Part - I

- FMS vs DBMS vs noSql

  The **key difference** between DBMS and File Management System is that **a DBMS stores data to the hard disk according to a structure while a file management system stores data to the hard disk without using a structure.**

  Advantages of DBMS:
  - High Security
  - User Friendly nature
  - Having a structure

  Disadvantages of DBMS:
  - Low performance

  Advantages of FMS
  - High Performance
  - User friendly nature

  Disadvantages of FMS
  - Low security (Everyone can access it without any kind of authentication)

- **NoSql**

  NoSQL databases (aka "not only SQL") are non tabular, and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, **key-value**, **wide-column**, and **graph**. They provide flexible schemas and scale easily with large amounts of data and high user loads.

  It covers the disadvantages of both DBMS and FMS

  NoSql => FMS + DBMS

  Advantages of NoSql
  - High performance
  - High Security

- Installation procedure of XAMPP | Oracle

  Install XAMPP (Linux)

  - For linux operating system, We've to download the software from official website (https://www.apachefriends.org/download.html)

  Open Terminal and enter:

  `sudo tar xvfz xampp-linux-1.7.7.tar.gz -C /opt`

  (replace *xampp-linux-1.7.7.tar.gz* with the version of xammp you downloaded). It has been reported that the MYSQL database of xampp 1.7.4 does not work with Joomla 1.5.22

  This installs ... Apache2, mysql and php5 as well as an ftp server.

  sudo /opt/lampp/lampp start

  And

  sudo /opt/lampp/lampp stop

  Install XAMPP in WINDOWS

  - For Windows operating system, We've to download the software from official website (https://www.apachefriends.org/download.html)
  - We can install the software by double clicking on the downloaded software. And we have to follow a few steps according to steps.

  Starting with the SQL environment

  - PHPMyAdmin includes sql environment. It provides us to run sql queries
  - By using URL (http://localhost/phpmyadmin) from your favorite browser to visit the sql environment.

- Database (Collection of tables)
    - CREATE DATABASE <database name>

      Creating a database does not select it for use; you must do that explicitly. To make menagerie the current database, use this statement:

      ```
      mysql> USE <database name>
      ```

- Tables

  Creating the database is the easy part, but at this point it is empty, as **SHOW TABLES** tells you:

  ```
  mysql> SHOW TABLES;
  Empty set (0.00 sec)
  ```

  Syntax for creating table

  ```
  mysql> CREATE TABLE <table_name> (name VARCHAR(20), email VARCHAR(60),species VARCHAR(20), gender CHAR(1), birth DATE);
  ```

  Getting list of Tables

  ```
  mysql> SHOW TABLES;
  +--------------------+
  | Tables in menagerie |
  +--------------------+
  | <table_name>       |
  +--------------------+
  ```

- Varchar vs varchar2
    - Varchar takes empty values too.
    - Varchar2 doesn't allow empty values or null values.

- Data Types

MySQL DATA TYPES

DATE TYPE	SPEC	DATA TYPE	SPEC
CHAR	String (0 - 255)	INT	Integer (-2147483648 to 214748-3647)
VARCHAR	String (0 - 255)	BIGINT	Integer (-9223372036854775808 to 9223372036854775807)
TINYTEXT	String (0 - 255)	FLOAT	Decimal (precise to 23 digits)
TEXT	String (0 - 65535)	DOUBLE	Decimal (24 to 53 digits)
BLOB	String (0 - 65535)	DECIMAL	"DOUBLE" stored as string
MEDIUMTEXT	String (0 - 16777215)	DATE	YYYY-MM-DD
MEDIUMBLOB	String (0 - 16777215)	DATETIME	YYYY-MM-DD HH:MM:SS
LONGTEXT	String (0 - 4294967295)	TIMESTAMP	YYYYMMDDHHMMSS
LONGBLOB	String (0 - 4294967295)	TIME	HH:MM:SS
TINYINT	Integer (-128 to 127)	ENUM	One of preset options
SMALLINT	Integer (-32768 to 32767)	SET	Selection of preset options
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN	TINYINT(1)

The following table shows the string data types in MySQL:

String Types	Description
CHAR	A fixed-length nonbinary (character) string
VARCHAR	A variable-length non-binary string
BINARY	A fixed-length binary string
VARBINARY	A variable-length binary string
TINYBLOB	A very small BLOB (binary large object)
BLOB	A small BLOB

MEDIUMBLOB	A medium-sized BLOB
LONGBLOB	A large BLOB
TINYTEXT	A very small non-binary string
TEXT	A small non-binary string
MEDIUMTEXT	A medium-sized non-binary string
LONGTEXT	A large non-binary string
ENUM	An enumeration; each column value may be assigned one enumeration member
SET	A set; each column value may be assigned zero or more SET members

The following table illustrates the MySQL date and time data types:

Date and Time Types	**Description**
DATE	A date value in CCYY-MM-DD format
TIME	A time value in hh:mm:ss format
DATETIME	A date and time value inCCYY-MM-DD hh:mm:ss format
TIMESTAMP	A timestamp value in CCYY-MM-DD hh:mm:ss format

YEAR	A year value in CCYY or YY format

Part - II

Dropping

```
DROP DATABASE <database name>    (Dropping a DataBase)
DROP TABLE <table name>   (Dropping a Table)
```

Inserting Data into Table

```
mysql> INSERT INTO <table_name>
VALUES ('Hanuman
Kumar','hanumankumar@gmail.com','hamster','m','1994-09-02');
```

- Operators :
  - Mathematical or Arithmetic
    - +, -, *, /, %
  - Logical
    - AND
    - OR
    - NOT
  - Comparison Operators and
    - =, !=,<>,<,>,<=,>=, IN, NOT IN, BETWEEN, NOT BETWEEN, EXISTS, NOT EXITS, LIKE, NOT LIKE
  - Set Operators
    - UNION, UNION ALL, MINUS, INTERSECT

- Numerical Functions :
  - ABS(-1) = 1
  - CEIL
    - 123.456 => 124
  - FLOOR
    - 123.456 => 123

  - LN
  - MOD (10,3) =>1
  - POWER(2,3) => 8

- ROUND(1.2345)=>1
- SQRT(25) => 5

Aggregation :

An aggregate function performs a calculation on multiple values and returns a single value.
For example, you can use the AVG() aggregate function that takes multiple numbers and returns the average value of the numbers.

- MAX()

  The MySQL MAX() function returns the maximum value in a set of values. The MAX() function comes in handy in many cases such as finding the greatest number, the most expensive product, and the largest payment from customers.

  ```
  SELECT MAX(<column name>) FROM <table name>
  ```

- MIN()

  The MIN() function returns the minimum value in a set of values. The MIN() function is very useful in some scenarios such as finding the smallest number, selecting the least expensive product, or getting the lowest credit limit.

  ```
  SELECT MIN(<column name>) FROM <table name>
  ```

- SUM()

  The SUM() function is an aggregate function that allows you to calculate the sum of values in a set.

  ```
  SELECT SUM(<column name>) FROM <table name>
  ```

- COUNT

  The COUNT() function is an aggregate function that returns the number of rows in a table. The COUNT() function allows you to count all rows or only rows that match a specified condition.

The COUNT() function has three forms: COUNT(*), COUNT(expression) and COUNT(DISTINCT expression).

COUNT(*) function

The COUNT(*) function returns the number of rows in a result set returned by a SELECT statement. The COUNT(*) returns the number of rows including duplicate, non-NULL and NULL rows.

COUNT(expression)

The COUNT(expression) returns the number of rows that do not contain NULL values as the result of the expression.

COUNT(DISTINCT expression)

The COUNT(DISTINCT expression) returns the number of distinct rows that do not contain NULL values as the result of the expression.

The return type of the COUNT() function is BIGINT. The COUNT() function returns 0 if there is no matching row found.

```
SELECT COUNT(*) FROM <table name>
```

```
SELECT COUNT(<column name>) FROM <table name>
```

MySQL supports the following aggregate functions:

Aggregate function	Description
AVG()	Return the average of non-NULL values.
BIT_AND()	Return bitwise AND.
BIT_OR()	Return bitwise OR.

BIT_XOR()	Return bitwise XOR.
COUNT()	Return the number of rows in a group, including rows with NULL values.
GROUP_CONCAT()	Return a concatenated string.
JSON_ARRAYAGG()	Return result set as a single JSON array.
JSON_OBJECTAGG()	Return result set as a single JSON object.
MAX()	Return the highest value (maximum) in a set of non-NULL values.
MIN()	Return the lowest value (minimum) in a set of non-NULL values.
STDEV()	Return the population standard deviation.
STDDEV_POP()	Return the population standard deviation.
STDDEV_SAMP()	Return the sample standard deviation.
SUM()	Return the summation of all non-NULL values a set.
VAR_POP()	Return the population standard variance.
VARP_SAM()	Return the sample variance.
VARIANCE()	Return the population standard variance.

Part - III

Sql Commands

- Data Definition Language
- The commands are reflects the structure of the table
    - CREATE
    - DROP
    - ALTER
        - Syntax : ALTER TABLE employee RENAME COLUMN name TO names (Oracle)
        - Syntax : ALTER TABLE employee CHANGE <old column name> <new column name along with datatype > (mySql)
        - Deleting the column : ALTER TABLE <table name> DROP COLUMN <column-name>
        - ALTER TABLE <table_name> ADD <column-name> <data_type with size>

Part - IV

    - RENAME : RENAME TABLE <old table name> TO <new table name>
    - TRUNCATE
        - Syntax: TRUNCATE TABLE employees
- Data Manipulation Language (DML):
    - INSERT
    - DELETE
        - Syntax : DELETE FROM <table_name> WHERE (clause) <condition>
        - Example:
            - `DELETE FROM employees WHERE id=1`
    - UPDATE
        - Syntax : UPDATE employees SET <column_name>=<value> WHERE <column_name>=<value>
        - Example:
            - `UPDATE employees SET name='Ranga' WHERE id=1`
            - `UPDATE employees SET name="Poojitha" WHERE id=1 AND name="Ranga"`

Auto Increment:

1 .The column must be initialized with any kind of key constraints

SYntax: ALTER TABLE <table_name> ADD PRIMARY KEY (<column_name>)

2. The column value should not be empty

Syntax :

ALTER TABLE <table_name> MODIFY <column_name> <datatype with length> NOT NULL AUTO_INCEREMT

Example:

```
ALTER TABLE employees MODIFY id integer(20) NOT NULL AUTO_INCREMENT
```

** Constraints:

These are the rules for enforcing on specific field (columns)

- NOT NULL
- DEFAULT
- PRIMARY KEY
    - 1. 1 table can have only one primary key
    - 2. The value should not be empty
    - 3. Won't allow duplicate values
- UNIQUE KEY
- FORIEGN KEY

Part - V

- DCL (Data Control Language)
    - GRANT
        - Granting Permissions to specific user.
    - REVOKE
        - Withdrawing the permissions which were given using GRANT command.
- TCL (Transaction Control Language)
    - COMMIT
    - ROLLBACK
    - SAVEPOINT

LIKE clause:

The LIKE operator is a logical operator that tests whether a string contains a specified pattern or not.

The LIKE operator is used in the WHERE clause of the SELECT , DELETE, and UPDATE statements to filter data based on patterns.

MySQL provides two wildcard characters for constructing patterns: percentage % and underscore _ .

> The percentage (%) wildcard matches any string of zero or more characters.
> The underscore (_) wildcard matches any single character.

For example, s% matches any string starts with the character s such as sun and six. The se_ matches any string starts with se and is followed by any character such as see and sea

Example :
`SELECT * FROM employees WHERE name LIKE 'K%'` (Retrieving data where name starts with character '**K**')

`SELECT * FROM employees WHERE name LIKE '%a%'` (Retrieving data where name contains character '**a**')

Part - VI

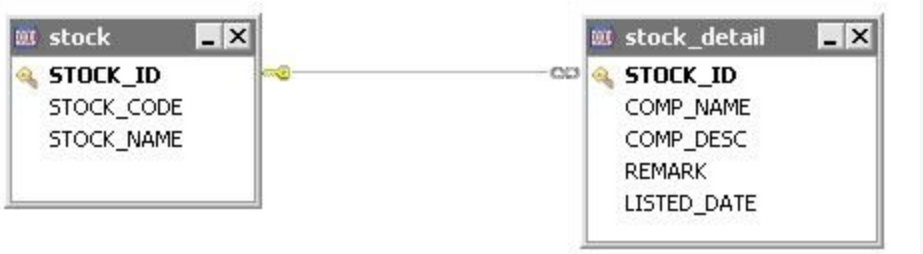
Relationships

- One-to-one

  One-to-one relationships occur when there is exactly one record in the first table that corresponds to exactly one record in the related table.

  MySQL does not contains any "ready" options to define the one-to-one relationship, but, if you want to enforce it, you can add a foreign key from one primary key to the other primary key, by doing this, both tables will have the one-to-one relationship automatically.

  Here's an example to define a one-to-one relationship in MySQL.



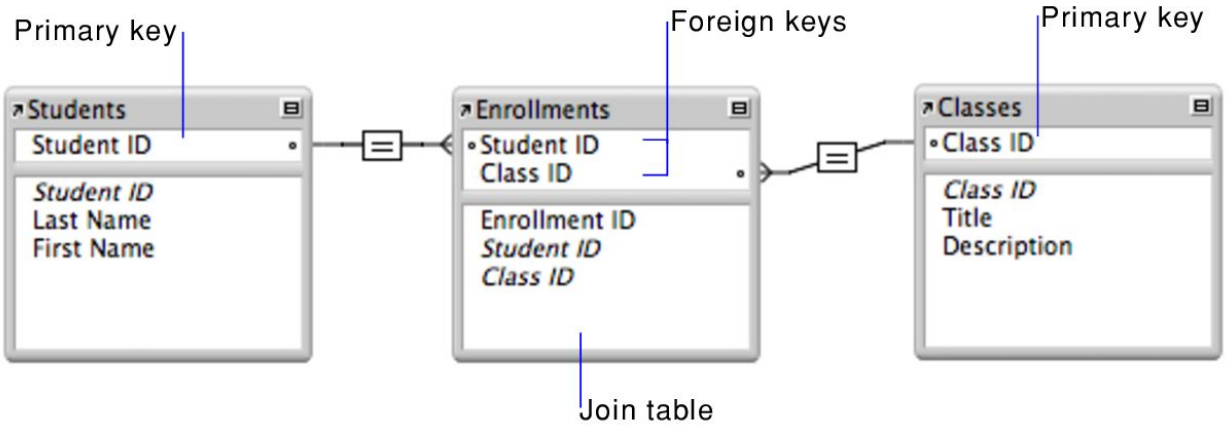
- One-to-many

  CREATE VIEW viewAreas AS SELECT * FROM Areas, Maps WHERE Areas.ID = Maps.AreaID;

- Many-to-many

  A *many-to-many relationship* occurs when multiple records in a table are associated with multiple records in another table. For example, a many-to-many relationship exists between customers and products: customers can purchase various products, and products can be purchased by many customers.

  The following example includes a Students table, which contains a record for each student, and a Classes table, which contains a record for each class. A join table, Enrollments, creates two one-to-many relationships—one between each of the two tables.



The primary key Student ID uniquely identifies each student in the Students table. The primary key Class ID uniquely identifies each class in the Classes table. The Enrollments table contains the foreign keys Student ID and Class ID.

Example

Table 1: (Primary Key)

- The values should be unique
- One table can have only one primary key
- Primary key doesn't allow any NULL values.

```
USE apssdc;

CREATE TABLE employees (
id int(20) AUTO_INCREMENT,
name varchar(30),
email varchar(70),
PRIMARY KEY(id))
```

Table2 : (Foriegn Key)

```
CREATE TABLE salaries(
id int(20) NOT NULL,
salary int(20),
FOREIGN KEY(id) REFERENCES employees(id)
)
```

Example :

Students

```
Roll name email
2    ccc  ccc@gmail.com
```

Marks

```
Roll marks
2    40
```

ON DELETE cascade: (REFERENTIAL INTEGRITY)

```
CREATE TABLE salaries(
```

```
id int(20), salary int(30),
FOREIGN KEY(id) REFERENCES employees(id) ON DELETE CASCADE )
```

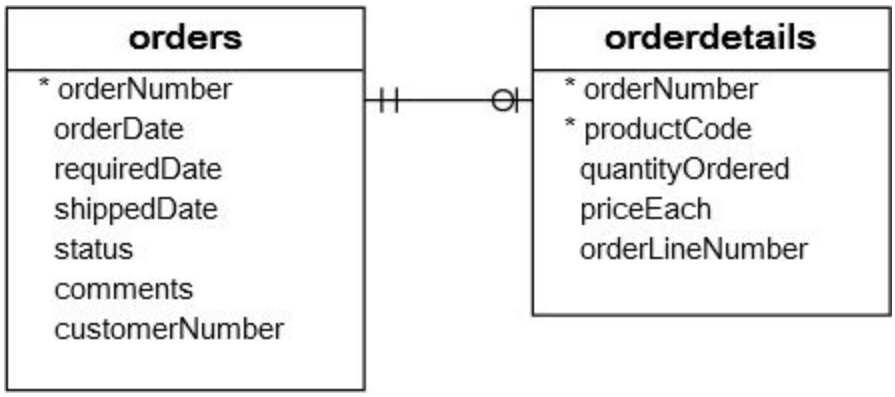
UNIQUE KEY

- Same as primary key (Doesn't allow duplicate values)
- We can specify multiple unique attributes for a single table.

Part - VII

Joins in SQL

A relational database consists of multiple related tables linking together using common columns which are known as foreign key columns. Because of this, data in each table is incomplete from the business perspective.
For example, in the sample database, we have the orders and orderdetails tables that are linked using the orderNumber column:



To get complete orders' information, you need to query data from both orders and orderdetails tables.

That's why joins come into the play.

A join is a method of linking data between one (self-join) or more tables based on values of the common column between the tables.

MySQL supports the following types of joins:

1. Inner join

2. Left join
3. Right join
4. Full join

To join tables, you use the cross join, inner join, left join, or right join clause for the corresponding type of join. The join clause is used in the SELECT statement appeared after the FROM clause.

- Inner Join

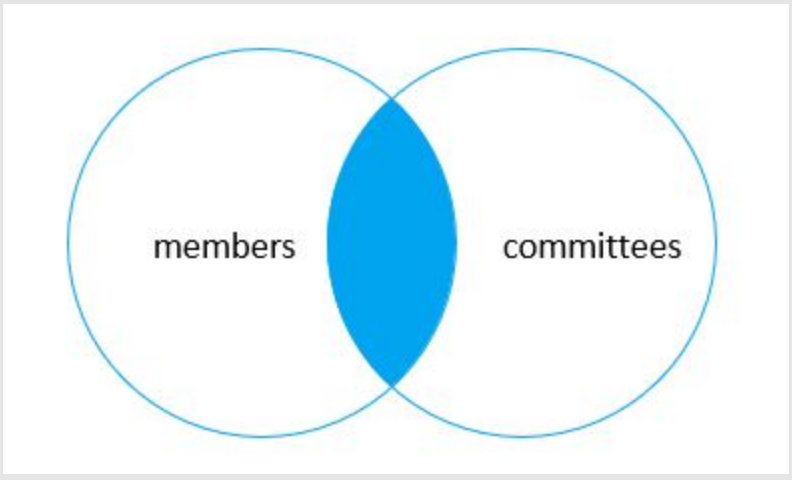
  The inner join clause joins two tables based on a condition which is known as a join predicate.

  The inner join clause compares each row from the first table with every row from the second table. If values in both rows cause the join condition evaluates to true, the inner join clause creates a new row whose column contains all columns of the two rows from both tables and include this new row in the final result set. In other words, the inner join clause includes only rows whose values match.

  - For getting common values between two tables
  - Example:

    ```
    SELECT
    employee.id,
    employee.name,
    salaries.salary,
    employee.address FROM employee
    INNER JOIN salaries ON employee.id=salaries.id
    ```

    The following Venn diagram illustrates the inner join:



- Left Join

  Similar to an inner join, a left join also requires a join-predicate. When two tables using a left join, the concepts of left and right tables are introduced.

  The left join selects data starting from the left table. For each row in the left table, the left join compares with every row in the right table. If the values in the two rows cause the join condition evaluates to true, the left join creates a new row whose columns contain all columns of the rows in both tables and includes this row in the result set.

  If the values in the two rows are not matched, the left join clause still creates a new row whose columns contain columns of the row in the left table and NULL for columns of the row in the right table.

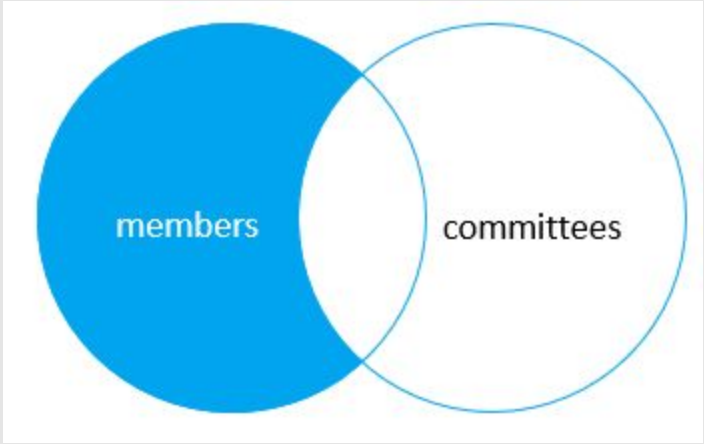
  In other words, the left join selects all data from the left table whether there are matching rows exist in the right table or not. In case there is no matching rows from the right table found, NULLs are used for columns of the row from the right table in the final result set.

  - It returns data from left table even the data is not available in right table
  - Example :

```
SELECT
employee.name,
Salaries.salary,
```

```
employee.address FROM employee
LEFT JOIN salaries ON employee.id=salaries.id
```

This Venn diagram illustrates how to use the left join to select rows that only exist in the left table:



- Right Join

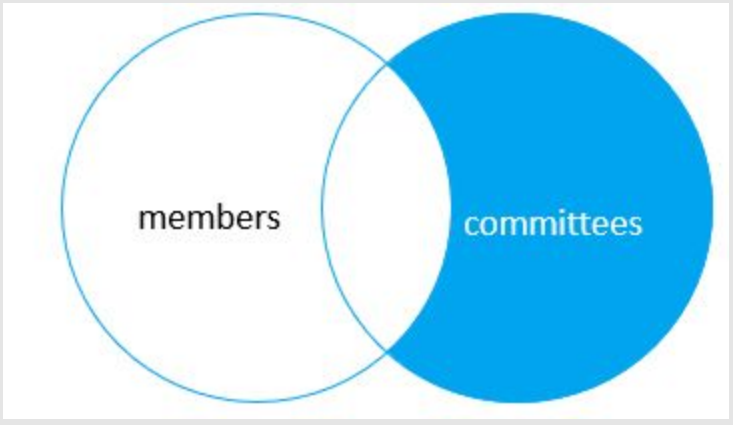
  The right join clause is similar to the left join clause except that the treatment of tables is reversed. The right join starts selecting data from the right table instead of the left table.

  The right join clause selects all rows from the right table and matches rows in the left table. If a row from the right table does not have matching rows from the left table, the column of the left table will have NULL in the final result set.

  - It returns data from right table even the data is not available in left table
  - Example :

    ```
    SELECT employee.name,
    Salaries.salary,
    employee.address FROM employee
    RIGHT JOIN salaries ON employee.id=salaries.id
    ```

This Venn diagram illustrates how to use the right join to select data that exists only in the right table:



- Full join

  Unlike the inner join, left join, and right join, the cross join clause does not have a join condition. The right join makes a Cartesian product of rows from the joined tables. The cross join combines each row from the first table with every row from the right table to make the result set.

  Suppose the first table has n rows and the second table has m rows. The cross join that joins the first with the second table will return nxm rows.

  - Left Join + Right Join
  - The result will contain all records from both tables, and fill in Null for missing matches.
  - Example :

```
SELECT employee.name,
salaries.salary,
employee.address FROM employee
RIGHT JOIN salaries ON employee.id=salaries.id
UNION
SELECT employee.name,
salaries.salary, employee.address FROM employee
LEFT JOIN salaries ON employee.id=salaries.id
```

- Storing Resulting data after applying different kinds of Joins

- Example:
  ```
  CREATE TABLE result AS
  SELECT employee.name,
  salaries.salary,
  employee.address FROM employee
  LEFT JOIN salaries ON employee.id=salaries.id
  ```

Importance of HAVING clause

- Selecting data from a table by filtering based on condition

Part - VIII

Clauses in SQL :

- HAVING

  The HAVING clause is used in the SELECT statement to specify filter conditions for a group of rows or aggregates.

  The HAVING clause is often used with the GROUP BY clause to filter groups based on a specified condition. If the GROUP BY clause is omitted, the HAVING clause behaves like the WHERE clause.

  ```
  SELECT product, SUM(quantity) AS "Total quantity" FROM order_details
  GROUP BY product HAVING SUM(quantity) > 10;
  ```

- GROUP BY
  - We can get data from multiple records and make them into one column.
  - Example :

    ```
    1.SELECT name,COUNT(address) FROM employee GROUP BY address
    2.SELECT name,SUM(marks) AS Total_marks FROM marks GROUP BY
    name
    3.SELECT name,MAX(marks) AS Total_marks FROM marks GROUP BY
    name
    ```

- ORDER BY
  - To sort the data (ascending order, descending order)
  - ASC - Ascending Order
  - DESC - Descending Order

Example :

1. SELECT name,address FROM employee ORDER BY name ASC
2. SELECT name,address FROM employee ORDER BY name DESC

Note : By Default we retrieve information in ascending order by using ORDER BY.

- WHERE
- SELECT
- FROM
- DISTINCT
  - For getting unique values from the table
  - Example :
    SELECT DISTINCT(id),salary FROM salaries

Relationships:

- One - one
- One - Many
  - Relationship between one field in one table and many fields in another table
  - Example :
    SELECT E.name,M.subject,M.marks FROM employee E RIGHT JOIN marks M ON E.name=M.name
- Many - One
  - Example :
    SELECT E.name,E.address,SUM(M.marks) AS Total_marks FROM employee E INNER JOIN marks M ON E.name=M.name GROUP BY E.name

    2. SELECT E.name, M.subject,M.marks,S.salary FROM employee E, marks M,salaries S WHERE E.id=M.id and E.id=S.id

Part - IX

LIMIT :
SELECT * FROM employee LIMIT 3

Example 2:
SELECT * FROM employee ORDER BY id DESC LIMIT 3

Example 3:
SELECT * FROM(SELECT * FROM employee ORDER BY id DESC LIMIT 3) employee ORDER BY id ASC

Wildcards :
%

R**% =>** The value must be starts with ‘R’
%R => The value must be ends with ‘R’
%R% => The value needs to have the character of ‘R’.

_

Matching the positional values.

Example :

```
SELECT * FROM `employee` WHERE name LIKE '_a%'
```

```
SELECT * FROM `employee` WHERE name LIKE '_%a'
```
(Matching at last character)

[]

This is used for matching any character in the given structure
‘[abc]%’ => The value must be starts with ‘a’ or ‘b’ or ‘c’

‘[!abc]%’ => The string should not start with ‘a’ or ‘b’ or ‘c’.

‘[a-z]%’ => The string should start with any kind of alphabet.

VIEWS :

This is a virtual table we can store the value by getting those from an expression

Example :

```
CREATE VIEW data AS SELECT name FROM employee;
```

The manipulating commands are similar to the commands in table.

Normalization :

We can break the complex table into pieces (Normalization)

1. Reducing redundancy of Data
2. Data integrity (Primary Key, Forien Key)
    a. Problems without normalization
        i. Insertion anomaly
        ii. Update anomaly
        iii. Deletion anomaly

Kinds of Normal forms:

- 1NF =>Need to avoid multiple values in a cell. A Single cell can't hold multiple values.
- 2NF =>
    - Must be satisfied 1st Normal form
    - Have to create multiple tables for assigning primary keys efficiently

- 3NF =>
    - Must be satisfied with the 2nd NF.
    - Non - primary key value should not be depend upon a non-primary key value

- BCNF => (Boyce–Codd normal form)
    - Must be satisfied with 3NF
        - Student name and subject
        - Teacher and subject
    -
        - Student and Professor in one table
        - Subjects in another table

Transaction management:

Atomicity
Consistency
Isolation
Durability

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

- **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of

a transaction, it must remain consistent after the execution of the transaction as well.

- **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

- **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.