**Operators**

**Day-01**

**05-03-2025**

**===================**

Operand:

======

 a value on which we can define any operation

Operator:

======

 a symbol which denote an operation

Expression:

========

 the combination of operators and operands is called as an "Expression".

x - y

x + y

Statement:

=======

-> a line of the program is called as statement.

-> That every statement must be terminated with semi-colon.

Types of Operators:

=============

-> On the basis of number of operands, the operators are classified into two types:

 1) Unary operators

 ----------------------

 -> The operators can be defined with single operand are called as "Unary Operators".

 2) Binary Operators

 -----------------------

 -> The operators which are defined with two operands are called as "Binary operators".

n = 7

if the above number is negative, ==> -7

a + b

-> Based on the usage the operators are classified into:

 1) Arithmetic Operators

 2) Comparison Operators

 3) Compound Operators

 4) Logical Operators

 5) Bitwise Operators

 6) Special Operators

Ex: update Employee\_Table set Employee\_Name = "Kishore" where Employee\_Id = 102 and Age = 26;

Note:

=====

In SQL, the operators can be used to represent with symbols and also with keywords.

**1) Arithmetic Operators**

**----------------------------**

 + ==> Plus ==> addition

 - ==> Hyphen ==> Minus ==> Subtraction

 \* ==> Asterisk ==> Multiplication

 / ==> Slash ==> Division ==> can give a quotient after the division

 % ==> Percentage ==> Modulus ==> can give the remainder after division

Ex: Employee

salary = 25000

allowance = 5000

total-salary ==> salary + allowance ==> 25000 + 5000 ==> 30000

paid-leaves-cost = 700

total-salary ==> 30000 - 700 ==> 29300/-

paid-leaves = 3

paid-leave-cost = 350/day

=> total paid leave cost ==> 3 \* 350 ==> 1050

**2) Comparison Operators**

**------------------------------**

-> Also called as "Relational operators"

-> The comparison operators are:

 = => Equal operator

 !=

 <> ==> Not equal operators

 < ==> less than

 > ==> greater than

 <= ==> less than equal

 >= ==> greater than equal

7 = 7 ==> true

10 < 5 ==> false

**3) Compound Operators**

**----------------------------**

x = 100

x = x + 10 ==> 110

x += 10 ==> 110

+=, -=, \*=, /=, %=, &=, |= etc.

x -= 2 ==> x = x - 2

**4) Logical Operators**

**------------------------**

-> Three logical operators:

 1) logical and ==> and

 2) logical or ==> or

 3) logical not ==> not

Truth table:

------------

 a b a and b a or b

 --------------------------------------------------------

 true true true true

 true false false true

 false true false true

 false false false false

logical and

-----------

If any of the input is "false", the output is "false"

when both inputs are "true", the output is "true"

logical or

----------

If any input is "true", the output is "true"

When both inputs are "false" then only the output is "false"

logical not:

------------

Syntax:

 not operand

-> not operator can alter the input

 not true --> false

 not false --> true

**Day-02**

**06-03-2025**

**================**

binary literal in SQL ==> b'1010101'

0b10101 ==> base-2 (0 and 1)

octal --> 012 ==> base-8 (0 to 7)

hexadecimal --> 0xc12 ==> base-16 (0 to 9 and a to f)

decimal ==> 1023 ==> base-10 (0 to 9)

**Bitwise Operators:**

**============**

-> The operations to be performed on the data bit by bit those operators are called as "Bitwise Operators".

-> Bitwise operators are:

 bitwise and --> &

 bitwise or --> |

 bitwise xor --> ^

 bitwise not --> ~

 left shift --> <<

 right shift --> >>

bitwise and --> &

--------------------------

-> binary operator

Syntax:

 operand1 & operand2

a b a & b

==========

0 0 0

0 1 0

1 0 0

1 1 1

Note:

----

bitwise operators can always allow to define on integer but not on the floats.

Ex: 10 & 12 = 8

bitwise or --> |

-------------------

a b a | b

-------------------

0 0 0

0 1 1

1 0 1

1 1 1

bitwise xor --> ^

--------------------------

a b a ^ b

----------------------

0 0 0

0 1 1

1 0 1

1 1 0

bitwise not --> ~

--------------------------

-> also called as "bitwise complement".

Syntax:

 ~ operand

-> Unary operator

-> can find the 2's complement of the given number.

-> positive --> negative and negative ==> positive