Select Command

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

Syntax

The basic syntax of the 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

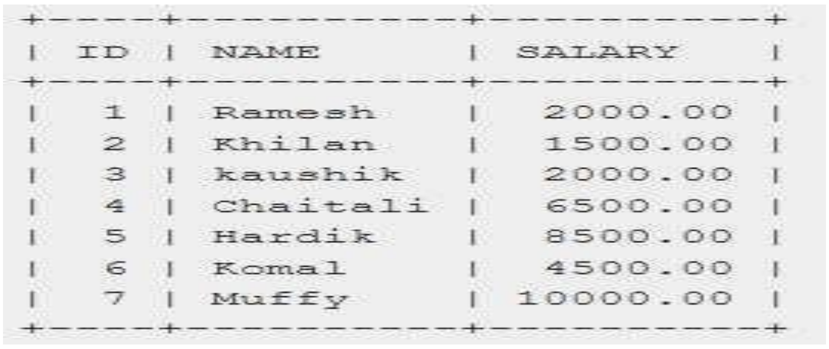
- Consider the **CUSTOMERS** table having the following records

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

- The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

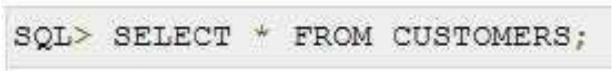
```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;
```

This would produce the following result



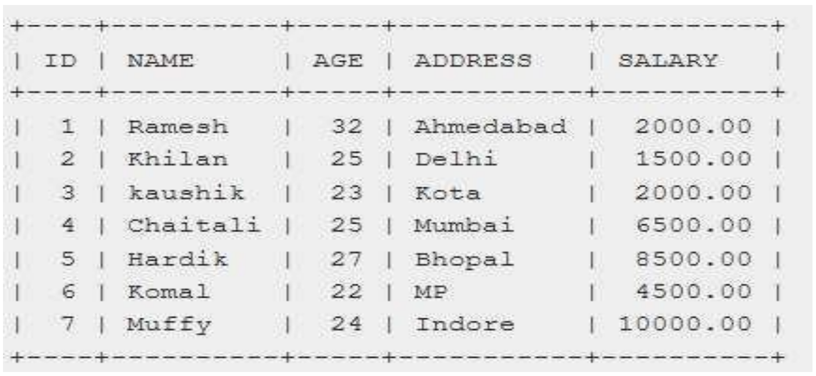
```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+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

- If you want to fetch all the fields of the **CUSTOMERS** table, then you should use the following query.



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

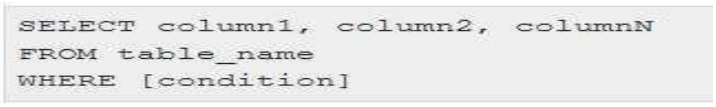
- This would produce the result as shown below.



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

Using **WHERE** clause

- The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables.
- If the given condition is satisfied, then only it returns a specific value from the table.
- You should use the **WHERE** clause to filter the records and fetching only the necessary records.
- The **WHERE** clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc.,
- **Syntax**
- The basic syntax of the **SELECT** statement with the **WHERE** clause is as shown below.

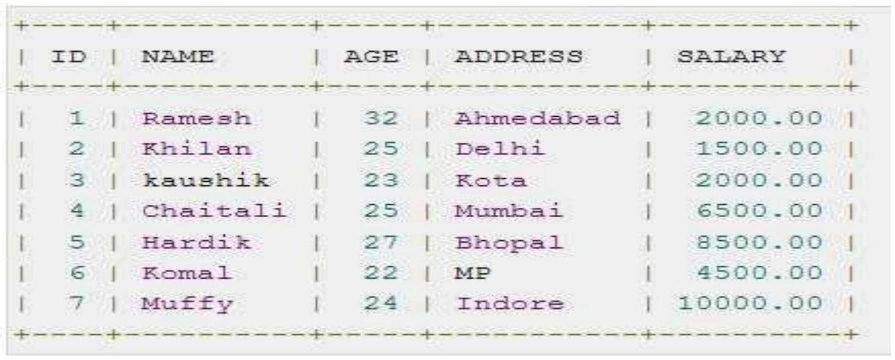


```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc

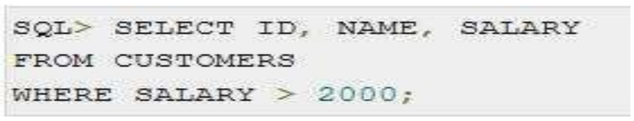
Example

- Consider the **CUSTOMERS** table having the following records



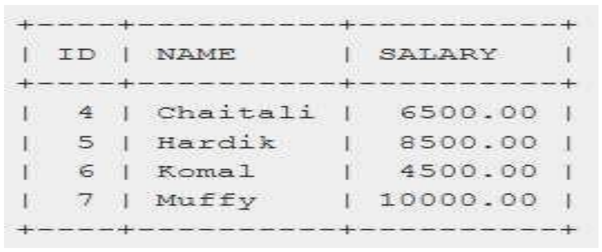
```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali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

- The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000.



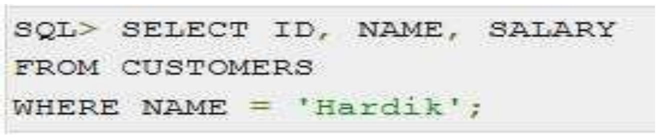
```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

- This would produce the following result.



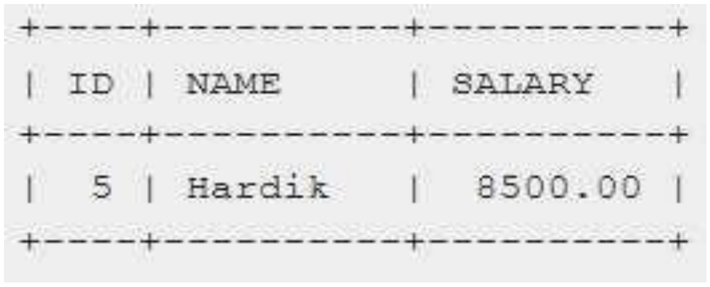
```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

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

What are SQL operators?

- SQL operators are reserved keywords used in the **WHERE** clause of a SQL statement to perform **arithmetic**, **logica**l and **comparison** operations. Operators act as conjunctions in SQL statements to fulfill multiple conditions in a statement.

- There are different types of operators in SQL
- Types of Operators:
    1. Arithmetic Operators
    2. Comparison Operators
    3. Logical Operators

Arithmetic Operators

These operators are used to perform operations such as addition, multiplication, subtraction etc.

Operator	**Operation**	**Description**
+	Addition	Add values on either side of the operator
–	Subtraction	Used to subtract the right hand side value from the left hand side value
*	Multiplication	Multiples the values present on each side of the operator
/	Division	Divides the left hand side value by the right hand side value
%	Modulus	Divides the left hand side value by the right hand side value; and returns the remainder

Addition (+) :

- It is used to perform **addition operation** on the data items, items include either single column or multiple columns.

- **Implementation**:

```
SELECT employee_id, employee_name, salary, salary + 100
   AS "salary + 100" FROM addition;
```

Output:

employee_id	employee_name	salary	salary+100
1	alex	25000	25100
2	rr	55000	55100
3	jpm	52000	52100
4	ggshmr	12312	12412

Here we have done addition of 100 to each Employee's salary i.e, addition operation on single column.

Let's perform **addition of 2 columns**:

```
SELECT employee_id, employee_name, salary, salary + employee_id
   AS "salary + employee_id" FROM addition;
```

Output:

employee_id	employee_name	salary	salary+employee_id
1	alex	25000	25001
2	rr	55000	55002
3	jpm	52000	52003
4	ggshmr	12312	12316

Here we have done addition of 2 columns with each other i.e, each employee's employee_id is added with its salary.

Subtraction (-) :

- It is use to perform **subtraction operation** on the data items, items include either single column or multiple columns.

Implementation:

```
SELECT employee_id, employee_name, salary, salary - 100
    AS "salary - 100" FROM subtraction;
```

Output:

employee_id	employee_name	salary	salary-100
12	Finch	15000	14900
22	Peter	25000	24900
32	Warner	5600	5500
42	Watson	90000	89900

Here we have done subtraction of 100 to each Employee's salary i.e, subtraction operation on single column.

- Let's perform **subtraction of 2 columns**:

```
SELECT employee_id, employee_name, salary, salary - employee_id
    AS "salary - employee_id" FROM subtraction;
```

Output:

employee_id	employee_name	salary	salary – employee_id
12	Finch	15000	14988
22	Peter	25000	24978
32	Warner	5600	5568
42	Watson	90000	89958

Here we have done subtraction of 2 columns with each other i.e, each employee's employee_id is subtracted from its salary.

Multiplication (*) :

- It is use to perform **multiplication** of data items.

Implementation:

```
SELECT employee_id, employee_name, salary, salary * 100
    AS "salary * 100" FROM addition;
```

Output:

employee_id	employee_name	salary	salary * 100
1	Finch	25000	2500000
2	Peter	55000	5500000
3	Warner	52000	5200000
4	Watson	12312	1231200

Let’s perform **multiplication of 2 columns**:

```
SELECT employee_id, employee_name, salary, salary * employee_id
    AS "salary * employee_id" FROM addition;
```

Output:

employee_id	employee_name	salary	salary * employee_id
1	Finch	25000	25000
2	Peter	55000	110000
3	Warner	52000	156000
4	Watson	12312	49248

- Here we have done multiplication of 2 columns with each other i.e, each employee’s employee_id is multiplied with its salary.

Modulus (%) :

- It is use to get **remainder** when one data is divided by another.

Implementation:

```
SELECT employee_id, employee_name, salary, salary % 25000
    AS "salary % 25000" FROM addition;
```

Output:

employee_id	employee_name	salary	salary % 25000
1	Finch	25000	0
2	Peter	55000	5000
3	Warner	52000	2000
4	Watson	12312	12312

- Here we have done modulus of 100 to each Employee’s salary i.e, modulus operation on single column.

- Let’s perform **modulus operation between 2 columns**:

```
SELECT employee_id, employee_name, salary, salary % employee_id
    AS "salary % employee_id" FROM addition;
```

Output:

employee_id	employee_name	salary	salary % employee_id
1	Finch	25000	0
2	Peter	55000	0
3	Warner	52000	1
4	Watson	12312	0

- Here we have done modulus of 2 columns with each other i.e, each employee’s salary is divided with its id and corresponding remainder is shown.

Concept of NULL :

- If we perform any arithmetic operation on **NULL**, then answer is *always* null.
- **Implementation**:

```
SELECT employee_id, employee_name, salary, type, type + 100
    AS "type+100" FROM addition;
```

Output:

employee_id	employee_name	salary	type	type + 100
1	Finch	25000	NULL	NULL
2	Peter	55000	NULL	NULL
3	Warner	52000	NULL	NULL
4	Watson	12312	NULL	NULL

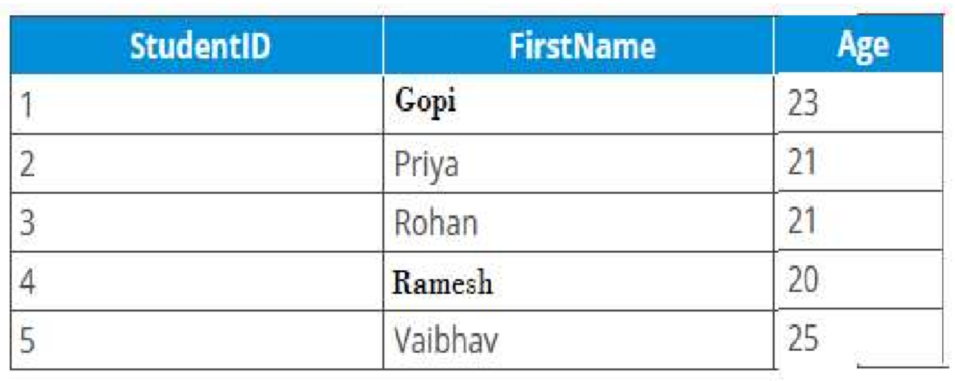
Here output always came null, since performing any operation on null will always result in a *null value*.

Comparison Operators

- **These operators are used to perform operations such as equal to, greater than, less than etc.**

Operator	Operation	Description
=	Equal to	Used to check if the values of both operands are equal or not. If they are equal, then it returns TRUE.
>	Greater than	Returns TRUE if the value of left operand is greater than the right operand.
<	Less than	Checks whether the value of left operand is less than the right operand, if yes returns TRUE.
>=	Greater than or equal to	Used to check if the left operand is greater than or equal to the right operand, and returns TRUE, if the condition is true.
<=	Less than or equal to	Returns TRUE if the left operand is less than or equal to the right operand.
<> or !=	Not equal to	Used to check if values of operands are equal or not. If they are not equal then, it returns TRUE.
!>	Not greater than	Checks whether the left operand is not greater than the right operand, if yes then returns TRUE.
!<	Not less than	Returns TRUE, if the left operand is not less than the right operand.

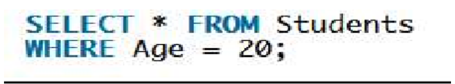
Example:



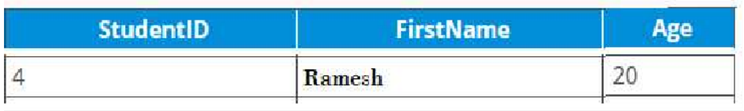
StudentID	FirstName	Age
1	Gopi	23
2	Priya	21
3	Rohan	21
4	Ramesh	20
5	Vaibhav	25

Equality Operator

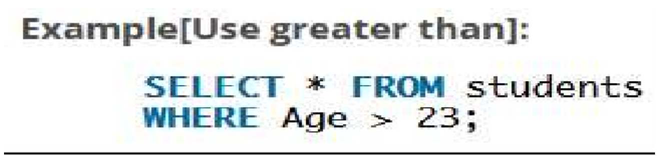
you can use the = operator to test for equality in a query.



```
SELECT * FROM Students
WHERE Age = 20;
```



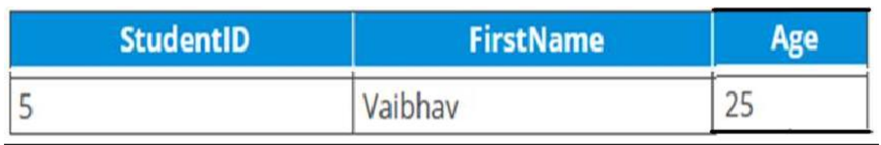
StudentID	FirstName	Age
4	Ramesh	20



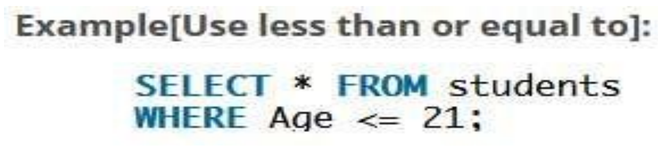
Example[Use greater than]:

```
SELECT * FROM students
WHERE Age > 23;
```

Output:



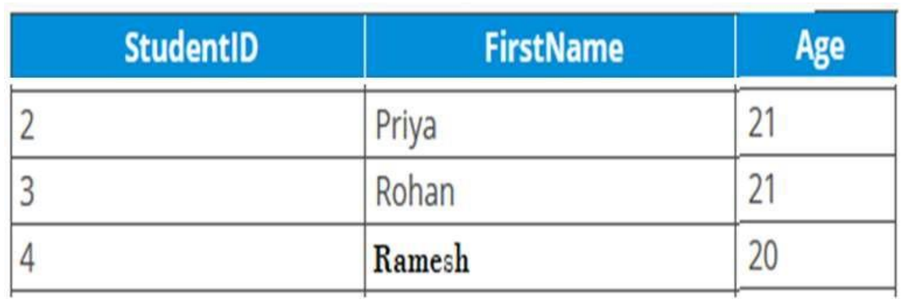
StudentID	FirstName	Age
5	Vaibhav	25



Example[Use less than or equal to]:

```
SELECT * FROM students
WHERE Age <= 21;
```

Output:



StudentID	FirstName	Age
2	Priya	21
3	Rohan	21
4	Ramesh	20

Logical Operators

The logical operators are used to perform operations such as ALL, ANY, NOT, BETWEEN etc.

Operator	Description
ALL	Used to compare a specific value to all other values in a set
ANY	Compares a specific value to any of the values present in a set.
IN	Used to compare a specific value to the literal values mentioned.
BETWEEN	Searches for values within the range mentioned.
AND	Allows the user to mention multiple conditions in a WHERE clause.
OR	Combines multiple conditions in a WHERE clause.
NOT	A negate operators, used to reverse the output of the logical operator.
EXISTS	Used to search for the row's presence in the table.
LIKE	Compares a pattern using wildcard operators.
SOME	Similar to the ANY operator, and is used compares a specific value to some of the values present in a set.

Example[ANY]

```
SELECT * FROM Students
WHERE Age > ANY (SELECT Age FROM Students WHERE Age > 21);
```

Output:

StudentID	FirstName	Age
1	Gopi	23
5	Vaibhav	25

Example[BETWEEN & AND]

```
SELECT * FROM Students
WHERE Age BETWEEN 22 AND 25;
```

Output:

StudentID	FirstName	Age
1	Gopi	23

Example[IN]

```
SELECT * FROM Students
WHERE Age IN('23', '20');
```

Output:

StudentID	FirstName	Age
1	Gopi	23
4	Ramesh	20