**DATABASE MANAGEMENT SYSTEM**

**LAB REPORT**

**PREPARED BY:**

STUDENT MIS:- 112015044

STUDENT NAME:- ETCHERLA SAI MANOJ

**SUBMITTED TO:**

DR. SANJEEV SHARMA

**

**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, PUNE**

**DECEMBER 2021**

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Title** | | **Page No.** |
| 1 | Introduction to DBMS | | 3 |
| 2. | Hardware and Software Requirements | | 5 |
| 3. | Introduction to SQL | | 5 |
| 4. | Schema of sample database with E-R Diagram | | 7 |
| 5 | SQL Commands | | 9 |
|  | 5.1 | CREATE | 9 |
|  | 5.2 | DROP | 9 |
|  | 5.3 | TRUNCATE | 10 |
|  | 5.4 | ALTER | 10 |
|  | 5.5 | NOT NULL | 11 |
|  | 5.6 | UNIQUE | 12 |
|  | 5.7 | CHECK | 12 |
|  | 5.8 | PRIMARY KEY | 13 |
|  | 5.9 | FOREIGN KEY | 13 |
|  | 5.10 | INSERT | 14 |
|  | 5.11 | UPDATE | 15 |
|  | 5.12 | DELETE | 15 |
|  | 5.13 | SELECT | 15 |
|  | 5.14 | DISTINCT | 16 |
|  | 5.15 | ORDER BY | 17 |
|  | 5.16 | GROUP BY | 17 |
|  | 5.17 | HAVING | 18 |
|  | 5.18 | AND, OR, NOT | 19 |
|  | 5.19 | BETWEEN, IN | 20 |
|  | 5.20 | LIKE OPERATOR | 21 |
|  | 5.21 | AGGREGATE FUNCTIONS | 22 |
|  | 5.22 | USE OF JOIN | 22 |

**1.Introduction to DBMS**

Database Management System (DBMS) is a software tool to organize data in a database. Basic terminology related to DBMS: -

* **Data:** Data is known facts that can be recorded and have implicit meaning.
* **Database:** A database is a collection of tables, with related data.
* **Table:** A table is a matrix with data. A table in a database looks like a simple spreadsheet.

These systems are designed to manage large bodies of information which includes both defining structures for storage of information and providing mechanisms for the manipulation of information.

**Why DBMS?**

In early days, file systems were used to maintain databases. These systems have lot of drawbacks compared to DBMS.

Drawbacks of file systems:

* **Data redundancy:** refers to duplication of data. If a duplicate of data needed to be stored in database extra space would be needed.
* **Data inconsistency:** Data redundancy leads to data inconsistency. If data needed to be updated it should also be updated in all duplicates.
* **Data isolation:** different files have different file formats, retrieving data is difficult.
* **Dependency on application programs:** changing files would lead to change in application programs.
* **Atomicity issues:** Atomicity of transaction refers to “All or nothing”, which means either all the operations in a transaction executes or none.
* **Data security:** Data should be secured from unauthorized access.

DBMS provides solution for all these types of drawbacks in file systems.

Advantages of DBMS over file systems:

* **No redundant data:** Redundancy removed by data normalization.
* **Data consistency and Integrity:** Since data redundancy is removed by data normalization, data inconsistency is also has also been taken care of as a part of it.
* **Data Security:** It is easier to apply constraints in database systems so that only authorized user is able to access the data.
* **Privacy:** Limited access means privacy of data of individual user.
* **Easy access of data:** DBMS manages data in a such way so that the data is easily accessible with fast response times.
* **Easy recovery:** DBMS keeps backup data, so it is easier to do a full recovery of data.
* **Flexible:** DBMS are more flexible than file processing systems.

Though DBMS has a lot of advantages. It comes with its own disadvantages.

Disadvantages of DBMS:

* **High Cost:** DBMS implementation cost is high compared to file system.
* **Complexity:** Database systems are complex to understand.
* **Performance:** Database systems are generic, making them suitable for various applications. But these wide range of features affect their performance.

Beside of these disadvantages, it makes management of data a lot easier than file systems. So, DBMS gained a large popularity over file systems and used today everywhere.

**What describes a database?**

* **Schema:** the basic structure of how the data will be stored in the database is called schema. It is overall description of the database.
* **Instance:** the collection of information stored in a database at a particular point of time.

**Building blocks of DBMS**

* **Entities:** Entities are real time objects that exist, represented by rectangle box containing the entity name in it.
* **Attributes:** The set of characteristics representing an entity, representing ellipse symbol with attribute name on it.
* **Relationship:** describes association between two entities, represented using diamond symbol containing relation name with it.
* **Constraints:** Constraints are conditions applied on the data. It provides the data integrity.

**Database Applications**

* Railway Reservation System
* Library Management System
* Banking
* Universities and Colleges
* Credit Card Transactions

And many more…

**Data Abstraction in DBMS**

Abstraction basically means hiding irrelevant data from user. Since Database Systems are made up of complex data structures. It is necessary to hide unnecessary details from users to ease the user interaction with database.

There are three levels of data abstraction in DBMS:

* **Physical level:** This is lowest level of data abstraction. This level describes how data is stored in database. One can get the complex data structure details at this level like schema of database etc.
* **Logical level:** This is next higher or middle level of data abstraction. This level describes what data is stored in database.
* **View level:** Highest level of data abstraction. This level describes the user interaction with the database. This can be used by all users and is least complex and easy to understand.

**2. Hardware and Software Requirements**

**Hardware**

Talking about hardware, we mean computer, hard disks, Input/Output channels for data, and any physical component involved before any data is successfully stored into the memory. When we run database software like Oracle or MySQL on our PC, then our computer’s hard disk, keyboard which we type in all commands, RAM, ROM, all become part of DBMS hardware.

**Software**

This is the main component, as this is the program which controls everything. The DBMS software is more like a wrapper around the physical database, which provides us with an easy-to-use interface to store, access and update data. The DBMS software is capable of understanding the Database Access Language and interpret it into actual database commands to execute them on the DB. For example, Oracle, My SQL, Microsoft SQL, etc.

**3. Introduction to SQL**

* Structured Query Language (SQL) is a database query language used for storing and managing data in databases.
* SQL is a standard relational database management language.
* The first commercial DBMS that supported SQL was Oracle in 1979.
* SQL is a “nonprocedural” or “declarative” language.
* SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.
* SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce after learning about the relational model from Edgar F. Codd in the early 1970s.

SQL have variety of commands for variety of tasks. These commands can be classified into four types as below.

**Classification of SQL Commands**

DBMS provides us with different languages and interfaces to express database queries and updates. Database languages can be used to read, store, and update the data in the database.

* **Data Definition Language (DDL):** This language offers a set of operations which are used to classify the database structure or schema. Tasks that come under DDL are:

Command Description

CREATE - It creates objects in the database.

ALTER - It alters structure(schema) of database.

DROP - It is used to delete objects from the database.

TRUNCATE - It is used to remove all records from a table, including all spaces allocated for records in database.

COMMENT - It is used to add comments to data dictionary.

RENAME - It is used to rename objects in the database.

* **Data Manipulation Language (DML):** This language offers a set of operations which are used to manipulate data int database and handle user requests. Tasks that come under DML are:

Command Description

SELECT - It retrieves data from a database.

INSERT - It inserts data into a table.

UPDATE - It updates existing data in the table.

DELETE - It deletes all records from table, but spaces for records remain undeleted.

MERGE - It is kind of UPSERT operation(insert or update)

* **Data Control Language:** This language offers a set of operations which are used to grant privilege or revoke privilege from a user. Tasks that come under DCL are:

Command Description

GRANT - It gives privilege to user to access the database.

REVOKE - It is used to take back the privilege given to user to access the database.

* **Transaction Control Language:** This language offers a set of operations which are used to run changes made by DML statement, basically manages transactions. Tasks that come under TCL are:

Command Description

COMMIT - It is used to save the work.

SAVEPOINT - It is used to set a point in transaction to roll back later. Basically, marks a transaction with certain name.

ROLLBACK - It is used to restore since last commit.

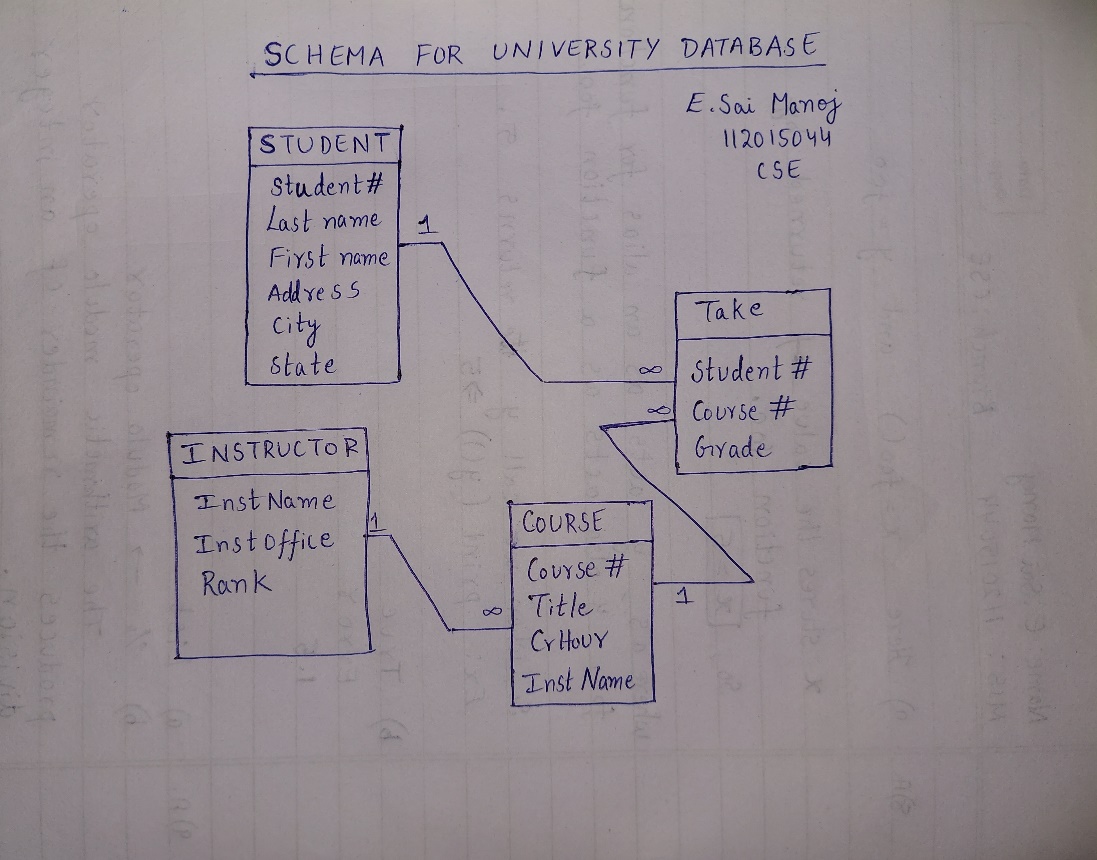
**Commonly used Datatypes in SQL**

* **INT:** used for columns which store integer values.
* **FLOAT:** used for columns which store float values.
* **DOUBLE:** used for columns which store float values but with higher precision.
* **VARCHAR:** used for columns which store combination of characters and integers, simply a string.
* **CHAR:** used for columns which store character values, only single character.
* **DATE:** used for columns which store date values.
* **TEXT:** used for columns which store text, generally text long in length, like a sentence/paragraph.

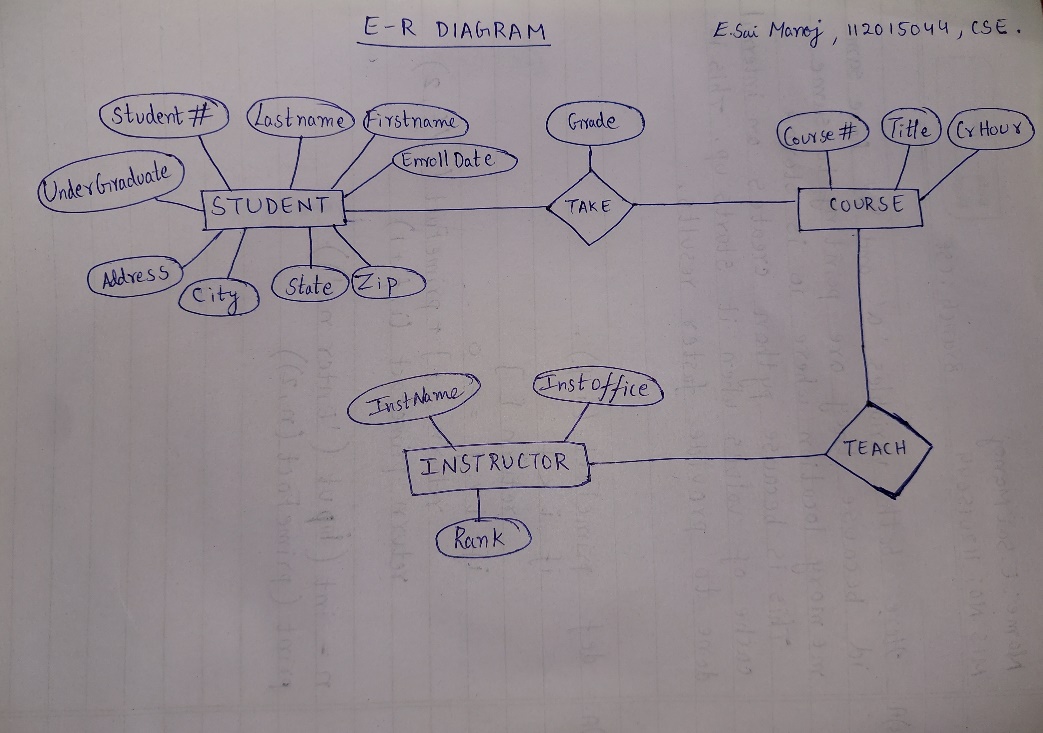
**4. Schema of sample database with E-R Diagram**

**Example:**

Schema for University Students Database:

****

**E-R DIAGRAM:**

****

**Tables:**

STUDENT(Student#, LastName, FirstName, Address, City, State, Zip, Enroll\_Date, Undergrad?)

COURSE(Course#, Title, CrHour, InstName)

INSTRUCTOR(InstName, InstOffice, Rank)

Take(Student#, Course#, Grade)

**5. SQL Commands**

1. **CREATE:**

CREATE command is used to create new database. Also, it can create new table in a database.

Creating a database:

**Syntax:** CREATE DATABASE <DB\_NAME>;

**Example:** CREATE DATABASE College;

Creating a Table:

**Syntax:** CREATE TABLE <TABLE\_NAME>

**(**

column\_name1 datatype1;

column\_name2 datatype2;

column\_name3 datatype3;

);

**Example:** CREATE TABLE Student

**(**

stu\_id INT;

stu\_name VARCHAR(100);

stu\_age INT;

);

1. **DROP:**

DROP command completely removes a table from the database. It also destroys the table structure and the data stored in it. It can also be used on Databases, to delete the complete database.

**Syntax:** DROP TABLE table\_name;

**Example:** DROP TABLE Student;

1. **TRUNCATE:**

TRUNCATE command removes all the records from a table along with spaces allotted for records. But this command will not destroy table structure and the data stored in it. When we use TRUNCATE command on a table its (auto-incremented) primary key is also initialized.

**Syntax:** TRUNCATE TABLE table\_name;

**Example:** TRUNCATE TABLE Student;

1. **ALTER:**

ALTER command is used for altering table structure, such as

* To add a column to existing table

**Syntax:**

ALTER TABLE table\_name ADD(

column\_name datatype);

**Example:**

ALTER TABLE Student ADD(

address VARCHAR(200)

);

* To rename any existing column.

**Syntax:**

ALTER TABLE table\_name RENAME

old\_column\_name TO new\_coumn\_name;

**Example:**

ALTER TABLE table\_name RENAME

address TO location;

* To change datatype of any column or to modify its size

**Syntax:**

ALTER TABLE table\_name MODIFY(

column\_name datatype

);

**Example:**

ALTER TABLE Student MODIFY(

address VARCHAR(300)

);

* To drop a column from the table

**Syntax:**

ALTER TABLE table\_name DROP(

column\_name);

**Example:**

ALTER TABLE Student DROP(

address);

1. **NOT NULL:**

The NOT NULL is a constraint that enforces a column NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

**Syntax:**

CREATE TABLE table\_Name

(

column1 data\_type(size) NOT NULL,

column2 data\_type(size) NOT NULL,

....

);

**Example:**

CREATE TABLE EMPLOYEES

(

empid INTEGER NOT NULL,

empname VARCHAR2(10) NOT NULL,

doj DATE

);

1. **UNIQUE:**

The UNIQUE constraint ensures that all values in a column are different. Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint. However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

**Syntax:**

SELECT table.ID

FROM table

WHERE UNIQUE (SELECT table2.ID

FROM table2

WHERE table.ID = table2.ID);

**Example:**

SELECT I.EMPLOYEEID, I.NAME

FROM Instructor as I

WHERE UNIQUE (SELECT Inst.EMPLOYEEID

FROM Instructor as Inst

WHERE I.EMPLOYEEID = Inst.EMPLOYEEID

and Inst.YEAR = 2017);

1. **CHECK:**

Check Constraint is used to specify a predicate that every tuple must satisfy in a given relation. It limits the values that a column can hold in a relation.

**Syntax:**

CREATE TABLE pets(

ID INT NOT NULL,

Name VARCHAR(30) NOT NULL,

Breed VARCHAR(20) NOT NULL,

Age INT,

GENDER VARCHAR(9),

PRIMARY KEY(ID),

check(GENDER in ('Male', 'Female', 'Unknown'))

);

**Example:**

CREATE TABLE student(

StudentID INT NOT NULL,

Name VARCHAR(30) NOT NULL,

Age INT NOT NULL,

GENDER VARCHAR(9),

PRIMARY KEY(ID),

check(Age >= 17)

);

1. **PRIMARY KEY:**

A primary key constraint depicts a key comprising one or more columns that will help uniquely identify every tuple/record in a table.

**Syntax:**

Create Table Person

(

Id int NOT NULL PRIMARY KEY,

Name varchar2(20),

Address varchar2(50)

);

1. **FOREIGN KEY:**

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables. A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

**Syntax:**

Create table people (no int references person(id),

Fname varchar2(20));

**Example:**

|  |  |  |
| --- | --- | --- |
| c\_id | Coustmer\_Name | address |
| 101 | Adam | Noida |
| 102 | Alex | Delhi |
| 103 | Stuart | Rohtak |

Coustmer \_detail

Order\_Detail

|  |  |  |
| --- | --- | --- |
| Order\_id | Order\_Name | c\_id |
| 10 | Order1 | 101 |
| 11 | Order2 | 103 |
| 12 | Order3 | 102 |

In Customer\_Detail table, c\_id is the primary key which is set as foreign key in Order\_Detail table.

FOREIGN KEY constraint at table level:

CREATE table Order\_Detail(

Order\_id int PRIMARY KEY,

Order\_Name varchar(60) NOT NULL,

c\_id int FOREIGN KEY REFERENCES Customer\_Detail(c\_id)

);

In this query, c\_id in table Order\_Detail is made as foreign key, which is a reference of c\_id column in Customer\_Detail table.

1. **INSERT:**

INSERT command is used to insert data into a table.

**Syntax:** INSERT INTO table\_name VALUES(data1, data2, ...)

**Example:**

Consider a table student with the following fields.

|  |  |  |
| --- | --- | --- |
| s\_id | name | age |

**Query:** INSERT INTO student VALUES(101, 'Adam', 15);

|  |  |  |
| --- | --- | --- |
| s\_id | name | age |
| 101 | Adam | 15 |

1. **UPDATE:**

UPDATE command is used to update any record of data in a table

**Syntax:** UPDATE table\_name SET column\_name = new\_value WHERE some\_condition;

**Example:**

|  |  |  |
| --- | --- | --- |
| student\_id | name | age |
| 101 | Adam | 15 |
| 102 | Alex |  |
| 103 | Chris | 14 |

**Query:** UPDATE student SET age=18 WHERE student\_id=102;

|  |  |  |
| --- | --- | --- |
| student\_id | name | Age |
| 101 | Adam | 15 |
| 102 | Alex | 18 |
| 103 | Chris | 14 |

1. **DELETE:**

DELETE command is used to delete data from a table.

**Syntax:** DELETE FROM table\_name;

**Example:** DELETE FROM student;

The above command will delete all the records from the table student.

1. **SELECT:**

SELECT query is used to retrieve data from a table. It is the most used SQL query. We can retrieve complete table data, or partial by specifying conditions using the WHERE clause.

**Syntax:**

SELECT

column\_name1,

column\_name2,

column\_name3,

...

column\_nameN

FROM table\_name;

**Example:**

Consider a table student ,

|  |  |  |  |
| --- | --- | --- | --- |
| s\_id | name | age | address |
| 101 | Adam | 15 | Chennai |
| 102 | Alex | 18 | Delhi |

**Query:** SELECT s\_id, name, age FROM student;

The above query will fetch information of s\_id, name and age columns of the student table and display them,

|  |  |  |
| --- | --- | --- |
| s\_id | name | age |
| 101 | Adam | 15 |
| 102 | Alex | 18 |

1. **DISTINCT:**

The DISTINCT keyword is used with SELECT statement to retrieve unique values from the table. Distinct removes all the duplicate records while retrieving records from any table in the database.

**Syntax:** SELECT DISTINCT column-name FROM table-name;

**Example:** Consider a table Emp,

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 101 | Anu | 24 | 5000 |
| 102 | Shanu | 25 | 8000 |
| 103 | Jay | 23 | 10000 |
| 104 | Om | 44 | 10000 |
| 105 | James | 35 | 8000 |

**Query:** SELECT DISTINCT salary FROM Emp;

The above query will return only the unique salary from Emp table.

|  |
| --- |
| salary |
| 5000 |
| 8000 |
| 10000 |

1. **ORDER BY**

Order by clause is used with SELECT statement for arranging retrieved data in sorted order. The Order by clause by default sorts the retrieved data in ascending order. To sort the data in descending order DESC keyword is used with Order by clause.

**Syntax:** SELECT column-list|\* FROM table-name ORDER BY ASC | DESC;

**Example:** Consider a table Emp,

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 101 | Anu | 34 | 9000 |
| 102 | Shanu | 43 | 8000 |
| 103 | Jay | 23 | 6000 |

**Query:** SELECT \* FROM Emp ORDER BY salary;

The above query will return the resultant data in ascending order of the salary.

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 103 | Jay | 23 | 6000 |
| 102 | Shanu | 43 | 8000 |
| 101 | Anu | 34 | 9000 |

1. **GROUP BY:**

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

**Syntax:**

SELECT column\_name, function(column\_name)

FROM table\_name

WHERE condition

GROUP BY column\_name

**Example:** Consider a table Emp,

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 101 | Anu | 22 | 9000 |
| 102 | Shanu | 29 | 8000 |
| 103 | Rohan | 34 | 6000 |
| 104 | Scott | 44 | 9000 |
| 105 | Tiger | 35 | 8000 |

**Query:**

SELECT name, age

FROM Emp GROUP BY salary

Result will be,

|  |  |
| --- | --- |
| name | age |
| Rohan | 34 |
| Shanu | 29 |
| Anu | 22 |

1. **HAVING:**

Having clause is used with SQL Queries to give more precise condition for a statement. It is used to mention condition in Group by based SQL queries, just like WHERE clause is used with SELECT query.

**Syntax:**

SELECT column\_name, function(column\_name)

FROM table\_name

WHERE column\_name condition

GROUP BY column\_name

HAVING function(column\_name) condition

**Example:** consider a table Sale,

|  |  |  |  |
| --- | --- | --- | --- |
| oid | order\_name | previous\_balance | customer |
| 1 | Ord1 | 2000 | Alex |
| 2 | Ord2 | 1000 | Adam |
| 3 | Ord3 | 2000 | Abhi |
| 4 | Ord4 | 1000 | Adam |
| 5 | Ord5 | 2000 | Alex |

**Query:**

SELECT \*

FROM sale GROUP BY customer

HAVING sum(previous\_balance) > 3000

Result will be,

|  |  |  |  |
| --- | --- | --- | --- |
| oid | order\_name | previous\_balance | customer |
| 1 | Ord1 | 2000 | Alex |

1. **AND, OR, NOT:**

These are logical operators in SQL.

* **AND:** It returns true if both the conditions are true.

**Example:** consider a table Emp,

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 101 | Anu | 22 | 5000 |
| 102 | Shanu | 29 | 8000 |
| 103 | Rohan | 34 | 12000 |
| 104 | Scott | 44 | 10000 |
| 105 | Tiger | 35 | 9000 |

**Query:**

SELECT \* FROM Emp WHERE salary < 10000 AND age > 25

The above query will return records where salary is less than 10000 and age greater than 25.

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 102 | Shanu | 29 | 8000 |
| 105 | Tiger | 35 | 9000 |

* **OR:** It returns True if either of the condition is true.

**Example:**

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 101 | Anu | 22 | 5000 |
| 102 | Shanu | 29 | 8000 |
| 103 | Rohan | 34 | 12000 |
| 104 | Scott | 44 | 10000 |
| 105 | Tiger | 35 | 9000 |

**Query:**

SELECT \* FROM Emp WHERE salary > 10000 OR age > 25

The above query will return records where either salary is greater than 10000 or age is greater than 25.

|  |  |  |  |
| --- | --- | --- | --- |
| 102 | Shanu | 29 | 8000 |
| 103 | Rohan | 34 | 12000 |
| 104 | Scott | 44 | 10000 |
| 105 | Tiger | 35 | 9000 |

* **NOT: It returns true if both conditions are false.**

**Example:**

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 101 | Anu | 22 | 5000 |
| 102 | Shanu | 29 | 8000 |

**Query:**

SELECT \* FROM Emp WHERE NOT salary = 8000

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 101 | Anu | 22 | 5000 |

1. **BETWEEN, IN:**

* **BETWEEN:** Allow to check the value within a range.

**Syntax:**

SELECT column\_name(s)

FROM table\_name

WHERE column\_name BETWEEN value1 AND value2;

**Example:** consider a table Emp,

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 101 | Anu | 22 | 5000 |
| 102 | Shanu | 29 | 8000 |
| 103 | Rohan | 34 | 12000 |
| 104 | Scott | 44 | 10000 |
| 105 | Tiger | 35 | 13000 |

**Query:**

SELECT name

FROM Emp

WHERE Salary

BETWEEN 4000 AND 9000;

|  |
| --- |
| name |
| Anu |
| Shanu |

* **IN:** Allows to check the value within a set.

**Syntax:**

SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (list\_of\_values);

**Example:** Consider a table Emp,

|  |  |  |  |
| --- | --- | --- | --- |
| eid | name | age | salary |
| 101 | Anu | 22 | 5000 |
| 102 | Shanu | 29 | 8000 |
| 103 | Rohan | 34 | 12000 |
| 104 | Scott | 44 | 10000 |
| 105 | Tiger | 35 | 13000 |

**Query:**

SELECT name

FROM Emp

WHERE Salary IN (50000, 80000, 12000);

|  |
| --- |
| name |
| Anu |
| Shanu |
| Rohan |

1. **LIKE OPERATOR:**

Used in matching pattern of characters.

**Example: Consider a Student table,**

|  |  |  |
| --- | --- | --- |
| id | name | age |
| 1 | Alex | 14 |
| 2 | Adam | 12 |
| 3 | Jay | 13 |

**Query:** SELECT \* FROM Student WHERE name LIKE 'A%';

The above query will return all records where name starts with character 'A'.

|  |  |  |
| --- | --- | --- |
| id | name | age |
| 1 | Alex | 14 |
| 2 | Adam | 12 |

1. **AGGREGATE FUNCTIONS:**

These functions return a single value after performing calculations on a group of values. Following are some of the frequently used Aggregate functions.

* **AVG() Function:** Average returns average value after calculating it from values in a numeric column.

**Syntax:** SELECT AVG(column\_name) FROM table\_name

* **COUNT() Function:** Count returns the number of rows present in the table either based on some condition or without condition.

**Syntax:**

SELECT COUNT(column\_name) FROM table-name

* **MAX() Function:** MAX function returns maximum value from selected column of the table.

**Syntax:** SELECT MAX(column\_name) from table-name;

* **MIN() Function:** MIN function returns minimum value from a selected column of the table.

**Syntax:** SELECT MIN(column\_name) from table-name;

* **SUM() Function:** SUM function returns total sum of a selected columns numeric values.

**Syntax:** SELECT SUM(column\_name) from table-name;

1. **USE OF JOIN:**

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

* **INNER JOIN:** The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e. value of the common field will be same.

**Syntax:**

SELECT table1.column1, table1.column2, table2.column1,.…

FROM table1

INNER JOIN table2

ON table1.matching\_column = table2.matching\_column;

* **LEFT JOIN:** This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

**Syntax:**

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching\_column = table2.matching\_column;

* **RIGHT JOIN:** RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

**Syntax:**

SELECT table1.column1, table1.column2, table2.column1,.

FROM table1

RIGHT JOIN table2

ON table1.matching\_column = table2.matching\_column;

* **FULL JOIN:** FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain NULL values.

**Syntax:**

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching\_column = table2.matching\_column;