Introduction Database System

Contents
❖ORACLE

ORACLE

It is a very large and multi-user database management system. Oracle is a relational database management system developed by 'Oracle Corporation'.
Oracle works to efficiently manage its resource, a database of information, among the multiple clients requesting and sending data in the network.
It is an excellent database server choice for client/server computing. Oracle supports all major operating systems for both clients and servers, including MSDOS, NetWare, UnixWare, OS/2 and most UNIX flavors.

History:

Oracle began in 1977 and celebrating its 32 wonderful years in the industry (from 1977 to 2009).

- 1977 - Larry Ellison, Bob Miner and Ed Oates founded Software Development Laboratories to undertake development work.
- 1979 - Version 2.0 of Oracle was released and it became first commercial relational database and first SQL database. The company changed its name to Relational Software Inc. (RSI).
- 1981 - RSI started developing tools for Oracle.
- 1982 - RSI was renamed to Oracle Corporation.
- 1983 - Oracle released version 3.0, rewritten in C language and ran on multiple platforms.
- 1984 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc.
- 1985 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc.
- 2007 - Oracle has released Oracle11g. The new version focused on better partitioning, easy migration etc.
- SQL is followed by unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax:
- All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).
- Important point to be noted is that SQL is **case insensitive**, which means SELECT and select have same meaning in SQL statements, but MySQL makes difference in table names. So if you are working with MySQL, then you need to give table names as they exist in the database.
- All the examples given in this tutorial have been tested with MySQL server.

SQL SELECT Statement:

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

SQL DISTINCT Clause:

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

SQL WHERE Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION;
```

SQL AND/OR Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION-1 {AND|OR} CONDITION-2;
```

SQL IN Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name IN (val-1, val-2,...val-N);
```

SQL BETWEEN Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name BETWEEN val-1 AND val-2;
```

SQL LIKE Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name LIKE { PATTERN };
```

SQL ORDER BY Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION
ORDER BY column_name {ASC|DESC};
```

SQL GROUP BY Clause:

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name;
```

SQL COUNT Clause:

```
SELECT COUNT(column_name)
FROM   table_name
WHERE  CONDITION;
```

SQL HAVING Clause:

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (arithematic function condition);
```

SQL CREATE TABLE Statement:

```
CREATE TABLE table_name(
column1 datatype,
column2 datatype,
column3 datatype,
.....
columnN datatype,
PRIMARY KEY( one or more columns )
);
```

SQL DROP TABLE Statement:

```
DROP TABLE table_name;
```

SQL CREATE INDEX Statement :

```
CREATE UNIQUE INDEX index_name
ON table_name ( column1, column2,...columnN);
```

SQL DROP INDEX Statement :

```
ALTER TABLE table_name
DROP INDEX index_name;
```

SQL DESC Statement :

```
DESC table_name;
```

SQL TRUNCATE TABLE Statement:

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement:

```
ALTER TABLE table_name {ADD|DROP|MODIFY} column_name {data_ype};
```

SQL ALTER TABLE Statement (Rename) :

```
ALTER TABLE table_name RENAME TO new_table_name;
```

SQL INSERT INTO Statement:

```
INSERT INTO table_name( column1, column2....columnN)
```

```
VALUES ( value1, value2....valueN);
```

SQL UPDATE Statement:

```
UPDATE table_name
SET column1 = value1, column2 = value2....columnN=valueN
[ WHERE  CONDITION ];
```

SQL DELETE Statement:

```
DELETE FROM table_name
WHERE  {CONDITION};
```

SQL CREATE DATABASE Statement:

```
CREATE DATABASE database_name;
```

SQL DROP DATABASE Statement:

```
DROP DATABASE database_name;
```

SQL USE Statement:

```
USE database_name;
```

SQL COMMIT Statement:

```
COMMIT;
```

SQL ROLLBACK Statement:

```
ROLLBACK;
```

- QL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.
- You would use these data types while creating your tables. You would choose a particular data type for a table column based on your requirement.
- SQL Server offers six categories of data types for your use –

Exact Numeric Data Types

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255

bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

- # Approximate Numeric Data Types

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

- # Date and Time Data Types

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

- **Note** – Here, datetime has 3.33 milliseconds accuracy where as smalldatetime has 1 minute accuracy.

- # Character Strings Data Types

DATA TYPE	Description
char	Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
varchar	Maximum of 8,000 characters.(Variable-length non-Unicode data).
varchar(max)	Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).
text	Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

- # Unicode Character Strings Data Types

DATA TYPE	Description
nchar	Maximum length of 4,000 characters.(Fixed length Unicode)
nvarchar	Maximum length of 4,000 characters.(Variable length Unicode)
nvarchar(max)	Maximum length of 231characters (SQL Server 2005 only).(Variable length Unicode)
ntext	Maximum length of 1,073,741,823 characters. (Variable length Unicode)

- **Binary Data Types**

DATA TYPE	Description
binary	Maximum length of 8,000 bytes(Fixed-length binary data)
varbinary	Maximum length of 8,000 bytes.(Variable length binary data)
varbinary(max)	Maximum length of 231 bytes (SQL Server 2005 only). (Variable length Binary data)
image	Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

- **Misc Data Types**

DATA TYPE	Description
sql_variant	Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
timestamp	Stores a database-wide unique number that gets updated every time a row gets updated
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only).
cursor	Reference to a cursor object
table	Stores a result set for later processing

What is an 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:

Show Examples

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:

Show Examples

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

<>	Checks if the values 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 then condition becomes true.	(a >= b) is not true.
<=	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.

Show Examples

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 a 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 a 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).

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value.
SQL EXPRESSIONs are like formulas and they are written in query language. You can also use them to query the database for specific set of data.

Syntax:

Consider the basic syntax of the SELECT statement as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];
```

There are different types of SQL expressions, which are mentioned below:

SQL - Boolean Expressions:

SQL Boolean Expressions fetch the data on the basis of matching single value.
Following is the syntax:

```
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHTING EXPRESSION;
```

Consider the CUSTOMERS table having the following records:

```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| 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 rows in set (0.00 sec)
```

Here is simple example showing usage of SQL Boolean Expressions:

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 10000;
+----+-------+-----+---------+----------+
| ID | NAME  | AGE | ADDRESS | SALARY   |
+----+-------+-----+---------+----------+
|  7 | Muffy |  24 | Indore  | 10000.00 |
+----+-------+-----+---------+----------+
1 row in set (0.00 sec)
```

SQL - Numeric Expression:

This expression is used to perform any mathematical operation in any query. Following is the syntax:

```
SELECT numerical_expression as  OPERATION_NAME
[FROM table_name
WHERE CONDITION] ;
```

Here numerical_expression is used for mathematical expression or any formula. Following is a simple examples showing usage of SQL Numeric Expressions:

```
SQL> SELECT (15 + 6) AS ADDITION
+----------+
| ADDITION |
+----------+
|       21 |
+----------+
1 row in set (0.00 sec)
```

There are several built-in functions like avg(), sum(), count(), etc., to perform what is known as aggregate data calculations against a table or a specific table column.

```
SQL> SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;
+---------+
| RECORDS |
+---------+
|       7 |
+---------+
1 row in set (0.00 sec)
```

SQL - Date Expressions:

Date Expressions return current system date and time values:

```
SQL>  SELECT CURRENT_TIMESTAMP;
+---------------------+
| Current_Timestamp   |
+---------------------+
| 2009-11-12 06:40:23 |
+---------------------+
1 row in set (0.00 sec)
```

Another date expression is as follows:

```
SQL>  SELECT  GETDATE();;
```

```
+-------------------------+
| GETDATE                 |
+-------------------------+
| 2009-10-22 12:07:18.140 |
+-------------------------+
1 row in set (0.00 sec)
```

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;
```

Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases as follows:

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
| orig               |
| test               |
| testDB             |
+--------------------+
7 rows in set (0.00 sec)
```

The SQL **DROP DATABASE** statement is used to drop an 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;
```

NOTE: Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database. Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it in the list of databases as follows:

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
| orig               |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed.

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;
```

Always database name should be unique within the RDBMS.

Example:

You can check available databases as follows:

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
```

```
| orig               |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

Now, if you want to work with AMROOD database, then you can execute the following SQL command and start working with AMROOD database:

```
SQL> USE AMROOD;
```

Creating a basic table involves naming the table and defining its columns and each column's data type.

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. You can check complete details atCreate Table Using another Table.

Example:

Following is an example, which creates a CUSTOMERS table with ID as primary key and NOT NULL are the constraints showing that these fields can not be NULL while creating records in this table:

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

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use **DESC** command as follows:

```
SQL> DESC CUSTOMERS;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| ID      | int(11)       | NO   | PRI |         |       |
| NAME    | varchar(20)   | NO   |     |         |       |
| AGE     | int(11)       | NO   |     |         |       |
| ADDRESS | char(25)      | YES  |     | NULL    |       |
| SALARY  | decimal(18,2) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Now, you have CUSTOMERS table available in your database which you can use to store required information related to customers.

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

NOTE: You have to be careful while using this command because once a table is deleted then all the information available in the table would also be lost forever.

Syntax:

Basic syntax of DROP TABLE statement is as follows:

```
DROP TABLE table_name;
```

Example:

Let us first verify CUSTOMERS table and then we would delete it from the database:

```
SQL> DESC CUSTOMERS;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| ID      | int(11)       | NO   | PRI |         |       |
| NAME    | varchar(20)   | NO   |     |         |       |
| AGE     | int(11)       | NO   |     |         |       |
| ADDRESS | char(25)      | YES  |     | NULL    |       |
| SALARY  | decimal(18,2) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

This means CUSTOMERS table is available in the database, so let us drop it as follows:

```
SQL> DROP TABLE CUSTOMERS;
Query OK, 0 rows affected (0.01 sec)
```

Now, if you would try DESC command, then you would get error as follows:

```
SQL> DESC CUSTOMERS;
ERROR 1146 (42S02): Table 'TEST.CUSTOMERS' doesn't exist
```

Here, TEST is database name which we are using for our examples.

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

Syntax:

There are two basic syntaxes of INSERT INTO statement 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 (value1,value2,value3,...valueN);
```

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 statements would produce the 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 |
+----+----------+-----+-----------+----------+
```

Populate one table using another table:

You can populate data into a table through select statement over another table provided another table has a set of fields, which are required to populate first table. Here is the syntax:

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]
   SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

QL **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 the following syntax:

```
SELECT * FROM table_name;
```

Example:

Consider the CUSTOMERS table having the 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 the 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 |
+----+----------+----------+
```

If you want to fetch all the fields of CUSTOMERS table, then use the following query:

```
SQL> SELECT * FROM CUSTOMERS;
```

This would produce the 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 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

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

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 is 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 comparison or logical operators like >, <, =, LIKE, NOT, etc. Below examples would make this concept clear.

Example:

Consider the CUSTOMERS table having the 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 the following result:

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

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table for a customer with name **Hardik**. Here, it is important to note that all the strings should be given inside single quotes ('') where as numeric values should be given without any quote as in above example:

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME = 'Hardik';
```

This would produce the following result:

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  5 | Hardik   |  8500.00 |
+----+----------+----------+
```

The SQL **AND** and **OR** operators are used to combine 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 the CUSTOMERS table having the 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 the following result:

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

The OR Operator:

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax:

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

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

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

Example:

Consider the CUSTOMERS table having the 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 OR age is less tan 25 years:

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 OR age < 25;
```

This would produce the following result:

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

he 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 affected.

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];
```

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

Example:

Consider the CUSTOMERS table having the 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 update ADDRESS for a customer whose ID is 6:

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

Now, CUSTOMERS table would have the 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 | Pune      |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to modify all ADDRESS and SALARY column values in CUSTOMERS table, you do not need to use WHERE clause and UPDATE query would be as follows:

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune', SALARY = 1000.00;
```

Now, CUSTOMERS table would have the following records:

```
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
|  1 | Ramesh   |  32 | Pune    | 1000.00 |
|  2 | Khilan   |  25 | Pune    | 1000.00 |
|  3 | kaushik  |  23 | Pune    | 1000.00 |
|  4 | Chaitali |  25 | Pune    | 1000.00 |
|  5 | Hardik   |  27 | Pune    | 1000.00 |
|  6 | Komal    |  22 | Pune    | 1000.00 |
```

```
|  7 | Muffy    |  24 | Pune     | 1000.00 |
+----+----------+-----+----------+---------+
```

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 the CUSTOMERS table having the 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 the 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 |
+----+----------+-----+-----------+----------+
```

If you want to DELETE all the records from CUSTOMERS table, you do not need to use WHERE clause and DELETE query would be as follows:

```
SQL> DELETE FROM CUSTOMERS;
```

Now, CUSTOMERS table would not have any record.

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_'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

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

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Example:

Here are number of examples showing WHERE part having different LIKE clause with '%' and '_' operators:

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE	Finds any values that have 00 in the second and third

'_00%'	positions
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY LIKE '%2'	Finds any values that end with 2
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

Let us take a real example, consider the CUSTOMERS table having the 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 the following result:

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

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

Note: All the databases do not support TOP clause. For example MySQL supports **LIMIT** clause to fetch limited number of records and Oracle uses**ROWNUM** to fetch limited number of records.

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 the CUSTOMERS table having the 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 on SQL server, which would fetch top 3 records from CUSTOMERS table:

```
SQL> SELECT TOP 3 * FROM CUSTOMERS;
```

This would produce the 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 |
+----+---------+-----+-----------+---------+
```

If you are using MySQL server, then here is an equivalent example:

```
SQL> SELECT * FROM CUSTOMERS
LIMIT 3;
```

This would produce the 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 |
+----+---------+-----+-----------+---------+
```

If you are using Oracle server, then here is an equivalent example:

```
SQL> SELECT * FROM CUSTOMERS
WHERE ROWNUM <= 3;
```

This would produce the 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 |
+----+---------+-----+-----------+---------+
```

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

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 the CUSTOMERS table having the 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 the 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 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would sort the result in descending order by NAME:

```
SQL> SELECT * FROM CUSTOMERS
```

```
    ORDER BY NAME DESC;
```

This would produce the following result:

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

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 the CUSTOMERS table is having the 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 the 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 |
+----------+-------------+
```

Now, let us have following table where CUSTOMERS table has the following records with duplicate names:

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

Now again, 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 the following result:

```
+---------+-------------+
| NAME    | SUM(SALARY) |
+---------+-------------+
| Hardik  |     8500.00 |
| kaushik |     8500.00 |
| Komal   |     4500.00 |
| Muffy   |    10000.00 |
| Ramesh  |     3500.00 |
+---------+-------------+
```

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 the CUSTOMERS table having the 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 the 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 the following result where we do not have any duplicate entry:

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

```
| 10000.00 |
+----------+
```

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 which would be used to sort result in ascending or descending order 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 the CUSTOMERS table having the 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 the 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 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would sort the result in descending order by NAME:

```
SQL> SELECT * FROM CUSTOMERS
     ORDER BY NAME DESC;
```

This would produce the following result:

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

To fetch the rows with own preferred order, the SELECT query would as follows:

```
SQL> SELECT * FROM CUSTOMERS
    ORDER BY (CASE ADDRESS
    WHEN 'DELHI'    THEN 1
    WHEN 'BHOPAL'   THEN 2
    WHEN 'KOTA'     THEN 3
    WHEN 'AHMADABAD' THEN 4
    WHEN 'MP'      THEN 5
    ELSE 100 END) ASC, ADDRESS DESC;
```

This would produce the following result:

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

This will sort customers by ADDRESS in your ownoOrder of preference first and in a natural order for the remaining addresses. Also remaining Addresses will be sorted in the reverse alpha order.