* A [*median*](https://en.wikipedia.org/wiki/Median) is defined as a number separating the higher half of a data set from the lower half. Query the *median* of the *Northern Latitudes* (*LAT\_N*) from **STATION** and round your answer to  decimal places.

- select ROUND(MEDIAN(LAT\_N), 4) from STATION;

- Generate the following two result sets:

1. Query an *alphabetically ordered* list of all names in **OCCUPATIONS**, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).
2. Query the number of ocurrences of each occupation in **OCCUPATIONS**. Sort the occurrences in *ascending order*, and output them in the following format:

There are a total of [occupation\_count] [occupation]s.

where [occupation\_count] is the number of occurrences of an occupation in **OCCUPATIONS** and [occupation] is the *lowercase* occupation name. If more than one *Occupation* has the same [occupation\_count], they should be ordered alphabetically.

**Note:** There will be at least two entries in the table for each type of 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**

An **OCCUPATIONS** table that contains the following records:



**Sample Output**

Ashely(P) Christeen(P) Jane(A) Jenny(D) Julia(A) Ketty(P) Maria(A) Meera(S) Priya(S) Samantha(D) There are a total of 2 doctors. There are a total of 2 singers. There are a total of 3 actors. There are a total of 3 professors.

**Explanation**

The results of the first query are formatted to the problem description's specifications.  
The results of the second query are ascendingly ordered first by number of names corresponding to each profession (), and then alphabetically by profession (, and ).

- select NAME||'('||SUBSTR(Occupation,1,1)||')' from OCCUPATIONS order by NAME;

- select 'There are a total of '||count(OCCUPATION)||' '||LOWER(OCCUPATION)||'s.' from OCCUPATIONS

group by OCCUPATION order by count(OCCUPATION), OCCUPATION;

[Pivot](https://en.wikipedia.org/wiki/Pivot_table" \t "_blank) 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

**Explanation**

The first column is an alphabetically ordered list of Doctor names.  
The second column is an alphabetically ordered list of Professor names.  
The third column is an alphabetically ordered list of Singer names.  
The fourth column is an alphabetically ordered list of Actor names.  
The empty cell data for columns with less than the maximum number of names per occupation (in this case, the Professor and Actor columns) are filled with **NULL** values.

**Current Buffer** (saved locally, editable)

[Pivot](https://en.wikipedia.org/wiki/Pivot_table" \t "_blank) 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

**Explanation**

The first column is an alphabetically ordered list of Doctor names.  
The second column is an alphabetically ordered list of Professor names.  
The third column is an alphabetically ordered list of Singer names.  
The fourth column is an alphabetically ordered list of Actor names.  
The empty cell data for columns with less than the maximum number of names per occupation (in this case, the Professor and Actor columns) are filled with **NULL** values.

set @r1 = 0, @r2 = 0, @r3 = 0, @r4 = 0;

select min(Doctor), min(Professor), min(Singer), min(Actor) from

(

Select case when Occupation = 'Doctor' then @r1 := @r1 + 1

when Occupation = 'Professor' then @r2 := @r2 + 1

when Occupation = 'Singer' then @r3 := @r3 + 1

when Occupation = 'Actor' then @r4 := @r4 + 1 end as RowNumber,

case when Occupation = 'Doctor' then Name end as Doctor,

case when Occupation = 'Professor' then Name end as Professor,

case when Occupation = 'Singer' then Name end as Singer,

case when Occupation = 'Actor' then Name end as Actor

from Occupations

order by Name

)temp

group by RowNumber

Write a query identifying the *type* of each record in the **TRIANGLES** table using its three side lengths. Output one of the following statements for each record in the table:

* **Equilateral**: It's a triangle with  sides of equal length.
* **Isosceles**: It's a triangle with  sides of equal length.
* **Scalene**: It's a triangle with  sides of differing lengths.
* **Not A Triangle**: The given values of *A*, *B*, and *C* don't form a triangle.

**Input Format**

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



Each row in the table denotes the lengths of each of a triangle's three sides.

**Sample Input**



**Sample Output**

Isosceles Equilateral Scalene Not A Triangle

**Explanation**

Values in the tuple  form an Isosceles triangle, because .  
Values in the tuple  form an Equilateral triangle, because . Values in the tuple  form a Scalene triangle, because .  
Values in the tuple  cannot form a triangle because the combined value of sides  and  is not larger than that of side .

My SQL

select if(A+B<=C or B+C<=A or A+C<=B, "Not A Triangle",

if(A=B and B=C, "Equilateral",

if(A=B or B=C or C=A,"Isosceles", "Scalene")))

from TRIANGLES;

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

**Explanation**

The *Binary Tree* below illustrates the sample:



SQL SERVER

select N, case when BT.P is null then 'Root'

when exists(select B.P from BST as B where B.P = BT.N) then 'Inner'

else 'Leaf'

end

from BST as BT

order by BT.N;

MY SQL

select N,if(P is null,'Root',if(select count(\*) from BST where BST.N > 0), 'Inner','Leaf') from BST order by N

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

**Explanation**

In company *C1*, the only lead manager is *LM1*. There are two senior managers, *SM1* and *SM2*, under *LM1*. There is one manager, *M1*, under senior manager *SM1*. There are two employees, *E1* and *E2*, under manager *M1*.

In company *C2*, the only lead manager is *LM2*. There is one senior manager, *SM3*, under *LM2*. There are two managers, *M2* and *M3*, under senior manager *SM3*. There is one employee, *E3*, under manager *M2*, and another employee, *E4*, under manager, *M3*.

My SQL

select C.company\_code, C.founder, count(distinct LM.lead\_manager\_code), count(distinct SM.senior\_manager\_code), count(distinct M.manager\_code), count(distinct E.employee\_code) from Company as C, Lead\_Manager as LM, Senior\_Manager as SM, Manager as M, Employee as E where C.company\_code = LM.company\_code and LM.company\_code = SM.company\_code and SM.company\_code = M.company\_Code and M.company\_code = E.company\_code group by C.company\_code, C.founder order by C.company\_code

*P(R)* represents a pattern drawn by Julia in *R* rows. The following pattern represents *P(5)*:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

Write a query to print the pattern *P(20)*.

DECLARE @var int

SELECT @var = 1

WHILE @var <= 20

BEGIN

PRINT replicate('\* ', @var)

SET @var = @var + 1

END;

MYSQL –

set @row := 0;

select repeat('\* ', @row := @row + 1) from information\_schema.tables where @row < 20

Write a query to print all *prime numbers* less than or equal to . 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  would be:

2&3&5&7

SELECT LISTAGG(PRIME\_NUMBER,'&') WITHIN GROUP (ORDER BY PRIME\_NUMBER)

FROM(

SELECT L PRIME\_NUMBER

FROM(

SELECT LEVEL L

FROM DUAL

CONNECT BY LEVEL <= 1000),

(SELECT LEVEL M FROM DUAL CONNECT BY LEVEL <= 1000)

WHERE M <= L

GROUP BY L

HAVING COUNT(CASE WHEN L/M = TRUNC(L/M) THEN 'Y' END) = 2

ORDER BY L);

Query a *count* of the number of cities in **CITY** having a *Population* larger than 100000

**Input Format**

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



select count(name) from CITY where POPULATION > 100000;

- Query the total population of all cities in **CITY** where *District* is **California**.

select SUM(POPULATION) from CITY where district = 'California';

- Query the average population of all cities in **CITY** where *District* is **California**.

select AVG(population) from city where district = 'California';

- Query the average population for all cities in **CITY**, rounded *down* to the nearest integer.

select ROUND(AVG(POPULATION)) from CITY

-Query the sum of the populations for all Japanese cities in **CITY**. The *COUNTRYCODE* for Japan is **JPN**.

select sum(population) from CITY where countrycode = 'JPN';

- Query the difference between the maximum and minimum populations in **CITY**.

select MAX(population) - MIN(population) from CITY;

- Samantha was tasked with calculating the average monthly salaries for all employees in the **EMPLOYEES** table, but did not realize her keyboard's  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.:  average monthly salaries), and round it up to the next integer.

select CEIL(AVG(SALARY) - AVG(REPLACE(SALARY,0,''))) from EMPLOYEES;

- We define an employee's *total earnings* to be their monthly  worked, and the *maximum total earnings* to be the maximum total earnings for any employee in the **Employee** table. Write a query to find the *maximum total earnings* for all employees as well as the total number of employees who have maximum total earnings. Then print these values as  space-separated integers.

select salary\*months as earnings, count(\*) from employee

group by earnings

order by earnings desc limit 1;

- Query the following two values from the **STATION** table:

1. The sum of all values in *LAT\_N* rounded to a scale of  decimal places.
2. The sum of all values in *LONG\_W* rounded to a scale of  decimal places.

select round(sum(LAT\_N), round(sum(long\_w)) from STATION

- Query the sum of *Northern Latitudes* (*LAT\_N*) from **STATION** having values greater than  and less than . Truncate your answer to  decimal places.

select round(sum(LAT\_N), 4) from STATION where LAT\_N > 38.7880 and LAT\_N < 137.2345;

- Query the greatest value of the *Northern Latitudes* (*LAT\_N*) from **STATION** that is less than . Truncate your answer to  decimal places.

select round(max(LAT\_N),4) from STATION where LAT\_N < 137.2345

- Query the *Western Longitude* (*LONG\_W*) for the largest *Northern Latitude* (*LAT\_N*) in **STATION** that is less than . Round your answer to  decimal places.

select ROUND(LONG\_W,4) from STATION where LAT\_N in (select MAX(LAT\_N) from STATION where LAT\_N < 137.2345);

- Query the smallest *Northern Latitude* (*LAT\_N*) from **STATION** that is greater than . Round your answer to  decimal places.

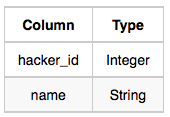
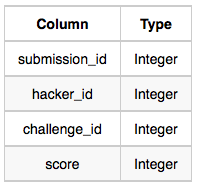
select ROUND(MIN(LAT\_N), 4) from STATION where LAT\_N > 38.7780;

-   
You did such a great job helping Julia with her last coding contest challenge that she wants you to work on this one, too!

The total score of a hacker is the sum of their maximum scores for all of the challenges. Write a query to print the *hacker\_id*, *name*, and total score of the hackers ordered by the descending score. If more than one hacker achieved the same total score, then sort the result by ascending *hacker\_id*. Exclude all hackers with a total score of  from your result.

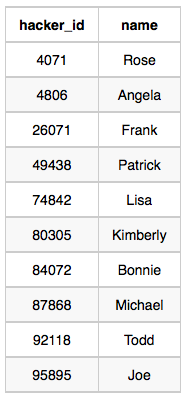
**Input Format**

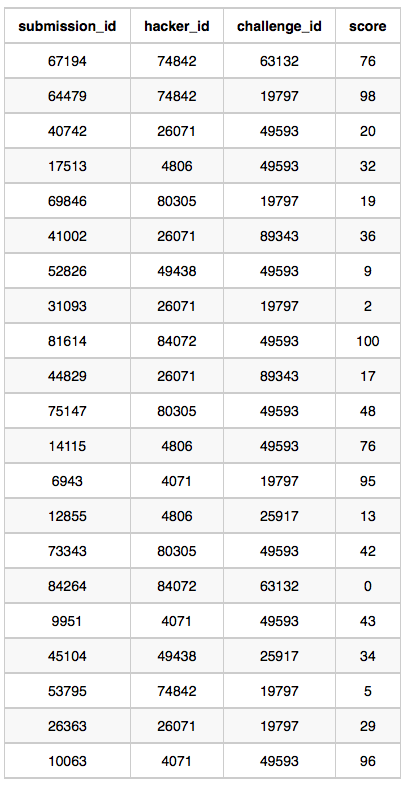
The following tables contain contest data:

* Hackers: The hacker\_id is the id of the hacker, and name is the name of the hacker.
* Submissions: The submission\_id is the id of the submission, hacker\_id is the id of the hacker who made the submission, challenge\_id is the id of the challenge for which the submission belongs to, and score is the score of the submission.
* 

**Sample Input**

Hackers Table:



*Submissions* Table:

**Sample Output**

4071 Rose 191 74842 Lisa 174 84072 Bonnie 100 4806 Angela 89 26071 Frank 85 80305 Kimberly 67 49438 Patrick 43

**Explanation**

Hacker *4071* submitted solutions for challenges *19797* and *49593*, so the total score .

Hacker *74842* submitted solutions for challenges *19797* and *63132*, so the total score

Hacker *84072* submitted solutions for challenges *49593* and *63132*, so the total score .

The total scores for hackers *4806*, *26071*, *80305*, and *49438* can be similarly calculated.

select h.hacker\_id, h.name,sum(score) as total\_score from hackers as h

inner join

(select hacker\_id, max(score) as score from submissions group by hacker\_id, challenge\_id) max\_score

on h.hacker\_id = max\_score.hacker\_id

group by h.hacker\_id, name

having total\_score > 0

order by total\_Score desc, h.hacker\_id;

- Given the **CITY** and **COUNTRY** tables, query the sum of the populations of all cities where the *CONTINENT* is *'Asia'*.

**Note:** *CITY.CountryCode* and *COUNTRY.Code* are matching key columns.

SELECT SUM(CITY.POPULATION)

FROM CITY, COUNTRY

WHERE CITY.COUNTRYCODE = COUNTRY.CODE AND COUNTRY.CONTINENT = 'Asia';

Given the **CITY** and **COUNTRY** tables, query the names of all cities where the *CONTINENT* is *'Africa'*.

**Note:** *CITY.CountryCode* and *COUNTRY.Code* are matching key columns.

select city.name from country, city where city.countrycode = country.code and country.continent = 'Africa';

Given the **CITY** and **COUNTRY** tables, query the names of all the continents (*COUNTRY.Continent*) and their respective average city populations (*CITY.Population*) rounded *down* to the nearest integer.

select country.continent, round(avg(city.population)-0.5) from country, city where country.code = city.countrycode group by country.continent;

*Ketty* gives *Eve* a task to generate a report containing three columns: *Name*, *Grade* and *Mark*. *Ketty* doesn't want the NAMES of those students who received a grade lower than *8*. The report must be in descending order by grade -- i.e. higher grades are entered first. If there is more than one student with the same grade (8-10) assigned to them, order those particular students by their name alphabetically. Finally, if the grade is lower than 8, use "NULL" as their name and list them by their grades in descending order. If there is more than one student with the same grade (1-7) assigned to them, order those particular students by their marks in ascending order.

Write a query to help Eve.

select if(Grades.Grade < 8, NULL, Students.Name) as Sname, Grades.Grade, Students.Marks from Students inner join Grades on Students.Marks between Grades.Min\_Mark and Grades.Max\_Mark order by Grades.Grade desc, Sname asc, Students.Marks asc;

select case when Grades.Grade > 7 then Students.Name when Grades.Grade <= 7 then NULL end, Grades.Grade, Students.Marks from Students inner join Grades on Students.Marks between Grades.Min\_Mark and Grades.Max\_Mark order by Grades.Grade desc, Students.Name asc, Students.Marks asc

-Julia just finished conducting a coding contest, and she needs your help assembling the leaderboard! Write a query to print the respective *hacker\_id* and *name* of hackers who achieved full scores for *more than one* challenge. Order your output in descending order by the total number of challenges in which the hacker earned a full score. If more than one hacker received full scores in same number of challenges, then sort them by ascending *hacker\_id*.

select h.hacker\_id, h.name

from submissions s

inner join challenges c

on s.challenge\_id = c.challenge\_id

inner join difficulty d

on c.difficulty\_level = d.difficulty\_level

inner join hackers h

on s.hacker\_id = h.hacker\_id

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

group by h.hacker\_id, h.name

having count(s.hacker\_id) > 1

order by count(s.hacker\_id) desc, s.hacker\_id asc

Harry Potter and his friends are at Ollivander's with Ron, finally replacing Charlie's old broken wand.

Hermione decides the best way to choose is by determining the minimum number of gold galleons needed to buy each *non-evil* wand of high power and age. Write a query to print the *id*, *age*, *coins\_needed*, and *power* of the wands that Ron's interested in, sorted in order of descending *power*. If more than one wand has same power, sort the result in order of descending *age*.

select w.id, p.age,w.coins\_needed, w.power

from Wands as w join Wands\_Property as p

on w.code = p.code

where p.is\_evil = 0 and w.coins\_needed = (select min(coins\_needed) from Wands as w1 join Wands\_Property as p1 on w1.code = p1.code where w1.power = w.power and p1.age = p.age)

order by w.power desc, p.age desc

Julia asked her students to create some coding challenges. Write a query to print the *hacker\_id*, *name*, and the total number of challenges created by each student. Sort your results by the total number of challenges in descending order. If more than one student created the same number of challenges, then sort the result by *hacker\_id*. If more than one student created the same number of challenges and the count is less than the maximum number of challenges created, then exclude those students from the result.

select c.hacker\_id, h.name, count(c.challenge\_id) as c\_count

from hackers h join challenges c

on h.hacker\_id = c.hacker\_id

group by c.hacker\_id, h.name

having c\_count = (

select MAX(temp1.cnt) from (

select count(hacker\_id) as cnt from challenges

group by hacker\_id

order by hacker\_id) temp1)

or

c\_count in (

select t.cnt from (select count(\*) as cnt from challenges group by hacker\_id)t group by t.cnt having count(t.cnt) = 1)

order by c\_count desc, c.hacker\_id

set @number = 21;

select REPEAT('\* ', @number := @number-1) from information\_schema.tables where @number > 0;

Julia asked her students to create some coding challenges. Write a query to print the *hacker\_id*, *name*, and the total number of challenges created by each student. Sort your results by the total number of challenges in descending order. If more than one student created the same number of challenges, then sort the result by *hacker\_id*. If more than one student created the same number of challenges and the count is less than the maximum number of challenges created, then exclude those students from the result.

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 .

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

select con.contest\_id, con.hacker\_id, con.name, sum(total\_submissions), sum(total\_accepted\_submissions), sum(total\_views), sum(total\_unique\_views) from contests con join colleges col on con.contest\_id = col.contest\_id join challenges cha on col.college\_id = cha.college\_id left join (select challenge\_id, sum(total\_views) as total\_views, sum(total\_unique\_views) as total\_unique\_views from view\_stats group by challenge\_id) vs on cha.challenge\_id = vs.challenge\_id left join (select challenge\_id, sum(total\_submissions) as total\_submissions, sum(total\_accepted\_submissions) as total\_accepted\_submissions from submission\_stats group by challenge\_id) ss on cha.challenge\_id = ss.challenge\_id group by con.contest\_id, con.hacker\_id, con.name having sum(total\_submissions)!=0 or sum(total\_accepted\_submissions)!=0 or sum(total\_views)!=0 or sum(total\_unique\_views)!=0 order by contest\_id;

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.

select submission\_date ,( SELECT COUNT(distinct hacker\_id)

FROM Submissions s2

WHERE s2.submission\_date = s1.submission\_date AND

(SELECT COUNT(distinct s3.submission\_date)

FROM Submissions s3

WHERE s3.hacker\_id = s2.hacker\_id AND

s3.submission\_date < s1.submission\_date) = dateDIFF(s1.submission\_date , '2016-03-01')) ,

(select hacker\_id from submissions s2

where s2.submission\_date = s1.submission\_date

group by hacker\_id

order by count(submission\_id) desc , hacker\_id limit 1) as hack,

(select name from hackers where hacker\_id = hack)

from

(select distinct submission\_date from submissions) s1

group by submission\_date;

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).

select s.name from students s, friends f, packages p, packages p2 where s.id = f.id and f.friend\_id = p2.id and s.id = p.id and p.salary < p2.salary order by p2.salary;;

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.

SELECT Start\_Date, MIN(End\_Date) FROM /\* Choose start dates that are not end dates of other projects (if a start date is an end date, it is part of the samee project) \*/ (SELECT Start\_Date FROM Projects WHERE Start\_Date NOT IN (SELECT End\_Date FROM Projects)) a, /\* Choose end dates that are not end dates of other projects \*/ (SELECT end\_date FROM PROJECTS WHERE end\_date NOT IN (SELECT start\_date FROM PROJECTS)) b /\* At this point, we should have a list of start dates and end dates that don't necessarily correspond with each other \*/ /\* This makes sure we only choose end dates that fall after the start date, and choosing the MIN means for the particular start\_date, we get the closest end date that does not coincide with the start of another task \*/ where start\_date < end\_date GROUP BY start\_date ORDER BY datediff(start\_date, MIN(end\_date)) DESC, start\_date