**Q1.** Create a table and store comma separated address into it,

**Eg:**

|  |
| --- |
| Address |
| A11, ganga colony, sirsi road, link road, jaipur |
| 122, shashtri nagar, jaipur |
| 35, Malviya nagar, jaipur |

Create a dynamic query to store comma separated values into different columns.

**Note:** columns should be dynamic.

**Expected Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address1 | Address2 | Address3 | Address4 | Address5 |
| A11 | ganga colony | sirsi road | link road | jaipur |
| 122 | shashtri nagar | jaipur | NULL | NULL |
| 35 | Malviya nagar | jaipur | NULL | NULL |

Ans.

create table addresss(

adress varchar(100)

)

insert into addresss values('10, bhavani peth,ghongade vasti,solapur')

insert into addresss values('20,yerwada,pune')

insert into addresss values('30,kachiguda,hyderabad')

with ct as(

select \*, row\_number() over(partition by adress order by adress) as rn from addresss

cross apply string\_split(adress,',')

)

--select \* from ct

select [1] as adrs1,[2] as adrs2,[3] as adrs3 ,[4] as adrs4 from ct

pivot (min(value) for rn in([1],[2],[3],[4])) as res

**Q2.** Write a query to find the sums of the amounts from the orders table, grouped by date, eliminating all those dates where the sum was not at least 1000.00 above the maximum amount for that date.

**Sample table: Order**

Table

Description automatically generated

**Sample table: Customer**

Table

Description automatically generated

Expected Output:

Table

Description automatically generated

Sol.

create table orders (ord\_no int, purch\_amt int,

ord\_date date, cust\_id int, salesman\_id int)

alter table orders alter column purch\_amt decimal

create table customer(

id int, name varchar(50),city varchar(50),grade int,salesman\_id int)

insert into customer values(301,'nick','solapur',100,501)

insert into customer values(302,'nick','solapur',100,501)

insert into customer values(302,'gram','Newyork',200,502)

insert into customer values(303,'pogo','london',300,503)

insert into customer values(304,'hung','Paris',300,504)

insert into customer values(305,'kids','Latur',300,505)

insert into customer values(301,'choco','Pune',200,501)

insert into customer values(302,'loco','Delli',200,502)

insert into customer values(303,'caren','Goa',200,506)

select ord\_date,SUM(purch\_amt) as sum from orders group by ord\_date having sum(purch\_amt)>500

**Task 17 (1 Day)** . Understand the concept of dimension tables in data modelling. Learn the importance and schema structure of **DimDate** table (date dimension table) in modelling and implement a stored procedure code to load 25 years(from today’s date) date data and its computed date fields in date dimension table.

**Task 18 (1 Day).** Learn the importance and schema structure of **DimTime** table (time dimension table) in modelling and implement a stored procedure code to load data and its computed time fields in time dimension table.

**Q3.** Create logic for the given Date Dimension table (use StartDate and EndDate variable for range):

create table DateDimension(

[Date] Date,

[Day\_of\_month] int,

[Day\_name] nvarchar(15),

[Day\_of\_week] nvarchar(15),

[Day\_of\_year] nvarchar(15),

[Quater\_Number] nvarchar(15),

[Week\_of\_year] nvarchar(15),

[Month] nvarchar(15),

[Month\_name] nvarchar(15),

[Day\_of\_Quater] nvarchar(15),

[Year] int,

[Week\_of\_Month] int,

[Week\_of\_Quarter] int,

[Month\_year] nvarchar(15),

[First\_Day\_of\_Month] DATE,

[Last\_Day\_of\_Month] DATE,

[First\_Day\_of\_Quater] DATE,

[Last\_Day\_of\_Quater] DATE,

[First\_Day\_of\_Year] DATE,

[Last\_Day\_of\_Year] DATE,

[Fiscal\_Year] varchar(50),

[Fiscal\_Year\_Month] int,

[Fiscal\_Year\_Month\_Name] nvarchar(15),

[Fiscal\_Year\_Week] nvarchar(15),

[Fiscal\_Year\_Day] nvarchar(15),

[Fiscal\_Year\_Day\_Name] nvarchar(15),

[Fiscal\_Year\_Day\_of\_Week] nvarchar(15),

[Fiscal\_Year\_Quater\_Number] int,

[Fiscal\_Year\_Day\_of\_Quater] nvarchar(15),

[Fiscal\_Year\_First\_Day\_of\_Month] DATE,

[Fiscal\_Year\_Last\_Day\_of\_Month] DATE,

[Fiscal\_Year\_First\_Day\_of\_Quater] DATE,

[Fiscal\_Year\_Last\_Day\_of\_Quater] DATE,

[Fiscal\_Year\_First\_Day\_of\_Year] DATE,

[Fiscal\_Year\_Last\_Day\_of\_Year] DATE,

[Is\_Weekday] nvarchar(15)

)

Reference:

Table

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated

Table

Description automatically generated

create or alter procedure dim\_date\_tbl @startdt date,@enddt date

as begin

declare @startdate date,@enddate date;

set @startdate=@startdt;

set @enddate=@enddt;

--declare @startdate date='2022-01-31',@enddate date='2022-03-01';

with ct as(

select @startdate as d

union all

select DATEADD(DAY,1,d) from ct

where d<@enddate)

select d,

DAY=DAY(d),

month=MONTH(d),

year=YEAR(d),

weekday=datename(WEEKDAY,d),

week=datename(week,d),

monthname=datename(MM,d),

shortyear=right(datename(yy,d),2),

shortmonth=left(datename(M,d),3),

quatr=datename(QUARTER,d),

weekdayy=left(datename(W,d),3),

dayyer=datename(dy,d)from ct

option (maxrecursion 32767)

end

exec dim\_date\_tbl '2022-04-01','2022-04-30'

2nd Method-----------------------------------------------------------------------------------------------------------------------

IF OBJECT\_ID('tempdb..#DateTmp') IS NOT NULL

BEGIN

DROP TABLE #DateTmp

END

CREATE TABLE #DateTmp (DateKey datetime NOT NULL)

-- Create temp table with all the dates we will use

DECLARE @StartDate datetime

DECLARE @EndDate datetime

SET @StartDate = '01/01/2005'

SET @EndDate = getdate()

DECLARE @LoopDate datetime

SET @LoopDate = @StartDate

WHILE @LoopDate <= @EndDate

BEGIN

INSERT INTO #DateTmp VALUES

(

@LoopDate

)

SET @LoopDate = DateAdd(dd, 1, @LoopDate)

END

INSERT INTO dbo.DimDate

SELECT

CAST(CONVERT(VARCHAR(8), DateKey, 112) AS int) , -- date key

DateKey, -- date alt key

Year(DateKey), -- calendar year

datepart(qq, DateKey), -- calendar quarter

Month(DateKey), -- month number of year

datename(mm, DateKey), -- month name

Day(DateKey), -- day number of month

datepart(dw, DateKey), -- day number of week

datename(dw, DateKey), -- day name of week

CASE

WHEN Month(DateKey) < 7 THEN Year(DateKey)

ELSE Year(DateKey) + 1

END, -- Fiscal year (assuming fiscal year runs from Jul to June)

CASE

WHEN Month(DateKey) IN (1, 2, 3) THEN 3

WHEN Month(DateKey) IN (4, 5, 6) THEN 4

WHEN Month(DateKey) IN (7, 8, 9) THEN 1

WHEN Month(DateKey) IN (10, 11, 12) THEN 2

END -- fiscal quarter

FROM #DateTmp

GO

CREATE TABLE [dbo].[DimDate]

(

[DateKey] [int] NOT NULL,

[DateAltKey] [datetime] NOT NULL,

[CalendarYear] [int] NOT NULL,

[CalendarQuarter] [int] NOT NULL,

[MonthOfYear] [int] NOT NULL,

[MonthName] [nvarchar](15) NOT NULL,

[DayOfMonth] [int] NOT NULL,

[DayOfWeek] [int] NOT NULL,

[DayName] [nvarchar](15) NOT NULL,

[FiscalYear] [int] NOT NULL,

[FiscalQuarter] [int] NOT NULL

)

WITH

(

DISTRIBUTION = REPLICATE,

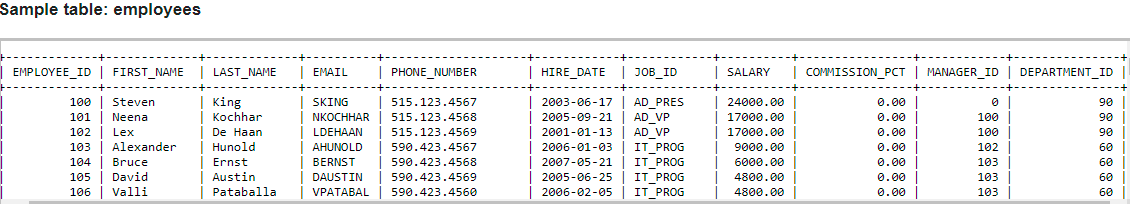
CLUSTERED COLUMNSTORE INDEX

);

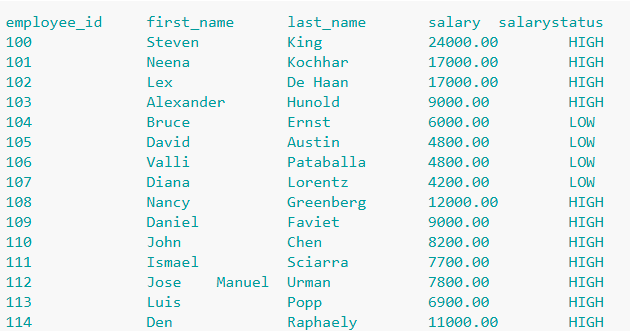
GO

**Q1.** Write a query to display the employee id, Full Name ( first name and last name ), salary and the Salary Status column with a title HIGH and LOW respectively for those employees whose salary is more than and less than the average salary of all employees.

**Eg: Sample table : Employee**



**Expected Output**



CREATE TABLE empl(EMPNO int NOT NULL, fNAME VARCHAR(10),LNAME VARCHAR(10), JOB

VARCHAR(9), MGR int, HIREDATE DATETIME,

SAL int, COMM int, DEPTNO tinyint);

INSERT INTO EMPl VALUES(7369, 'SMITH','SMITH', 'CLERK', 7902,'1980-DEC-17', 800, NULL, 20);

INSERT INTO EMPl VALUES(7499, 'ALLEN','ALLEN', 'SALESMAN', 7698,'1981-FEB-20', 1600, 300, 30);

INSERT INTO EMPl VALUES(7521, 'WARD', 'WARD', 'SALESMAN', 7698,'1981-FEB-22', 1250, 500, 30);

INSERT INTO EMPl VALUES(7566, 'JONES','JONES', 'MANAGER', 7839,'2-APR-1981', 2975, NULL, 20);

INSERT INTO EMPl VALUES(7654, 'MARTIN','MARTIN', 'SALESMAN', 7698,'28-SEP-1981', 1250, 1400, 30);

INSERT INTO EMPl VALUES(7698, 'BLAKE', 'BLAKE', 'MANAGER', 7839,'1-MAY-1981', 2850, NULL, 30);

INSERT INTO EMPl VALUES(7782, 'CLARK', 'CLARK', 'MANAGER', 7839,'9-JUN-1981', 2450, NULL, 10);

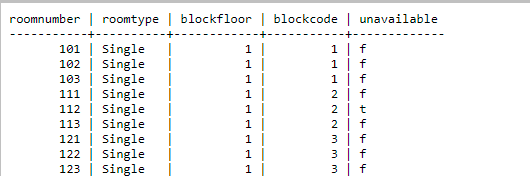
INSERT INTO EMPl VALUES(7788, 'SCOTT', 'SCOTT', 'ANALYST', 7566, '09-DEC-1982', 3000, NULL, 20);

INSERT INTO EMPl VALUES(7839, 'KING', 'KING', 'PRESIDENT', NULL,'17-NOV-1981', 5000, NULL, 10);

select empno,fNAME+' '+LNAME as fullname,SAL ,IIF(sal>(select avg(sal) from empl),'High','Low') as sal\_status from empl

**Q2.** Write a query in SQL to count the number of available rooms in each block.

**Eg: Sample table: Room**

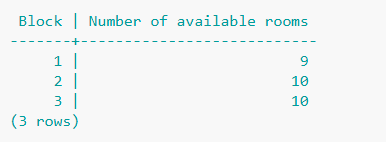


**This table has a column name unavailable which contains two values**

1. **F (F means False)**
2. **T (T Mean True)**

Write the query to see this Expected output.

Expected Output:



create table room (r\_no int,r\_type varchar(20),block\_floor int,block\_code int,availabilityy varchar(2))

insert into room values(101,'single',1,1,'f')

insert into room values(102,'single',1,1,'f')insert into room values(101,'single',1,1,'t')

insert into room values(103,'single',1,2,'f')

insert into room values(104,'single',1,2,'f')

insert into room values(105,'single',1,3,'f')

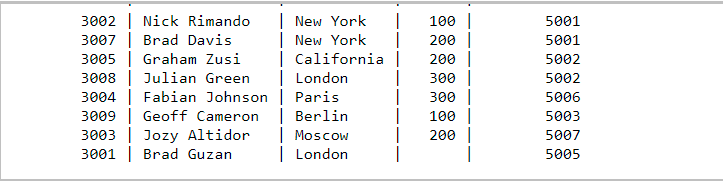
insert into room values(106,'single',1,3,'t')

insert into room values(107,'single',1,3,'t')

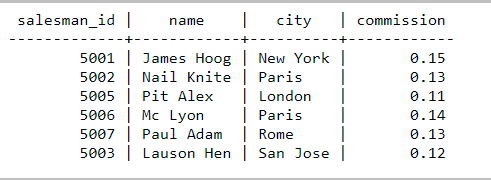
select block\_code , count(availabilityy) as no\_of\_available\_rooms from room where availabilityy='t' group by block\_code

**Q3.** Write a query to find all the salesmen who worked for only one customer.

**Eg: Sample table : Customer**

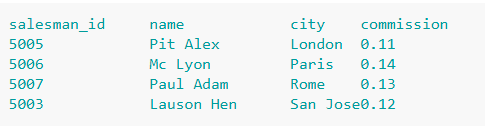


**Sample table : Salesman**

****

Write the query to see this Expected output.

**Expected Output:**

****

create table salesman(s\_id int,s\_name varchar(50),city varchar(30),commission decimal)

insert into salesman values(501,'james hoog','solapur',0.0)

insert into salesman values(502,'nail knite','surat',0.5)

insert into salesman values(503,'pit alex','goa',0.6)

insert into salesman values(504,'mc lyon','solar',0.7)

insert into salesman values(505,'paul adam','asam',0.3)

insert into salesman values(506,'lauson hen','deli',0.1)

insert into salesman values(507,'g','kerla',0.2)

select salesman.s\_id,salesman.s\_name,salesman.city,salesman.commission from salesman join customer on customer.salesman\_id=salesman.s\_id group by salesman.s\_id,salesman.s\_name,salesman.city,commission

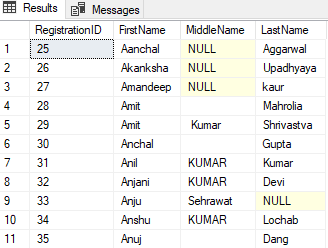
having COUNT(salesman.s\_id)<2

**Q2.** Create three tables named Registration, MonthMaster and PayMonthly . Registration table have some Employee Data and MonthMaster has some information about month with their ID’s and PayMonthly table contains salary data of each employee on month and yearly basis.

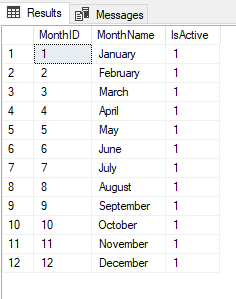
Write a dynamic SQL query to generate the expected output.

**Note :- RegistrationID in registration table equivalent to Rgenid of PayMonthly table and MonthID of MonthMaster equivalent to PayMonth of PayMonthly table.**

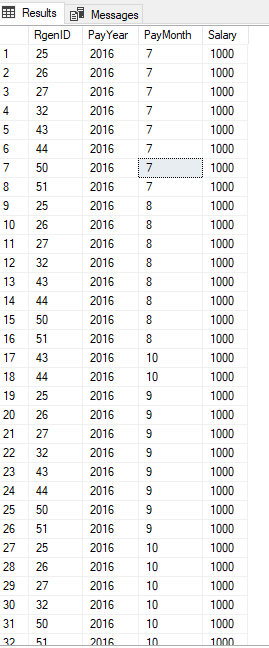
**Eg: Sample table: Registration**



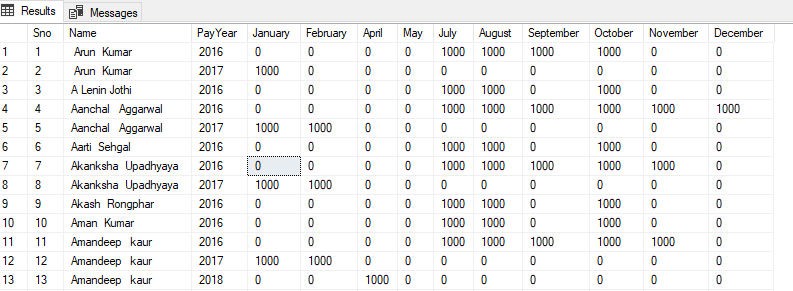
**Eg: Sample table: MonthMaster**



**Eg: Sample table: PayMonthly**



**Expected output :**



create table registraion (id int,fname varchar(50),mname varchar(50),lname varchar(50))

insert into registraion values(1,'anchal','a','agrwal')

insert into registraion values(2,'aman','a','bhalla')

insert into registraion values(3,'anju','','')

insert into registraion values(4,'deep','','koth')

insert into registraion values(5,'akansha','','upadhyay')

insert into registraion values(6,'amit','','')

insert into registraion values(7,'bunti','','')

insert into registraion values(8,'venky','','macha')

insert into registraion values(9,'kedar','','m')

insert into registraion values(10,'anil','kumar','null')

insert into registraion values(11,'ravi','kumar','null')

insert into registraion values(12,'mahesh','kumar','')

insert into registraion values(13,'raj','','patil')

create table monthmaster(m\_no int,m\_name varchar(50),is\_active int)

insert into monthmaster values(1,'January',1)

insert into monthmaster values(2,'February',1)

insert into monthmaster values(3,'March',1)

insert into monthmaster values(4,'April',1)

insert into monthmaster values(5,'May',1)

insert into monthmaster values(6,'June',1)

insert into monthmaster values(7,'July',1)

insert into monthmaster values(8,'august',1)

insert into monthmaster values(9,'September',1)

insert into monthmaster values(10,'october',1)

insert into monthmaster values(11,'November',1)

insert into monthmaster values(12,'December',1)

create table paymonthly(id int ,year int,month int,salary money)

insert into paymonthly values(1,2016,5,1000)

insert into paymonthly values(2,2016,5,1000)

insert into paymonthly values(3,2016,5,1000)

insert into paymonthly values(4,2016,4,1000)

insert into paymonthly values(5,2016,4,1000)

insert into paymonthly values(6,2016,4,1000)

insert into paymonthly values(7,2016,3,1000)

insert into paymonthly values(8,2016,3,1000)

insert into paymonthly values(9,2016,2,1000)

insert into paymonthly values(10,2016,2,1000)

insert into paymonthly values(11,2016,1,1000)

insert into paymonthly values(12,2016,6,1000)

insert into paymonthly values(1,2016,6,1000)

insert into paymonthly values(2,2016,7,1000)

insert into paymonthly values(3,2016,7,1000)

insert into paymonthly values(4,2016,8,1000)

insert into paymonthly values(5,2016,9,1000)

insert into paymonthly values(6,2016,10,1000)

insert into paymonthly values(7,2016,11,1000)

insert into paymonthly values(8,2016,12,1000)

with ct as(

select fname+' '+mname+' '+lname AS fullName,p.year,p.salary,m.m\_name

from registraion r join paymonthly p on r.id=p.id join monthmaster m on m.m\_no=p.month

)

select row\_number() over(order by year) as Sr\_no, fullname,year,January,[february],[march],[april],[May],[June],

[July],[August],[September],[October],[november],[December] from ct

pivot(max([salary]) for [m\_name] in([January],[february],[march],[april],[May],[June],

[July],[August],[September],[October],[november],[December])) as months

**Q3.**

Step1: Create a table named Record which have 2 columns are there named ID and Name.

Step2 : Create a another table named Audit which have 3 columns named ID, Check-in and Check-out.

Q3 A): Create a stored procedure to insert the data in the Record table with ID and name.

Q3 B): whenever the data is inserted or updated in the record table then the audit table should store the check-in and check-out date and time with respective their ID’s.

When the data inserted then Check-in column of audit table is updated with current date and time and when the data is Deleted then update the Check-out column with current date and time with their respective ID’s.

**Note: ID should be unique and auto incremental and do not use triggers.**

create table record (id int identity, name varchar(50))

create table audit(id int identity ,check\_in datetime,check\_out datetime)

create or alter procedure insert\_records @name varchar(50)

as begin

declare @id int

set @id=SCOPE\_IDENTITY()

declare @check\_in datetime

set @check\_in=getdate()

insert into record values(@name)

insert into audit values(@check\_in,null)

end

exec insert\_records 'tom'

create or alter procedure del\_record @name varchar(50),@id int

as begin

declare @idd int

set @idd=SCOPE\_IDENTITY()

declare @check\_out datetime

set @check\_out=getdate()

delete from record where @id=id

update audit set check\_out=@check\_out where id=@id

end

exec del\_record 'clc',2

Q. You have to write a query to find out whether the column can be consider as a primary key or not.

|  |  |  |
| --- | --- | --- |
| Id | Name | Dept |
| 1 | Ram | CS |
| 2 | Shyam | Mech |
| 3 | Sania | EC |
| 4 | Aditi | CS |
| 5 | Ram | EC |

So in the above table assume that it contains millions of records. I want a query in which I will insert **tablename** and **columnname** and when I will execute the quert I will come to know whether I can consider it as a Primary Key or Not in the output

Ans.

create or alter procedure check\_pk @tbl\_name varchar(50),@col\_name varchar(50)

as begin

declare @sql nvarchar(max)

set @sql ='select case when COUNT(distinct '+@col\_name+')=COUNT('+@col\_name+') then ''Pk'' else '' no '' end from '+@tbl\_name

execute sp\_executesql @sql

end

exec check\_pk 'powerbi','fees'

exec check\_pk 'empl','empno'

|  |  |  |  |
| --- | --- | --- | --- |
| **Task 2** | | | |
| Repetation of same record with its count | | | |
|  | | | |
| You have to write a query to find out the count of repetation of value in a table. For example: | | | |
| Id | Name | Dept |
| 1 | Ram | CS |
| 2 | Shyam | Mech |
| 3 | Sania | EC |
| 4 | Aditi | CS |
| 5 | Ram | EC |

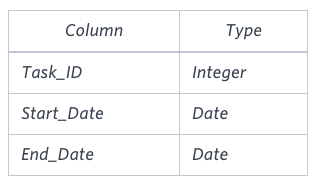
If the column name is **Name** in the query the then the expected output is-->

|  |  |
| --- | --- |
| Name | Count |
| Ram | 2 |
| Shyam | 1 |
| Sania | 1 |
| Aditi | 1 |

Ans.

select college, COUNT(\*) as count,COUNT(college) cnt from powerbi group by college

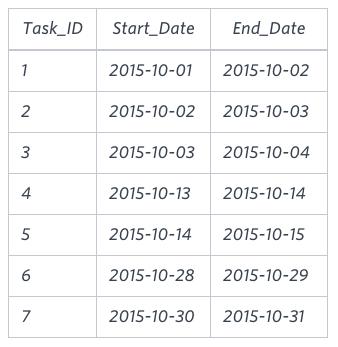
**Task 1 (1 Hour)**.You are given a table, Projects, containing three columns: Task\_ID, Start\_Date and End\_Date. It is guaranteed that the difference between the End\_Date and the Start\_Date is equal to 1 day for each row in the table.



If the End\_Date of the tasks are consecutive, then they are part of the same project. Samantha is interested in finding the total number of different projects completed.

Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

**Sample Input**



**Sample Output**

2015-10-28 2015-10-29

2015-10-30 2015-10-31

2015-10-13 2015-10-15

2015-10-01 2015-10-04

create table projects(task\_id int,start\_Dates date,End\_Dates date)

insert into projects values (1,'2015-10-01','2015-10-02'),(2,'2015-10-02','2015-10-03'),(3,'2015-10-03','2015-10-04'),(4,'2015-10-13','2015-10-14'),(5,'2015-10-14','2015-10-15'),(6,'2015-10-28','2015-10-29'),(7,'2015-10-30','2015-10-31')

SELECT Start\_Dates,min(End\_Dates) FROM

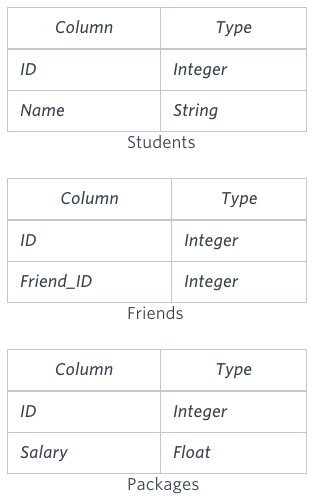
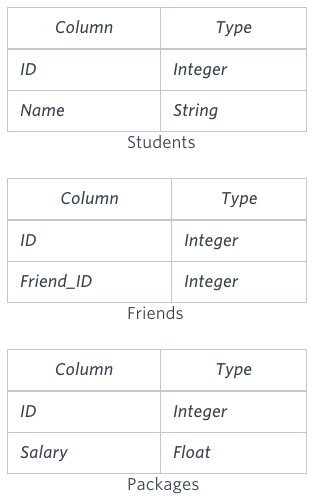
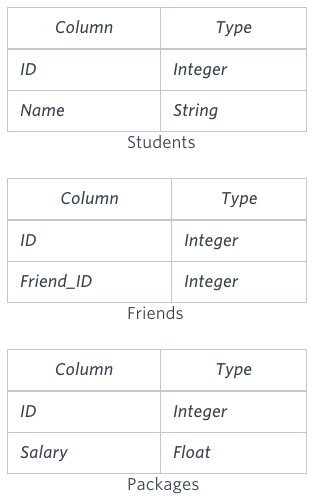
(SELECT start\_Dates FROM Projects WHERE start\_Dates NOT IN (SELECT End\_Dates FROM Projects)) a,

(SELECT End\_Dates FROM Projects WHERE End\_Dates NOT IN (SELECT start\_Dates FROM Projects)) b

WHERE Start\_Dates < End\_Dates group by start\_Dates

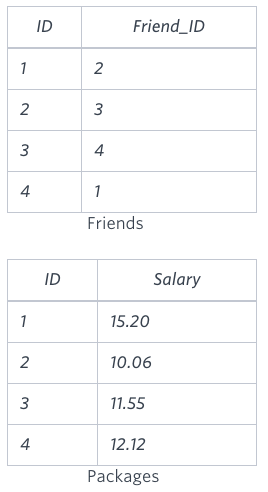
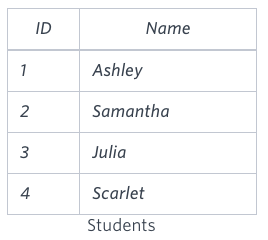
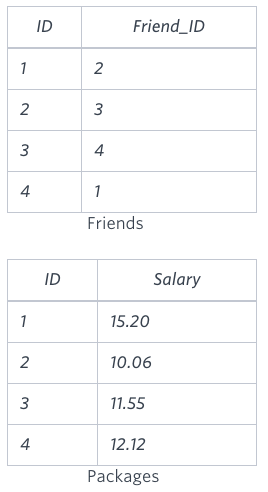
ORDER BY DATEDIFF(DD,min(End\_Dates), Start\_Dates)DESC , Start\_Dates ASC;

**Task 2 (1 hour).** You are given three tables: Students, Friends and Packages. Students contains two columns: ID and Name. Friends contains two columns: ID and Friend\_ID (ID of the ONLY best friend). Packages contains two columns: ID and Salary (offered salary in $ thousands per month).



Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students got same salary offer.

**Sample Input**



**Sample Output**

Samantha

Julia

Scarlet

SELECT s.Name

FROM Students s

JOIN Packages p1 ON s.ID = p1.ID

JOIN Friends f ON s.ID = f.ID

JOIN Packages p2 ON f.Friend\_ID = p2.ID

WHERE p2.Salary > p1.Salary

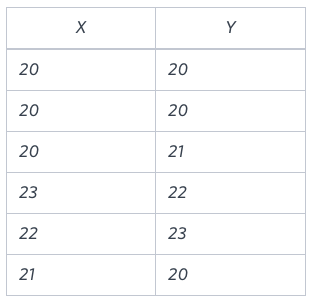
ORDER BY p2.Salary

**Task 3 (20 mins).** You are given a table, Functions, containing two columns: X and Y.

Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.

Write a query to output all such symmetric pairs in ascending order by the value of X.

**Sample Input**



**Sample Output**

20 20

20 21

22 23

SELECT f1.X, f1.Y FROM funtions AS f1

WHERE f1.X = f1.Y AND

(SELECT COUNT(\*) FROM funtions WHERE X = f1.X AND Y = f1.Y) > 1

UNION

SELECT f1.X, f1.Y from funtions AS f1

WHERE EXISTS(SELECT X, Y FROM funtions WHERE f1.X = Y AND f1.Y = X AND f1.X < X)

order BY X;

**Task 6 (20 mins).** Consider P1(a,b) and P2(c,d) to be two points on a *2D* plane.

* happens to equal the minimum value in *Northern Latitude* (*LAT\_N* in **STATION**).
* happens to equal the minimum value in *Western Longitude* (*LONG\_W* in **STATION**).
* happens to equal the maximum value in *Northern Latitude* (*LAT\_N* in **STATION**).
* happens to equal the maximum value in *Western Longitude* (*LONG\_W* in **STATION**).

Query the [Manhattan Distance](https://xlinux.nist.gov/dads/HTML/manhattanDistance.html) between points P1 and P2 and round it to a scale of  decimal places.

**Input Format**

The **STATION** table is described as follows:



where *LAT\_N* is the northern latitude and *LONG\_W* is the western longitude.

create table station (id int,city varchar(26),LAT\_N floAT ,LONG\_W FLOAT)

insert into station values (1,'solapur',78.564646,45.5897987),(2,'pune',22.568846,78.6868),(3,'latur',62.524556,89.56545),(4,'surat',28.664565,74.56465)

select ROUND(ABS(MAX(LAT\_N) - MIN(LAT\_N)) + ABS(MAX(LONG\_W) - MIN(LONG\_W)), 4) FROM STATION;

**Task 7 (1 hour).** Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line, and use the ampersand (&) character as your separator (instead of a space).

For example, the output for all prime numbers <=10 would be:

WITH PRIME AS (

SELECT 2 AS NUM

UNION ALL

SELECT A.NUM+1 AS VALUE

FROM PRIME A

WHERE A.NUM <100

)

---- by using recursvive cte print all the numbers between 1 to 100

select \* from prime

SELECT NUM FROM PRIME a

WHERE NOT EXISTS (

SELECT num FROM PRIME B WHERE A.NUM % B.NUM = 0 AND A.NUM != B.NUM

)

--- the above correlated query returns the numbers which are not prime so by using not exists condition

--- we extract prime numbers from prime cte

--- the where condition inside the correlated sub query checks by using logic and (True and false) if both conditions are True then it returns the row.

Declare @i int=2 ,@prime int=0

Declare @out table (num int)

while @i<100

begin

Declare @j int

set @j = @i-1

set @prime=1

while @j>1

begin

if @i % @j = 0

begin

set @prime=0

end

set @j=@j-1

end

if @prime=1

begin

insert @out values (@i)

end

set @i=@i+1

end

select \* from @out

**Task 8 (2 hours).** [Pivot](https://en.wikipedia.org/wiki/Pivot_table) the Occupation column in **OCCUPATIONS** so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

**Note:** Print **NULL** when there are no more names corresponding to an occupation.

**Input Format**

The **OCCUPATIONS** table is described as follows:

Occupation will only contain one of the following values: **Doctor**, **Professor**, **Singer** or **Actor**.

**Sample Input**



**Sample Output**

Jenny Ashley Meera Jane

Samantha Christeen Priya Julia

NULL Ketty NULL Maria

create table occupations(name varchar(50), occupation varchar(50))

INSERT INTO occupations(name, occupation) values('Samantha','Doctor')

INSERT INTO occupations(name, occupation) values('Julia','Actor')

INSERT INTO occupations(name, occupation) values('Maria','Actor')

INSERT INTO occupations(name, occupation) values('Meera','Singer')

INSERT INTO occupations(name, occupation) values('Ashely','Professor')

INSERT INTO occupations(name, occupation) values('Ketty','Professor')

INSERT INTO occupations(name, occupation) values('Christeen','Professor')

INSERT INTO occupations(name, occupation) values('Jane','Actor')

INSERT INTO occupations(name, occupation) values('Jenny','Doctor')

INSERT INTO occupations(name, occupation) values('Priya','Singer')

ans.

select Doctor, Professor, Singer, Actor from

(select a.name, a.occupation, count(\*) as row\_num from Occupations a join Occupations b

on a.occupation=b.occupation and (a.name>=b.name)

group by a.name, a.occupation

) as chill

PIVOT(max(name) for occupation in (Doctor, Professor, Singer, Actor))as pvt;

**Task 9 (2 hours).** You are given a table, *BST*, containing two columns: *N*and *P,* where *N* represents the value of a node in *Binary Tree*, and *P* is the parent of *N*.

Write a query to find the node type of *Binary Tree* ordered by the value of the node. Output one of the following for each node:

* *Root*: If node is root node.
* *Leaf*: If node is leaf node.
* *Inner*: If node is neither root nor leaf node.

**Sample Input**



**Sample Output**

1 Leaf

2 Inner

3 Leaf

5 Root

6 Leaf

8 Inner

9 Leaf

create table bst(N int ,p int)

insert into bst values(1,2),(3,2),(6,8),(9,8),(2,5),(8,5),(5,null)

select n, case when p is null then 'Root'

when n in (select distinct p from chill4) then 'Inner'

else 'Leaf'

end

from bst as AB

order by n

**Task 12.** Find top 5 employees according to salary (without order by).

select \* from emp e where 5 > (Select count(distinct(Salary)) from emp m where m.salary > e.salary)

**Task 13.** Swap value of two columns in a table without using third variable or a table.

create table swaptable (id int ,names varchar(78),sname varchar(28))

insert into swaptable values (1,'kumar','suresh'),(2,'latha','madhu'),(3,'babu','ramesh')

update swaptable set names=sname,sname=names

select \* from swaptable

**Task 14.** Create a user ,create a login for that user provide permissions of DB\_owner to the user.

create login sqldev with PASSWORD = 'p@assw0rd'

create user sqldev for login sqldev

alter role [db\_owner] add member sqldev

**Task 19 (1 Day)**. Understand the concept of Slowing Changing Dimension and its types in Data Modelling. Create a stored procedure to implement SCD-type1 logic for a sample dimension table(take it any table).

**Task 20 (2 Day)**. Create a stored procedure to implement SCD-type2 logic for a sample dimension table(take it any table).

create table Target\_Emp

(ID int, Ename varchar(70), Location varchar(70),Salary money, Role varchar(80), StartDate date,

EndDate date, Active\_Flag varchar(3))

drop table target\_emp

insert into Target\_Emp values (101, 'Kedar', 'Hyderabad', 20000, 'Big Data Associate','1-05-2022',null,'Y')

insert into Target\_Emp values (102, 'Vishwanath', 'Mumbai', 25000, 'Back End Developer Trainee','1-05-2022',null,'Y')

insert into Target\_Emp values (103, 'Manikanth', 'Pune', 15000, 'Data Intergration Associate','1-05-2022',null,'Y')

insert into Target\_Emp values (104, 'Vikram', 'Solapur', 25000, 'Data Science Intern','5-05-2022',null,'Y')

insert into Target\_emp values (105, 'Sandeep', 'Chennai', 20000, 'Cloud Infra & Security Trainee','1-05-2022',null,'Y')

create table Source\_Emp

(ID int, Ename varchar(70), [Location] varchar(70),Salary money, Role varchar(40))

insert into Source\_Emp values (101, 'Kedar', 'Hyderabad', 20000, 'Big Data Associate')

insert into Source\_Emp values (102, 'Vishwanath', 'Mumbai', 25000, 'Back End Developer Trainee')

insert into Source\_Emp values (103, 'Manikanth', 'Pune', 15000, 'Data Intergration Associate')

insert into Source\_Emp values (104, 'Vikram', 'Solapur', 25000, 'Data Science Intern')

insert into source\_emp values (105, 'Sandeep', 'Chennai', 20000, 'Cloud Infra & Security Trainee')

insert into source\_emp values (106, 'deep', 'Chennai', 30000, 'Cloud')

update source\_emp set ename= 'Venky' where ID=104

update source\_emp set role= 'IT' where ID=102

create or alter procedure scd2

as begin

merge target\_emp as t

using source\_emp as s

on t.id=s.id and t.active\_flag='Y'

when matched and

T.ename <> S.ename or T.Location <> S.Location or T.Salary <> S.Salary or T.Role <> S.Role and T.Active\_Flag = 'Y'

then update set T.EndDate = getdate(),T.Active\_Flag = 'N';

merge target\_emp as t

using source\_emp as s

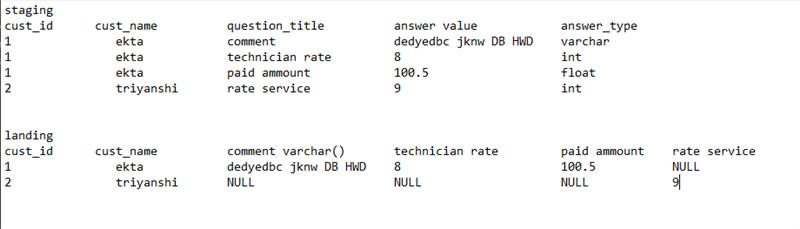
on t.id=s.id and t.active\_flag='Y'

when not matched then

insert values (s.id,s.ename,s.location,s.salary,s.role,getdate(),null,'Y');

end

exec SCD2

**Task---create columns without using pivot function..  
**

create table staging (id int,names varchar(50),quen varchar(50),ansr varchar(50))

insert into staging values(1,'ekta','coment','kadfl')

insert into staging values(1,'ekta','Tech\_rate','8')

insert into staging values(1,'ekta','Paid\_amt','100.5')

insert into staging values(2,'tara','rate\_service','9')

select id,names,

min(case when [quen]='coment' then [ansr] else null end) as [comment],

min(case when [quen]='Tech\_rate' then [ansr] else null end) as [tech\_rate],

min(case when [quen]='paid\_amt' then [ansr] else null end) as [paid\_amt],

min(case when [quen]='rate\_service' then [ansr] else null end) as [rate\_service]

from (select ID,names,quen,ansr,ROW\_NUMBER() over(partition by quen order by id) as rn from staging) stg

group by rn,id,names order by rn

-- the min agg function is used because it is not contained in group by function & also to take value from column

**Task 22**— you are given Customer\_Details table having 2 columns customer Id as Cust\_id and check-in/checkout date and time as check-in/checkout.

Write a query to output 3 columns as cust\_id, check-in and checkout.

Note: - do not use min, max type of aggregation.

**Input Table:-**

***Cust\_id   Check-in/checkout***

102         21-02-2020 10:32:10

101         22-02-2020 12:45:36

101         20-02-2020 12:45:36

102         22-02-2020 11:12:56

**Output Table:-**

***Cust\_id Check-in Checkout***

101 20-02-2020 12:45:36 22-02-2020 12:45:36

102 21-02-2020 10:32:10 22-02-2020 11:12:56

create table customerr(id int,check\_iO date)

insert into customerr values(101,'2022-01-01'),(101,'2022-01-01'),(102,'2022-01-02'),(102,'2022-01-04');

insert into customerr values(103,'18/06/12 10:34:09 AM'),(103,'18/06/12 7:34:09 PM')

select ID, (select min(check\_iO) from customerr c1 where c2.id=c1.id) as c\_in,

(select max(check\_iO) from customerr c1 where c2.id=c1.id) as c\_out from customerr c2 group by id

**Task 3**: You need to combine the values of column name with respect to schema name and table name and it should be separated by ',' and if schema name and column name are same all the column names should come in one row.

**Input:**

|  |  |  |
| --- | --- | --- |
| TableName:Information |  |  |
| Schema\_Name | Table\_Name | Column\_Name |
| dbo | Project | Project\_Id |
| dbo | Project | Project\_Name |

**Output:**

|  |  |  |
| --- | --- | --- |
| Schema\_Name | Table\_Name | Column\_Name |
| dbo | Project | Project\_Id,Project\_Name |

create table information (schema\_name varchar(25),table\_name varchar(25), column\_name varchar(25))

insert into information values('dbo','project','P\_id' ),('dbo','project','P\_name' )

insert into information values('dbo1','project','P\_id' ),('dbo','project1','P\_id' )

insert into information values('dbo1','project','P\_name' ),('dbo','project1','P\_name' )

select i2.SCHEMA\_NAME,i2.table\_name,

STUFF((select ', ' + i1.column\_name from information i1

where i1.schema\_name=i2.schema\_name

for XML path ('')),1,1,'') columns

from information i2 group by schema\_name,table\_name

select min(schema\_name) schema\_names ,min(table\_name) table\_names ,STRING\_AGG(column\_name,',') column\_name from information

group by table\_name,schema\_name;

**Task 4:** I have so many columns in a table but only want to apply aggregation on several particular Characteristics not all the columns. (Create Dynamic Procedure)

**Input:** Exec Procedure**\_**Name @Prod\_Desc=’Detergent’,@Prod\_Code= (‘12012’,’12011’,’12014’)

**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| Prod\_Desc | Prod\_Code | Prod\_Qty | Prod\_Price |
| Detergent | 12012 | 1 | 120 |
| Detergent | 12011 | 1 | 90 |
| Detergent | 12014 | 1 | 130 |

**Input:** Exec Procedure**\_**Name @Prod\_Desc=’Detergent’,@Prod\_Code= ‘’

**Output:**

|  |  |  |
| --- | --- | --- |
| Prod\_Desc | Prod\_Qty | Prod\_Price |
| Detergent | 3 | 340 |

create table store\_table4(prod\_desc varchar(100),prod\_code int,prod\_qty int,prod\_price int)

insert into store\_table4 values ('Detergent',12012,1,120),('Detergent',12011,1,90),('Detergent',12014,1,120)

insert into store\_table4 values ('Deter',12012,1,120)

select \* from store\_table4

drop table store\_table4

select min(prod\_desc) as prod\_desc ,sum(prod\_qty) prod\_qty ,sum(prod\_price) total\_prod\_price from store\_table4 where prod\_code in (12012,12011)

CREATE OR ALTER PROCEDURE TASK4 @list varchar(max)

AS BEGIN

declare @sql nvarchar(max)

set @sql = 'SELECT PROD\_DESC,sum(Prod\_Qty) as Prod\_Qty,sum(Prod\_Price) as Prod\_Price FROM store\_table4

where prod\_code in ('+@list+') group by prod\_desc'

execute sp\_executesql @sql

END

declare @pcode varchar(max)='12011,12012'

exec TASK4 @List = @pcode

**Task 5:** You need to write the logic of previous start date with respect to particular ID like example for 101 first row should be start date= '2020-10-20' and for this row the previous start date should be '2020-10-19' and for start date='2020-10-19' previous start date should be NULL because for 101 we don't have further row available same goes for ID 102 where we have two small start dates in comparision with ‘2020-10-20’.

**Input:**

|  |  |
| --- | --- |
| TableName:Project |  |
| Id | StartDate |
| 101 | 10/19/2020 |
| 101 | 10/20/2020 |
| 102 | 10/19/2020 |
| 102 | 10/20/2020 |
| 102 | 10/18/2020 |

**Output:**

|  |  |  |
| --- | --- | --- |
| id | StartDate | Previous Start Date |
| 101 | 10/20/2020 | 10/19/2020 |
| 101 | 10/19/2020 | Null |
| 102 | 10/20/2020 | 10/19/2020 |
| 102 | 10/19/2020 | 10/18/2020 |
| 102 | 10/18/2020 | Null |

create table project5

(id int , startdate date)

insert into project5 values(101,'10/19/2020'),(101,'10/20/2020'),(102,'10/19/2020'),(102,'10/20/2020'),(102,'10/18/2020')

drop table project5

select \* from project5

select id,startdate,LEAD(STARTDATE) OVER(PARTITION BY ID ORDER BY STARTDATE desc) AS Previous\_Start\_Date FROM project5

**Task 6:** Given a table events with the following structure:

 create table events (

      event\_type integer not null,

      value integer not null,

      time timestamp not null,

      unique(event\_type, time)

  );

write an SQL query that, for each event\_type that has been registered more than once, returns the difference between the latest (i.e. the most recent in terms of time) and the second latest value. The table should be ordered by event\_type (in ascending order).

For example, given the following data:

  event\_type | value      | time

  ------------+------------+--------------------

   2          | 5          | 2015-05-09 12:42:00

   4          | -42        | 2015-05-09 13:19:57

   2          | 2          | 2015-05-09 14:48:30

   2          | 7          | 2015-05-09 12:54:39

   3          | 16         | 2015-05-09 13:19:57

   3          | 20         | 2015-05-09 15:01:09

your query should return the following rowset:

  event\_type | value

  ------------+-----------

   2          | -5

   3          | 4

For the event\_type 2, the latest value is 2 and the second latest value is 7, so the difference between them is −5.

 The names of the columns in the rowset don't matter, but their order does.

create table events (event\_type integer not null, value integer not null,time datetime not null, unique(event\_type, time));

insert into events values(2,5,'2015/05/09 12:42:00'),(2,2,'2015-05-09 14:48:30'),(2,7,'2015-05-09 12:54:39')

insert into events values(4,-42,'2015-05-09 13:19:57'),(3,16,'2015-05-09 13:19:57'),(3,20,'2015-05-09 15:01:09')

select A1.event\_type, A1.value - A2.value from

(select event\_type, value,rank() over(partition by event\_type order by time desc) rnk from events)A1,

(select event\_type,value,rank() over(partition by event\_type order by time desc) rnk from events)A2

where A1.event\_type = A2.event\_type

and A1.rnk = 1

and A2.rnk = 2

**Task 8:**

id is the primary key column for this table.

Each row of this table indicates the ID of an employee, their name, salary, and the ID of their manager.

Write an SQL query to find the employees who earn more than their managers.

Return the result table in **any order**.

The query result format is in the following example.

**Input:**

Employee table:

+----+-------+--------+-----------+

| id | name | salary | managerId |

+----+-------+--------+-----------+

| 1 | Joe | 70000 | 3 |

| 2 | Henry | 80000 | 4 |

| 3 | Sam | 60000 | Null |

| 4 | Max | 90000 | Null |

+----+-------+--------+-----------+

**Output:**

+----------+

| Employee |

+----------+

| Joe |

+----------+

create table employee\_table (id int,name varchar(100),salary int,mangerid int)

insert into employee\_table values (1,'joe',70000,3),(2,'henry',80000,4),(3,'sam',60000,null),(4,'max',90000,null)

select \* from employee\_table

select a.Name from employee\_table a,employee\_table b where a.Salary > b.Salary and a.mangerid = b.Id

**Task 10:**

Table: Trips

| id | client\_id | driver\_id | city\_id | status | request\_at |

+----+-----------+-----------+---------+---------------------+------------+

| 1 | 1 | 10 | 1 | completed | 2013-10-01 |

| 2 | 2 | 11 | 1 | cancelled\_by\_driver | 2013-10-01 |

| 3 | 3 | 12 | 6 | completed | 2013-10-01 |

| 4 | 4 | 13 | 6 | cancelled\_by\_client | 2013-10-01 |

| 5 | 1 | 10 | 1 | completed | 2013-10-02 |

| 6 | 2 | 11 | 6 | completed | 2013-10-02 |

| 7 | 3 | 12 | 6 | completed | 2013-10-02 |

| 8 | 2 | 12 | 12 | completed | 2013-10-03 |

| 9 | 3 | 10 | 12 | completed | 2013-10-03 |

| 10 | 4 | 13 | 12 | cancelled\_by\_driver | 2013-10-03 |

+----+-----------+-----------+---------+---------------------+------------+

id is the primary key for this table.

The table holds all taxi trips. Each trip has a unique id, while client\_id and driver\_id are foreign keys to the users\_id at the Users table.

Status is an ENUM type of ('completed', 'cancelled\_by\_driver', 'cancelled\_by\_client').

Table: Users

| users\_id | banned | role |

+----------+--------+--------+

| 1 | No | client |

| 2 | Yes | client |

| 3 | No | client |

| 4 | No | client |

| 10 | No | driver |

| 11 | No | driver |

| 12 | No | driver |

| 13 | No | driver |

users\_id is the primary key for this table.

The table holds all users. Each user has a unique users\_id, and role is an ENUM type of ('client', 'driver', 'partner').

banned is an ENUM type of ('Yes', 'No').

The **cancellation rate** is computed by dividing the number of canceled (by client or driver) requests with unbanned users by the total number of requests with unbanned users on that day.

Write a SQL query to find the **cancellation rate** of requests with unbanned users (**both client and driver must not be banned**) each day between "2013-10-01" and "2013-10-03". Round Cancellation Rate to **two decimal** points.

Return the result table in **any order**.

The query result format is in the following example.

**Input: the input tables trips & users are as same as above**

**Output:**

+------------+-------------------+

| Day | Cancellation Rate |

+------------+-------------------+

| 2013-10-01 | 0.33 |

| 2013-10-02 | 0.00 |

| 2013-10-03 | 0.50 |

+------------+-------------------+

create table trips(client\_id int, Driver\_id int, city\_id int,

status varchar(50),request date)

insert into trips values(1,10,1,'completed','2013-10-1')

insert into trips values(2,11,1,'cancelled\_by\_driver','2013-10-1')

insert into trips values(3,12,6,'completed','2013-10-1')

insert into trips values(4,13,6,'cancelled\_by\_client','2013-10-1')

insert into trips values(1,10,1,'completed','2013-10-2')

insert into trips values(2,11,6,'completed','2013-10-2')

insert into trips values(3,12,6,'completed','2013-10-2')

insert into trips values(2,12,12,'completed','2013-10-3')

insert into trips values(3,10,12,'completed','2013-10-3')

insert into trips values(4,13,12,'cancelled\_by\_driver','2013-10-3')

select \* from trips

create table users (users\_id int, banned varchar(7), role varchar(10))

insert into users values(1,'No','Client'),(2,'Yes','Client')

insert into users values(3,'No','Client'),(4,'No','Client')

insert into users values(10,'No','Driver'),(11,'No','Driver')

insert into users values(12,'No','Driver'),(13,'No','Driver')

select \* from users

with ct as(

select request,u1.users\_id,status,CUME\_DIST() over(PARTITION by request order by status) as cd from trips,users u1, users u2

where u1.users\_id=trips.client\_id and u2.users\_id=trips.driver\_id and u1.banned='No' and u2.banned='No'

)

select request,round(cd,2) cancellation\_rate from ct where status like 'canc%' group by request,cd

select t.request day,

cast(count( distinct case when status like '%cancelled%' then id end)\*1.0/ count(distinct t.id) as decimal(12,2)) as "CR"

from trips t join users u1 on u1.users\_id = t.driver\_id and u1.banned = 'No'

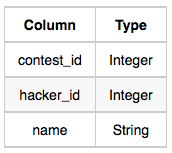
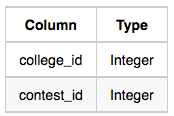
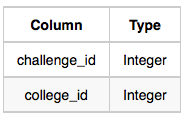
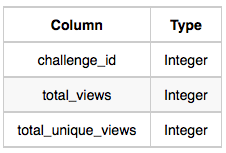
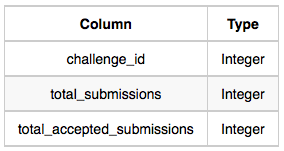
join users u2 on u2.users\_id = t.client\_id and u2.banned = 'No' group by t.request

**Task 4(2 hour).** Samantha interviews many candidates from different colleges using coding challenges and contests. Write a query to print the *contest\_id*, *hacker\_id*, *name*, and the sums of *total\_submissions*, *total\_accepted\_submissions*, *total\_views*, and *total\_unique\_views* for each contest sorted by *contest\_id*. Exclude the contest from the result if all four sums are zero.

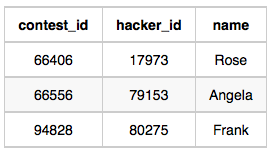
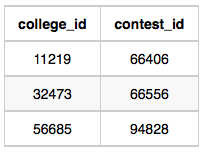
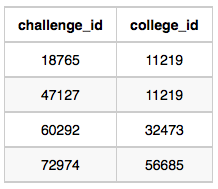
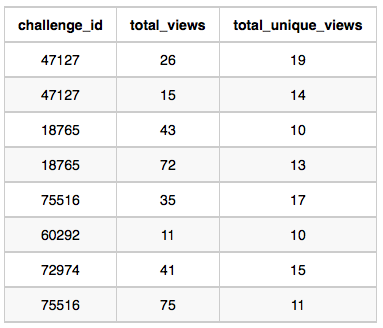
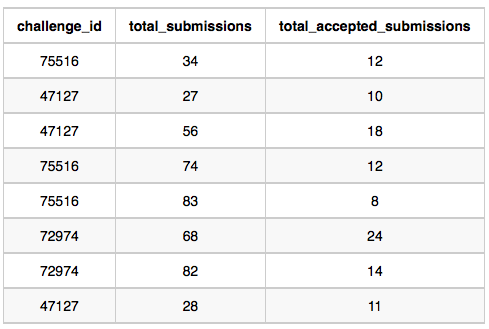
**Note:** A specific contest can be used to screen candidates at more than one college, but each college only holds  screening contest.

**Input Format**

The following tables hold interview data:

* *Contests:* The *contest\_id* is the id of the contest, *hacker\_id* is the id of the hacker who created the contest, and *name* is the name of the hacker. 
* *Colleges:* The *college\_id* is the id of the college, and *contest\_id* is the id of the contest that Samantha used to screen the candidates. 
* *Challenges:* The *challenge\_id* is the id of the challenge that belongs to one of the contests whose contest\_id Samantha forgot, and *college\_id* is the id of the college where the challenge was given to candidates. 
* *View\_Stats:* The *challenge\_id* is the id of the challenge, *total\_views* is the number of times the challenge was viewed by candidates, and *total\_unique\_views* is the number of times the challenge was viewed by unique candidates. 
* *Submission\_Stats:* The *challenge\_id* is the id of the challenge, *total\_submissions* is the number of submissions for the challenge, and *total\_accepted\_submission* is the number of submissions that achieved full scores. 

**Sample Input**

*Contests* Table:  *Colleges* Table:   *Challenges* Table:  *View\_Stats* Table: *Submission\_Stats* Table: 

**Sample Output**

66406 17973 Rose 111 39 156 56

66556 79153 Angela 0 0 11 10

94828 80275 Frank 150 38 41 15

**Task 10 (2 hours).** Amber's conglomerate corporation just acquired some new companies. Each of the companies follows this hierarchy: 

Given the table schemas below, write a query to print the *company\_code*, *founder* name, total number of *lead* managers, total number of *senior* managers, total number of *managers*, and total number of *employees*. Order your output by ascending *company\_code*.

**Note:**

* The tables may contain duplicate records.
* The *company\_code* is string, so the sorting should not be **numeric**. For example, if the *company\_codes* are *C\_1*, *C\_2*, and *C\_10*, then the ascending *company\_codes* will be *C\_1*, *C\_10*, and *C\_2*.

**Input Format**

The following tables contain company data:

* *Company:* The *company\_code* is the code of the company and *founder* is the founder of the company. 
* *Lead\_Manager:* The *lead\_manager\_code* is the code of the lead manager, and the *company\_code* is the code of the working company. 
* *Senior\_Manager:* The *senior\_manager\_code* is the code of the senior manager, the *lead\_manager\_code* is the code of its lead manager, and the *company\_code* is the code of the working company. 
* *Manager:* The *manager\_code* is the code of the manager, the *senior\_manager\_code* is the code of its senior manager, the *lead\_manager\_code* is the code of its lead manager, and the *company\_code* is the code of the working company. 
* *Employee:* The *employee\_code* is the code of the employee, the *manager\_code* is the code of its manager, the *senior\_manager\_code* is the code of its senior manager, the *lead\_manager\_code* is the code of its lead manager, and the *company\_code* is the code of the working company. 

**Sample Input**

*Company* Table: 

*Lead\_Manager* Table: *Senior\_Manager* Table: 

*Manager* Table:  *Employee* Table: 

**Sample Output**

C1 Monika 1 2 1 2

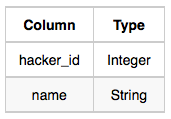
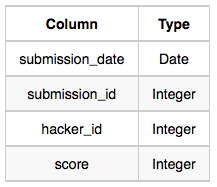
C2 Samantha 1 1 2 2

**Task 5 (45 mins).** Julia conducted a  days of learning SQL contest. The start date of the contest was *March 01, 2016* and the end date was *March 15, 2016*.

Write a query to print total number of unique hackers who made at least submission each day (starting on the first day of the contest), and find the *hacker\_id* and *name* of the hacker who made maximum number of submissions each day. If more than one such hacker has a maximum number of submissions, print the lowest *hacker\_id*. The query should print this information for each day of the contest, sorted by the date.

**Input Format**

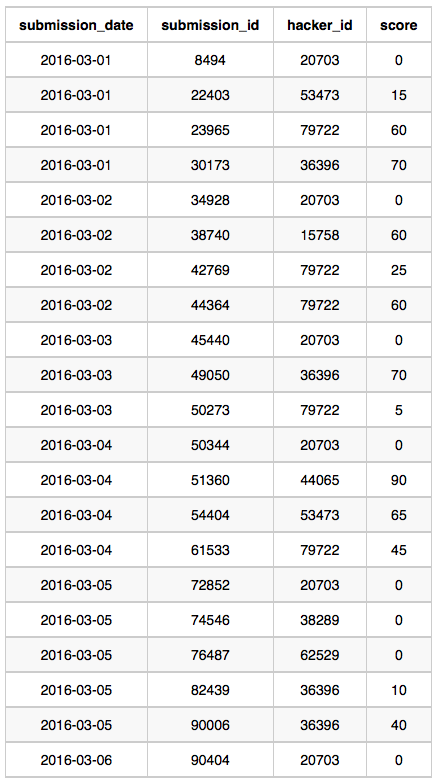
The following tables hold contest data:

* *Hackers:* The *hacker\_id* is the id of the hacker, and *name* is the name of the hacker.
* *Submissions:* The *submission\_date* is the date of the submission, *submission\_id* is the id of the submission, *hacker\_id* is the id of the hacker who made the submission, and *score* is the score of the submission. 

**Sample Input**

For the following sample input, assume that the end date of the contest was *March 06, 2016*.

*Hackers* Table: 

*Submissions* Table: 

**Sample Output**

2016-03-01 4 20703 Angela

2016-03-02 2 79722 Michael

2016-03-03 2 20703 Angela

2016-03-04 2 20703 Angela

2016-03-05 1 36396 Frank

2016-03-06 1 20703 Angela

**Task 15.** Samantha was tasked with calculating the average monthly salaries for all employees in the **EMPLOYEES** table, but did not realize her keyboard's 0  key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeroes removed), and the actual average salary.

Write a query calculating the amount of error (i.e.:  actual – miscalculated average monthly salaries), and round it up to the next integer.

select cast(CEILING(avg(cast(salary as float)) - avg(cast(replace(salary,0,'') as float)))as int) from Employess

create table employess (id int,names varchar(39),salary int)

insert into employess values(1,'rahul',1420),(2,'nikhil',2006),(3,'vinay',2210),(4,'maria',3000)

select \* from employess

**Task 16.** Copy new data of one table to another( you do not have indicator for new data and old data).

SELECT \* INTO NewTable FROM OldTable

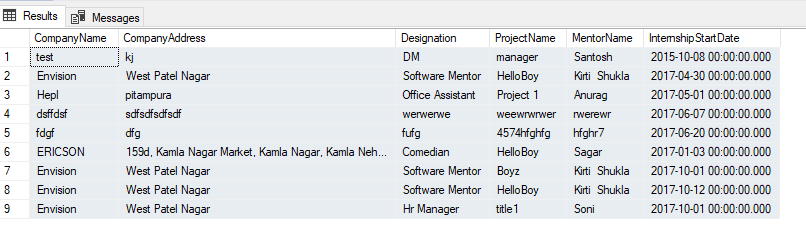
**Task 21 (3 Day)**. Load the following configuration table in your database. Create a dynamic stored procedure that will work over the following configuration table (existing in your DB) to create new tables in database, if the status is ‘New’ (along with the primary keys, Clustered indexes, Non- clustered indexes) or add columns for tables with status ‘Old’ with Alter command (which should run only once, even if there are multiple columns listed to be added in your ‘Old’ status tables.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **NewColumn** | **DataTypes** | **Table** | **TableStatus** | **If\_New\_Table\_then\_PrimaryKey** | **Any\_Index(CLI)** | **Any\_Index(NCLI)** |
| Camp\_id | nvarchar(100) | DimCampaign | New | Yes | Yes |  |
| Total | float | Orders | Old |  |  | Yes |
| Camp\_type | nvarchar(100) | DimCampaign | Old |  |  |  |
| Department | nvarchar(100) | DimEmployees | Old |  |  |  |
|  |  |  |  |  |  |  |

**Question---Scd 2 without merger logic**

**Q1.** You have a table named tblInternship and this table contained record row wise. Write a dynamic query in SQL to create a json format like the output figure without using System’s inbuilt functions like (JSON path, OPENJSON etc.)

**Eg: Sample table :** tblInternship



**Expected Output :**



create table Internship(company\_name varchar(50),company\_add varchar(50),designation varchar(50),

project\_name varchar(50),mentor varchar(50),Internship\_start datetime)

insert into Internship values ('Infosys','Hinjewadi Pune','DA','Manager','Santosh',2020/11/01)

insert into Internship values ('TCS','Hinjewadi Pune','Software','Proj1','Sanjay',2021/4/01)

insert into Internship values ('TCS','Hinjewadi Pune','Associate','Project2','Sunny',2021/06/01)

insert into Internship values ('Celebal','Hinjewadi Pune','office','Man','amit',2021/09/01)

insert into Internship values ('Wipro','Hinjewadi Pune','HR','Money','yemul',2022/01/01)

insert into Internship values ('Accenture','Hinjewadi Pune','Devloper','proj5','Santosh',2020/03/01)

select \* from Internship for json auto

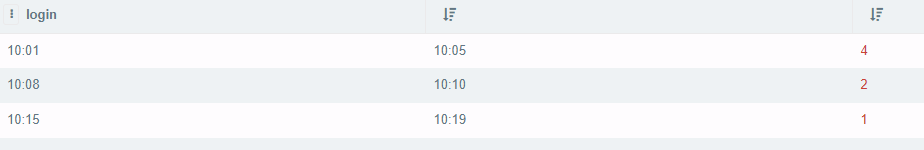
select \* from Internship for json path

**Task**

input



output



create table input1 (time time, status varchar(4) )

insert into input1 values('10:01','in')

insert into input1 values('10:02','in')

insert into input1 values('10:03','in')

insert into input1 values('10:04','in')

insert into input1 values('10:05','Out')

insert into input1 values('10:08','in')

insert into input1 values('10:09','in')

insert into input1 values('10:10','Out')

insert into input1 values('10:15','in')

insert into input1 values('10:19','Out')

with ct as(

select t1,t2,

ROW\_NUMBER() over (partition by t1 order by t1,t2) as rn

from (select time t1 from input1 where status='in') i1 ,(select TIME t2 from input1 where status='out') i2

where t1<t2 )

select min(t1) login,t2 logout,count(t2) countt from ct where rn=1 group by t2

OR soln.2

select min(check\_in),check\_out,count(\*)-1 as in\_count from  
(select check\_in,status,min(check\_out) over (order by check\_in rows between current row and unbounded following) as check\_out from  
(select a.time as check\_in,a.status,b.time as check\_out from  
log\_table as a left join (select \* from log\_table where status=="out") as b on a.time=b.time)) group by check\_out

--------------------------- select max marks of students from its given exam ---------

create table marks (id int, m1 int,m2 int, m3 int)

insert into marks values(1,50,null,50),(1,null,60,null),(2,34,45,54)

insert into marks values(2,67,null,44)

select ID, MAX(m1) m1,MAX(m2) m2,MAX(m3) m3 from marks group by id