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

The sum of all values in LONG\_W rounded to a scale of decimal places.

* SELECT ROUND(SUM(LAT\_N),2) AS lat, ROUND(SUM(LONG\_W),2) as lon FROM station;

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

* SELECT ROUND(SUM(LAT\_N),4) FROM station WHERE (STATION.LAT\_N > 38.7880 AND STATION.LAT\_N < 137.2345);

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

* SELECT ROUND(MAX(LAT\_N),4) FROM station WHERE station.LAT\_N < 137.2345;

1. Query the Western Longitude (LONG\_W) for the largest Northern Latitude (LAT\_N) in **STATION** that is less than xxxxx. Round your answer to 4 decimal places.

* SELECT ROUND(LONG\_W,4) FROM station WHERE LAT\_N = (SELECT MAX(LAT\_N) FROM station WHERE station.LAT\_N < 137.2345);

1. Query the Western Longitude (LONG\_W)where the smallest Northern Latitude (LAT\_N) in STATION is greater than xxxx. Round your answer to 4 decimal places.

* SELECT ROUND(LONG\_W,4) FROM station WHERE LAT\_N = (SELECT MIN(LAT\_N) FROM station WHERE station.LAT\_N > 38.7780);

1. 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
* JOIN COUNTRY ON CITY.CountryCode = COUNTRY.Code
* WHERE continent = 'Asia'

1. 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 CITY
* JOIN COUNTRY ON CITY.COUNTRYCODE = COUNTRY.CODE
* WHERE CONTINENT = 'Africa'

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

* SELECT CONTINENT, FLOOR(AVG(CITY.POPULATION)) FROM CITY
* JOIN COUNTRY ON CITY.CountryCode = Country.code
* GROUP BY CONTINENT

1. 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.

* WITH cte AS
* (
* SELECT
* h.hacker\_id,
* h.name,
* COUNT(c.challenge\_id) AS challenges\_created,
* COUNT(COUNT(c.challenge\_id)) OVER (PARTITION BY COUNT(c.challenge\_id)) AS temp
* FROM hackers h
* JOIN challenges c ON h.hacker\_id = c.hacker\_id
* GROUP BY
* h.hacker\_id,
* h.name
* )
* SELECT
* hacker\_id,
* name,
* challenges\_created
* FROM cte
* WHERE temp = 1 or challenges\_created = (SELECT max(challenges\_created) FROM cte)
* ORDER BY challenges\_created DESC, hacker\_id

1. 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).

* SET @i:=21;
* SELECT REPEAT('\* ', @i:= @i - 1)
* FROM INFORMATION\_SCHEMA.TABLES
* WHERE @i > 0;

1. 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).

* SET @i:=0;
* SELECT REPEAT('\* ', @i:= @i + 1)
* FROM INFORMATION\_SCHEMA.TABLES
* WHERE @i < 20;

# The CROSS JOIN keyword returns all records from both tables (table1 and table2).

# CROSS JOINs don't have ON clauses as everything is joined with everything. FULL OUTER JOIN is a combination of LEFT OUTER and the RIGHT OUTER JOINs. FULL OUTER JOIN returns those rows in two tables that match the WHERE clause, and shows null values for the rows the ON condition isn't met for.

1. 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.

* 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
* JOIN Lead\_Manager as LM USING(company\_code)
* JOIN Senior\_Manager as SM USING(company\_code)
* JOIN Manager as M USING(company\_code)
* JOIN Employee as E USING(company\_code)
* GROUP BY C.company\_code, C.founder
* ORDER BY C.company\_code

2nd Ans => SELECT C.company\_code, C.founder, COUNT(DISTINCT E.lead\_manager\_code), COUNT(DISTINCT E.senior\_manager\_code), COUNT(DISTINCT E.manager\_code), COUNT(DISTINCT E.employee\_code)

FROM Company as C

JOIN Employee as E USING(company\_code)

GROUP BY C.company\_code, C.founder

ORDER BY C.company\_code

1. Window functions (AVERAGE, MIN,MAX etc), OVER, PARTITION BY

* SELECT Name, Age, Department, Salary, AVERAGE(Salary) OVER( PARTITION BY Department) AS Avg\_Salary FROM employee

1. 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.

* select sd.start\_date, ed.end\_date from
* (SELECT start\_date, row\_number() over (order by start\_date) as rn
* FROM projects
* WHERE start\_date NOT IN (SELECT end\_date FROM projects)) sd
* join
* (SELECT end\_date, row\_number() over (order by end\_date) as rn
* FROM projects
* WHERE end\_date NOT IN (SELECT start\_date FROM projects)) ed
* on sd.rn=ed.rn
* order by ed.end\_date-sd.start\_date

1. Consider and 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 between points and and round it to a scale of decimal places.

* SELECT ROUND(
* SQRT(
* POWER((MAX(LAT\_N) - MIN(LAT\_N)), 2) +
* POWER((MAX(LONG\_W) - MIN(LONG\_W)), 2)
* ),
* 4)
* FROM STATION

1. A 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(lat\_n, 4)
* FROM (SELECT lat\_n, row\_number() over (ORDER BY lat\_n DESC) AS ROWNUM FROM station) a
* WHERE a.ROWNUM = (SELECT CEILING(COUNT(\*)/2) FROM station);

1. 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.

* SELECT h.hacker\_id, h.name, sum(maxscore) FROM Hackers H
* JOIN (SELECT S.hacker\_id, S.challenge\_id, max(score) as maxscore
* FROM Submissions S GROUP BY S.hacker\_id, S.challenge\_id) M ON H.hacker\_id=M.hacker\_id
* GROUP BY h.hacker\_id, h.name HAVING sum(maxscore)> 0
* ORDER BY sum(maxscore) DESC, hacker\_id ASC

1. Julia conducted 15 a days of learning SQL contest. The start date of the contest was March 01, 2016 and the end date was March 15, 2016.

SQL ADVANCE

<https://www.hackerrank.com/challenges/15-days-of-learning-sql/problem?isFullScreen=true&h_r=next-challenge&h_v=zen>

part 01: <https://www.youtube.com/watch?v=ed59ow-nvb0>

part 02: <https://www.youtube.com/watch?v=2wrdMoLBOtU>

Output Goal:

1. Submission date
2. Number of DISTINCT Hacker who made submission each day
3. Hacker\_id of hacker who made max number of submissions
4. Hacker name of 3.

Order Condition 3 & 4:

1. Maximum submission 2. Lowest hacker\_id

1st Approch :

Do solution for 1st of March

1. 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. List the rows such that X1 ≤ Y1.

* WITH F AS
* (SELECT X,Y,
* ROW\_NUMBER () OVER (ORDER BY X) AS ROW\_NUM
* FROM Functions)
* SELECT DISTINCT F1.X, F1.Y
* FROM F AS F1, F AS F2
* WHERE F1.X = F2.Y AND
* F2.X = F1.Y AND
* F1.X <= F1.Y AND
* F1.ROW\_NUM <> F2.ROW\_NUM
* ORDER BY F1.X

1. 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 C.contest\_id, C.hacker\_id, C.name, SUM(SS.total\_submissions), SUM(SS.total\_accepted\_submissions), SUM(VS.total\_views), SUM(VS.total\_unique\_views)
* FROM Contests AS C
* JOIN Colleges AS CO ON CO.contest\_id = C.contest\_id
* JOIN Challenges AS CH ON CH.college\_id = CO.college\_id
* JOIN View\_Stats AS VS ON VS.challenge\_id = CH.challenge\_id
* JOIN Submission\_Stats AS SS ON SS.challenge\_id = CH.challenge\_id
* GROUP BY C.contest\_id, C.hacker\_id, C.name
* HAVING SUM(SS.total\_submissions) + SUM(SS.total\_accepted\_submissions) + SUM(VS.total\_views) + SUM(VS.total\_unique\_views) <> 0
* ORDER BY C.contest\_id
* **845 579 Rose 2566 841 2447 869**
* **858 1053 Angela 1930 441 1464 511**
* **883 1055 Frank 2689 734 1794 593**
* **1793 2655 Patrick 2488 655 2257 762**
* **2374 2765 Lisa 6349 1924 7405 2026**
* **2963 2845 Kimberly 8499 2395 7261 2460**
* SELECT c.contest\_id, c.hacker\_id, c.name, t0.s1, t0.s2, t1.s1, t1.s2
* FROM Contests c
* JOIN (SELECT u.contest\_id, sum(ss.total\_submissions) s1, sum(ss.total\_accepted\_submissions) s2
* FROM Colleges u
* JOIN Challenges ch ON ch.college\_id=u.college\_id
* JOIN Submission\_Stats ss ON ss.challenge\_id=ch.challenge\_id
* GROUP BY u.contest\_id) t0 ON t0.contest\_id=c.contest\_id
* JOIN (SELECT u.contest\_id, sum(vs.total\_views) s1, sum(vs.total\_unique\_views) s2
* FROM Colleges u
* JOIN Challenges ch ON ch.college\_id=u.college\_id
* JOIN View\_Stats vs ON vs.challenge\_id=ch.challenge\_id
* GROUP BY u.contest\_id) t1 ON t1.contest\_id=c.contest\_id
* GROUP BY c.contest\_id, c.hacker\_id, c.name
* HAVING t0.s1+t1.s1>0;
* 845 579 Rose 1987 580 1635 566
* 858 1053 Angela 703 160 1002 384
* 883 1055 Frank 1121 319 1217 338
* 1793 2655 Patrick 1337 360 1216 412
* 2374 2765 Lisa 2733 815 3368 904
* 2963 2845 Kimberly 4306 1221 3603 1184

1. Display the list of products orderes along with top 5 products which were ordered more than once. The final list should not contains the top 5 products. The top 5 products are to be computed based on the count of orders.

* SELECT distinct orderid
* FROM orderdetails
* WHERE productid in (Select Productid from (Select productid, quantity from orderdetails) group by Productid, quantity having count(\*) > 2)
* Display Top 5th Quantity
* Select Quantity FROM OrderDetails
* Order BY Quantity DESC
* LIMIT 4,1;

LIMIT 4,1 => skips the first 4 rows and gives the Top 5th Quantity (1 = single value)

* Display the list of products orderes along with top 5 products which were ordered more than once. The final list should not contains the top 5 products. The top 5 products are to be computed based on the count of orders.
* SELECT DISTINCT(ProductID) FROM (SELECT ProductID, COUNT(ProductID)
* FROM OrderDetails
* GROUP BY ProductID
* HAVING COUNT(ProductID) > 2
* ORDER BY COUNT(ProductID) DESC)
* LIMIT 5

1. Second Highest Salary (we can solve it using limit 1,1 but it will not give null if column have only 1 row) by using sub-query it will give corre

* SELECT MAX(salary) AS SecondHighestSalary FROM Employee
* WHERE salary < (SELECT MAX(salary) from employee ORDER BY salary DESC)

1. Nth Highest Salary

CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT

BEGIN

  set N = N - 1;

  RETURN (

      SELECT distinct(salary) from employee

      order by salary desc

      limit 1 offset N

  );

END