1 | Hardik   | 2000.00 |
|  2 | Hardik   | 1500.00 |
|  3 | Hardik   | 2000.00 |
|  4 | Hardik   | 6500.00 |
|  6 | Hardik   | 4500.00 |
|  1 | Komal    | 2000.00 |
|  2 | Komal    | 1500.00 |
|  3 | Komal    | 2000.00 |
|  1 | Muffy    | 2000.00 |
|  2 | Muffy    | 1500.00 |
|  3 | Muffy    | 2000.00 |
|  4 | Muffy    | 6500.00 |
|  5 | Muffy    | 8500.00 |
|  6 | Muffy    | 4500.00 |
+----+----------+---------+
```

6) CARTESIAN JOIN:

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from the two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to True or where the join-condition is absent from the statement.

Syntax:

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


Example: Consider table customers and demand as follows,

```
mysql> select *from customers;
+------+--------+------+----------+--------+
| id   | name   | age  | address  | salary |
+------+--------+------+----------+--------+
|    1 | ram    |   25 | shegaon  |  25766 |
|    2 | shyam  |   27 | akola    |  52766 |
|    3 | ganesh |   29 | buldhana |  87766 |
|    4 | akash  |   23 | pune     |  78766 |
|    5 | samir  |   24 | mumbai   |  99766 |
+------+--------+------+----------+--------+
5 rows in set (0.30 sec)

mysql> select *from demand;
+------+------+------+
| oid  | amt  | cid  |
+------+------+------+
|   10 | 2500 |    1 |
|   20 | 3500 |    2 |
|   30 | 4500 |    4 |
+------+------+------+
3 rows in set (0.05 sec)
```

Query for Cartesian product would be as follows,

Select id,name,amt from customers,demand:

And output for above mention query would be as follows,

```
mysql> select id,name,amt from customers,demand;
+------+--------+------+
| id   | name   | amt  |
+------+--------+------+
|    1 | ram    | 2500 |
|    1 | ram    | 3500 |
|    1 | ram    | 4500 |
|    2 | shyam  | 2500 |
|    2 | shyam  | 3500 |
|    2 | shyam  | 4500 |
|    3 | ganesh | 2500 |
|    3 | ganesh | 3500 |
|    3 | ganesh | 4500 |
|    4 | akash  | 2500 |
|    4 | akash  | 3500 |
|    4 | akash  | 4500 |
|    5 | samir  | 2500 |
|    5 | samir  | 3500 |
|    5 | samir  | 4500 |
+------+--------+------+
15 rows in set (0.01 sec)
```

Result: Thus we have studied and implemented various join operations by using SQL queries successfully.

Suggested questions:

1) What are various JOINS?
2) What is INNER JOIN?

3) Explain LEFT JOIN and RIGHT JOIN?
4) What is SELF JOIN?
5) Explain how FULL JOIN and CARTESIAN JOIN works?

Chapter No. 7

Aim: To study and implement VIEWS.

Learning outcome: At the end of this Chapter, students will be able to perform various operations of VIEWS by using CREATE, ALTER and DROP command.

Resources:

Software Required: MySQL/Oracle, Windows 7 or any other compatible operating system.

Hardware Required:

- Processor: Pentium dual core or any other similar or higher configuration recommended.
- Ram: Minimum 1 GB recommended.
- Hard-disk: 500 GB or higher recommended.

Theory:

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query. A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depend on the written SQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following: 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 see and (sometimes) modify exactly what they need and no more. Summarize data from various tables which can be used to generate reports.

(1) Creating Views:

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view.
To create a view, a user must have the appropriate system privilege according to the specific implementation.

Syntax:

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


You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

Example:
Consider the CUSTOMERS table having the following records:

```
mysql> select *from customers;
+------+---------+------+-----------+---------+
| id   | name    | age  | address   | salary  |
+------+---------+------+-----------+---------+
|    1 | ram     |   25 | shegaon   |   25766 |
|    2 | shyam   |   27 | akola     |   52766 |
|    3 | ganesh  |   29 | buldhana  |   87766 |
|    4 | akash   |   23 | pune      |   78766 |
|    5 | samir   |   24 | mumbai    |   99766 |
+------+---------+------+-----------+---------+
5 rows in set (0.05 sec)
```

Query for creating view is as follows,

create view customersview as select id,name,age from customers;

Now, you can query on customersview view in similar way as you query an actual table. Following is the example:

select *from customersview;

This would produce the following result:

```
mysql> create view customersview as select id,name,age from customers;
Query OK, 0 rows affected (0.03 sec)

mysql> select *from customersview;
+------+---------+------+
| id   | name    | age  |
+------+---------+------+
|    1 | ram     |   25 |
|    2 | shyam   |   27 |
|    3 | ganesh  |   29 |
|    4 | akash   |   23 |
|    5 | samir   |   24 |
+------+---------+------+
5 rows in set (0.00 sec)
```

2) Updating a View:

A view can be updated under certain conditions:

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.

- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

  Example:

  Consider the CUSTOMERS table having the following records:

```
mysql> select *from customers;
+------+--------+------+----------+--------+
| id   | name   | age  | address  | salary |
+------+--------+------+----------+--------+
|    1 | ram    |   25 | shegaon  |  25766 |
|    2 | shyam  |   27 | akola    |  52766 |
|    3 | ganesh |   29 | buldhana |  87766 |
|    4 | akash  |   23 | pune     |  78766 |
|    5 | samir  |   24 | mumbai   |  99766 |
+------+--------+------+----------+--------+
5 rows in set (0.05 sec)
```

  So if a view satisfies all the above mentioned rules then you can update a view. Following is an example to update the age of ram:

  update customersview set age=’35’ where id=’1’;

  This would ultimately update the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result:

```
mysql> update customersview set age='35' where id='1';
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select *from customers;
+------+--------+------+----------+--------+
| id   | name   | age  | address  | salary |
+------+--------+------+----------+--------+
|    1 | ram    |   35 | shegaon  |  25766 |
|    2 | shyam  |   27 | akola    |  52766 |
|    3 | ganesh |   29 | buldhana |  87766 |
|    4 | akash  |   23 | pune     |  78766 |
|    5 | samir  |   24 | mumbai   |  99766 |
+------+--------+------+----------+--------+
5 rows in set (0.00 sec)

mysql> select *from customersview;
+------+--------+------+
| id   | name   | age  |
+------+--------+------+
|    1 | ram    |   35 |
|    2 | shyam  |   27 |
|    3 | ganesh |   29 |
|    4 | akash  |   23 |
|    5 | samir  |   24 |
+------+--------+------+
5 rows in set (0.00 sec)
```


3) Inserting Rows into a View:

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here we cannot insert rows in CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in similar way as you insert them in a table.

Syntax:

insert into view_name values(col_name data-type);

Example:

Consider the customersview view having the following records:

```
mysql> select *from customersview;
+------+--------+------+
| id   | name   | age  |
+------+--------+------+
|    1 | ram    |   35 |
|    2 | shyam  |   27 |
|    3 | ganesh |   29 |
|    4 | akash  |   23 |
|    5 | samir  |   24 |
+------+--------+------+
5 rows in set (0.00 sec)
```

Query for inserting row into view is as follows.

insert into customersview values(6,'sachin',34);

and output for above query would be as follows,

```
mysql> insert into customersview values(6,'sachin',34);
Query OK, 1 row affected (0.09 sec)

mysql> select *from customersview;
+------+--------+------+
| id   | name   | age  |
+------+--------+------+
|    1 | ram    |   35 |
|    2 | shyam  |   27 |
|    3 | ganesh |   29 |
|    4 | akash  |   23 |
|    5 | samir  |   24 |
|    6 | sachin |   34 |
+------+--------+------+
6 rows in set (0.00 sec)
```

4) Deleting Rows into a View:

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Syntax: delete from view-name where [condition];

Example:
Consider the customersview view having the following records:

```
mysql> select *from customersview;
+------+--------+------+
| id   | name   | age  |
+------+--------+------+
|    1 | ram    |   35 |
|    2 | shyam  |   27 |
|    3 | ganesh |   29 |
|    4 | akash  |   23 |
|    5 | samir  |   24 |
+------+--------+------+
5 rows in set (0.00 sec)
```

Query for deleting row from view is as follows.

delete from customersview where id='6';

and output for above query would be as follows,

```
mysql> delete from customersview where id='6';
Query OK, 1 row affected (0.08 sec)

mysql> select *from customersview;
+------+--------+------+
| id   | name   | age  |
+------+--------+------+
|    1 | ram    |   35 |
|    2 | shyam  |   27 |
|    3 | ganesh |   29 |
|    4 | akash  |   23 |
|    5 | samir  |   24 |
+------+--------+------+
5 rows in set (0.00 sec)

mysql>
```

This would ultimately delete a row from the base table CUSTOMERS and same would reflect in the view itself.

5) Dropping Views:
Obviously, when you have a view, you need a way to drop the view if it is no longer needed.

Syntax: drop view view_name;

Example: to drop CUSTOMERS_VIEW from CUSTOMERS table:

drop view customersview;

Above query would drop entire customersview view.

Result: In this way, we have successfully implemented VIEWS by using various commands.

Suggested questions:

1) What is VIEW?
2) How to create VIEW?
3) How to insert record into VIEW?
4) What is the syntax to update VIEW?
5) How to drop VIEW?

Chapter No. 8

Aim: To study SQL queries using union, union all, intersect and minus clause/operator.

Learning outcome: At the end of this Chapter, students will be able to implement various SQL queries by using operators such as union, union all, intersect and minus clause/operator.

Resources:

Software Required: MySQL/Oracle, Windows 7 or any other compatible operating system.

Hardware Required:

- Processor: Pentium dual core or any other similar or higher configuration recommended.
- Ram: Minimum 1 GB recommended.
- Hard-disk: 500 GB or higher recommended.

Theory:

1) UNION:

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.
To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order, but they do not have to be the same length.

Syntax: select col1,col2 from table_name1 left join table_name2 on table1.col=table2.col union select col1,col2 from table_name1 right join table_name2 on table1.col=table2.col;

Example:
Consider the following two tables CUSTOMERS and DEMAND table as follows:

```
mysql> select *from customers;
+------+--------+------+----------+--------+
| id   | name   | age  | address  | salary |
+------+--------+------+----------+--------+
|    1 | ram    |   35 | shegaon  |  25766 |
|    2 | shyam  |   27 | akola    |  52766 |
|    3 | ganesh |   29 | buldhana |  87766 |
|    4 | akash  |   23 | pune     |  78766 |
|    5 | samir  |   24 | mumbai   |  99766 |
+------+--------+------+----------+--------+
5 rows in set (0.00 sec)

mysql> select *from demand;
+------+------+------+
| oid  | amt  | cid  |
+------+------+------+
|   10 | 2500 |    1 |
|   20 | 3500 |    2 |
|   30 | 4500 |    4 |
+------+------+------+
3 rows in set (0.06 sec)
```

Now, let us join these two tables in our SELECT statement as follows:
Query to use union clause/operator is as follows,

select id,name,amt from customers left join demand on customers.id=demand.cid union select id,name,amt from customers right join demand on customers.id=demand.cid;

And output for above query would be as follows;

```
mysql> select id,name,amt from customers left join demand on customers.id=demand
.cid union select id,name,amt from customers right join demand on customers.id=d
emand.cid;
+------+--------+------+
| id   | name   | amt  |
+------+--------+------+
|    1 | ram    | 2500 |
|    2 | shyam  | 3500 |
|    3 | ganesh | NULL |
|    4 | akash  | 4500 |
|    5 | samir  | NULL |
+------+--------+------+
5 rows in set (0.05 sec)
```

2) UNION ALL:

This operator is used to combine the results of two SELECT statements including duplicate rows.
The same rules that apply to UNION apply to the UNION ALL operator.

Syntax:
Syntax: select col1,col2 from table_name1 left join table_name2 on table1.col=table2.col union all select col1,col2 from table_name1 right join table_name2 on table1.col=table2.col;

Example:
Consider the following two tables CUSTOMERS and DEMAND table as follows:

```
mysql> select *from customers;
+------+--------+------+----------+--------+
| id   | name   | age  | address  | salary |
+------+--------+------+----------+--------+
|    1 | ram    |   35 | shegaon  |  25766 |
|    2 | shyam  |   27 | akola    |  52766 |
|    3 | ganesh |   29 | buldhana |  87766 |
|    4 | akash  |   23 | pune     |  78766 |
|    5 | samir  |   24 | mumbai   |  99766 |
+------+--------+------+----------+--------+
5 rows in set (0.00 sec)

mysql> select *from demand;
+------+------+------+
| oid  | amt  | cid  |
+------+------+------+
|   10 | 2500 |    1 |
|   20 | 3500 |    2 |
|   30 | 4500 |    4 |
+------+------+------+
3 rows in set (0.06 sec)
```

Now, let us join these two tables in our SELECT statement as follows:
Query to use union all clause/operator is as follows,

select id,name,amt from customers left join demand on customers.id=demand.cid union all select id,name,amt from customers right join demand on customers.id=demand.cid;

And output for above query would be as follows;

```
mysql> select id,name,amt from customers left join demand on customers.id=demand
.cid union all select id,name,amt from customers right join demand on customers.
id=demand.cid;
+------+--------+------+
| id   | name   | amt  |
+------+--------+------+
|    1 | ram    | 2500 |
|    2 | shyam  | 3500 |
|    3 | ganesh | NULL |
|    4 | akash  | 4500 |
|    5 | samir  | NULL |
|    1 | ram    | 2500 |
|    2 | shyam  | 3500 |
|    4 | akash  | 4500 |
+------+--------+------+
8 rows in set (0.00 sec)

mysql>
```


3) INTERSECT:

The SQL INTERSECT clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements.
Just as with the UNION operator, the same rules apply when using the INTERSECT operator. As MySQL does not support INTERSECT operator, we will simulate this concept by using distinct and inner join.

Syntax: By using distinct and inner join

select distinct column_name from table_name1 inner join table_name2 using (column_name);

Example:
Consider the following two tables CUSTOMER and DEMANDS table as follows:

```
mysql> select *from customer;
+------+--------+---------+------+---------+
| id   | name   | address | age  | salary  |
+------+--------+---------+------+---------+
|    1 | sachin | pune    |   25 | 2544.30 |
|    2 | akash  | mumbai  |   27 | 3544.30 |
|    3 | sameer | delhi   |   29 | 5544.30 |
+------+--------+---------+------+---------+
3 rows in set (0.03 sec)

mysql> select *from demands;
+------+---------+------+
| oid  | amount  | id   |
+------+---------+------+
|   10 | 5644.20 |    1 |
|   20 | 7644.20 |    3 |
|   30 | 4644.20 |    4 |
+------+---------+------+
3 rows in set (0.00 sec)
```

Now, let us join these two tables in our SELECT statement as follows:
Query to use intersect clause/operator is as follows,

select distinct id from customer inner join demands using (id) ;

And output for above query would be as follows;

```
mysql> select distinct id from customer inner join demands using (id);
+------+
| id   |
+------+
|    1 |
|    3 |
+------+
2 rows in set (0.03 sec)
```

4) MINUS:

The Minus Operator in SQL is used with two SELECT statements. The MINUS operator is used to subtract the result set obtained by first SELECT query from the result set obtained by second SELECT query. In simple words, we can say that MINUS operator will return only those rows which are unique in only first SELECT query and not those rows which are common to both first and second SELECT queries. As MySQL does not support MINUS operator, we will simulate this concept by using left join.

Syntax: By using left join

select column_name from table_name1 left join table_name2 using (id);

Example:

Consider the following two tables CUSTOMER and DEMANDS table as follows:

```
mysql> select *from customer;
+------+--------+---------+------+---------+
| id   | name   | address | age  | salary  |
+------+--------+---------+------+---------+
|    1 | sachin | pune    |   25 | 2544.30 |
|    2 | akash  | mumbai  |   27 | 3544.30 |
|    3 | sameer | delhi   |   29 | 5544.30 |
+------+--------+---------+------+---------+
3 rows in set (0.03 sec)

mysql> select *from demands;
+------+---------+------+
| oid  | amount  | id   |
+------+---------+------+
|   10 | 5644.20 |    1 |
|   20 | 7644.20 |    3 |
|   30 | 4644.20 |    4 |
+------+---------+------+
3 rows in set (0.00 sec)
```

Now, let us join these two tables in our SELECT statement as follows:

Query to use minus clause/operator is as follows,

select id from customer left join demand using (id);

And output for above query would be as follows;

```
mysql> select id from customer left join demands using (id);
+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.52 sec)
```

Result: Thus, we have executed SQL queries using union, union all, intersect and minus operator/ clause successfully.

Suggested questions:

1) What is UNION clause/operator?
2) Explain working of UNION ALL clause/operator?
3) Explain working of INTERSECT and MINUS clause/operator?

Chapter No. 9

Aim: To study integrity constraints.

Learning outcome: At the end of this Chapter, students will be able illustrate various integrity constraints such as NOT NULL, DEFAULT, UNIQUE, PRIMARY KEY, FOREIGN KEY and CHECK constraint.

Resources:

Software Required: MySQL/Oracle, Windows 7 or any other compatible operating system.

Hardware Required:

- Processor: Pentium dual core or any other similar or higher configuration recommended.
- Ram: Minimum 1 GB recommended.
- Hard-disk: 500 GB or higher recommended.

Theory:

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database. Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL.

- NOT NULL Constraint: Ensures that a column cannot have NULL value.
- DEFAULT Constraint: Provides a default value for a column when none is specified.
- UNIQUE Constraint: Ensures that all values in a column are different.
- PRIMARY Key: Uniquely identified each rows/records in a database table.
- FOREIGN Key: Uniquely identified a rows/records in any another database table.
- CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.

1) NOT NULL:

By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such constraint on this column specifying that NULL is now not allowed for that column.

A NULL is not the same as no data, rather, it represents unknown data.

Example:

The following SQL creates a new table called CUSTOMERS and adds five columns, three of which, ID and NAME and AGE, specify not to accept NULLs:

```
CREATE TABLE CUSTOMERS(
                        ID INT(2)NOT NULL,
                        NAME VARCHAR(20)NOT NULL,
                        AGE INT(2) NOT NULL,
                        ADDRESS CHAR25),
                        SALARY DECIMAL(18,2)
                       );
```

2) DEFAULT:

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

Example:

Above SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 7000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 7000.00.

```
CREATE TABLE CUSTOMERS(
                        ID INT(2)NOT NULL,
                        NAME VARCHAR(20)NOT NULL,
                        AGE INT(2) NOT NULL,
                        ADDRESS CHAR25),
                        SALARY DECIMAL(18,2) DEFAULT 7000.00
                       );
```

3) UNIQUE:

Constraint prevents two records from having identical values in a particular column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having identical age.

Example:

The following SQL creates a new table called CUSTOMERS and adds five columns. Here, AGE column is set to UNIQUE, so that you cannot have two records with same age:

```
CREATE TABLE CUSTOMERS(
                        ID INT(2) NOT NULL,
                        NAME VARCHAR(20) NOT NULL,
                        AGE INT(2)NOT NULL UNIQUE,
                        ADDRESS CHAR(25),
                        SALARY DECIMAL(18,2)
                        );
```

If CUSTOMERS table has already been created, then to add a UNIQUE constraint to AGE column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS MODIFY AGE INT(2) NOT NULL UNIQUE;
```

You can also use following syntax, which supports naming the constraint in multiple columns as well:

```
ALTER TABLE CUSTOMERS ADD CONSTRAINT myUniqueConstraintUNIQUE(AGE, SALARY);
```

4) PRIMARY KEY:

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary key must contain unique values. A primary key column cannot have NULL values.
A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.
If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).
Note: You would use these concepts while creating database tables.
Here is the syntax to define ID attribute as a primary key in a CUSTOMERS table.

```
CREATE TABLE CUSTOMERS(
                        ID INT(2) NOT NULL,
                        NAME VARCHAR(20) NOT NULL,
                        AGE INT(2) NOT NULL,
                        ADDRESS CHAR(25),
                        SALARY DECIMAL (18,2),
                        PRIMARY KEY(ID)
                        );
```

To create a PRIMARY KEY constraint on the "ID" column when CUSTOMERS table already exists, use the following SQL syntax:

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
```


Note: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

For defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE CUSTOMERS(
                        ID INT(2) NOT NULL,
                        NAME VARCHAR(20) NOT NULL,
                        AGE INT(2) NOT NULL,
                        ADDRESS CHAR(25),
                        SALARY DECIMAL (18,2),
                        PRIMARY KEY (ID, NAME)
                        );
```

5) FOREIGNE KEY:

A foreign key is a key used to link two tables together. This is sometimes called a referencing key. Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
The relationship between two tables matches the Primary Key in one of the tables with a Foreign Key in the second table.
If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).
Example:
Consider the structure of the two tables as follows:
CUSTOMERS table:

```
CREATE TABLE CUSTOMERS(
                        ID INT(2) NOT NULL,
                        NAME VARCHAR(20) NOT NULL,
                        AGE INT(2) NOT NULL,
                        ADDRESS CHAR(25),
                        SALARY DECIMAL(18,2),
                        PRIMARY KEY (ID)
                        );
```

ORDERS table:

```
CREATE TABLE ORDERS (
                      ID INT(2) NOT NULL,
                      DATE DATETIME,
                      CUSTOMER_ID INT references CUSTOMERS(ID),
                      PRIMARY KEY (ID)
                      );
```


If ORDERS table has already been created, and the foreign key has not yet been set, use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE ORDERS ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);
```

6) CHECK:

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered into the table.

Example:
For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER below 18 years:

```
CREATE TABLE CUSTOMERS(
                        ID INT(2) NOT NULL,
                        NAME VARCHAR(20) NOT NULL,
                        AGE INT(2) NOT NULL CHECK (AGE >=18),
                        ADDRESS CHAR(25),
                        SALARY   DECIMAL (18,2)
                       );
```

If CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS MODIFY AGE INT NOT NULL CHECK (AGE >=18);
```

Result: In this way, we have studied various integrity constraints successfully.

Suggested questions:

1) What are various integrity constraints?
2) Explain NOT NULL and DEFAULT constraints?
3) What is UNIQUE constraint?
4) Explain PRIMARY KEY and FOREIGN KEY?
5) What is CHECK constraint?