SQL

What is SQL?

SQL is structured Query Language which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server uses SQL as standard database language.

Als they are using different dialects, Such as: MS SQL Server using T-SQL, Oracle using

PL/SQL,MS Access version of SQL is called JET SQL (native format)etc

Why SQL?

Allow users to access data in relational database management systems.

Allow users to describe the data.

Allow users to define the data in database and manipulate that data.

Allow to embed within other languages using SQL modules, libraries &

pre-compilers. Allow users to create and drop databases and tables.

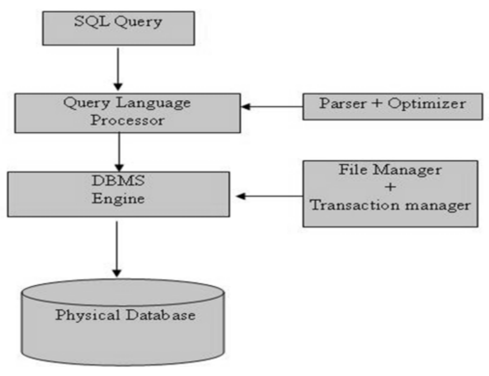
Allow users to create view, stored procedure, functions in a database.
Allow users to set permissions on tables, procedures, and views

SQL Process:

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization engines, Classic Query Engine and SQL query engine etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:



SQL DATABASE :

MySQL

MySQL is open source SQL database, which is developed by Swedish company MySQL AB. MySQL is pronounced "my ess-que-ell," in contrast with SQL, pronounced "sequel."

MySQL is supporting many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS X.

MySQL has free and paid versions, depending on its usage (non-commercial/commercial) and features. MySQL comes with a very fast, multi-threaded, multi-user, and robust SQL database server.

Features:

High
Performance.
High Availability.

Scalability and Flexibility Run nything.

Robust Transactional Support.

Web and Data Warehouse
Strengths. Strong Data Protection.

Comprehensive Application
Development. Management Ease.

Open Source Freedom and 24 x 7 Support.

Lowest Total Cost of Ownership.

Operator in SQL

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.

Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

Arithmetic operators
Comparison
operators Logical
operators

Operators used to negate conditions

SQL Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns Remainder	b % a will give 0

SQL Comparison Operators:

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
=	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true	(a != b) is tru
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.

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

SQL Logical Operators:

Here is a list of all the logical operators available in SQL

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

EXISTS	The EXISTS operator is used to search for the presence of a row in specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg. NOT EXISTS, NOT BETWEEN, NOT IN etc. **This is negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

COMMANDS IN SQL :

1. **CREATE DATABASE**

The SQL **CREATE DATABASE** statement is used to create new SQL database.

Syntax:

Basic syntax of CREATE DATABASE statement is as follows:

```
CREATE DATABASE DatabaseName;
```

Always database name should be unique within the RDBMS.

Example:

If you want to create new database <testDB>, then CREATE DATABASE statement would be as follows:

```
SQL> CREATE DATABASE testDB;
```

2. **DROP DATABASE**

The SQL **DROP DATABASE** statement is used to drop any existing database in SQL schema.

Syntax: Basic syntax of DROP DATABASE statement is as follows:

```
DROP DATABASE DatabaseName;
```

Always database name should be unique within the RDBMS.

Example:

If you want to delete an existing database <testDB>, then DROP DATABASE statement would be as follows:

```
SQL> DROP DATABASE testDB;
```

3. **USE**

The SQL **USE** statement is used to select any existing database in SQL schema.

Syntax:

Basic syntax of USE statement is as follows:

```
USE DatabaseName;
```

4. **CREATE TABLE**

The SQL **CREATE TABLE** statement is used to create a new table.

Syntax:

Basic syntax of CREATE TABLE statement is as follows:

```
CREATE TABLE table_name(
  column1
  datatype,
  column2
  datatype,
  column3
  datatype,

  ....
  .

  columnN
  datatype,

  PRIMARY KEY( one or more
  columns )

);
```

CREATE TABLE is the keyword telling the database system what you want to do.in this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with an example below.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement.

5. **DROP TABLE**

The SQL **DROP TABLE** statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

Syntax:

Basic syntax of DROP TABLE statement is as follows:

DROP TABLE table_name;

6. **INSERT INTO**

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax:

There are two basic syntax of INSERT INTO statement is as follows:

INSERT INTO TABLE_NAME (column1, column2,
column3,...columnN)] VALUES (value1, value2,
value3,...valueN);

Here column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table. The SQL INSERT INTO syntax would be as follows:

INSERT INTO TABLE_NAME VALUES

Example

Following statements would create six records in CUSTOMERS table:

INSERT INTO CUSTOMERS
(ID,NAME,AGE,ADDRESS,SALARY) VALUES (1,

```
'Ramesh', 32, 'Ahmedabad', 2000.00 );
```

```
INSERT INTO CUSTOMERS
(ID,NAME,AGE,ADDRESS,SALARY) VALUES (2,
'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS
(ID,NAME,AGE,ADDRESS,SALARY) VALUES (3,
'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS
(ID,NAME,AGE,ADDRESS,SALARY) VALUES (4,
'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS
(ID,NAME,AGE,ADDRESS,SALARY) VALUES (5,
'Hardik', 27, 'Bhopal', 8500.00 )

INSERT INTO CUSTOMERS
(ID,NAME,AGE,ADDRESS,SALARY) VALUES (6,
'Komal', 22, 'MP', 4500.00 );
```

You can create a record in CUSTOMERS table using second syntax as follows:

```
INSERT INTO CUSTOMERS

VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statement would product following records in CUSTOMERS table:

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

7. **SELECT**

SQL **SELECT** Statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Syntax:

The basic syntax of SELECT statement is as follows:

SELECT column1, column2, columnN FROM table_name;

Here column1, column2...are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field then you can use following syntax:

SELECT * FROM table_name;

Example:

Consider CUSTOMERS table is having following records:

```
+----+----------+-----+-----------+----------+

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

Following is an example which would fetch ID, Name and Salary fields of the customers available in

CUSTOMERS table:

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;

This would produce following result:

```
+----+----------+----------+

| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

8. WHERE CLAUSE

The SQL **WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple table.

If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

The WHERE clause not only used in SELECT statement, but it is also used in UPDATE, DELETE

statement etc. which we would examine in subsequent chapters.

Syntax:

The basic syntax of SELECT statement with WHERE clause is as follows:

```
SELECT column1, column2,
columnN FROM table_name

WHERE [condition]
```

You can specify a condition using comparision or logical operators like >, <, =, LIKE, NOT etc. Below examples would make this concept clear.

Example:

Consider CUSTOMERS table is having following records:

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

Following is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000:

```
SQL> SELECT ID, NAME,
SALARY FROM
CUSTOMERS

WHERE SALARY > 2000;
```

This would produce following result:

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

9. AND and OR OPERATORS

The SQL **AND** and **OR** operators are used to combile multiple conditions to narrow data in an SQL statement. These two operators are called conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND Operator:

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax:

The basic syntax of AND operator with WHERE clause is as follows:

```
SELECT column1, column2,
columnN FROM table_name

WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine N number of conditions using AND operator. For an action to be taken by the SQL statement, whether it be a transaction or query, all conditions separated by the AND must be TRUE.

Example:

Consider CUSTOMERS table is having following records:

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

Following is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000 AND age is less tan 25 years:

```
SQL> SELECT ID, NAME,
SALARY FROM CUSTOMERS

WHERE SALARY > 2000 AND age < 25;
```

This would produce following result:

```
+----+-------+----------+
| ID | NAME  | SALARY   |
+----+-------+----------+
|  6 | Komal |  4500.00 |
|  7 | Muffy | 10000.00 |
+----+-------+----------+
```

10. **UPDATE**

The SQL **UPDATE** Query is used to modify the existing records in a table.

You can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be effected.

Syntax:

The basic syntax of UPDATE query with WHERE clause is as follows:

```
UPDATE table_name

SET column1 = value1, column2 = value2...., columnN
= valueN WHERE [condition];
```

11. **DELETE**

The SQL **DELETE** Query is used to delete the existing records from a table.

You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax:

The basic syntax of DELETE query with WHERE clause is as follows

```
DELETE FROM table_name

WHERE [condition];
```

You can combine N number of conditions using
AND or OR operators

Example:

Consider CUSTOMERS table is having following records:

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

Following is an example which would DELETE a customer whose ID is 6:

```
SQL> DELETE FROM
CUSTOMERS WHERE ID = 6;
```

Now CUSTOMERS table would have following records:

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

12. **LIKE**

The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax:

The basic syntax of % and _ is as follows:

```
SELECT FROM table_name
WHERE column LIKE

'XXXX%' or

SELECT FROM table_name
WHERE column LIKE

'%XXXX%' or

SELECT FROM table_name
WHERE column LIKE 'XXXX_'
```

Consider CUSTOMERS table is having following records:

```
+----+----------+-----+-----------+----------
+

| ID | NAME     | AGE | ADDRESS   |
SALARY   |

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

Following is an example which would display all the records from CUSTOMERS table where SALARY starts with 200:

```
SQL> SELECT * FROM
CUSTOMERS WHERE
SALARY LIKE '200%';
```

This would produce following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
+----+----------+-----+-----------+----------+
```

13. **TOP**

The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.

Syntax:

The basic syntax of TOP clause with SELECT statement would be as follows:

```
SELECT TOP number|percent
column_name(s) FROM table_name

WHERE [condition]
```

Example:

Consider CUSTOMERS table is having following records:

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

This would produce following result:

```
+----+---------+-----+-----------+---------+
| ID | NAME    | AGE | ADDRESS   | SALARY  |
+----+---------+-----+-----------+---------+
|  1 | Ramesh  |  32 | Ahmedabad | 2000.00 |
|  2 | Khilan  |  25 | Delhi     | 1500.00 |
|  3 | kaushik |  23 | Kota      | 2000.00 |
+----+---------+-----+-----------+---------+
```

14. **ORDER BY**

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

Syntax:

The basic syntax of ORDER BY clause is as follows:

```
SELECT column-
list FROM
table_name
[WHERE
condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example:

Consider CUSTOMERS table is having following records

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

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

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

This would produce following result:

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

15. **GROUP BY**

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

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

```
SELECT column1,
column2

FROM table_name

WHERE [ conditions ]

GROUP BY column1,
column2

ORDER BY column1,
column2
```

Example:

Consider CUSTOMERS table is having following records:

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

If you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

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

This would produce following result:

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

16. DISTINCT KEYWORD

The SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

Syntax:

The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

```
SELECT DISTINCT column1,
column2,.....columnN FROM table_name
```

WHERE [condition]

Example:

Consider CUSTOMERS table is having following records:

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

First let us see how the following SELECT query returns duplicate salary records:

```
SQL> SELECT SALARY FROM
   CUSTOMERS ORDER BY SALARY;
```

This would produce following result where salary 2000 is coming twice which is a duplicate record from the original table.

```
+----------+
| SALARY   |
+----------+
|  1500.00 |
|  2000.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------+
```

Now let us use DISTINCT keyword with the above SELECT query and see the result:

```
SQL> SELECT DISTINCT SALARY FROM
   CUSTOMERS ORDER BY SALARY;
```

This would produce following result where we do not have any duplicate entry:

```
+----------+
| SALARY   |
+----------+
|  1500.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------+
```

17. CONSTRAINTS

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Contraints could be column level or table level. Column level constraints are applied only to one column where as table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL. These constraints have already been discussed in SQL - RDBMS Concepts chapter but its worth to revise them at this point.

NOT NULL Constraint: Ensures that a column cannot have NULL value.

DEFAULT Constraint : Provides a default value for a column when none is specified. UNIQUE Constraint: Ensures that all values in a column are different.

PRIMARY Key: Uniquely identified each rows/records in a database table. FOREIGN Key: Uniquely identified a rows/records in any another database table. CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.

INDEX: Use to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use ALTER TABLE statment to create constraints even after the table is created.

Dropping Constraints:

Any constraint that you have defined can be dropped using the ALTER TABLE command with the

DROP CONSTRAINT option.

For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command:

ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK;

Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command:

ALTER TABLE EMPLOYEES DROP PRIMARY KEY;

Some implementations allow you to disable constraints. Instead of permanently dropping a constraint from the database, you may want to temporarily disable the constraint, and then enable it later.

Integrity Constraints:

Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

There are many types of integrity constraints that play a role in referential integrity (RI). These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints mentioned above.

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider following two tables, (a) CUSTOMERS table is as follows:

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

(b) Another table is ORDERS as follows:

```
+-----+---------------------+-------------+--------+
OID   | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

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

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

This would produce following result:

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

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

18. SQL JOIN TYPES

There are different type of joins available in SQL:

INNER JOIN: returns rows when there is a match in both tables.

LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.

RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.

FULL JOIN: returns rows when there is a match in one of the tables.

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

CARTESIAN JOIN: returns the cartesian product of the sets of records from the two or more joined tables.