DBMS

Database → A database system is basically just a computerized record keeping system. The database itself can be regarded as a kind of electronic filing cabinet; that is it is a container for a collection of computerized data files.

The collection of data, usually referred to as the database, contains information about one particular enterprise. It maintains any information that may be necessary to the decision-making processes involved in the management of that organization.

A database may also be defined as a collection of interrelated data stored together to serve multiple applications.

User of the DB system can perform a variety of operations involving such files — for example:-

→ Adding new files to the database.
→ Inserting data into existing files.
→ Retrieving data from existing files.
→ Deleting data from existing files.
→ changing data in existing files.
→ Removing existing files from the database.

A database system involves 4 major components:

1. ~~data~~ data
→ hardware
→ software
→ user.

(i) Single-user system → A single user system is a system in which at most one user can access the database at any given time.

(ii) A multi-user system is a system in which many users can access the database at the same time.

Data can be of two types:

→ Integrated → No duplication.
→ shared — It mean the database can be shared among different user in the sense that different users can have access to the same data, possibly even at the same time. A DB that is not shared is said to be "personal" or "application-specific".

Ex —
A manufacturing company
A bank
A hospital
A University
A govt. Dept.

Purpose of Database → on a DB

→ The data should be accurate, private and protected from damage. Different application programmers and different end user will have different views of data which must be derived from a common data structure.

On a typical file-processing system, permanent records are stored in various files. A number of different application programs are written to extract records from and add records to the appropriate files. But this scheme has a number of major limitations and disadvantages such as data redundancy (duplication of data), data inconsistency, unshareable data, unstandarized data, insecure data, incorrect data. A database management system is answer to all these problems as it provides a centralized control of the data.

Advantages provided by DBMS and how a DBMS overcomes the above mentioned problems:

→ DB reduces the data redundancy to a large extent.
Duplication of data is known as data redundancy.
The DB system do not maintain separate copies of the same data. Rather, all the data are kept at one place, and all the applications that require data refer to the centrally maintained DB.
Now, if any change is to be made to data, it will be made at just one place, and the same changed information will be available to all the applications referring to it.

→ DB can control data inconsistency to a large extent.
When the redundancy is not controlled, there may be occasions on which the two entries about the same data do not agree (that is, when one of them store the updated information and the other does not). At such times, db is said to be inconsistent.
An inconsistent db will provide incorrect or conflicting information.

→ DB facilitate sharing of data.

Sharing of data means that individual pieces of data in the DB may be shared among several different users, in the sense that each of these user may have access to the same piece of data and each of them may use it for different purposes.

→ DB enforces standards. → The DBMS can ensure that all the data follow the applicable standards. There may be certain standards laid by the company or organization using the DB. Or there may be certain industry standards that must be satisfied by the data. There may be national or international standards.

→ DB can ensure data security → The info stored inside a DB is sometimes of great value to a corporation. Therefore, it must be kept secure and private.

* Data security refer to protection of data against accidental or intentional disclosure to unauthorized

persons or unauthorised modification or destruction.

* And privacy of data refers to the rights of individuals and organizations to determine for themselves, when, how, and to what extent information about them is to be transmitted to others.

→ Integrity can be maintained through DB.
H/w failures and various types of accidents will occur occasionally. The storage of data and its updation, and insertion procedures defined by the database, are such that the system can easily recover from these circumstances without harm to the data.

In addition to protecting data from system problems, the DB management system design certain integrity checks to ensure that data values conform to certain specified rules.

Ex — ① 25/25/06 It is an invalid date
② No. of days worked for an employee cannot exceed the number of working days in a month.

Data Abstraction :-

We know that a collection of interrelated files and a set of programs that allow users to access and modify these files is known as DBMS. A major purpose of a DBMS is to provide the users only that much information that is required by them. This means that the system does not disclose all the details of data, rather it hides certain details of how the data is stored and maintained.

A good database system ensures easy, smooth and efficient data structures in such a way so that every type of database user;

→ end user.
→ application system analyst and
→ physical storage system analyst

is able to access its desired information efficiently.

END USER → An end user is a person who is not a computer-trained person but uses the database to retrieve some information.
For ex — In a bank database, a

customer, who wants to know how much balance remains in his account, is an end-user.

APPLICATION SYSTEM ANALYST → He is the one who is concerned about all of the database at logical level i.e, what all data constitute the database, what are the relationships between the data-entities etc. without considering the physical implementation details.

PHYSICAL STORAGE SYSTEM ANALYST → He is concerned with the physical implementation details of the database i.e how would the database be stored on which storage devices? what will be the starting address of the DB? what will be the storage technique? etc. etc).

Since the requirements of different user differ from one another, the complexity of the database is hidden from them, if needed.

Various Levels of Database Implementation :-

A database is implemented through three general levels: internal, conceptual and external so as to cater to the needs of its users.

* Internal Level (Physical Level)

→ The lowest level of abstraction, the internal level, is the one closest to physical storage. This level is also sometimes termed as physical level. It describes how the data are actually stored on the storage medium. At this level, complex low-level data structures are described in details.

* → Conceptual Level ⇒ This level of abstraction describes what data are actually stored in the database. It also describes the relationships existing among data. At this level, the database is described logically in terms of simple data structures. The users of this level are not concerned with how these logical data structures will be implemented at the physical level.

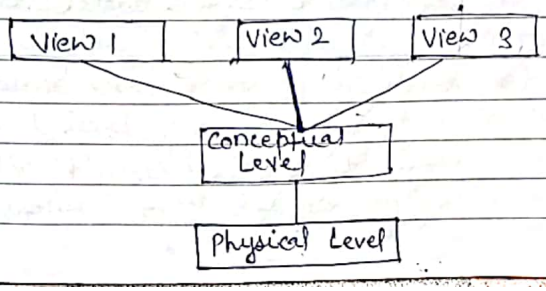
Rather, they just are concerned about what information is to be kept in the database.

→ External Level (View level)

This is the level ~~closest~~ closest to the users and is concerned with the way in which the data are viewed by individual users. Most of the users of the database are not concerned with ~~how these logical data~~ all the information contained in the database. Instead, they need only a part of the database relevant to them.

for example: even though the bank database stores a lot many information an account holder (a user) is interested only in his account details and not with the rest of the information stored in the database. To simply such users' interaction with the system, this level of abstraction is defined.

External View,



Sales office

View 1
Item-Name
Price

Purchase Office

View 2
Item-Name
Reorder-Quantity

Conceptual	
Item-Number	character (6)
Item-Name	character (20)
Price	Numeric (5+2)
Reorder-quantity	Numeric (4)

Internal	
Stored-Item	Length = 40
Item #	Type = Byte (6), offset = 0, Indx = By
Name	Type = Byte (20), offset = 6,
Price	Type = Byte (8), offset = 26
ROQ	Type = Byte (4), offset = 34,

Concept of Data Independence

Data Independence ⇒ The ability to modify a scheme definition in one level without affecting a scheme definition in the next higher level is called Data Independence.

As a database may be viewed through three levels of abstraction, any change at a level may affect other level's schemes. Since the databases keep on growing, there may be frequent changes at times. This should not lead to the redesigning and reimplementation of the database. The concept of data independence proves beneficial in such a context.

1. **Physical Data Independence** → Physical Data Independence refers to the ability to modify the scheme followed at the physical level without affecting the scheme followed at the conceptual level. That is, the application programs remain the same even though the scheme at physical level gets modified. Modifications at the physical level are occasionally necessary in order to

Improve performance of the system.

(2) Logical Data Independence → It refers to the ability to modify the conceptual schema without causing any changes in the schema followed at view levels. The logical data independence ensures that the application programs remain the same. Modifications at the conceptual level are necessary whenever logical structures of the database get altered because of some unavoidable reasons.
for example:- The introduction of paternity leave in the employee database for the first time.

It is more difficult to achieve logical data independence than the physical data independence. The reason being that the application programs are heavily dependent on the logical structure of the database.

STRUCTURED QUERY LANGUAGE

Introduction:-
The Structured Query Language (SQL) is a language that enables us to create and operate on relational databases, which are sets of related information stored in tables.

The SQL has proved to be a standard language as it allows users to learn one set of commands and use it to create, retrieve, alter and transfer information regardless of whether they are working on a PC, a workstation, a mini, or a mainframe.

There are numerous versions of SQL. The original version was developed at IBM's San Jose Research Laboratory. Latest ISO standard of SQL was released in 2008 and named as SQL:2008.

SQL has proved to be a language that can be used by both casual users as well as skilled programmers. It offers a variety of processing capabilities, simpler ones of which may be used.

Ex of DDL → CREATE, ALTER, DROP, GRANT, RENAME
DML – INSERT, UPDATE, DELETE, SELECT, LOCK TABLE.
TCL – COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACT

by the former and the more complex by the latter class of users.

Various processing capabilities of SQL are :-

(1) DDL → Data Definition Language. It provides commands for defining relation schemas, deleting relations, creating indexes, ~~and~~ ~~insertion and retrieval and modifications~~ of data stored in a database.

(2) Interactive Data Manipulation Language (DML) :- Commands that allow us to perform data manipulation e.g retrieval, insertion, deletion and modification of data stored in a database.

(3) TCL (Transaction Control Language) Commands that allow us to manage and control the transactions.
→ making changes to DB, permanent
→ undoing changes to DB, permanent
→ creating savepoints.
→ Setting properties for current transactions.

Transaction ⇒ A transaction is a

complete logical unit of work.

for example: If we say that a file is to be updated, then this transaction will include opening of file, reading of file, making the changes in it and then finally storing it back. Thus this complete unit includes four small steps that make it a transaction.

SQL ELEMENTS

(1) Literals → It refers to a fixed data value. This may be of character type or numeric literal.
Ex – 1, 2, 'Sonam', 'Azad'.

All character literals are enclosed in single quotation marks or double quotations marks.
Numbers are not enclosed in quotation marks are numeric literals e.g 22, 18, 1997, 2003 are all numeric literals.

Numeric literals can either be integer literals i.e without any decimal or be real literals i.e with a decimal point. 17 an integer literal but 17.0 and 17.5 are real literals.

Data Types :-

MySQL uses many different data types, divided into three categories.

→ Numeric
→ Date and time and
→ String types.

Numeric Data Types :-

① INT → -2147483648 to 2147483647
If unsigned → 0 to 4294967295 (11) b/k

② TINYINT → -128 to 127 (4)
If unsigned → 0 to 255

③ SMALLINT → -32768 to 32767 (5)
If unsigned → 0 to 65535

④ MEDIUMINT → -8388608 to 8388607 (9)
If unsigned → 0 to 16777215

⑤ BIGINT → -9223372036854775808 to 9223372036854775807
If unsigned → 0 to 18446744073709551615 (11)

⑥ FLOAT(M,D) ⇒ default (10,2);

⑦ DOUBLE(M,D) ⇒ default (16,4)

⑧ DECIMAL(M,D) ⇒

DATE and TIME Types :-

① DATE → YYYY-MM-DD

② DATETIME → YYYY-MM-DD HH:MM:SS

③ TIMESTAMP → YYYYMMDD HHMMSS

④ TIME → HH:MM:SS

⑤ YEAR(M) → default length is 4

String | Text Types :

① char(M) ⇒ A fixed-length string between 1 and 255 characters in length.

② Varchar(M) ⇒ A variable-length string between 1 and 255 characters in length; for example varchar(25). You must define a length when creating a varchar field.

③ BLOB or TEXT

④ TINYBLOB or TINYTEXT

⑤ MEDIUMBLOB or MEDIUMTEXT

⑥ LONGBLOB or LONGTEXT

⑦ ENUM

Q. Difference between char and varchar datatypes :-

In order to understand the difference between char and varchar datatypes, one must be clear about fixed length and variable length fields.

A field is a data element which can store one type of information which is composed of characters. Number of characters that a field can store, is called field length or field width.
As each character requires one byte for its storage, number of characters inside a field determine its field width in bytes. There can be fixed length fields as well as variable length fields.

Fixed length fields :- It have fixed lengths i.e they occupy fixed number of bytes for every data element they store. These number of bytes are determined by maximum number of characters the field can store. Not necessarily, all data elements inside a field are of same length but the field length is determined by determining the maximum possible characters in data element. In fixed

length fields, some space is always wasted as all data elements do not use all the space reserved but processing is much simpler in case of fixed length fields.

Variable length fields :- They have varied lengths i.e field length is determined separately for every data element inside the field. The number of characters in the data elements become its field width.

Ans: The difference between char and varchar is that of fixed length and variable length. The char datatype specifies a fixed length character string. When a column is given datatype as CHAR(n), then MYSQL ensures that all values stored in that column have this length i.e. n bytes. If a value is shorter than this length n then blanks are added, but the size of value remains n bytes.

VARCHAR, on the other hand, specifies a variable length string, when a column is given datatype as VARCHAR(n), then the maximum size a value in this

column can have is n bytes. Each value that is stored in this column stores exactly as we specify; i.e. no blanks are added if the length is shorter than maximum length n. However, if we exceed the maximum length n, then an error message is displayed.

* NULL VALUES

If a column in a row has no value, then column is said to be null, or to contain a null. Nulls can appear in columns of any data type provided they are not restricted by NOT NULL or PRIMARY KEY integrity constraints. We should use a null value when the actual value is not known or when a value would not be meaningful.

Note:- One thing that you should make sure is, not to use null to represent a value of zero, because they are not equivalent. Any arithmetic expression containing a null, always evaluate to null.

for example, null added to 10 is null. In fact, all operators (except

concatenation) return null when given a null operand.

COMMENTS :-

A comment is a text that is not executed; it is only for documentation purpose. Comments within SQL statements do not affect the statement execution, but they may make our application easier for you to read and maintain. A comment generally describes the statement's purpose within an application. There are three different ways to write comment in SQL :-

1. → /* ---- */
2. → --
3. → #

The SQL provides a predefined set of commands that help us work on relational database.

Keywords → keywords are words that have a special meaning in SQL.

To fully use the power of an RDBMS, we need to communicate with it. A powerful way of communicating with it is making queries. That is, asking

the RDBMS to show our desired data in desired format. This thing can be achieved through SELECT command.

Q. How to access an already existing database.

Ans → Before we start making queries upon the data in tables of a database we need to open the database for use.
for this after logging into MySQL, we need to issue a command.
use <databasename>;
Ex:-
Use AZTECH;
Database changed.

The SELECT Command:
The SELECT command of SQL lets us make queries on the DB. A query is a command that is given to produce certain specified information from the database table. There are various ways and combination a SELECT statement can be used into. It can be used to retrieve a subset of rows or columns from one or more tables. In its simplest form,

SELECT statement is used as given below:
Select <column name>, <column name2>, ...
FROM <table name>;

① Select roll, sname FROM student;

② Select * FROM student;

Selecting columns
for example, if we want to view only the information of two columns. name and owner of table pet, we may write our query as:
SELECT name, owner FROM pet;

Selecting All Columns
If we want to see the entire table i.e every column of a table, we need not give a complete list of columnns. The asterisk (*) can be substituted for a complete list of columns.
Ex-
Select * from pet;

Reordering Columns in Query Results:
while giving a querying the result can be obtained in any order.
for example: If we give

Select roll, sname, sex from student;

The result will be having roll as first column, sname as second column, and sex as third column, will be displayed. We can write the column names in any order and the output will be having information in exactly the same order.

Eliminating Redundant Data (with Keyword DISTINCT)

By default, data is selected from all the rows of the table, even if the data appearing in the result gets duplicated. The DISTINCT keyword eliminates duplicate rows from the results of a SELECT statement.

Ex:-

Select Distinct city
from student;

City
Delhi
Mumbai
Jaipur
Bangalore

On the output, there would be no duplicate rows whenever DISTINCT is used, only one NULL value is returned in the results, no matter how many

NULL values are encountered.

Selecting FROM all the rows — ALL Keyword

If we in place of DISTINCT, we give ALL then the results retains the duplicate output rows.

SELECT ALL city FROM student;

o/p

city
Delhi
Mumbai
Delhi
Banglore

* To show all the tables in a database.

show tables;

* If we want to know the structure of a table, we can use DESCRIBE or DESC command as per the following syntax:

DESCRIBE | DESC <table name>;

➡ How to perform Simple Calculations?

Often a simple calculation needs to be done, for example, 4*3. The only SQL verb to cause an output to be written to monitor is SELECT.

SELECT 1+6;

1+6
7

MySQL also provides a dummy table called Dual to provide compatibility with SQL of other DBMSs.
Dual table is a small worktable, which has just one row and one column. It can be used for obtaining calculation results and also system-date. The following query:
SELECT 4*3 FROM dual;

4*3
12

→ The current date can be obtained, using function curdate(), as shown below:

Select curdate(); Select NOW(); Only date

curdate()
Date Time

→ Scalar Expressions with Selected fields
If you want to perform simple numeric computations on the data to put it in a form more appropriate to your needs,

SQL allows us to place scalar expressions and constants among the selected fields.

For example, we want to increase marks by 5.
Select roll, sname, marks+5 FROM STUDENT;

O/P

1	Sonam	85
2	Azad	90

The operators that we can use in arithmetic expressions are plus (+), minus (-), divide (/), multiply (*) and modulo (%) for remainder.

Using Column Aliases

The columns that we select in a query can be given a different name i.e column alias-name for output purposes.
Ex:
① Select roll, sname, marks+5 as "INCREASED MARKS" FROM STUDENT;

② Select 22/7 as PI;

PI
3.1429

③ Select roll, sname, total/5 as Percentage FROM STUDENT;

HANDLING NULL

As we know that, the empty values are represented as NULLS in a table. When we list a column having null values, only non-null values are displayed.

Ex:-

Select roll, sname, marks from student;

Roll	sname	marks
1	Azad	NULL
2	Sonam	85
3	Ram	65
4	Prasoon	NULL

We have seen that null values appear as NULL. If we want to substitute null with a value in the output, we can use IFNULL() function. IFNULL() function may be used as per following syntax.

Ex- IFNULL(<columnname>, value-to substituted)

Select roll, sname, IFNULL(marks, "FAIL");

o/p

Roll	Sname	marks.
1	Azad	FAIL
2	Sonam	85
3	Ram	65
4	Prasoon	FAIL

Note: The value being substituted for NULL values through IFNULL(), should be of same type as that of the column.

Putting Text in the Query Output

① Select roll, sname, marks/5, '%' from student.

② Select roll, sname, 'has got', marks/5, '%' from student;

Selecting Specific Rows - WHERE clause

In real life, tables can contain unlimited rows. There is no need to view all the rows when only certain rows are needed. SQL enables us to define criteria to determine which rows are selected for output.

The WHERE clause in SELECT statement specifies the criteria for selection of rows to be retrieved. The SELECT statement with WHERE clause takes the following general form:

Select <column name1>, <column name2>...
FROM <tablename>
WHERE <condition>;

When a WHERE clause is present, the database program goes through the entire table one row at a time and examines each row to determine if the given condition is true. If it is true for a row, that row is displayed in the output.

Ex-

1. Select roll, sname from student where marks > 300;
2. Select roll, sname, age from student where age >= 18;

Relational operators

To compare two values, a relational operator is used. The result of the comparison is true or false. The SQL recognizes following relational operators:
=, >, <, >=, <=, <> (not equal to).

Ex:- Select * from student where city <> 'Patna';

LOGICAL OPERATORS

The logical operators OR(||), AND(&&) and NOT(!) are used to connect search conditions in the WHERE clause.

1. Select ecode, ename, grade from emp where (grade = 'E2' OR grade = 'E3');

Symbol || may also be used as OR operator.

2. Select ecode, ename, grade, sal from emp where (grade = 'E4' AND sal > 9000);
3. Select ecode, ename, grade, gross from emp where (Not grade = 'G1');

Q. Write a query to display all the details from pet table for species cat/dog having gender (sex) as male ('m')

Ans→
```
Select * from pet
where (species = 'cat' || species = 'dog')
&& sex = 'm';
```

CONDITION BASED ON RANGE

The BETWEEN operator defines a range of values that the column value must fall in to make the condition true. The range include both lower value and upper value.

Ex:-
```
Select roll, sname, marks
from student
where marks between 40 AND 60
```

The operator NOT BETWEEN is reverse of BETWEEN operator, that is the rows not satisfying the BETWEEN conditions are retrieved.

Ex:
```
Select roll, sname
from student
where ROL NOT BETWEEN 50 AND 80.
```


The above query will list the students whose roll is below 50 or above 80.

CONDITION BASED ON A LIST:-

To specify a list of values IN operator is used. The IN operator selects values that match any value in a given list of values.

for example, to display a list of members from 'DELHI', 'MUMBAI', 'CHENNAI' or 'BANGALORE' cities, we may give:

Ex:-
```
Select * FROM student
where city IN('DELHI','MUMBAI',
'CHENNAI','BANGALORE');
```

The NOT IN operator finds rows that do not match in the list. So, if we write
```
Select * FROM STUDENT
where city NOT IN('DELHI','MUMBAI',
'CHENNAI');
```

CONDITION BASED ON PATTERN MATCHES

SQL also include a string-matching operator, LIKE, for comparisons on character strings using patterns. Patterns are described using two

Special wildcard characters:

⇒ percent (%) → The % character matches any substring.

⇒ underscore (_) → The _ character matches any character.

Patterns are case-sensitive, that is uppercase character do not match lower-case character, or vice-versa.

Ex:-

⇒ 'San%' ⇒ matches any string beginning with 'San'.

⇒ '%idge%' ⇒ matches any string containing 'idge' as a substring
Ex - Bridges, Catridge.

⇒ "___" matches any string of exactly 4 character.

⇒ '___%' matches any string of at least 3 characters.

The LIKE keyword is used to select rows containing columns that match a wildcard pattern.

① Select firstname, lastname, city
from emp
where pin LIKE '13%';

It will list members which are in areas with pin codes starting with 13.

② To list names of students ending with 'y', the command would be:
Select name
from student
where name LIKE '%y';

③ To list member which are not in areas with pin codes starting with 13, the command is:
Select firstname, lastname, city
from members
where pin NOT LIKE '13%';

The keyword NOT LIKE is used to select rows that do not match the specified pattern of characters.

Q Write query to display the names of pets beginning with 'f'.
⇒ Select * from pet
where name LIKE "f%";

Q Write query to display the names of pets having exactly four letters names.
⇒ Select * from pet
where name LIKE "____";

Searching for NULL

The NULL value in a column can be searched for in a table using IS NULL in the where clause. (Relational operator like =, <>, etc. can't be used with NULL).

For example :- To list details of all employees whose departments contain NULL (novalue), we use the command:

```
Select empno, empname, job
from emp
where deptno IS NULL;
```

Sorting Results - ORDER BY clause

Whenever a SELECT query is executed, the resulting rows emerge in a pre-decided order. You can sort the results or a query in a specific order using ORDER BY clause. The ORDER BY clause allows sorting of query results by one or more columns. The sorting can be done either in ascending or descending order, the default order is ascending. The data in the table is not sorted; only the results that appear on the screen are sorted. The ORDER BY clause is used as:


```
Select columnname1, columnname2,
from <tablename>
where <condition>
ORDER BY <column name> ;
```

→ for example, to display the list of employees in the alphabetical order of their names, we use the command:

```
Select * from Employee
order by ename;
```

To display the list of students having aggregate more than 400 in the alphabetical order of their names, we may give the command:

→
```
Select name, aggregate from student
where aggregate >400
order by name;
```

→ To display the list of employees in the descending order of employee code, we use the command:

```
select * from employee
order by ecode DESC;
```

* To specify the sort order, we may specify DESC for descending order or ASC for ascending order.

Select * from employee
order by grade DESC, ename ASC;

Q Write a query to display name, age, aggregate of students whose aggregate is between 300 and 400. Order the query in descending order by name.

Ans → Select name, age, aggregate FROM student
where aggregate between 300 AND 400
order By name DESC;

ASSIGNMENTS

Q1 Display all the records (all columns) from table emp.

Q2 Display EmpNo and EName of all employees from table empl.

Q3 Display Ename, Sal, and sal added with comm from table empl.

Q4 ~~Select~~ Write a query to display employee name, salary and department number who are not getting commission from table empl.

MySQL FUNCTIONS

Data can be manipulated in queries through functions supported by SQL. A function is a special type of predefined command set that performs some operation and returns a single value. Functions operate on zero, one, two or more values that are provided to them. The values that are provided to functions are called parameters or arguments.

The MySQL functions have been categorised into various categories, such as String functions, Mathematical functions, Date and Time functions and so on.

String functions :-

The String functions of MySQL can manipulate the text string in many ways.

(1) CHAR() → This function returns the character for each integer passed.

Ex — ASCII code
Select CHAR(70, 65, 67, 69);
o/p FACE

Ex — ~~selected~~ Select CHAR(65, 67.3, 69.3);
o/p ACE

(2) CONCAT() → This function concatenates two strings.

Ex – Select CONCAT(name, marks) AS "Name Marks" from student where age >= 14 OR age = 16;

Select CONCAT(CONCAT(ename, "is a"), job); from emp where empno = 8900;

(3) LOWER/LCASE → This function converts a string into lowercase.

Ex –

Select LOWER("AZTECH");

(4) SUBSTR() → This function extracts a substring from a given string.

Ex –

Select SUBSTR("ABCDEFGH", 3, 4);

o/p

CDEF

(5) UPPER/UCASE() → This function converts the given string into upper case.

Select UCASE("aztech");

(6) LTRIM() → This function removes leading spaces i.e spaces from the left of given string.

Select LTRIM(' AZTECH');

(7) RTRIM() → This function removes trailing spaces i.e spaces from the right of given string.

Select RTRIM('AZTECH ');

(8) TRIM() → This function removes leading and trailing spaces from a given string. It performs combined functions of LTRIM() and RTRIM().

Select TRIM(' AZTECH ');

(9) INSTR() → This function searches for given second string into the given first string.

Select INSTR('CORPORATE FLOOR', 'OR');

(10) LENGTH() → This function returns the length of a given string in bytes.

Select LENGTH('AZTECH');

(11) LEFT() → This function returns the leftmost number of characters as specified.

Select LEFT('USS/23/67/09', 3);

(12) RIGHT() → This function returns the rightmost number of character as specified.
Select RIGHT('USA/23/67/09', 2);

(13) MID() → This function returns a substring starting from the specified position.

Select SUBSTRING('Quadratically', 5, 6);
OR
Select MID('Quadratically', 5, 6);

NUMERIC FUNCTIONS

The number functions are those functions that accept numeric values and after performing the required operation, return numeric values.

(1) MOD() → This function returns modulus (i.e remainder) of given two numbers.
Ex:-
Select MOD(11,4);

(2) POWER/POW → This function return m^n i.e a number m raised to the n^{th} power.

Ex:- Select POWER(3,2);

(3) ROUND() → This functions returns a number rounded off as per given specifications.
Ex:-
Select ROUND(15.193,1);

(4) SIGN() → This function returns sign of a given number.
Ex:- Syntax:
SIGN(n)
If argument n < 0, the function returns -1,
If argument n = 0, the function return 0;
If argument n > 0, the function returns 1

Select SIGN(-15);

(5) SQRT() → This function returns the square root of a given number.
Ex:-
Select SQRT(25);

(6) TRUNCATE → This function returns a number with some digits truncated.
Ex:-
Select TRUNCATE(15.79,1);
O/P
15.7

DATE/TIME FUNCTIONS:

① CURDATE() / CURRENT-DATE() → This function return the current date

Ex:-
Select curdate();
Select curdate() +10;

② DATE() → This function extracts the date part of a date or datetime expression.

Ex:- Select DATE('2008-12-31 01:02:03');

③ MONTH() → This function returns the month from the date passed.

Ex:-
Select MONTH('2009-02-03');

④ YEAR() → This function returns the year part of a date.

Ex:
Select YEAR('2009-02-03');

⑤ DAYNAME() → This function returns the name of weekday.

Ex:
Select DAYNAME('2009-02-03');

⑥ DAYOFMONTH() → This function returns the day of month.

Ex:- Select DAYOFMONTH('2009-02-03');

⑦ DAYOFWEEK() → This function returns the day of week.

1 = sunday
2 = Monday
7= Saturday.

Ex:
Select DAYOFWEEK('2009-02-13');

⑧ DAYOFYEAR() → This function returns the day of the year. Returns the day of the year for date, in the range 1 to 366.

Ex:-
Select dayofyear('2009-02-13');

⑨ NOW() → This function returns the current date and time.

Ex:
Select NOW();

⑩ SYSDATE() → This function returns the time at which the function execute.

Ex:-
Select now(), sleep(2), NOW();

O/P
NOW() → 2015-07-20 15:23:34
sleep(2) → 0
NOW() → 2015-07-20 15:23:34

Select sysdate(), sleep(2), sysdate();

o/p

sysdate - 2015-06-03 15:33:15

sleep(2) - 0

sysdate - 2015:06-03 15:33:17

CREATING DATABASES:-

(1) On simplest form the Create Database command takes the following syntax:

CREATE DATABASE IF NOT EXISTS
<database name>;

IF NOT EXISTS clause, if used, will first test whether a database by the mentioned name already exists or not. If it does, then create database command is simply ignored, otherwise a database with the mentioned name is created.

Ex:-
Create database AZTECH;
⇒ Creates database having name as AZTECH.

Create database IF NOT EXISTS AZTECH

Creates a database having name as AZTECH if there is no database by the name AZTECH already exists.

OPENING DATABASES:-

Creating database is not enough. Before we create tables in it, we need to open the database. To open a database, we simple need to write the statement given below:

Use databasename;
Use AZTECH;
The only thing we need to ensure before opening a database is that it must already exist i.e it must be already created. To check the names of existing databases, we may write following command:

SHOW DATABASES;

Removing Database :-

Sometimes, we need to remove a database, when we don't need it anymore. But before making this decision, do make sure that

we don't need data stored in different tables of the database. This is because, when we drop a database, all its tables also get removed along with a database.

To remove a database, we need to issue the following command:

DROP DATABASE <databasename>;
Drop database AZTECH;

CREATING TABLES :-

Tables are defined with the CREATE TABLE command. When a table is created, its columns are named, datatypes and sizes are supplied for each column. Each table must have at least one column. The syntax of CREATE TABLE command is :-

CREATE TABLE <tablename>
(<columnname> <datatype> <size>,
<columnname> <datatype> <size>....
.....);

To create an employee table

whose schema is as follows:
employee (ecode, ename, sex, grade, gross)

Syntax:

```
CREATE TABLE employee
( ecode integer,
  ename char(20),
  sex   char(1),
  grade char(2),
  gross decimal) ;
```

DATA INTEGRITY THROUGH CONSTRAINTS :-

A constraints, in general, refers to a condition or a check that is applied to a column (field) or set of columns in a table. The constraints applied to maintain data integrity are also known as Integrity constraints.

Once an integrity constraints is enabled, all data in the table must confirm to the rule that it specifies. Any subsequent SQL statement that tries to enter

or modify data in the table, is successfully carried out if and only if the data being entered or modified satisfies the integrity constraints.

A constraint is a condition or check applicable on a field or set of fields.

The two basic types of constraints are Column constraints and Table Constraints. The difference between the two is that column constraints apply only to individual columns, where as table constraints apply to groups of one or more columns.

The following is the syntax for the create table command, expanded to include constraints :-

```
CREATE TABLE <tablename>
(
<columnname><datatype> [(size)] column constraints,
"       "       "       "
);
```

for example, if we write the keywords NOT NULL immediately after the

data type (and size) of a column, this means the column can never have empty values (i.e. NULL values). Otherwise SQL will assume that NULL are permitted. Consider the following SQL command.

```
CREATE TABLE employee
(ecode integer NOT NULL,
 ename char(20) NOT NULL,
 sex   char(1)  NOT NULL,
 grade char(2),
 gross decimal );
```

The above command creates a table called employee in which ecode column (integer type) can never be empty as its definition is followed by keywords NOT NULL. Similarly, the columns ename (char(20)) and sex (char(1)) can never have NULL values. Any attempts to put NULL values in these columns will be rejected.

DIFFERENT CONSTRAINTS :-

These constraints ensure database integrity, thus are sometimes called database integrity constraints. A few of them are:

1) UNIQUE Constraint
2) PRIMARY KEY Constraint
3) Default Constraint
4) Check Constraint
5) Foreign Key Constraint

MySQL also supports some other constraints such as ENUM and SET constraints that are used to define columns that can contain only a given set of values.

1. Unique Constraint ⇒ This constraints ensures that no two rows have the same value in the specified column(s).
for example UNIQUE constraint applied on ecode of employee table ensures that no rows have the same ecode value.

```
CREATE TABLE employee
( ecode integer NOT NULL UNIQUE,
  ename char(20) NOT NULL,
  sex char(1) NOT NULL,
  grade char(2),
  gross decimal);
```


2. Primary key Constraint :-
This constraint declares a column as the primary key of the table. This constraint is similar to unique constraint except that only one column (or one group of columns) can be applied in this constraint. The primary keys cannot allow NULL values, thus this constraint must be applied to columns declared as NOT NULL.

Ex:-

```
CREATE TABLE employee
( ecode integer NOT NULL PRIMARY KEY,
  ename char(20) NOT NULL,
  sex char(1) NOT NULL,
  grade char(2),
  gross decimal);
```

Q. Difference between Primary and Unique key.
⇒ Both key ensures unique value for each row in a column, but UNIQUE allows NULL values where as PRIMARY KEY does not. Also, there can exist multiple columns with UNIQUE constraints in a table, but there can exist only one column or one combination with PRIMARY KEY

constraint.

3. DEFAULT Constraint → When a user does not enter a value for the column (having default value), automatically the defined default value is inserted in the field. A default value can be specified for a column using the DEFAULT clause.

Ex—

```
CREATE TABLE employee
(
  ecode integer NOT NULL PRIMARY KEY,
  ename char(20) NOT NULL,
  sex   char(1)  NOT NULL,
  grade char(2)  DEFAULT 'E1',
  gross decimal);
```

According to above command, if no value is provided for grade, the default value of 'E1', will be entered. The datatype of the default value has to be compatible with the datatype of the column to which it is assigned. Insertion of NULL (as default value) is possible only if the column definition permits.

NOT NULL column cannot have NULL as default. A column can have only one default value.

4. CHECK CONSTRAINT → This constraint limits value that can be inserted into a column of a table. For, instance, consider the following SQL statement:

Ex:-

```
CREATE TABLE employee
(
  ecode integer NOT NULL PRIMARY KEY,
  ename char(20) NOT NULL,
  sex   char(1)  NOT NULL,
  grade char(2)  DEFAULT 'E1',
  gross decimal CHECK (gross > 2000));
```

This statement ensures that the values inserted for gross must be greater than 2000.

When a check constraint involves more than one column from the same table, it is specified after all the columns have been defined.

```
Create table items
( icode char(5) NOT NULL PRIMARY KE
```

desc char(20) NOT NULL,
rol integer,
qoh integer,
CHECK (ROL < QOH));
This statement compares two columns ROL and QOH, thus, these two columns must be defined before this CHECK constraint.
CHECK constraint can consist of:

* A list of constant expressions specified using IN. For example:

city char(20) CHECK
(city IN ('Patna', 'UP', 'MP'))

* Range of constant expressions specified using BETWEEN. The upper and lower boundary values are included in the range. For example:

price decimal CHECK (price BETWEEN 250.00 and 800.00)

* A pattern specified using LIKE. For example, if we give a date in char format, then we may chek:

DOJ char(10) NOT NULL CHECK (DOJ LIKE '--/--/----')

Ex:-
CREATE TABLE Customer
(SID integer CHECK (SID > 0),
Last_name varchar(30),
First_name varchar(30));

Column SID has a constraint – its value must only include integers greater than 0. So attempting to execute the following statement,
INSERT INTO Customer VALUES
('-3', 'Sonam', 'Azad');
should result in an error because the value for SID must be greater than 0.

FOREIGN KEY CONSTRAINT

CREATING TABLE FROM EXISTING TABLE

We can define a table and put data into it without going through the usual data definition process. This can be done by using SELECT statement with CREATE TABLE. The new table stores the result produced by the SELECT statement. The name of the new table must be unique.

ex:-

```
CREATE TABLE STUDENT1 AS        <- New Table
( SELECT ROLL, SNAME
    FROM STUDENT                <- existing table
    WHERE ROLL > 5
);
```

This will create a new table called STUDENT1 that stores two columns : ROLL, SNAME.

MODIFYING DATA WITH UPDATE COMMAND:

Sometimes we need to change some or all the values in an existing row. This can be done using the UPDATE command of SQL. The UPDATE command specifies the rows to be changed using the WHERE clause, and the new data using the SET keyword. The new data can be a specified constant, an expression or data from other tables.

Ex:-

```
Alter Table Customer MODIFY
Address varchar(20);
```

ORACLE ALTER TABLE Customer MODIFY Address varchar(25); 60

SQL Server → ALTER TABLE Customer ALTER COLUMN Address varchar(

```
UPDATE Student
Set marks = 496
where ROLLNO = 5;
```

UPDATING MULTIPLE COLUMNS :-

To update multiple columns, multiple column assignments can be specified with SET clause, separated by commas.

①
```
UPDATE STUDENT
set ROL = 400, MARKS = 700
where sid < 15;
```

②
```
UPDATE Employee
set gross = gross + 900;
```

③
```
UPDATE employee
Set gross = gross * 2
where (grade = 'E3' or grade = 'E4');
```

④
```
UPDATE employees
Set grade = NULL
where grade = 'E4';
```

DELETING DATA WITH DELETE Command :

While working with tables, we may reach at a situation where we no longer need some rows of data.

In such a case, we would like to remove such rows. This can be done by using DELETE command.

The DELETE command removes rows from a table. This removes the entire rows, not individual field value, so no field argument is needed or accepted. The DELETE statement takes the following general form:

```
DELETE from <tablename>
where <condition>;
```

* To remove all the contents of student table, we have to use the following command :

```
DELETE FROM STUDENTS;
```

*
```
Delete from employee
where gross <2200.00;
```

ALTER TABLE COMMAND

→ When we define a system, we specify what data we need to store, the size and data type of that data. what can we do when the requirements change? we can alter the tables to accommodate the changed requirements.

The ALTER TABLE command is used to change definitions of existing tables. Usually, it can add columns to a table. Sometimes it can delete columns (depending on privileges) or change their sizes. In general, ALTER TABLE command is used:

→ to add a column
→ to add an integrity constraint
→ to redefine a column (datatype, size, default values).

Adding Columns

→ We use the following syntax to add a column to a table :-

ALTER TABLE <tablename> ADD <columnname> <datatype> <size> <constraint name> ;

The new columns will be added with NULL values for all rows currently in the table. It is generally possible to add several new columns, separated by commas, in a single command.

Ex:- ALTER TABLE emp
ADD (tel_number integer);

ALTER TABLE emp
ADD (gross NUMBER(7,2),

ecode char(1) CHECK (ecode IN ('E', 'C', 'H', 'P')));

While adding columns, we can define these types of integrity constraints (NOT NULL, UNIQUE, PRIMARY KEY, DEFAULT and CHECK constraints) on existing columns using the ADD clause and in the table constraint syntax.

MODIFYING COLUMN DEFINITIONS:-

We can use the MODIFY clause to change any of the following parts of a column definition:

→ datatype
→ size
→ default value
→ NOT NULL column constraint
→ Order of column

The MODIFY clause need only specify the column name and the modified part of the definition, rather than the entire column definition.

Examples:-

ALTER TABLE <tablename>
MODIFY (columnname newdatatype(new size));

* ALTER TABLE emp
MODIFY (Job char(20));

* ALTER TABLE emp
MODIFY empID INT FIRST;

* ALTER TABLE emp
MODIFY (gross NUMBER(9,2));

* ALTER TABLE ACCOUNTS
MODIFY (bal DEFAULT 0);

* ALTER TABLE emp
ADD PRIMARY KEY (eid);

CHANGING A COLUMN NAME

Sometimes we may need to change the name of one of our columns. For this, we can use CHANGE clause of ALTER TABLE command as per following syntax:

ALTER TABLE
CHANGE <old_column_name> <new column definition>;

Ex:-
ALTER TABLE CUSTOMERS
CHANGE First_Name firstName VARCHAR(20);

REMOVING TABLE Components

If we plan to remove a component of a table, then we may use the DROP clause of ALTER TABLE. The keywords mostly used with DROP clause of ALTER TABLE command are:-

Primary key → Drop's the table's primary key Constraint

Column <columnname> → Removed mentioned column from the table.

Ex:- ALTER TABLE dept
DROP column dname;

The DROP TABLE Command

The DROP TABLE Command of SQL lets us drop a table from the database. The syntax for using a DROP TABLE command is:

DROP TABLE [IF EXISTS] <tablename>;

Ex:-
DROP TABLE emp;
DROP TABLE IF EXISTS emp;

AGGREGATE (GROUP) FUNCTIONS.

MySQL also supports and provides group functions or aggregate functions.
Group functions or aggregate functions work upon groups of rows, rather than on single rows. That is why, these functions are sometimes also called multiple row functions.

Many group functions accept the following options:

- DISTINCT → This option causes a group function to consider only distinct values of the argument expression.
- ALL → This option causes a group function to consider ~~only distinct~~ all values including all duplicate.

Let us create a table empl.
empno, ename, Job, MGR, HIREDATE, SAL, COMM, DEPTNO

(1) AVG → The function computes the average of given data.
Ex—
Select AVG(sal) "Average Salary"
From empl;

(2) COUNT → This function counts the number of rows in a given column or expression.
Ex:—
① Select COUNT(*) "Total" from empl;
② Select COUNT(Job) "Job Count" from empl;
③ Select COUNT(DISTINCT JOB) "Distinct Jobs" from empl;

(3) MAX → This function returns the maximum value from a given column or expression.
Ex:—
Select MAX(sal) "Maximum Salary" from empl;

(4) MIN → This function returns the minimum value from a given column or expression.
Ex:—
Select MIN(sal) "Minimum Salary" from empl;

(5) SUM → This function returns the sum of values in a given column or expression.
Ex:—
⇒ Select SUM(sal) "Total Salary" from empl;

⇒ Select SUM(gross) from emp where grade = 'E2';

⇒ Select AVG(gross) from emp where (grade = 'E1' or grade = 'E2');

⇒ Select COUNT(*) from empl;

⇒ Select COUNT(DISTINCT city) from empl;

If we want to count the enteries including repeats, the keyword ALL is used.

⇒ Select COUNT(ALL city) from empl;

On General, GROUP functions
- Return a single value for a set of rows
- Can be applied to any numeric values, and some Text Types and DATE values.

TYPES OF SQL FUNCTIONS :-

SQL supports many and many functions. All these functions can be categorized into following two types :-
- Single Row (or scalar) functions.
- Multiple Row (or Group or Aggregate) functions.

* Single Row Functions: Single row function work with a single row at a time. A single row function returns a result for every row of a queried table.
Ex:- year(), day() etc.

* Multiple Row or Group functions :- These functions work with data of multiple rows at a time and return aggregated value.
Ex:- sum(); count(), max(), min(), avg() etc.

The difference between these two types of functions is in the number of rows they act upon. A single row function works with the data of a single row at a time and returns a single result for each row queried upon. A multiple row function works with a group of rows and returns a single result for that group.

GROUPING RESULT - GROUP BY

The GROUP BY clause combines all those records that have identical

value in a particular field or a group of fields. This grouping results into one summary record per group if group-functions are used with it. In other words, the GROUP BY clause is used in SELECT statements to divide the table into groups. Grouping can be done by a column name, or with aggregate functions in which case the aggregate produces a value for each group.

Ex:—

① Select job, COUNT(*)
from empl
GROUP By Job;

o/p

Job	count(*)
Analyst	2
Clerk	4
Manger	3
President	1
Saleman	4

② Select deptno, count(*), SUM(sal)
from empl
Group by deptno;

Nested Groups — Grouping on Multiple Columns :-

With Group By clause, we can create groups within groups. Such type of grouping is called Nested grouping. This can be done by specifying in GROUP BY expression, where the first field determines the highest group level, the second field determines the second group level, and so on. The last field determines the lowest level of grouping.

① Let us see this example:
Select COUNT(empno) FROM empl
GROUP BY Deptno;

o/p

Count
2
5
6

But from this we can not make out that these employee-counts for which departments? To get this information, we may modify the SELECT list as:

Select Deptno, COUNT(empno) from empl

GROUP BY Deptno;

o/p

deptno	count(empno)
10	3
20	5
30	6

Now the result is more clear.

① Select Deptno, Job, COUNT(empno)
from empl
GROUP BY Deptno, Job;

o/p

deptno	job	count(empno)
10	clerk	1
10	Manager	1
10	President	1
20	Analyst	2
20	Clerk	2
20	Manager	1
30	clerk	1
30	Manager	1
30	Salesman	4

PLACING CONDITIONS ON GROUPS – Having clause:

The HAVING clause places conditions on groups in contrast

to WHERE clause that places conditions on individual rows. While WHERE conditions cannot include aggregate functions, HAVING conditions can do so.

Ex:– To calculate the average gross and total gross for employees belonging to 'E4' grade, the command would be:

Ex:–

→ Select AVG(gross), SUM(gross)
from employee
Group By grade
Having grade = 'E4';

→ Select Job, COUNT(*)
from empl
Group By Job
Having count(*) < 3;

→ Select Deptno, AVG(comm), AVG(sal)
from empl
GROUP BY Deptno
Having AVG(comm) > 750 AND
AVG(sal) > 2000;

→ Select Deptno, AVG(sal)
from empl

```
Group By Deptno
Having COUNT(*) <= 3;
```

→
```
Select Deptno, Job, AVG(sal)
from empl
group By Deptno, Job
Having Job IN('CLERK', 'SALESMAN');
```

→
```
Select Deptno, Job, sum(sal)
from empl
Group By Deptno, Job
Having SUM(sal) BETWEEN
3000 AND 7000;
```

EMPL TABLE

EmpNO	Ename	Job	MGR	HIREDATE	SAL	COMM	DEPT.NO	CITY
8369	SMITH	CLERK	8902	1990-12-18	800.00	NULL	20	
8499	ANYA	SALESMAN	8698	1991-02-20	1600.00	300.00	30	
8521	SETH	SALESMAN	8698	1991-02-22	1250.00	500.00	30	
8566	MAHADEVAN	MANAGER	8839	1991-04-02	2985.00	NULL	20	
8654	MOMIN	SALESMAN	8698	1991-09-28	1250.00	1400.00	30	
8698	BINA	MANAGER	8839	1991-05-01	2850.00	NULL	30	
8839	AMIR	PRESIDENT	NULL	1991-11-18	5000.00	NULL	10	
8844	KULDEEP	SALESMAN	8698	1991-09-08	1500.00	0.00	30	
8882	SHIVANSH	MANAGER	8839	1991-06-09	2450.00	NULL	10	
8886	ANOOP	CLERK	8888	1991-01-12	1100.00	NULL	20	
8888	SCOTT	ANALYST	8566	1991-12-09	3000.00	NULL	20	
8900	JATIN	CLERK	8698	1991-12-03	950.00	NULL	30	
8902	FAKIR	ANALYST	8566	1991-12-03	3000.00	NULL	20	
8934	MITA	CLERK	8882	1991-01-23	1300.00	NULL	10	