



ANJUMAN

COLLEGE OF ENGINEERING & TECHNOLOGY

(MANAGED BY : ANJUMAN HAMI-E-ISLAM, NAGPUR)

Database Management System Lab Manual



Computer Science & Engineering Department

Roll No: _____

Name: _____

Sem: _____ Section _____



ANJUMAN COLLEGE OF ENGINEERING & TECHNOLOGY

ESTD. 1999

Approved by A.I.C.T.E. New Delhi, Recognized by DTE, Mumbai, Affiliated to RTM Nagpur University, Nagpur.

CERTIFICATE

Certified that this file is submitted by

Shri/Ku. _____

Roll No. _____ a student of _____ year of the course _____

_____ as a part of PRACTICAL/ORAL as

prescribed by the Rashtrasant Tukadoji Maharaj Nagpur University for the

subject _____ in the laboratory of

_____ during the academic year

_____ and that I have instructed him/her for the said work,

from time to time and I found him/her to be satisfactory progressive.

And that I have accessed the said work and I am satisfied that the same is up to that

standard envisaged for the course.

Date:-

Signature & Name
Of Subject Teacher

Signature & Name
of HOD

Anjuman College of Engineering and Technology

Vision


- To be a centre of excellence for developing quality technocrats with moral and social ethics, to face the global challenges for the sustainable development of society.

Mission

- To create conducive academic culture for learning and identifying career goals.
- To provide quality technical education, research opportunities and imbibe entrepreneurship skills contributing to the socio-economic growth of the Nation.
- To inculcate values and skills, that will empower our students towards development through technology.

Vision and Mission of the Department

Vision:

- 
- To achieve excellent standards of quality education in the field of computer science and engineering, aiming towards development of ethically strong technical experts contributing to the profession in the global society.


Mission:

- To create outcome based education environment for learning and identifying career goals.
- Provide latest tools in a learning ambience to enhance innovations, problem solving skills, leadership qualities team spirit and ethical responsibilities.
- Inculcating awareness through innovative activities in the emerging areas of technology.

Program Educational Objectives (PEOs)

- The graduates will have a strong foundation in mathematical, scientific and engineering fundamentals necessary to formulate, solve and analyze engineering problem in their career.
- Graduates will be able to create and design computer support systems and impart knowledge and skills to analyze, design, test and implement various software applications.
- Graduates will work productively as computer science engineers towards betterment of society exhibiting ethical qualities.

Program Specific Outcomes (PSOs)

- 
- Foundation of mathematical concepts: To use mathematical methodologies and techniques for computing and solving problem using suitable mathematical analysis, data structures, database and algorithms as per the requirement.
 - Foundation of Computer System: The capability and ability to interpret and understand the fundamental concepts and methodology of computer systems and programming. Students can understand the functionality of hardware and software aspects of computer systems, networks and security.
 - Foundations of Software development: The ability to grasp the software development lifecycle and methodologies of software system and project development.

PROGRAM: CSE	DEGREE: B.E
COURSE: Database Management System	SEMESTER: IV CREDITS: 3
COURSE CODE: BECSE403T	COURSE TYPE: REGULAR
COURSE AREA/DOMAIN: Practical knowledge about SQL	CONTACT HOURS: 2 hours/Week.
CORRESPONDING LAB COURSE CODE : BECSE403P	LAB COURSE NAME : Database Management System Lab

COURSE PRE-REQUISITES:

C.CODE	COURSE NAME	DESCRIPTION	SEM
		Basic concept of file processing and fundamentals of operating systems.	

LAB COURSE OBJECTIVES:

- Students learn how to design and create a good database and use various SQL operations.
- To learn the fundamental concepts of SQL queries.
- To understand the concept of designing a database with the necessary attributes.
- To know the methodology of Accessing, Modifying and Updating data & information from the relational databases.

COURSE OUTCOMES: Database Management System

After completion of this course the students will be able -

SNO	DESCRIPTION	BLOOM'S TAXONOMY LEVEL
CO.1	Transform an information model into a relational database schema and to use a data definition language and/or utilities to implement the schema using a DBMS.	LEVEL 5
CO.2	Use an SQL interface of a multi-user relational DBMS package to create, secure, populate, maintain, and query a database.	LEVEL 3
CO.3	Formulate query, using SQL, solutions to a broad range of query and data update problems.	LEVEL 5
CO.4	Use a desktop database package to create, populate, maintain, and query a database.	LEVEL 3
CO.5	Demonstrate a rudimentary understanding of programmatic interfaces to a database and be able to use the basic functions of one such interface.	LEVEL 3
CO.6	Analyze an information storage problem and derive an information model expressed in the form.	LEVEL 4

Lab Instructions:

- Make entry in the Log Book as soon as you enter the Laboratory.
- All the students should sit according to their Roll Numbers.
- All the students are supposed to enter the terminal number in the Log Book.
- Do not change the terminal on which you are working.
- Strictly observe the instructions given by the Faculty / Lab. Instructor.
- Take permission before entering in the lab and keep your belongings in the racks.
- NO FOOD, DRINK, IN ANY FORM is allowed in the lab.
- TURN OFF CELL PHONES! If you need to use it, please keep it in bags.
- Avoid all horseplay in the laboratory. Do not misbehave in the computer laboratory. Work quietly.
- Save often and keep your files organized.
- Don't change settings and surf safely.
- Do not reboot, turn off, or move any workstation or PC.
- Do not load any software on any lab computer (without prior permission of Faculty and Technical Support Personnel). Only Lab Operators and Technical Support Personnel are authorized to carry out these tasks.
- Do not reconfigure the cabling/equipment without prior permission.
- Do not play games on systems.
- Turn off the machine once you are done using it.

- Violation of the above rules and etiquette guidelines will result in disciplinary action.

DBMS

Continuous Assessment Practical

Exp No	NAME OF EXPERIMENT	Date	Sign	Remark
1	Implement Data Definition Language (Create, Alter, Drop, Truncate, and Rename).			
2	Implement Data Manipulation Language (Insert, Update, and Delete).			
3	Implement SELECT command with different clauses (Where clause, having clause, Group by clause, Order by clause).			
4	Implement Single Row function (character, numeric, data functions).			
5	To implement Group function (AVG, MIN, MAX, SUM).			
6	Implement various types of SET operators (Union, Intersect, Minus).			
7	Implement various types of integrity constraints (NOT NULL Constraint, DEFAULT Constraint, UNIQUE Constraint, PRIMARY Key, FOREIGN Key, CHECK Constraint).			
8	Implement various types of joins (Left Join, Right Join, Outer Join, and Inner Join).			
9				
10				

CONTENTS

Exp No	NAME OF EXPERIMENT	PAGE NO.
1	Implement Data Definition Language (Create, Alter, Drop, Truncate, and Rename).	8
2	Implement Data Manipulation Language (Insert, Update, and Delete).	17
3	Implement SELECT command with different clauses (Where clause, having clause, Group by clause, Order by clause).	24
4	Implement Single Row function (character, numeric, data functions).	32
5	To implement Group function (AVG, MIN, MAX, SUM).	41
6	Implement various types of SET operators (Union, Intersect, Minus).	45
7	Implement various types of integrity constraints (NOT NULL Constraint, DEFAULT Constraint, UNIQUE Constraint, PRIMARY Key, FOREIGN Key, CHECK Constraint).	55
8	Implement various types of joins (Left Join, Right Join, Outer Join, and Inner Join).	66
9		74
10		90

EXPERIMENT NO – 1

Aim: Implement Data Definition Language (Create, Alter, Drop, Truncate, and Rename) & Data Manipulation Language (Insert, Update, and Delete).

Theory:

DDL-(Data Definition Language)

The data definition language is used to create an object, alter the structure of an object and also drop already created object. The Data Definition Languages used for table definition can be classified into following:

- CREATE TABLE - to create objects in the database
- ALTER TABLE - alters the structure of the database
- DROP TABLE- delete objects from the database
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

Before creating table follow these steps to create table in MySql.

```
mysql => mysql -u root -p
        show databases;
        create database sectionB34;
        use sectionB34;
```

CREATE TABLE

It defines each column of the table uniquely. Each column has minimum of three attributes, a name, data type and size.

Here, a few items need explanation:

- Field Attribute NOT NULL is being used because we do not want this field to be NULL. So, if a user will try to create a record with a NULL value, then MySQL will raise an error.

- Field Attribute AUTO_INCREMENT tells MySQL to go ahead and add the next available number to the id field.
- Keyword PRIMARY KEY is used to define a column as a primary key. You can use multiple columns separated by a comma to define a primary key.

Syntax:

Create table <table name> (<col1> <datatype>(<size>),<col2> <datatype>(<size>));

Ex:

create table stud(RollNo int, Name varchar(20), DOB date);

SHOW TABLES

SHOW TABLES lists the non-TEMPORARY tables in a given database.

Syntax: Show Tables

DESCRIBE TABLE: The DESCRIBE and EXPLAIN statements are synonyms, used either to obtain information about table structure or query execution plans.

Syntax: DESCRIBE <TABLE NAME>;

Ex: describe stud;

MODIFYING THE STRUCTURE OF TABLES USING ALTER TABLE STATEMENT

ALTER is used to modify existing database data structures (database, table).

a) Add new columns

Syntax: Alter table <tablename> add(<new col><datatype(size)>,<new col>datatype(size));

Ex: alter table stud add enno varchar(20);
alter table stud add email varchar(20) after name;

b) Dropping a column from a table

Syntax: Alter table <tablename> drop <column-name>;

Ex: alter table stud drop email;

c) Modifying existing columns.

Syntax: Alter table <tablename> modify (<column name><newdatatype>(<newsized>));

Ex: alter table stud modify email varchar(30);

d) Renaming the tables

RENAME command is used to rename SQL table

Syntax: Alter table <old table name> RENAME to <new table name>;

Ex: Alter table stud rename to student;

Destroying tables

DROP statement allows you to remove database, table, index or stored procedure.

Syntax: Drop <table name>;

Ex: drop stud;

Conclusion: Thus we, have studied Data Definition Language (Create, Alter, Drop, Truncate, and Rename) Successfully.

Viva Voce Question

1. Differentiate between 'TRUNCATE' and 'DROP' commands.

2. Why we ALTER command in SQL?

3. What is the difference between DBMS and RDBMS?

Signature of Subject Teacher

DBMS

EXPERIMENT NO - 2

Aim: Implement Data Manipulation Language (Insert, Update, and Delete).

Theory:

DML- (Data Manipulation Language)

DML- Data Manipulation Language (DML) statements are used for managing data within schema objects. DML deals with data manipulation, and therefore includes most common SQL statements such as SELECT, INSERT, etc. DML allows adding / modifying / deleting data itself.

DML is used to manipulate with the existing data in the database objects (insert, select, update, delete).

DML Commands:

- INSERT
- SELECT
- UPDATE
- DELETE

INSERT INTO:

To insert data into a MySQL table, you would need to use the SQL INSERT INTO command. To insert string data types, it is required to keep all the values into double or single quotes. For example - "value".

Syntax: INSERT INTO table_name (field1, field2,.....fieldN) VALUES (value1, value2,...valueN)

EX: insert into stud (RollNo, Name, DOB, enno) values (5,'Naaz','1998-01-10',115);

SELECT:

The SQL SELECT command is used to fetch data from the MySQL database.

- You can use one or more tables separated by comma to include various conditions using a WHERE clause, but the WHERE clause is an optional part of the SELECT command.
- You can fetch one or more fields in a single SELECT command.
- You can specify star (*) in place of fields. In this case, SELECT will return all the fields.
- You can specify any condition using the WHERE clause.

Syntax: SELECT field1, field2,...fieldN table_name1, table_name2...
[WHERE Clause]

EX: select * from stud;

UPDATE:

There may be a requirement where the existing data in a MySQL table needs to be modified. You can do so by using the SQL UPDATE command. This will modify any field value of any MySQL table.

- You can update one or more field altogether.
- You can specify any condition using the WHERE clause.
- You can update the values in a single table at a time.
- The WHERE clause is very useful when you want to update the selected rows in a table.

Syntax: UPDATE table_name SET field1=new-value1, field2=new-value2 [WHERE Clause]

EX: update stud set Name='Aisha' where RollNo='2';

DELETE:

- If you want to delete a record from any MySQL table, then you can use the SQL command DELETE FROM.

- If the WHERE clause is not specified, then all the records will be deleted from the given MySQL table.
- You can specify any condition using the WHERE clause.
- You can delete records in a single table at a time.
- The WHERE clause is very useful when you want to delete selected rows in a table

Syntax: DELETE FROM table_name [WHERE Clause]

EX: Delete from stud where Rollno=10;

Conclusion: Thus we, have studied Data Definition Language (Create, Alter, Drop, Truncate, and Rename) & Data Manipulation Language (Insert, Update, and Delete) Successfully.

Viva Voce Question

4. What is SQL? What are Tables in SQL?

5. Differentiate between 'DELETE', 'TRUNCATE' and 'DROP' commands.

6. What are different types of statements supported by SQL?

Signature of Subject Teacher

EXPERIMENT NO - 3

Aim: Implement SELECT command with different clauses (Where clause, having clause, Group by clause, Order by clause).

Theory:

SELECT: The SQL SELECT command is used to fetch data from the MySQL database.

- You can use one or more tables separated by comma to include various conditions using a WHERE clause, but the WHERE clause is an optional part of the SELECT command.
- You can fetch one or more fields in a single SELECT command.
- You can specify star (*) in place of fields. In this case, SELECT will return all the fields.
- You can specify any condition using the WHERE clause.

Syntax: SELECT field1, field2,...fieldN table_name1, table_name2... [WHERE Clause]

EX: select * from stud;

SELECT STATEMENT WITH WHERE CLAUSE

We have seen the SQL SELECT command to fetch data from a MySQL table. We can use a conditional clause called the WHERE Clause to filter out the results. Using this WHERE clause, we can specify a selection criteria to select the required records from a table.

Syntax: SELECT field1, field2,...fieldN table_name1, table_name2... [WHERE condition1 [AND [OR]] condition2.....

EX: Select name, age from stud where rollno=2;

- You can use one or more tables separated by a comma to include various conditions using a WHERE clause, but the WHERE clause is an optional part of the SELECT command.
- You can specify any condition using the WHERE clause.
- You can specify more than one condition using the AND or the OR operators.
- A WHERE clause can be used along with DELETE or UPDATE SQL command also to specify a condition.

The **WHERE** clause works like an **if condition** in any programming language. This clause is used to compare the given value with the field value available in a MySQL table. If the given value from outside is equal to the available field value in the MySQL table, then it returns that row.

Here is the list of operators, which can be used with the **WHERE** clause.

Assume field A holds 10 and field B holds 20, then -

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

>=	Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.	(A <= B) is true.

BETWEEN OPERATOR

Ex: select * from emp_master where salary between 5000 and 8000;

The above select statement will display only those rows where salary of employee is between 5000 and 8000.

IN Operator:

The in operator can be used to select rows that match one of the values in a list.

EX: Select * from emp where deptno in (10, 30);

The above query will retrieve only those rows where deptno is either in 10 or 30.

Logical Operators:

Logical operators are used to combine the results of two conditions to produce a single result. The logical operators are AND, NOT and OR.

AND Operator: The Oracle engine will process all rows in a table and display the result only when all the conditions specified using the AND operator are satisfied.

EX: select * from emp_master where salary > 5000 and comm < 750;

The select statement will return only those rows where salary is greater than 5000 and comm is less than 750. If both the conditions are true then only it will retrieve rows.

OR Operator:

The Oracle engine will process all rows in a table and display the result only when any of the conditions specified using the OR operators are satisfied.

EX: select * from emp_master where salary > 5000 or comm < 750;

This select statement will check either salary is greater than 5000 or comm is less than 750. I.e. it will return all the records either of any one condition returns true.

NOT Operator:

The Oracle engine will process all rows in a table and display the result only when none of the conditions specified using the NOT operator are satisfied.

EX: select * from emp_master where not salary = 10000;

This select statement will return all the records where salary is NOT equal to 10000.

LIKE Operator:

Like operator is used to search character pattern, we need not know the exact character value. The like operator is used with special character % and _ (underscore).

Syntax: SELECT field1, field2,...fieldN table_name1, table_name2... WHERE field1 LIKE condition1 [AND [OR]] field2 = 'somevalue'

- You can specify any condition using the WHERE clause.
- You can use the LIKE clause along with the WHERE clause.

- You can use the LIKE clause in place of the equal to sign.
- When LIKE is used along with % sign then it will work like a Meta character search.
- You can specify more than one condition using AND or OR operators.
- A WHERE...LIKE clause can be used along with DELETE or UPDATE SQL command also to specify a condition.

EX: select * from emp_master where job like 'M%';

The above select statement will display only those rows where job is starts with 'M' followed by any number of any characters. % sign is used to refer number of characters (it similar to * asterisk wildcard in DOS), while _ (underscore) is used to refer single character.

EX: Select * from emp_master where job like '_lerk';

In above query, it will display only those rows where job is start with any single character but ends with 'clerk'.

DISTINCT CLAUSE

To prevent the selection of distinct rows, we can include distinct clause with select command. The following command will exclude duplicate empno.

EX: select distinct deptno from emp_master;

Order by Clause

Order by clause is used to arrange rows in either ascending or descending order. The order by clause can also be used to arrange multiple columns.

Syntax: SELECT field1, field2,...fieldN table_name1, table_name2... ORDER BY field1, [field2...] [ASC [DESC]]

- You can sort the returned result on any field, if that field is being listed out.
- You can sort the result on more than one field.
- You can use the keyword ASC or DESC to get result in ascending or descending order. By default, it's the ascending order.
- You can use the WHERE...LIKE clause in the usual way to put a condition.

If you want to view salary in ascending order the following command can be performed:

EX: Select empno,ename,salary from emp_master order by salary;

If you have not specified any order by default it will consider ascending order and salary will be displayed in ascending order. To retrieve data in descending order the desc keyword is used after order by clause.

EX: Select empno,ename,salary from emp_master order by salary desc;

Group by Clause

Group by clause is used with group functions only. Normally group functions return only one row. But group by clause will group on that column. The group by clause tells Oracle to group rows based on distinct values for specified columns, i.e. it creates a data set, containing several sets of records grouped together based on a condition.

Syntax: SELECT statements... GROUP BY column_name1[,column_name2,...] ;

EX: select deptno,count(*) from emp_master group by deptno;

Having Clause

The having clause is used to satisfy certain conditions on rows, retrieved by using group by clause. Having clause should be proceeding by a group by clause. Having clause further filters the rows returned by group by clause.

Syntax: SELECT statements... GROUP BY column_name1[,column_name2,...]
[HAVING condition];

EX: select deptno,count(*) from emp_master group by deptno having Deptno =1;

Conclusion: Thus we, have Implement SELECT command with different clauses
(Where clause, having clause, Group by clause, Order by clause) successfully.

Viva Voce Question

1. Difference between having and Group by clause?

2. Explain LIKE operator with example.

3. What is the use of distinct operator?

Signature of Subject Teacher

EXPERIMENT NO - 4

Aim: Implement Single Row function (character, numeric, date functions).

Theory:

Single Row Functions (Scalar Functions):

Functions that act on only one value at a time are called as Single Row Functions. A Single Row function returns one result for every row of a queried table or view. Single row functions can be character functions, numeric functions, date functions, and conversion functions. Note that these functions are used to manipulate data items. These functions require one or more input arguments and operate on each row, thereby returning one output value for each row. Argument can be a column, literal or an expression. Single row functions can be used in SELECT statement, WHERE and ORDER BY clause.

Single Row functions can be further grouped together by the data type of their arguments and return values. Functions can be classified corresponding to different data types as:

- String Functions: Work for String Data type
- Numeric Functions: Work for number Data type
- Conversion Functions: Work for conversion of one data type to another
- Date Functions: Work for Date Data type

Case Conversion functions - Accepts character input and returns a character value. Functions under the category are UPPER and LOWER.

- UPPER function converts a string to upper case.
- LOWER function converts a string to lower case.

String functions - Accepts character input and returns number or character value. Functions under the category are CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD and TRIM.

- CONCAT function concatenates two string values.
- LENGTH function returns the length of the input string.
- SUBSTR function returns a portion of a string from a given start point to an end point.
- LPAD and RPAD functions pad the given string upto a specific length with a given character.
- TRIM function trims the string input from the start or end.

Date functions - Date arithmetic operations return date or numeric values. Functions under the category are MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND and TRUNC.

- MONTHS_BETWEEN function returns the count of months between the two dates.
- ADD_MONTHS function adds 'n' number of months to an input date.
- NEXT_DAY function returns the next day of the date specified.
- LAST_DAY function returns last day of the month of the input date.
- ROUND and TRUNC functions are used to round and truncate the date value.

Number functions - Accepts numeric input and returns numeric values. Functions under the category are ROUND, TRUNC, and MOD.

- ROUND and TRUNC functions are used to round and truncate the number value.
- MOD is used to return the remainder of the division operation between two numbers.

“The Dual Table”

Dual is a small MySql worktable, which consists of only one row and one column, and contains the value x in that column. Besides arithmetic calculations, it also supports date retrieval and it's formatting.

```
SQL> select 2*2 from dual;
```

```
2*2
```

```
4
```

String Functions:

String functions accept string input and return either string or number values.

1. **ASCII:** Returns the numeric value of the leftmost character of the string str. Returns 0 if str is the empty string. Returns NULL if str is NULL. ASCII () works for characters with numeric values from 0 to 255.

Syntax: ASCII(string);

Example: SELECT ASCII('2') from dual;

Output: 50

Syntax: SELECT ASCII('dx') from dual;

Output: 100

2. **BIN:** Returns a string representation of the binary value of N, where N is a long long (BIGINT) number. This is equivalent to CONV (N, 10, 2). Returns NULL if N is NULL.

Syntax: BIN (Number)

Example: SELECT BIN(12) from dual;

Output: 1100

3. Lower: This String function will convert input string in to lower case.

Syntax: Lower (string)

Example: select lower ('AZURE') from dual;

Output: azure

4. Upper: This string function will convert input string in to upper case.

Syntax: Upper (string)

Example: select upper ('azure') from dual;

Output: AZURE

5. LTrim (Left trim): Ltrim function accepts two string parameters; it will fetch only those set of characters from the first string from the left side of the first string, and displays only those characters which are not present in second string. If same set of characters are not found in first string it will display whole string

Syntax: Ltrim(string,set)

Example: select ltrim('azuretech','azure') from dual;

Output: tech

6. Rtrim (Right Trim): Rtrim function accepts two string parameters; it will fetch only those characters from the first string, which is present in set of characters in second string from the right side of the first string.

Syntax: Rtrim(string,set)

Example: select rtrim('azuretrim','trim') from dual;

Output: azure

7. Substr: Substring fetches out a piece of the string beginning at start and going for count characters, if count is not specified, the string is fetched from start and goes till end of the string.

Syntax: Substr(string, starts [, count])

Example: select substr('azuretechnology',4,6) from dual;

Output: retech

8. Chr: Character function except character input and returns either character or number values. The first among character function is chr. This returns the character value of given number within braces.

Syntax: Chr(number)
Example: select chr(65) from dual;
Output: A

9. Lpad (Left Pad): This function takes three arguments. The first argument is character string, which has to be displayed with the left padding. Second is a number, which indicates total length of return value and third is the string with which left padding has to be done when required.

Syntax: Lpad(String,length,pattern)
Example: select lpad('Welcome',15,'*') from dual;
Output: *****Welcome

10.Rpad (Right Pad): Rpad does exact opposite then Lpad function.

Syntax: Lpad(String,length,pattern)
Example: select rpad('Welcome',15,'*') from dual;
Output: Welcome*****

11.Length: When the length function is used in a query. It returns length of the input string.

Syntax: Length(string)
Example: select length('auzre') from dual;
Output: 5

12.Concatenation (||) Operator: This operator is used to merge two or more strings.

Syntax: Concat(string1,string2)
Example: select concat('Azure',' Technology') from dual;
Output: Azure Technology
Example: select 'ename is '||ename from emp_master;
Output: 'ENAME IS' || ENAME

ename is Allen
 ename is King
 ename is Martin
 ename is Tanmay

Numeric Functions:

1. Abs (Absolute): Abs() function always returns positive number.

Syntax: Abs (Negative Number)

Example: select Abs (-10) from dual;

Output: 10

2. Ceil: This function will return ceiling value of input number. i.e. if you enter 20.10 it will return 21 and if you enter 20.95 then also it will return 21. so if there is any decimal value it will add value by one and remove decimal value.

Syntax: Ceil (Number)

Example: select Ceil (23.77) from dual;

Output: 24

3. Floor: This function does exactly opposite of the ceil function.

Syntax: Floor (Number)

Example: select Floor (45.3) from dual;

Output: 45

4. Power: This function will return power of raise value of given number.

Syntax: Power (Number, Raise)

Example: Select power (5, 2) from dual;

Output: 25

5. Mod: The function gives the remainder of a value divided by another value.

Syntax: Mod (Number, Division Value)

Example: select Mod (10, 3) from dual;

Output: 1

Date Function:

1. CURDATE(): Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

Example: SELECT CURDATE () from dual;

Output: 1997-12-15

- 2. CURTIME ():** Returns the current time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

Example: SELECT CURTIME ();

Output: 23:50:26

- 3. DAYNAME (date):** Returns the name of the weekday for date.

Example: SELECT DAYNAME ('1998-02-05') from dual;

Output: Thursday

- 4. DAYOFMONTH (date):** Returns the day of the month for date, in the range 0 to 31.

Example: SELECT DAYOFMONTH ('1998-02-03') from dual;

Output: 3

- 5. DAYOFWEEK (date):** Returns the weekday index for date (1 = Sunday, 2 = Monday..., 7 = Saturday). These index values correspond to the ODBC standard.

Example: SELECT DAYOFWEEK ('1998-02-03') from dual;

Output: 3

- 6. DAYOFYEAR (date):** Returns the day of the year for date, in the range 1 to 366.

Example: SELECT DAYOFYEAR ('1998-02-03') from dual;

Output: 34

- 7. MONTH (date):** Returns the month for date, in the range 0 to 12.

Example: SELECT MONTH ('1998-02-03') from dual;

Output: 2

- 8. MONTHNAME (date):** Returns the full name of the month for date.

Example: SELECT MONTHNAME ('1998-02-05') from dual;

Output: February

Conclusion: Thus we, have Implement Single Row function (character, numeric, date functions) successfully.

Viva Voce Question

1. What is Scalar function in MySql?

2. Differentiate between LPAD and RPAD commands.

3. What are different types Date functions?

Signature of Subject Teacher

EXPERIMENT NO – 5

Aim: To implement Group function (AVG, MIN, MAX, SUM).

Theory:

Group Functions:

These are the functions in MySQL that performs some calculation on a set of values and then returns a single value. A group functions returns a result based on a group of rows. Some of these are just purely mathematical functions.

List of aggregate functions

- COUNT function
- MIN function
- MAX function
- AVG function
- SUM function

1. **Avg (Average):** This function will return the average of values of the column specified in the argument of the column.

Syntax: SELECT AVG (column name) FROM table name;

Example: select avg(comm) from emp_master;

AVG with WHERE clause

Syntax: SELECT AVG (column name) FROM table name WHERE condition;

2. **Min (Minimum):** The function will give the least of all values of the column present in the argument.

Syntax: SELECT MIN (column name) FROM table name;

Example: Select min (salary) from emp_master;

MIN with WHERE clause

Syntax: SELECT MIN (column name) FROM table name WHERE condition;

3. **Max (Maximum):** To perform an operation, which gives the maximum of a set of values the max, function can be made use of.

Syntax: SELECT MAX (column name) FROM table name;

Example: select max (salary) from emp_master;

MAX with WHERE clause

Syntax: SELECT MAX (column name) FROM table name WHERE condition;

This query will return the maximum value of the column specified as the argument.

4. **Sum:** The sum function can be used to obtain the sum of a range of values of a record set.

Syntax: SELECT SUM (column name) FROM table name;

Example: Select sum (comm) from emp_master;

SUM with WHERE clause

Syntax: SELECT SUM (column name) FROM table name WHERE condition;

5. **Count:** This function is used to count number rows. It can take three different arguments, which mentioned below.

Count (Column name): It counts the number of values present in the column without including nulls.

Syntax: SELECT COUNT (column name) FROM table name;

Example: select count (comm) from emp_master;

Count (*): This will count all the rows, including duplicates and nulls.

Syntax: Select Count (*) FROM table name;

Example: Select count (*) from emp_master;

Count (distinct column name): It is similar to count (column name) but eliminates duplicate values while counting.

Syntax: Select Count (distinct column name) FROM table name;

Example: Select count (distinct deptno) from emp_master;

Conclusion: Thus we, have implement Group function (AVG, MIN, MAX, SUM). successfully.

Viva Voce Question

1. What is aggregate function in MySql?

2. What is the usage of aggregate functions?

Signature of Subject Teacher

DBMS

EXPERIMENT NO – 6

Aim: Implement various types of SET operators (Union, Intersect, Minus).

Theory:

SET Operators:

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

In this tutorial, we will cover 4 different types of SET operations, along with example:

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

UNION operator

MySQL UNION operator allows you to combine two or more result sets of queries into a single result set. The following illustrates the syntax of the UNION operator:

Syntax: SELECT column_list UNION [DISTINCT | ALL] SELECT column list
UNION [DISTINCT | ALL]
SELECT column list...

To combine result set of two or more queries using the UNION operator, there are the basic rules that you must follow:

- First, the number and the orders of columns that appear in all SELECT statements must be the same.
- Second, the data types of columns must be the same or convertible.

By default, the UNION operator removes duplicate rows even if you don't specify the DISTINCT operator explicitly.

Let's see the following sample tables: t1 and t2:

Example: DROP TABLE IF EXISTS t1;
DROP TABLE IF EXISTS t2;

```
CREATE TABLE t1 (
  id INT PRIMARY KEY
);
```

```
CREATE TABLE t2 (
  id INT PRIMARY KEY
);
```

```
INSERT INTO t1 VALUES (1),(2),(3);
```

```
INSERT INTO t2 VALUES (2),(3),(4);
```

The following statement combines result sets returned from t1 and t2 tables:

```
SELECT id FROM t1 UNION SELECT id FROM t2;
```

The final result set contains the distinct values from separate result sets returned by the queries:

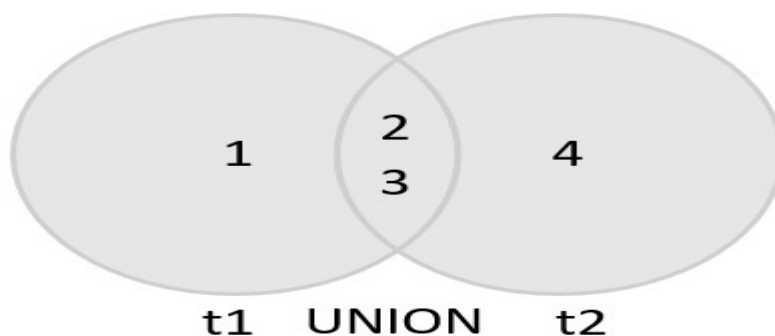
```
+----+
| id |
+----+
| 1 |
| 2 |
```

```
| 3 |
| 4 |
+----+
```

Because the rows with value 2 and 3 are duplicates, the UNION operator removed it and kept only distinct ones.

The following Venn diagram illustrates the union of two result sets that come from t1 and t2 tables:

As you can see, the duplicates appear in the combined result set because of the **UNION ALL** operation.



If you use the UNION ALL explicitly, the duplicate rows, if available, remain in the result. Because UNION ALL does not need to handle duplicates, it performs faster than UNION DISTINCT.

```
SELECT id FROM t1 UNION ALL SELECT id FROM t2;
```

```
+----+
| id |
+----+
| 1 |
| 2 |
| 3 |
| 2 |
| 3 |
| 4 |
```

UNION vs. JOIN

A JOIN combines result sets horizontally; a UNION appends result set vertically. The following picture illustrates the difference between UNION and JOIN:

id
1
2
3

UNION

id
2
3
4



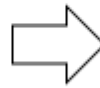
id
1
2
3
4

Append
result sets
vertically

id
1
2
3

INNER
JOIN

id
2
3
4



id	id
2	2
3	3

Append
result sets
horizontally

INTERSECT operator

The INTERSECT operator is a set operator that returns only distinct rows of two queries or more queries.

The following illustrates the syntax of the INTERSECT operator.

Syntax:

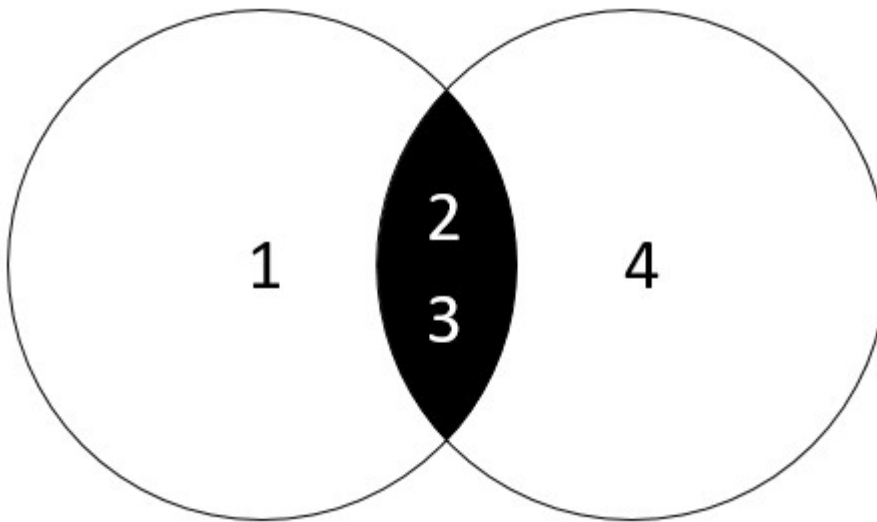
```
(SELECT column _list FROM table_1)
INTERSECT
(SELECT column _list FROM table_2);
```

The INTERSECT operator compares the result of two queries and returns the distinct rows that are output by both left and right queries.

To use the INTERSECT operator for two queries, the following rules are applied:

1. The order and the number of columns must be the same.
2. The data types of the corresponding columns must be compatible.

The following diagram illustrates the INTERSECT operator.



The left query produces a result set of (1,2,3).

The right query returns a result set of (2,3,4).

The INTERSECT operator returns the distinct rows of both result sets which include (2,3). Unlike the UNION operator, the INTERSECT operator returns the intersection between two circles.

Note that SQL standard has three set operators that include UNION, INTERSECT, and MINUS.

MySQL INTERSECT simulation

Unfortunately, MySQL does not support the INTERSECT operator. However, you can simulate the INTERSECT operator.

Let's create some sample data for the demonstration.

The following statements create table's t1 and t2, and then insert data into both tables.

Example:

```
CREATE TABLE t1 (  
  id INT PRIMARY KEY  
);
```

```
CREATE TABLE t2 LIKE t1;
```

```
INSERT INTO t1 (id) VALUES (1), (2), (3);
```

```
INSERT INTO t2 (id) VALUES (2), (3), (4);
```

The following query returns rows from the t1 table.

```
SELECT id FROM t1;
```

```
id
```

```
----
```

```
1
```

```
2
```

```
3
```

The following query returns the rows from the t2 table:

```
SELECT id FROM t2;
```

```
Id
```

```
---
```

```
2
```

```
3
```

```
4
```

Simulate MySQL INTERSECT operator using DISTINCT operator and INNER JOIN clause.

The following statement uses DISTINCT operator and INNER JOIN clause to return the distinct rows in both tables:

Example:

```
SELECT DISTINCT
  id
FROM t1
  INNER JOIN t2 USING (id);
```

```
id
```

```
----
```

```
2
```

```
3
```

How it works.

1. The INNER JOIN clause returns rows from both left and right tables.
2. The DISTINCT operator removes the duplicate rows.

Simulate MySQL INTERSECT operator using IN operator and sub query

The following statement uses the IN operator and a sub query to return the intersection of the two result sets.

Example:

```
SELECT DISTINCT
  id
```

```
FROM
  t1
WHERE
  id IN (SELECT
    id
  FROM
    t2);
```

```
id
----
2
3
```

How it works.

1. The sub query returns the first result set.
2. The outer query uses the IN operator to select only values that are in the first result set. The DISTINCT operator ensures that only distinct values are selected.

MINUS OPERATOR

MINUS is one of three set operations in the SQL standard that includes UNION, INTERSECT, and MINUS.

MINUS compares results of two queries and returns distinct rows from the first query that isn't output by the second query.

The following illustrates the syntax of the MINUS operator:

Syntax:

```
SELECT column_list_1 FROM table_1
MINUS
SELECT columns_list_2 FROM table_2;
```

The basic rules for a query that uses MINUS operator are the following:

- The number and order of columns in both column_list_1 and column_list_2 must be the same.
- The data types of the corresponding columns in both queries must be compatible.

Suppose we have two tables' t1 and t2 with the following structure and data.

Example:

```
CREATE TABLE t1 (
```



```
id INT PRIMARY KEY
);
```

```
CREATE TABLE t2 (
  id INT PRIMARY KEY
);
```

```
INSERT INTO t1 VALUES (1), (2), (3);
```

```
INSERT INTO t2 VALUES (2), (3), (4);
```

The following query returns distinct values from the query of the t1 table that are not found on the result of the query of the t2 table.

```
SELECT id FROM t1
MINUS
SELECT id FROM t2;
```

id
1
2
3

t1 table

MINUS

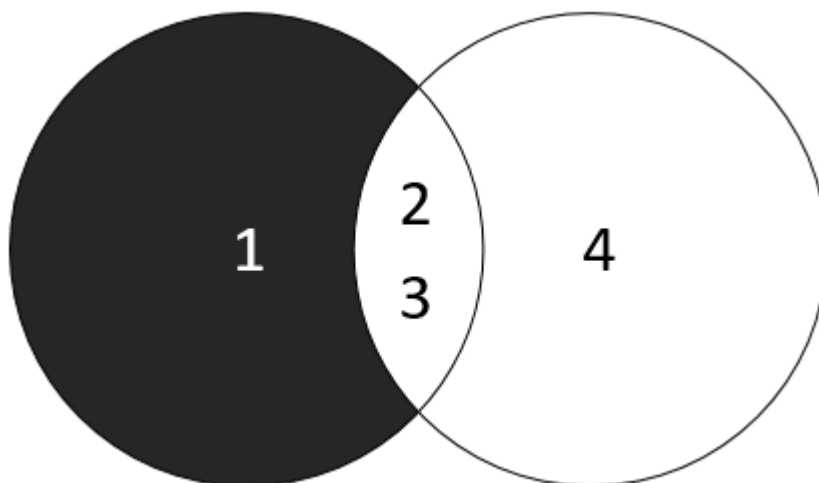
id
2
3
4

t2 table



id
1

The following Venn diagram illustrates the MINUS operator:



Unfortunately, MySQL does not support MINUS operator. However, you can use the MySQL join to simulate it.

To emulate the MINUS of two queries, you use the following syntax:

```
SELECT
  column_list
```

```

FROM
    table_1
    LEFT JOIN table_2 ON join_predicate
WHERE
    table_2.id IS NULL;

```

For example, the following query uses the LEFT JOIN clause to return the same result as the MINUS operator:

```

SELECT
    id
FROM
    t1
    LEFT JOIN
    t2 USING (id)
WHERE
    t2.id IS NULL;

```

In this tutorial, you have learned about the SQL MINUS operator and how to implement MySQL MINUS operator using LEFT JOIN clause.

Conclusion: Thus we, have implement various types of SET operators (Union, Intersect, Minus) successfully.

Viva Voce Question

1. What is Set Operation in MySql?

2. What is the difference between Union and Intersection operation?

3. What is the difference between SET and JOIN operation?

Signature of Subject Teacher

EXPERIMENT NO- 7

Aim: Implement various types of integrity constraints (NOT NULL Constraint, DEFAULT Constraint, UNIQUE Constraint, PRIMARY Key, FOREIGN Key, CHECK Constraint).

Theory:

Integrity Constraints:

MySQL CONSTRAINT is used to define rules to allow or restrict what values can be stored in columns. The purpose of inducing constraints is to enforce the integrity of a database.

MySQL CONSTRAINTS are used to limit the type of data that can be inserted into a table.

MySQL CONSTRAINTS can be classified into two types - column level and table level.

The column level constraints can apply only to one column where as table level constraints are applied to the entire table.

MySQL CONSTRAINT is declared at the time of creating a table.

MySQL CONSTRAINTs are:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

CONSTRAINT	DESCRIPTION
NOT NULL	In MySQL NOT NULL constraint allows to specify that a column can not contain any NULL value. MySQL NOT NULL can be used to CREATE and ALTER a table.
UNIQUE	The UNIQUE constraint in MySQL does not allow inserting a duplicate value in a column. The UNIQUE constraint maintains the uniqueness of a column in a table. More than one UNIQUE column can be used in a table.
PRIMARY KEY	A PRIMARY KEY constraint for a table enforces the table to accept unique data for a specific column and this constraint creates a unique index for accessing the table faster.
FOREIGN KEY	A FOREIGN KEY in MySQL creates a link between two tables by one specific column of both tables. The specified column in one table must be a PRIMARY KEY and referred by the column of another table known as FOREIGN KEY.
CHECK	A CHECK constraint controls the values in the associated column. The CHECK constraint determines whether the value is valid or not from a logical expression.
DEFAULT	In a MySQL table, each column must contain a value (including a NULL). While inserting data into a table, if no value is supplied to a

	column, then the column gets the value set as DEFAULT.
--	--

NOT NULL Constraint

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 a constraint on this column specifying that NULL is now not allowed for that column.

A NULL is not the same as any data; rather, it represents unknown data.

Example

the following SQL query creates a new table called CUSTOMERS and adds five columns, three of which, are ID NAME and AGE, In this we specify not to accept NULLs –

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

If CUSTOMERS table has already been created, then to add a NOT NULL constraint to the SALARY column in Oracle and MySQL, you would write a query like the one that is shown in the following code block.

ALTER TABLE CUSTOMERS

```
MODIFY SALARY DECIMAL (18, 2) NOT NULL;
```

Default Constraint

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

Example

The following SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 5000.00 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 5000.00.

```
CREATE TABLE CUSTOMERS (
    ID INT          NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT         NOT NULL,
    ADDRESS CHAR (25),
    SALARY DECIMAL (18, 2) DEFAULT 5000.00,
    PRIMARY KEY (ID)
);
```

If the CUSTOMERS table has already been created, then to add a DEFAULT constraint to the SALARY column, you would write a query like the one which is shown in the code block below.

```
ALTER TABLE CUSTOMERS
MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00;
```

Drop Default Constraint

To drop a DEFAULT constraint, use the following SQL query.

```
ALTER TABLE CUSTOMERS
    ALTER COLUMN SALARY DROP DEFAULT;
```

UNIQUE Constraint

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

Example

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

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

If the CUSTOMERS table has already been created, then to add a UNIQUE constraint to the AGE column. You would write a statement like the query that is given in the code block below.

```
ALTER TABLE CUSTOMERS
    MODIFY AGE INT NOT NULL UNIQUE;
```

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

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

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL query.

```
ALTER TABLE CUSTOMERS
    DROP CONSTRAINT myUniqueConstraint;
```

If you are using MySQL, then you can use the following syntax –

```
ALTER TABLE CUSTOMERS
```


DROP INDEX myUniqueConstraint;

Primary key Constraint

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys 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.

Create Primary Key

Syntax to define the ID attribute as a primary key in a CUSTOMERS table.

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

To create a PRIMARY KEY constraint on the "ID" column when the 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) should have already been declared to not contain NULL values (when the table was first created).

For defining a PRIMARY KEY constraint on multiple columns, use the SQL syntax given below.

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

To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use the following SQL syntax.

```
ALTER TABLE CUSTOMERS
    ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME);
```

Delete Primary Key

You can clear the primary key constraints from the table with the syntax given below.

```
ALTER TABLE CUSTOMERS DROP PRIMARY KEY;
```

Foreign key Constraint

A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.

A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

The relationship between 2 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 following two tables.

CUSTOMERS table

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

ORDERS table

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

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

ALTER TABLE ORDERS

```
ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);
```

DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL syntax.

ALTER TABLE ORDERS

```
DROP FOREIGN KEY;
```

CHECK Constraint

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 the table.

Example

The following program 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 who is below 18 years.

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

If the CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement like the one given below.

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

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

```
ALTER TABLE CUSTOMERS
    ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18);
```

DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL syntax. This syntax does not work with MySQL.

ALTER TABLE CUSTOMERS

DROP CONSTRAINT myCheckConstraint;

Conclusion: Thus, we have successfully Implemented various types of integrity constraints (NOT NULL Constraint, DEFAULT Constraint, UNIQUE Constraint, PRIMARY Key, FOREIGN Key, and CHECK Constraint).

Viva Voce Question

1. Explain different constraints to maintain data integrity in SQL Server?

2. What is the difference between primary key and unique key constraints?

3. What is the Referential Integrity and foreign keys?

Signature of Subject Teacher

EXPERIMENT NO- 8

Aim: Implement various types of joins (Left Join, Right Join, Outer Join, and Inner Join).

Theory:

JOINS:

We understand the benefits of taking a Cartesian product of two relations, which gives us all the possible tuples that are paired together. But it might not be feasible for us in certain cases to take a Cartesian product where we encounter huge relations with thousands of tuples having a considerable large number of attributes.

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

We will briefly describe various join types in the following sections.

Theta (θ) Join

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol θ .

Notation

$R1 \bowtie_{\theta} R2$

$R1$ and $R2$ are relations having attributes $(A1, A2, \dots, An)$ and $(B1, B2, \dots, Bn)$ such that the attributes don't have anything in common, that is $R1 \cap R2 = \Phi$.

Theta join can use all kinds of comparison operators.

Student		
SID	Name	Std
101	Alex	10
102	Maria	11
Subjects		
Class	Subject	
10	Math	
10	English	
11	Music	

11	Sports

Student_Detail –

STUDENT ⋈_{Student.Std = Subject.Class} SUBJECT

Student_detail				
SID	Name	Std	Class	Subject
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin

When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Natural Join (⋈)

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Courses			
CID		Course	Dept
CS01		Database	CS
ME01		Mechanics	ME
EE01		Electronics	EE
HoD			
Dept		Head	
CS		Alex	
ME		Maya	
EE		Mira	
Courses ⋈ HoD			
Dept	CID	Course	Head
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

Outer Joins

Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the

participating relations in the resulting relation. There are three kinds of outer joins – left outer join, right outer join, and full outer join.

Left Outer Join($R \bowtie S$)

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Left			
A	B		
100	Database		
101	Mechanics		
102	Electronics		
Right			
A	B		
100	Alex		
102	Maya		
104	Mira		
Courses ⋈ HoD			
A	B	C	D
100	Database	100	Alex
101	Mechanics	---	---

102	Electronics	102	Maya
-----	-------------	-----	------

Right Outer Join: ($R \bowtie_r S$)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Courses \bowtie_r HoD			
A	B	C	D
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join: ($R \bowtie_{fs} S$)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Courses \bowtie_{fs} HoD			
A	B	C	D
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira

Conclusion: Thus, we have successfully implemented various types of JOINS.

Viva Voce Question

1. What are the different types of SQL JOIN clauses, and how are they used?

1. What is the difference between Inner Join and Outer Join?

2. What is the difference between left Join and Right Join?

Signature of Subject Teacher

EXPERIMENT NO- 9

Aim:

Theory:

Introduction

Conclusion:

DBMS

Viva Voce Question

1. ?

2. ?

Signature of Subject Teacher

EXPERIMENT NO- 10

Aim:.

Theory:

Conclusion:

DBMS

Viva Voce Question

1. ?

2. ?

Signature of Subject Teacher