STL

String

Input Functions

1. getline() :- This function is used to store a stream of characters as entered by the user in the object memory.
```
getline(cin,str);
```
2. push_back() :- This function is used to input a character at the end of the string.
```
str.push_back('s');
```
3. pop_back() :- Introduced from C++11(for strings), this function is used to delete the last character from the string.
```
str.pop_back();
```
4. capacity() :- This function returns the capacity allocated to the string, which can be equal to or more than the size of the string. Additional space is allocated so that when the new characters are added to the string, the operations can be done efficiently.
```
str.capacity()
```
5. resize() :- This function changes the size of string, the size can be increased or decreased.
```
str.resize()
```
6.shrink_to_fit() :- This function decreases the capacity of the string and makes it equal to its size. This operation is useful to save additional memory if we are sure that no further addition of characters have to be made.

7. begin() :- This function returns an iterator to beginning of the string.

8. end() :- This function returns an iterator to end of the string.

9. rbegin() :- This function returns a reverse iterator pointing at the end of string.

10. rend() :- This function returns a reverse iterator pointing at beginning of string.

11. copy("char array", len, pos) :- This function copies the substring in target character array mentioned in its arguments. It takes 3 arguments, target char array, length to be copied and starting position in string to start copying.
```
str1.copy(ch,13,0);
```
12. swap() :- This function swaps one string with other.

13. strtok() :- Splits str[] according to given delimiters and returns next token. It needs to be called in a loop to get all tokens. It returns NULL when there are no more tokens.

```
                        char * strtok(char str[], const char *delims);
14. reverse(s.begin(), s.end())
```

toupper / tolower=>

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;

int main()
{
    string s;
    s= "Shivam";
    for(int i=0;i<s.length();i++)
    {
        s[i] = tolower(s[i]);
    }
    cout<<s<<endl;
    for(int i=0;i<s.length();i++)
    {
        s[i] = toupper(s[i]);
    }
    cout<<s;
}
```

Count set bits

```
//count set bits-> __builtin_popcount(n);
#include <bits/stdc++.h>
using namespace std;

int setbits(int n)
{
    if(n==0)
    {
        return 0;
    }
    return 1+ setbits(n & (n-1));
}

int main()
{
    int n;
    cin>>n;
    cout<<setbits(n)<<endl;
}

// Iterative C program to implement pow(x, n)
#include <bits/stdc++.h>
using namespace std;

int power(int x, unsigned int y)
{
    int res = 1;

    while (y > 0)
    {
        // If y is odd, multiply x with result
        if (y & 1)
        {
            res = res * x;
        }
        // n must be even now
        y = y >> 1; // y = y/2
        x = x * x; // Change x to x^2
    }
    return res;
}

int main()
{
    int x,y;
    cin>>x>>y;
    cout<<power(x, y);
    return 0;
}



#include <stdio.h>
#include <string.h>

int main()
{
    char str[] = "Geeks-for-Geeks";
    // Returns first token
    char* token = strtok(str, "-");

    // Keep printing tokens while one of the
    // delimiters present in str[].
    while (token != NULL) {
        printf("%s\n", token);
        token = strtok(NULL, "-");
    }

    return 0;
}


// appending to string
#include <iostream>
#include <string>

int main ()
{
  std::string str;
  std::string str2="Writing ";
  std::string str3="print 10 and then 5 more";

  // used in the same order as described above:
  str.append(str2);                       // "Writing "
  str.append(str3,6,3);                   // "10 "
  str.append("dots are cool",5);          // "dots "
  str.append("here: ");                   // "here: "
  str.append(10u,'.');                    // ".........."
  str.append(str3.begin()+8,str3.end());  // " and then 5 more"
  str.append<int>(5,0x2E);                // "....."

  std::cout << str << '\n';
  return 0;
}
```

stoi	Convert string to integer (function template)
stol	Convert string to long int (function template)
stoul	Convert string to unsigned integer (function template)
stoll	Convert string to long long (function template)
stoull	Convert string to unsigned long long (function template)
stof	Convert string to float (function template)

stod	Convert string to double (function template)
stold	Convert string to long double (function template)

Convert to strings

to_string	Convert numerical value to string (function)
to_wstring	Convert numerical value to wide string (function)

Algorithms=>

```
cout<<min_element(arr, arr+n);
cout<<max_element(arr, arr+n);
```

Heaps->

```
// range heap example
#include <iostream>     // std::cout
#include <algorithm>    // std::make_heap, std::pop_heap, std::push_heap, std::sort_heap
#include <vector>       // std::vector

bool compare(int a, int b)
{
  return a>b; //for min heap
}

int main () {
  int myints[] = {10,20,30,5,15};
  std::vector<int> v(myints,myints+5);

  std::make_heap (v.begin(),v.end(), compare); //Heapify function
  std::cout << "initial max heap   : " << v.front() << '\n';

  std::pop_heap (v.begin(),v.end()); //moves the top element to last
  v.pop_back(); //pop the last element
  std::cout << "max heap after pop : " << v.front() << '\n';

  v.push_back(99); //push element to last
  std::push_heap (v.begin(),v.end()); //heapify the pushed element
  std::cout << "max heap after push: " << v.front() << '\n';

  std::sort_heap (v.begin(),v.end()); //heap sort

  std::cout << "final sorted range :";
  for (unsigned i=0; i<v.size(); i++)
    std::cout << ' ' << v[i];

  std::cout << '\n';

  return 0;
}
```

count->

```
cout<<count(v.begin(), v.end(), 10);
```

lower and upper bound of->

```
cout<<lower_bound (v.begin(), v.end(), 20);
cout<<upper_bound (v.begin(), v.end(), 20);
```

merging 2 vector in sorted manner->

```
  int first[] = {5,10,15,20,25};
  int second[] = {50,40,30,20,10};
  vector<int> v(10);

  sort (first,first+5);
  sort (second,second+5);
  merge (first,first+5,second,second+5,v.begin());
```

reverse->

```
reverse(myvector.begin(),myvector.end());
```

reverse_copy->

```
int myints[] ={1,2,3,4,5,6,7,8,9};
vector<int> myvector;
myvector.resize(9);    // allocate space
reverse_copy (myints, myints+9, myvector.begin());
```

rotate->

```
for (int i=1; i<10; ++i) myvector.push_back(i); // 1 2 3 4 5 6 7 8 9
rotate(myvector.begin(),myvector.begin()+3,myvector.end()); // 4 5 6 7 8 9 1 2 3
```

includes->

Returns true if the sorted range [first1,last1) contains all the elements in the sorted range [first2,last2).

```
cout<<includes(container,container+10,continent,continent+4);
```

Set->
```
  std::vector<int> v(10);                        // 0  0  0  0  0  0  0  0  0  0
  std::vector<int>::iterator it;
  std::sort (first,first+5);      //  5 10 15 20 25
  std::sort (second,second+5);   // 10 20 30 40 50
```
Insertion
```
  it=std::set_intersection (first, first+5, second, second+5, v.begin());
                                               // 10 20 0  0  0  0  0  0  0  0
  v.resize(it-v.begin());
```
Difference
```
  it=std::set_difference (first, first+5, second, second+5, v.begin());
                                               //  5 15 25  0  0  0  0  0  0  0
  v.resize(it-v.begin());                      //  5 15 25
```
Union
```
  it=std::set_union (first, first+5, second, second+5, v.begin());
                                               // 5 10 15 20 25 30 40 50  0  0
  v.resize(it-v.begin());                      // 5 10 15 20 25 30 40 50
```

Permutations->

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    set<string> a;
    string s;
    cin>>s;
    sort(s.begin(), s.end());
    a.insert(s); //insert 1st element
    while(next_permutation(s.begin(), s.end()))
    {
        a.insert(s);
    }
    for(auto x:a)
    {
        cout<<x<<endl;
    }
    cout<<a.size();
    return 0;
}
```

vector=>

```
v.swap(v2);
v.clear();
```

```
cout<<_builtin_popcount(n) << endl; //complxity O(1)
```

OOPS

Class in C++ is a blu-print representing a group of objects which shares some common properties and behaviours.

====> dynamic memory alloation

```
Student *s1 = new Student;
(*s1).age = 21;
s1 -> age = 21;
```

====> Dynamic Binding: In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has virtual functions to support this.
====> Static Binding: The binding which can be resolved at compile time by compiler is known as static or early binding. Binding of all the static, private and final methods is done at compile-time .

====>Pre and Post Increment

```
//Pre Increment
int i=5;
int j = ++i; // first increment and then copy the value
cout<<i<<" "<<j; // i=6; j=6;

int overloading -> Fraction operator++(){}

//Post Increment
int i=5;
int j = i++; // first copy the value  n then increment
cout<<i<<" "<<j; // i=6; j=5;

int overloading -> Fraction operator++(int) {}
```

====> This Pointer
=> If there are 2 variables with same name, then the one which has the closest scope us used.

```
int a=10;
if(a>5)
{
a =1;
cout<<a;
}
```

this will print a=1; because it has the closest scope.
=>this pointer holds the address of current object,

```
cout<<"address = "<<&s1<<endl;
cout<<"this = "<<this<<endl
```

====> Default Constructor
=> Default Constructor is atutomatically called, as soon as objet of the class is created.
=> If u have made your parameterized constructor, then default constructor has to be made mandatorily.
=> if we have constant values then it is compulory forus to make the constructor.
=> with the help of constant object we can call only the constant functions, other normal fucntions cant be called.
=> contant function-> which do not change any property of the current object.

====> Copy Constructor
When is user-defined copy constructor needed?
If we don’t define our own copy constructor, the C++ compiler creates a default copy constructor for each class which does a member-wise copy between objects. The compiler created copy constructor works fine in general. We need to define our own copy constructor only if an object has pointers or any runtime allocation of the resource like file handle, a network connection..etc.

```
Student s2(s1);
Student s4(*s3); // if s3 was created dynamically

Student *s5 = new Student(*s3);
Student *s6 = new Student(s1);
```

=> Default copy constructor does only shallow copy, changes in one, is reflected in another.
=> Deep copy is possible only with user defined copy constructor, changes in one, is not reflected in another.
=> In user defined copy constructor, we make sure that pointers (or references) of copied object point to new memory locations.

=> Default Constructor is atutomatically called, as soon as objet of the class is created.
=> If u have made your parameterized constructor, then default constructor has to be made mandatorily.
=> Copy constructor can be used only at the time of creation of the object. If u want to copy values from one object to another after creation of object 2, then use the assignment operator"=";
eg-> s1=s2;

==> Copy constructor vs Assignment Operator
Which of the following two statements call copy constructor and which one calls assignment operator?

```
MyClass t1, t2;
MyClass t3 = t1;  // ----> (1)
t2 = t1;          // -----> (2)
```

Copy constructor is called when a new object is created from an existing object, as a copy of the existing object. Assignment operator is called when an already initialized object is assigned a new value from another existing object. In the above example (1) calls copy constructor and (2) calls assignment operator.

Can we make copy constructor private?
https://www.geeksforgeeks.org/can-constructor-private-cpp/
Yes, a copy constructor can be made private. When we make a copy constructor private in a class, objects of that class become non-copyable.

Why argument to a copy constructor must be passed as a reference?
A copy constructor is called when an object is passed by value. Copy constructor itself is a function. So if we pass an argument by value in a copy constructor, a call to copy constructor would be made to call copy constructor which becomes a non-terminating chain of calls. Therefore compiler doesn’t allow parameters to be passed by value.

====> Destructor
=> same name as constructor
=> no return type

=> no input arguments
~Student()
{

}
=> can be used only once in a function
=> to delete dynamicaly allocated memory we need to explicitly delete it
eg-> delete s3;

int i=5;
int &k =i; // any changes made using i or k will be reflected.

int const &k =i; // we will be wable to change the value by using i, but not by k // here k will only point to it, it will not change the value.

====> Static data member
=> the property that r common for all the objects of a class, that they are property of class
=> therefore we use static for them
=> these static proerties ant be accessed by the objects
=> therefore we need to acces them in different manner
eg-> cout<<Student::totalStudents;
=> These properties are to be initialized outside the class
eg-> int Student :: totalStudents =0;
===> Static function
=> write static before these functions and then these can be accessed by
eg-> Student:: getTotalStudents();
=> these can acess the static properties
=> thses do not have this pointer

====> Operator Overloading // jaha koi operator work nahi karta waha bhi kuch code arke unski functioning bana dena, like + in case of fraction
=> int i = j+k;
Fraction i = fraction j+ fraction k; // this will not work.
=> therefore we need to write the code of it, i.e. add fraction n then apply operator overloading to it
eg-> fraction operator+(Fraction const &f2)
Fraction f4 = f2+f1;
f2 -> goes with this

f1 -> goes with argument

====> Inheritance
The capability of a class to derive properties and characteristics from another class is called Inheritance.
Sub Class: The class that inherits properties from another class is called Sub class or Derived Class.
Super Class:The class whose properties are inherited by sub class is called Base Class or Super class.

==> Using inheritance, we have to write the functions only one time instead of three times as we have inherited rest of the three classes from base class.

==> Modes of Inheritance
->Public mode: public member -> public and protected members -> protected.
->Protected mode: public member and protected members -> protected.
->Private mode: public member and protected members -> Private.

==> Types of Inheritance in C++
-> Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.
-> Multiple Inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one sub class is inherited from more than one base classes.
-> Multilevel Inheritance: In this type of inheritance, a derived class is created from another derived class.
-> Hierarchical Inheritance: In this type of inheritance, more than one sub class is inherited from a single base class. i.e. more than one derived class is created from a single base class.
-> Hybrid (Virtual) Inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.
-> Virtual Inheritance -> double copy is created in this case, to avoide that we use virtual keyword, virtual class

====> Emcapsulation
=> clubbing data and its function together, this is done with the help of class.

====> Abstraction
=> hiding the un necessary details
=> cruial properties should be abstracted i.e. hidden from outside world.
=> Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
=> Benifits
1) changes can be done easily without affecting the user view
2) avoide errors, as only a pt is visible or accessable to user, so no changes can be made int the function.

=====> Friend function
=> helps to access the private members outside the class.
=> friend function can be put under any access modifiers, no affect.
=> do not have the access of this pointer.
=> friend function can access all the properties of a class, may be private, public or protected
eg-> friend class Bus; // should be included in the main class.
=> friend can access all the properties of main class but not vice=versa.
=> now the definition should be given outside the bus class

=====> Polymorphism
=> Compile time
-> can be achieved by 1) fuction/operator overloading
-> base class wala child classs ke sir vo access kar sata hia jo base class mai avialable hai. (parent child ki cheeein nahi le sakta but child can take everything of parent)

=> Run time by function overridig
-> decision are taken at run time
-> to achieve this we need virtual functions
-> virtual functions-> present in base class and are overridden int he child class.
-> runtime pe decide hoga, jisko point kar raha hai uska print kar dega, n agar nahi hai vo to uske parent ka print kar dega, n if parent bhi present nahi hai to fir error dega.

====> overriding-> same function used in both parent and child.

====> Pure Virtual Functions
=> those which do not have any defination
eg-> virtual void print() = 0; // pure virtual function
eg-> virtual void print(){   // normal virtual function
}
=> any class with any pure virtual function becomes an abstract class. // abstract means incomplete
=> we can't make object of this abstract class
=> if parent is abstract, then the child can either complete all the pure virtual functions of the parent class or itself become an abstract class.
=> if it also becomes an abstract class, then we won't be able to ceat object of child class also.
=> so in such a case to work, we implement all the pure virtual functions of the parent class in the child class.
=> functons whose definitation changes for each childs, are made pure virtual fucntons, and are then completed in the child class.

DBMS

TRANSACTION=>

ATOMICITY-> Transaction control component
ISOLATION-> Concurrency Control Component
DURABILITY-> Recovery Control Component
COnsistency-> held automaticallly if the other 3 r present

Dirty Read Problem-> ram na shyam cheating problem, solution-> read the value after commit in data base by the first process.
Unrepeatable read-> one value has been read multiple times n e have received different value of it, so this leads to inconsistency. Maggie Rajma Chawal problem
Phantom Read Problem-> after first read by T2, T1 has deleted the value / changed the structure. kitchen khali ho gai, maggie ke sath sath
Lost Update Problem=> Write-Write Conflict-> value changed by T2 n comitted, then T1 commits the value cchanged value, n thinks that he is commiting the old value.

Data independency
-> physical data independeny-> an changes made at physical layer(ie if we change the data structure of storage of data), then it should not affect the conceptual level.
-> logical data dependency-> user should be able to access the data base in the same way as earlier. i.e. changes made at conceptual level should now be affect the user view of database.

Data Model-> concepts that defines the struture of our data base, n the constraints our data base will have to obey.
-> defines how the different layers will interact with each other.

Entity-> abt wich data will be stored
attribute-> characterstics of entity, columns
relationship-> 1:1, 1:M, M:M, M:N
Constraint->restrictions to be placed on data base
Degree-> No of attributes in a table.
Cardinality-> no of rows in a table.

SQL Concepts

one table can have only 1 primary key, but more than one unique keys.
primary key cant have NULL value, but unique key can have null values

DDL commands->
Creat
Rename
Truncate
Drop
Alter

```
creat table employee{ name varchar(100), id int, salary int};
insert into employee values("Shivam" , 123, 50000);
Describe employee-> shows the structure of the employee table
```

Order By

```
SELECT * FROM Customers
ORDER BY Country, City, ContactName;

SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

Insert Into

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal','Tom B. Erichsen','Skagen 21','Stavanger','4006','Norway');

INSERT INTO Customers (CustomerName, City, Country) //enter data into specified columns,
rest r set to null
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

IS NULL/ IS NOT NULL

```
SELECT CustomerName, ContactName, Address
```

FROM Customers
WHERE Address IS NULL;

SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;

CONSTRAINTS
creat table Faculty (id int primary Key, name varchar(100) NOT NULL default 'abc' , age int unique);

RENAME
rename table emloyee to emp, student to stud; // change table name

TRUNCATE
deletes the data of the table but not the structure
truncate employee;

DROP
deletes the entire table along with its structure
drop table employee; // deleted

ALTER
Can add column in table
Alter table employee add(address varchar(100), EmailId varchar(100));
alter table employee add(marks int default '10');
alter table employee modify column ContactNumber varchar(100);
alter table employee change ContactNumber CntNo varchar(100); // cange the name of column ContactNumber
alter table employee drop address; // drop a column

DML Commands->
Select
Insert
Update
Delete

UPDATE

UPDATE Customers
SET ContactName='Alfred Schmidt', City='Frankfurt' WHERE CustomerID=1;

```
UPDATE Customers
SET ContactName='Juan' WHERE Country='Mexico';

UPDATE Customers //update everywhere
SET ContactName='Juan';
```

Delete

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
DELETE FROM Customers; //deletes all rows in the "Customers" table, without deleting the table:
```

TOP / Percent

```
SELECT TOP 3 * FROM Customers;
SELECT TOP 3 * FROM Customers where Country="Germany";
SELECT TOP 50 PERCENT * FROM Customers;
SELECT * FROM Customers WHERE Country='Germany' LIMIT 4; //shows maximun 4 elements.
```

MIN/MAX/AVG/SUM   // use all the not null values

```
SELECT MIN(Price) AS SmallestPrice FROM Products;
SELECT MAX(Price) AS LargestPrice FROM Products;
SELECT AVG(Price) FROM Products;
SELECT AVG(Name) FROM Products; // 0
SELECT SUM(Quantity) FROM OrderDetails;
Select sum(City) from Customers; // ans=0
```

COUNT // uses the null values also, repeated values ko bhi count karta hai

```
select count(*) from Customers; // total entries in table // considers the NULL values also
select count(PostalCode) from Customers; //total entries in this particular column // doesnt consider the NULL values
```

LIKE

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents a range of characters	c[a-b]t finds cat and cbt

```
SELECT * FROM Customers WHERE CustomerName LIKE 'a%'; // starting with a
SELECT * FROM Customers WHERE CustomerName NOT LIKE 'a%'; //not starting with a
SELECT * FROM Customers WHERE City LIKE '[acs]%'; //all records where the first letter of the
City is an "a" or a "c" or an "s".
SELECT * FROM Customers WHERE City LIKE '[!bsp]%';
```

IN

```
SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK');
SELECT * FROM Customers WHERE Country NOT IN ('Germany', 'France', 'UK');
select * from Customers where Country IN(select Country from Suppliers);
```

Between

```
SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;
SELECT * FROM Products WHERE Price NOT BETWEEN 10 AND 20;
SELECT * FROM Products WHERE Price BETWEEN 10 AND 20 AND NOT CategoryID IN
(1,2,3);
SELECT * FROM Products WHERE ProductName BETWEEN 'Carnarvon Tigers' AND
'Mozzarella di Giovanni' ORDER BY ProductName;
SELECT * FROM Products WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef
Anton's Cajun Seasoning" ORDER BY ProductName;
SELECT * FROM Orders WHERE OrderDate BETWEEN #01/07/1996# AND #31/07/1997#;
SELECT * FROM Orders WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

ALIAS

```
SELECT CustomerID AS ID, CustomerName AS Customer FROM Customers;
SELECT CustomerName AS Customer, ContactName AS [Contact Person] FROM Customers;
// It requires double quotation marks or square brackets if the alias name contains spaces
SELECT CustomerName as Customer, Address + " , " + PostalCode + City +" , "+ Country as
Address from Customers;

SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
```

```
WHERE c.CustomerName="Around the Horn" AND c.CustomerID=o.CustomerID;
```

JOINS

(INNER) JOIN:	Returns records that have matching values in both tables
LEFT (OUTER) JOIN:	Returns all records from the left table, and the matched records from the right table
RIGHT (OUTER) JOIN:	Returns all records from the right table, and the matched records from the left table
FULL (OUTER) JOIN:	Returns all records when there is a match in either left or right table

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

INNER JOIN=>

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

JOINING 3 TABLES

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

Left Join=>

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Right Join=>

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Full Join=>

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Self Join=>

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2,
A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

UNION

```
SELECT City FROM Customers UNION SELECT City FROM Suppliers ORDER BY City;
//distinct cities
SELECT City FROM Customers UNION ALL SELECT City FROM Suppliers ORDER BY City;
//reperation allowed
SELECT City, Country FROM Customers WHERE Country='Germany' UNION SELECT City,
Country FROM Suppliers WHERE Country='Germany' ORDER BY City;
SELECT City, Country FROM Customers WHERE Country='Germany' UNION ALL SELECT
City, Country FROM Suppliers WHERE Country='Germany' ORDER BY City;
SELECT 'Customer' As Type, ContactName, City, Country FROM Customers UNIO SELECT
'Supplier', ContactName, City, Country FROM Suppliers;
```

GROUP BY

```
SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;
SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country ORDER BY
COUNT(CustomerID) DESC;
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID GROUP BY ShipperName;
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID GROUP BY
Shippers.ShipperID;
```

Having // used when grp clause is used, coz with group where doesn't work.

```
SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING
COUNT(CustomerID) > 5;
```

https://www.w3schools.com/sql/sql_having.asp

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID) GROUP BY
LastName HAVING COUNT(Orders.OrderID) > 10;
select FirstName + LastName as NAME from Employees LEFT JOIN Employees ON
Orders.EmployeeID=Employees.EmployeeID group by Orders.EmployeeID Having
count(Orders.OrderID) > 10;

select Employees.LastName, count(Orders.OrderID) from Orders LEFT JOIN EMPLOYEES ON
Orders.EmployeeID = Employees.EmployeeID where LastName IN ('Davolio' , 'Fuller') group by
LastName Having count(Orders.OrderID) >= 25;
```

Exists

```
=> select distinct Suppliers.SupplierName from Products left join suppliers on
Products.SupplierID= Suppliers.SupplierID where price < 20;
=> SELECT SupplierName FROM Suppliers WHERE EXISTS (SELECT ProductName FROM
Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);
```

ANY/ALL

```
SELECT ProductName FROM Products WHERE ProductID = ANY (SELECT ProductID FROM
OrderDetails WHERE Quantity = 10);
SELECT ProductName FROM Products WHERE ProductID = ALL (SELECT ProductID FROM
OrderDetails WHERE Quantity = 10);
```

Select INto

```
SELECT * INTO CustomersGermany FROM Customers WHERE Country = 'Germany';
SELECT Customers.CustomerName, Orders.OrderID INTO CustomersOrderBackup2017
FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

INSERT INTO SELECT

```
INSERT INTO Customers (CustomerName, City, Country) SELECT SupplierName, City,
Country FROM Suppliers;
```

CASE=>

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN "The quantity is greater than 30"
    WHEN Quantity = 30 THEN "The quantity is 30"
    ELSE "The quantity is under 30"
```

```
END
AS QuantityText
FROM OrderDetails;
```

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
   WHEN City IS NULL THEN Country
   ELSE City
END);
```

DCL Commands -> Data Control language
by default the created user has only 2 access-> 1) show databse 2) show tables.
controls user's access in database
1) grant
2) Revoke
Grant [Prevailages] on [table_name] to [user]
Revoke [prevailages] ON [table_name] from [user]

FUNCTIONAL DEPENDECIES====>
if column A uniquely identifies column B of same table, then A->B //A functionally determines B, B is functionally dependent on A
Trivial FD -> yes if B is subset of A;
Non Trivial FD -> holds true where B is not a subset of A;
FD Set-> set of all FD of a table.
(F+)Closure of FD set-> all the rules implied by the given rules in F.

Axioms====> used during normlaization

Reflexive property ->	if X is superset of Y, then X->Y //X determines Y
Augmentation Property->	If W->Z and X->Y, then WX->ZY
Transitive Property->	X->Y and Y->Z then X->Z
Union Rule->	If X->Y and X->Z, then X->YZ
Decomposition Rule->	If X->YZ, then X->Y and X->Z

Super key-> set of attributes which uniquely identifies all the attributes of a table.
if broken down, this may still uniquely identify

Candidate key-> It is also a super key, but this is minimal
if broken down, this will not uniquely identify

NORMALIZATION====> rules which r kept in mind so that no anamoly no inconsistency no duplicasy is seen in our database.
Insertion anomaly -> data cant be inserted becuase of some reason.
Deletion Anamoly -> on deleteion info can be lost which present only with that value. // kuch ki value delete karnia hai n kuch ki nahi.
Updation Anamoly -> infromation has to be updated at multiple places, because it is present at more than place.

1NF==> remove multi-valued attribute. // break it into different tupples, a student has many subjects.
remove composite attributes, i.e. break them into different column, i.e. creat more attributes. // Address.
2NF==> Table must be in 1NF
if table is not in 2NF then there can be updation anamoly
NO partial Dependencies should be there.
X->a is partial dependency if (both the conditions have to be satisfied)-
1) if X is proper subset of any candicdate key
2) a in non-prime or non-key attribute. //non-prime -> whichr not a part of cadidate key
2NF can be achieved by splitting the table into sub-tables using a single primary key source.
3NF==> Table must be in 2NF
No transitive Dependency
X->a is transitive dependency if //i.e. if non-prime determines non-prime
Both X and a are non-prime or non-key attribute.
3NF can be achieved by splitting the table into sub-tables using a single non-prime key source.
BCNF==> if X->Y exists, then X should be a super key.
if not in BCNF -> deletion, insertion, updation anomaly may exist even after the table is in 1,2,3NF
BCNF can be achieved by splitting the table into subtables.

====== even after following these NF we cant gaurentee that there is no anomaly, but we have minimized the anomalys.

OS

Deadlock->

Four conditions hold simultaneously (Necessary Conditions)
Mutual Exclusion: One or more than one resource are non-sharable (Only one process can use at a time)
Hold and Wait: A process is holding at least one resource and waiting for resources.
No Preemption: A resource cannot be taken from a process unless the process releases the resource.
Circular Wait: A set of processes are waiting for each other in circular form.

Methods for handling deadlock

There are three ways to handle deadlock

1) Deadlock prevention or avoidance: The idea is to not let the system into deadlock state.

One can zoom into each category individually, Prevention is done by negating one of above mentioned necessary conditions for deadlock.

Avoidance is kind of futuristic in nature. By using strategy of "Avoidance", we have to make an assumption. We need to ensure that all information about resources which process WILL need are known to us prior to execution of the process. We use Banker's algorithm (Which is in-turn a gift from Dijkstra) in order to avoid deadlock.

2) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred.

3) Ignore the problem all together: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take