Phone Directory Application

1. ABC organization requires a phone directory application that can be used to fetch the contact details department-wise to notify the announcements, reminders, wishes, etc. Our role is to create a web application with user credentials. Only the registered users must be able to fetch the required details

The portal must contain the login page with email/login id and password. User Registration page must contain first name, last name, department, employeeId, contact details like primary number, secondary number, official mailId, secondary email, address information like door no, street name, city, state, Country, Pincode, country code, Nationality, NRI/resident

We have to define the entities, attributes to the entities, and cardinalities between the entities Our system design should follow the MVC design principles with reusable components, Separation of concerns.

Find the List of Entities from the above description?

UserDetails

Department

Address

Contact

Fill all the attributes related to Contact entity.

contactid

Countrycode

primarymobileno

Secondarymobileno

userid

To make a clone of a database

Copy the path of the Bin

Cd path of the Bin🡪mysqldump –u root -p training>F:\trainingdb.sql

Enter Password:

Enter into Shell -🡪 mysql –u root -p

1.To Find Highest Salary of an employee

select emp\_name,salary from EMPLOYEE\_DETAILS order by salary desc Limit 1 offset 2;

select emp\_name,salary from EMPLOYEE\_DETAILS a where 3=

(select count(DISTINCT salary)from EMPLOYEE\_DETAILS b

where b.salary>a.salary);

select emp\_name,salary from EMPLOYEE\_DETAILS a where 3=

(select count(DISTINCT salary)from EMPLOYEE\_DETAILS b

where a.salary<=b.salary);

2.select count(\*) number\_of\_users from (select userid,count(eventTime)doc\_per\_user

from EVENT\_DATE\_TIME group by userId) doc\_per\_user where doc\_per\_user>=3;

3. select cust\_name Customers from CUSTOMERS where id not in

(select customer\_id from ORDERS group by customer\_id)

order by cust\_name;

4. select max(num) num from

(select num from NUMBERS group by num having count(num)=1) t;

SELECT

SUM(INSURANCE.INV\_2019) AS INV\_2019

FROM

INSURANCE

WHERE

INSURANCE.INV\_2018 IN

(

SELECT

INV\_2018

FROM

INSURANCE

GROUP BY INV\_2018

HAVING COUNT(\*) > 1

)

AND CONCAT(LAT, CONCAT(',', LON)) IN

(

SELECT

CONCAT(LAT, CONCAT(',', LON))

FROM

INSURANCE

GROUP BY LAT , LON

HAVING COUNT(\*) = 1

)

;

6. select product\_name,list\_price,category\_id from PRODUCTS a

where list\_price in(select max(b.list\_price)from PRODUCTS b

where b.category\_id=a.category\_id group by b.category\_id)

order by category\_id,product\_name;

7. # You are using MYSQL

select product\_name,list\_price,category\_id from PRODUCTS p1 where list\_price in

(select max(p2.list\_price) from PRODUCTS p2 where

p2.category\_id=p1.category\_id group by p2.category\_id)

order by category\_id,product\_name;

Joins

1. select d.dept\_no as Dept,sum(salary)+sum(i.Incentive\_amount) Amount

from Employee\_LM e inner join Department\_LM d on e.Employee\_id=d.emp\_no

inner join Incentives\_LM i on e.Employee\_id=i.Id

where Year(e.Joining\_Date)=2010 group by d.dept\_no order by d.dept\_no asc;

# select s.name name from SALESPERSON s where s.sales\_id not in(

select o.sales\_id from ORDERS o JOIN COMPANY c on o.com\_id=c.com\_id

where c.name='ABC') order by name;

3. select a.Id,a.Month,sum(b.credit) credit from CREDIT a join CREDIT b on

a.Id=b.Id and a.Month>=b.Month where a.Month<>(select max(MONTH)

from CREDIT where Id=a.Id)group by a.Id,a.Month order by a.Id asc,a.Month des

4. select name Name from CANDIDATE join (select CandidateId from VOTE

group by CandidateId order by count(\*) desc limit 1) winner where

CANDIDATE.Id=winner.CandidateId;

5.select o.Order\_Name ORDER\_NAME,c.Cust\_Name CUST\_NAME from ORDER\_DETAILS o

left join CUSTOMER\_DETAILS c on o.Cust\_ID=c.Cust\_ID order by c.Cust\_Name;

6. select h.hacker\_id,h.hacker\_name from (

select a.hacker\_id,count(\*) cnt from (

select s.hacker\_id from Submissions s join Challenges c on

s.challenge\_id=c.challenge\_id join Difficulty d on

c.difficulty\_level=d.difficulty\_level where d.score=s.score

)a

group by a.hacker\_id having count(\*)>1)

b join Hackers h on h.hacker\_id=b.hacker\_id order by cnt desc,hacker\_id;

7.select a.Id Id from HUMIDITY a join HUMIDITY b on DATEDIFF(

a.RecordDate, b.RecordDate) = 1 and a.percentage<b.percentage;

Department table

create table department(departmentId varchar(30) not null,

departmentName varchar(30) not null,details varchar(50) not null,

primary key(departmentId));

Reorder Column

alter table department modify column details varchar(50) after departmentId;

**User Details**

create table userdetails(userId int not null auto\_increment,

firstName varchar(30) not null,lastName varchar(30) not null,

primaryEmailId varchar(30) not null,secondaryEmailId varchar(30) not null,

passKey varchar(30) not null,confirm\_passKey varchar(30) not null,

department\_id varchar(30) not null,designation varchar(30) not null,

primary key(userId),foreign key(department\_id)

references department(departmentId));

Address Table

create table address(

addressId int not null auto\_increment,

t\_doorNo varchar(30)not null,

t\_streetName varchar(30) not null,

t\_city varchar(30) not null,

t\_pinCode varchar(30) not null,

t\_state varchar(30)not null,

t\_country varchar(30)not null,

p\_doorNo varchar(30) not null,

p\_streetName varchar(30) not null,

p\_city varchar(30) not null,

p\_pinCode varchar(30) not null,

p\_state varchar(30) not null,

p\_country varchar(30) not null,

userId int not null,

primary key(addressId),Foreign key(userId)

references userdetails(userId));

**Contact Table**

create table if not exists contact(contactId int not null auto\_increment,

countryCode varchar(10) not null,primaryMobileNumber varchar(30) not null,

secondaryMobileNumber varchar(30) not null,userId int not null,

primary key(contactId),foreign key (userId) references userdetails(userId));

**Joins**

select u.userid,firstname,primaryEmailId,departmentName,t\_country

from userdetails u inner join department d on u.department\_id=d.departmentId

inner join address a on u.userId=a.userId;

select u.userid,u.firstname,u.primaryEmailId,d.departmentName from userdetails u

inner join department d on u.department\_id=d.departmentId;

n=int(input())

hash\_table = [[] for \_ in range(n)]

#print (hash\_table)

def insert(hash\_table, key, value):

hash\_key = hash(key) % len(hash\_table)

key\_exists = False

bucket = hash\_table[hash\_key]

for i, kv in enumerate(bucket):

k, v = kv

if key == k:

key\_exists = True

break

if key\_exists:

bucket[i] = ((key, value))

else:

bucket.append((key, value))

for i in range(n):

insert(hash\_table, int(input()), input())

#print(hash\_table)

def delete(hash\_table, key):

hash\_key = hash(key) % len(hash\_table)

key\_exists = False

bucket = hash\_table[hash\_key]

for i, kv in enumerate(bucket):

k, v = kv

if key == k:

key\_exists = True

break

if key\_exists:

del bucket[i]

print ('Key {} deleted'.format(key))

else:

print ('Key {} not found'.format(key))

Merge Sort

def mergeSort(myList):

if len(myList) > 1:

mid = len(myList) // 2

left = myList[:mid]

right = myList[mid:]

# Recursive call on each half

mergeSort(left)

mergeSort(right)

# Two iterators for traversing the two halves

i = 0

j = 0

# Iterator for the main list

k = 0

while i < len(left) and j < len(right):

if left[i] <= right[j]:

# The value from the left half has been used

myList[k] = left[i]

# Move the iterator forward

i += 1

else:

myList[k] = right[j]

j += 1

# Move to the next slot

k += 1

# For all the remaining values

while i < len(left):

myList[k] = left[i]

i += 1

k += 1

while j < len(right):

myList[k]=right[j]

j += 1

k += 1

myList = [54,26,93,17,77,31,44,55,20]

mergeSort(myList)

print(myList)

Kanpsack:

def knapSack(W, wt, val, n):

K = [[0 for x in range(W + 1)] for x in range(n + 1)]

# Build table K[][] in bottom up manner

for i in range(n + 1):

for w in range(W + 1):

if i == 0 or w == 0:

K[i][w] = 0

elif wt[i-1] <= w:

K[i][w] = max(val[i-1]

+ K[i-1][w-wt[i-1]],

K[i-1][w])

else:

K[i][w] = K[i-1][w]

return K[n][W]

# Driver code

val=list(map(int,input().split(' ')))

wt=list(map(int,input().split(' ')))

W=int(input())

n=len(val)

print(knapSack(W, wt, val, n))

def QuickSort(arr):

elements = len(arr)

#Base case

if elements < 2:

return arr

current\_position = 0 #Position of the partitioning element

for i in range(1, elements): #Partitioning loop

if arr[i] <= arr[0]:

current\_position += 1

temp = arr[i]

arr[i] = arr[current\_position]

arr[current\_position] = temp

temp = arr[0]

arr[0] = arr[current\_position]

arr[current\_position] = temp #Brings pivot to it's appropriate position

left = QuickSort(arr[0:current\_position]) #Sorts the elements to the left of pivot

right = QuickSort(arr[current\_position+1:elements]) #sorts the elements to the right of pivot

arr = left + [arr[current\_position]] + right #Merging everything together

return arr

array\_to\_be\_sorted = list(map(int,input().split(' ')))

print("Original Array: ",array\_to\_be\_sorted)

print("Sorted Array: ",QuickSort(array\_to\_be\_sorted))

class Node:

def \_\_init\_\_(self, val):

self.data = val

self.next = None

class QuickSortLinkedList:

def \_\_init\_\_(self):

self.head=None

def addNode(self,data):

if (self.head == None):

self.head = Node(data)

return

curr = self.head

while (curr.next != None):

curr = curr.next

newNode = Node(data)

curr.next = newNode

def printList(self,n):

while (n != None):

print(n.data, end=" ")

n = n.next

def paritionLast(self,start, end):

if (start == end or start == None or end == None):

return start

pivot\_prev = start

curr = start

pivot = end.data

while (start != end):

if (start.data < pivot):

pivot\_prev = curr

temp = curr.data

curr.data = start.data

start.data = temp

curr = curr.next

start = start.next

temp = curr.data

curr.data = pivot

end.data = temp

return pivot\_prev

def sort(self, start, end):

if(start == None or start == end or start == end.next):

return

pivot\_prev = self.paritionLast(start, end)

self.sort(start, pivot\_prev)

if(pivot\_prev != None and pivot\_prev == start):

self.sort(pivot\_prev.next, end)

elif (pivot\_prev != None and pivot\_prev.next != None):

self.sort(pivot\_prev.next.next, end)

if \_\_name\_\_ == "\_\_main\_\_":

ll = QuickSortLinkedList()

ll.addNode(30)

ll.addNode(3)

ll.addNode(4)

ll.addNode(20)

ll.addNode(5)

n = ll.head

while (n.next != None):

n = n.next

print("Linked List before sorting")

ll.printList(ll.head)

ll.sort(ll.head, n)

print("\nLinked List after sorting");

ll.printList(ll.head)