Sure, let's tackle each task one by one. Here's the solution for each task described:

\*\*Task 1: Finding start and end dates of projects based on consecutive End\_Date\*\*

```sql

WITH ProjectsCTE AS (

SELECT Task\_ID, Start\_Date, End\_Date,

ROW\_NUMBER() OVER (ORDER BY Start\_Date) as rn

FROM Projects

), GroupedProjects AS (

SELECT Task\_ID, Start\_Date, End\_Date,

DATEADD(DAY, -ROW\_NUMBER() OVER (ORDER BY Start\_Date), End\_Date) as grp

FROM ProjectsCTE

)

SELECT MIN(Start\_Date) as Start\_Date, MAX(End\_Date) as End\_Date,

COUNT(\*) as Days

FROM GroupedProjects

GROUP BY grp

ORDER BY Days ASC, Start\_Date ASC;

```

\*\*Task 2: Names of students whose best friends got offered a higher salary\*\*

```sql

SELECT S.Name

FROM Students S

JOIN Friends F ON S.ID = F.ID

JOIN Packages P1 ON S.ID = P1.ID

JOIN Packages P2 ON F.Friend\_ID = P2.ID

WHERE P2.Salary > P1.Salary

ORDER BY P2.Salary;

```

\*\*Task 3: Finding symmetric pairs (X, Y) and (Y, X)\*\*

```sql

SELECT F1.X, F1.Y

FROM Functions F1

JOIN Functions F2 ON F1.X = F2.Y AND F1.Y = F2.X

WHERE F1.X < F1.Y

ORDER BY F1.X;

```

\*\*Task 4: Contest stats for each contest\*\*

```sql

SELECT c.contest\_id, c.hacker\_id, c.name,

SUM(ss.total\_submissions) AS total\_submissions,

SUM(ss.total\_accepted\_submissions) AS total\_accepted\_submissions,

SUM(vs.total\_views) AS total\_views,

SUM(vs.total\_unique\_views) AS total\_unique\_views

FROM Contests c

JOIN Colleges col ON c.contest\_id = col.contest\_id

JOIN Challenges ch ON col.college\_id = ch.college\_id

LEFT JOIN Submission\_Stats ss ON ch.challenge\_id = ss.challenge\_id

LEFT JOIN View\_Stats vs ON ch.challenge\_id = vs.challenge\_id

GROUP BY c.contest\_id, c.hacker\_id, c.name

HAVING SUM(ss.total\_submissions) > 0 OR SUM(ss.total\_accepted\_submissions) > 0 OR

SUM(vs.total\_views) > 0 OR SUM(vs.total\_unique\_views) > 0

ORDER BY c.contest\_id;

```

\*\*Task 5: Unique hackers and maximum submissions each day during contest\*\*

```sql

SELECT submission\_date,

COUNT(DISTINCT hacker\_id) AS unique\_hackers,

hacker\_id,

(SELECT name FROM Hackers WHERE hacker\_id = max\_hacker\_id) AS name

FROM (

SELECT submission\_date, hacker\_id,

DENSE\_RANK() OVER (PARTITION BY submission\_date ORDER BY COUNT(submission\_id) DESC, hacker\_id ASC) AS rnk,

MAX(hacker\_id) OVER (PARTITION BY submission\_date ORDER BY COUNT(submission\_id) DESC, hacker\_id ASC) AS max\_hacker\_id

FROM Submissions

GROUP BY submission\_date, hacker\_id

) ranked\_submissions

WHERE rnk = 1

GROUP BY submission\_date, hacker\_id, max\_hacker\_id

ORDER BY submission\_date;

```

\*\*Task 6: Manhattan Distance between points\*\*

```sql

SELECT ROUND(ABS(MIN(LAT\_N) - MAX(LAT\_N)) + ABS(MIN(LONG\_W) - MAX(LONG\_W)), 4) AS manhattan\_distance

FROM STATION;

```

\*\*Task 7: Prime numbers <= 1000\*\*

```sql

WITH RECURSIVE primes AS (

SELECT 2 AS num

UNION ALL

SELECT num + 1

FROM primes

WHERE num + 1 <= 1000 AND NOT EXISTS (

SELECT 1

FROM primes p

WHERE p.num <= FLOOR(SQRT(primes.num + 1))

AND (primes.num + 1) % p.num = 0

)

)

SELECT STRING\_AGG(num, '&') AS prime\_numbers

FROM primes;

```

\*\*Task 8: Pivot Occupation\*\*

```sql

SELECT

MAX(CASE WHEN occupation = 'Doctor' THEN name END) AS Doctor,

MAX(CASE WHEN occupation = 'Professor' THEN name END) AS Professor,

MAX(CASE WHEN occupation = 'Singer' THEN name END) AS Singer,

MAX(CASE WHEN occupation = 'Actor' THEN name END) AS Actor

FROM (

SELECT name, occupation,

ROW\_NUMBER() OVER (PARTITION BY occupation ORDER BY name) AS row\_num

FROM Occupations

) AS t

GROUP BY row\_num;

```

\*\*Task 9: Node type of Binary Tree\*\*

```sql

SELECT N,

CASE

WHEN P IS NULL THEN 'Root'

WHEN NOT EXISTS (SELECT 1 FROM BST WHERE P = BST.N) THEN 'Leaf'

ELSE 'Inner'

END AS node\_type

FROM BST

ORDER BY N;

```

\*\*Task 10: Company hierarchy statistics\*\*

```sql

SELECT c.company\_code, c.founder,

COUNT(DISTINCT lm.lead\_manager\_code) AS total\_lead\_managers,

COUNT(DISTINCT sm.senior\_manager\_code) AS total\_senior\_managers,

COUNT(DISTINCT m.manager\_code) AS total\_managers,

COUNT(DISTINCT e.employee\_code) AS total\_employees

FROM Company c

LEFT JOIN Lead\_Manager lm ON c.company\_code = lm.company\_code

LEFT JOIN Senior\_Manager sm ON lm.lead\_manager\_code = sm.lead\_manager\_code

LEFT JOIN Manager m ON sm.senior\_manager\_code = m.senior\_manager\_code

LEFT JOIN Employee e ON m.manager\_code = e.manager\_code

GROUP BY c.company\_code, c.founder

ORDER BY c.company\_code;

```

\*\*Task 11: Students whose best friends got a higher salary\*\*

```sql

SELECT S.Name

FROM Students S

JOIN Friends F ON S.ID = F.ID

JOIN Packages P1 ON S.ID = P1.ID

JOIN Packages P2 ON F.Friend\_ID = P2.ID

WHERE P2.Salary > P1.Salary

ORDER BY P2.Salary;

```

\*\*Task 12: Ratio of cost of job family in percentage by India and international\*\*

```sql

SELECT JobFamily,

ROUND((SUM(CASE WHEN Country = 'India' THEN Cost ELSE 0 END) / SUM(Cost)) \* 100, 2) AS India\_Percentage,

ROUND((SUM(CASE WHEN Country != 'India' THEN Cost ELSE 0 END) / SUM(Cost)) \* 100, 2) AS International\_Percentage

FROM SimulationData

GROUP BY JobFamily;

```

\*\*Task 13: Ratio of cost and revenue of a BU month on month\*\*

```sql

SELECT BU, Month,

ROUND(SUM(Cost) / NULLIF(SUM(Revenue), 0), 2) AS Cost\_Revenue\_Ratio

FROM BU\_MonthlyData

GROUP BY BU, Month

ORDER BY BU, Month;

```

\*\*Task 14: Headcounts of sub-band and percentage without join, subquery, and inner query\*\*

```sql

SELECT SubBand,

COUNT(\*) AS Headcount,

ROUND((COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM EmployeeData)), 2) AS Percentage

FROM EmployeeData

GROUP BY SubBand;

```

\*\*Task 15: Top 5 employees according to salary (without ORDER BY)\*\*

```sql

SELECT TOP 5 WITH TIES EmployeeID, Salary

FROM EmployeeSalaries

WHERE Salary IS NOT NULL

QUALIFY ROW\_NUMBER() OVER (PARTITION BY 1 ORDER BY Salary DESC) <= 5;

```

\*\*Task 16: Swap value of two columns without using a third variable or a table\*\*

```sql

UPDATE TableName

SET Column1 = Column1 + Column2,

Column2 = Column1 - Column2,

Column1 = Column1 - Column2;

```

\*\*Task 17: Create a user, create a login for that user and provide DB\_owner permissions\*\*

```sql

CREATE LOGIN NewUserLogin WITH PASSWORD = 'password';

CREATE USER NewUser FOR LOGIN NewUserLogin;

ALTER ROLE db\_owner ADD MEMBER NewUser;

```

\*\*Task 18: Weighted average cost of employees month on month in a BU\*\*

```sql

SELECT BU, Month,

SUM(Cost \* Weight) / SUM(Weight) AS Weighted\_Avg\_Cost

FROM EmployeeCosts

GROUP BY BU, Month

ORDER BY BU, Month;

```

\*\*Task 19: Difference between actual and miscalculated average monthly salaries\*\*

```sql

SELECT CEILING(ABS(

(SELECT AVG(Salary) FROM Employees) -

(SELECT AVG(CAST(REPLACE(CAST(Salary AS VARCHAR), '0', '') AS INT)) FROM Employees)

)) AS Difference;

```

\*\*Task 20: Copy new data from one table to another\*\*

```sql

INSERT INTO TargetTable (Column1, Column2, ...)

SELECT Source.Column1, Source.Column2, ...

FROM SourceTable Source

LEFT JOIN TargetTable Target ON Source.PrimaryKey = Target.PrimaryKey

WHERE Target.PrimaryKey IS NULL;

```

Feel free to adjust the table and column names as necessary to fit your specific schema.