SQL Project: Analyse Employee Data

Introduction

This SQL project explores employee data using real-world queries and database concepts. The goal is to analyze staff information across departments to uncover insights into hiring patterns, salary distribution, department structure, and employee trends.

Using a combination of SQL techniques including basic filtering, joins, aggregate functions, window functions, and ranking—we answer practical business questions such as:

- Who are the highest-paid employees in each department?
- What is the average salary in the Sales team?
- Which departments have the most employees?
- When were employees hired?
- Are there employees with duplicate first names?

This report demonstrates how structured SQL logic can be used to gain actionable insights from relational datasets.

Dataset Overview

This project uses two related tables:

Employees Table

- Contains information about each employee.
- Key columns include:
    - `EmployeeID` – Unique ID for each employee.
    - `FirstName` and `LastName` – Employee's name.
    - `DepartmentID` – Links the employee to a department.
    - `HireDate` – Date the employee joined the company.
    - `Salary` – Employee's salary amount.

Departments Table

- Contains details about the departments in the company.
- Key columns include:

    - `DepartmentID` – Unique ID for each department.
    - `DeptName` – Name of the department (e.g., IT, Sales, Marketing).

Relationship Between Tables

- The `DepartmentID` is the common column between both tables.
- This allows us to **join** the tables and analyze employees within their respective departments.

What This Report Covers

This project includes 21 SQL tasks grouped by topic. Each task is designed to solve a real-world business question using structured SQL queries. Here's a breakdown of what's included:

- **Basic SELECT & WHERE**
  - List all employees who work in the Engineering department.
  - Find all employees hired after January 1, 2020.
    Show employees whose salary is between 50,000 and 90,000.
- **ORDER BY Practice**
  - List all employees ordered by hire date (oldest first).
  - Show top 10 employees with the highest salaries.
  - List employees in the Marketing department, sorted by salary descending.
- **DISTINCT & AGGREGATES**
  - Get a list of unique department IDs from the Employees table.
  - Find the average salary of employees in the Sales department.
  - Count how many employees each department has.
- **GROUP BY & HAVING**
  - Show each department with the total salary payout (group by DepartmentID).
  - List departments where the average salary is over 80,000.
  - Count departments with more than 5 employees.
- **JOINS**
  - Write a query to show Employee Name, DeptName, and Salary using a JOIN.
  - Show all departments even if they have no employees (LEFT JOIN).
  - Show employee details for those in the IT department.
- **WINDOW FUNCTIONS (ADVANCED)**
  - Add a column showing Max salary across all employees next to each row.
  - For each department, show each employee with their department's max salary.
  - Add a column to show employee rank by salary within each department.
- **CHALLENGE TASKS**
  - Show the top 2 highest paid employees in each department.
  - Write a query that finds employees with duplicate first names.
  - Create a new column that shows the year only from the HireDate.

SQL Tasks and Analysis

Task 1: List all employees who work in the Engineering department.

SQL Query:

```sql
SELECT E.*,D.DeptName
FROM Employees E
JOIN Departments D ON E.DepartmentID = D.DepartmentID
WHERE D.DeptName = 'Engineering';
```

Explanation:
This query joins the `Employees` and `Departments` tables on `DepartmentID`, then filters for employees in the 'Engineering' department.

Results / Insights:
Lists all employees with full details working in Engineering, including department name for clarity.

Task 2: Find all employees hired after January 1, 2020

SQL Query:

```sql
SELECT *
FROM Employees
where HireDate > '2020-01-01';
```

Explanation:
This query selects all employees whose `HireDate` is later than January 1, 2020, effectively filtering for recently hired staff.

Results / Insights:
The output shows employees hired since 2020, which can help identify recent additions to the company workforce.

Task 3: Show employees whose salary is between 50,000 and 90,000

SQL Query:

```sql
SELECT *
FROM Employees
WHERE Salary BETWEEN 50000 AND 90000;
```

Explanation:
This query retrieves employees whose salary falls within the range of $50,000 to $90,000, inclusive.

Results / Insights:
This data helps analyze mid-range salary employees and understand the distribution within this pay bracket.

Task 4: List all employees ordered by hire date (oldest first)

SQL Query:

```sql
SELECT *
FROM Employees
```

```
Order by HireDate ASC;
```

Explanation:
This query retrieves all employee records sorted by their hire date in ascending order. The oldest hires appear first, allowing analysis of employee tenure.

Results / Insights:
Helps identify long-standing employees and see hiring trends over time.

Task 5: Show top 10 employees with the highest salaries

SQL Query:

```
SELECT Distinct FirstName, Salary
FROM Employees
ORDER BY Salary DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

Explanation:
This query lists the top 10 highest-paid employees, ordered by salary descending. The `DISTINCT` keyword is used to avoid duplicate first names, though it may exclude some entries.

Results / Insights:
Useful to identify top earners in the company. Note: If multiple employees share the same first name, some may be omitted due to `DISTINCT`.

Task 6: List employees in the Marketing department, sorted by salary descending

SQL Query:

```
SELECT E.*,D.DeptName
FROM Employees E
JOIN Departments D ON E.DepartmentID = D.DepartmentID
WHERE D.DeptName = 'Marketing'
ORDER BY Salary DESC;
```

Explanation:
This query joins `Employees` with `Departments`, filters for the Marketing department, and orders the employees by salary in descending order to highlight the highest paid Marketing staff.

Results / Insights:
Shows salary distribution within Marketing, useful for budget and resource planning.

Task 7: Get a list of unique department IDs from the Employees table

SQL Query:

```
SELECT
   E.FirstName,
```

```
    E.LastName,
    E.Salary,
    D.DeptName,
    AVG(E.Salary) OVER (PARTITION BY E.DepartmentID) AS AvgDeptSalary
FROM Employees E
JOIN Departments D ON E.DepartmentID = D.DepartmentID
WHERE D.DeptName = 'Sales'
ORDER BY E.Salary DESC;
```

Explanation:
This query retrieves a unique list of department IDs from the `Employees` table using the `DISTINCT` keyword to avoid duplicates.

Results / Insights:
Helps understand which departments currently have employees assigned. Useful for auditing active departments in the employee dataset.

Task 8: Find the average salary of employees in the Sales department

SQL Query:

```
SELECT
    E.FirstName,
    E.LastName,
    E.Salary,
    D.DeptName,
    AVG(E.Salary) OVER (PARTITION BY E.DepartmentID) AS AvgDeptSalary
FROM Employees E
JOIN Departments D ON E.DepartmentID = D.DepartmentID
WHERE D.DeptName = 'Sales'
ORDER BY E.Salary DESC;
```

Explanation:
This query uses a `JOIN` to bring in department names and a **window function** (`AVG OVER PARTITION BY`) to calculate the average salary within the Sales department. It displays the average salary on every row for context.

Results / Insights:
Provides a detailed breakdown of each Sales employee's salary along with their department's average. Useful for comparing individual earnings against departmental averages.

Task 9: Count how many employees each department has

SQL Query:

```
SELECT D.DeptName, COUNT(*) AS EmployeeCount
FROM Employees E
JOIN Departments D ON E.DepartmentID = D.DepartmentID
GROUP BY D.DeptName
```

```
ORDER BY EmployeeCount DESC;
```

Explanation:
This query joins `Employees` and `Departments`, then uses `GROUP BY` to count how many employees are in each department. Results are sorted from largest to smallest department size.

Results / Insights:
Shows department sizes, which can inform workforce planning, department budgets, or staffing evaluations.

Task 10: Show each department with the total salary payout (group by DepartmentID)

SQL Query:

```
SELECT DepartmentID, SUM(Salary) AS TotalSalaryPayout
FROM Employees
GROUP BY DepartmentID;
```

Explanation:
This query groups all employees by their `DepartmentID` and calculates the **total salary** paid out in each department using `SUM(Salary)`.

Results / Insights:
Gives a department-wise breakdown of total payroll expenditure. Useful for finance or HR departments to evaluate budget allocation per team.

Task 11: List departments where the average salary is over 80,000

SQL Query:

```
SELECT DepartmentID, AVG(Salary) AS AvgSalary
FROM Employees
GROUP BY DepartmentID
HAVING AVG(Salary) > 80000;
```

Explanation:
This query calculates the average salary per department and filters results to show only those departments where the average salary exceeds 80,000. The `HAVING` clause is used to filter aggregated data.

Results / Insights:
Identifies high-paying departments. Useful for analyzing salary structures or reviewing compensation strategies.

Task 12: Count departments with more than 5 employees

SQL Query:

```
SELECT DepartmentID, COUNT(*) AS EmployeeCount
FROM Employees
```

```
GROUP BY DepartmentID
HAVING COUNT(*) > 5;
```

Explanation:
This query counts how many employees are in each department and filters for departments with more than 5 employees. The `HAVING` clause is used to apply the filter on grouped results.

Results / Insights:
Helps identify larger departments that may need more resources, management attention, or space planning.

Task 13: Show Employee Name, DeptName, and Salary using a JOIN

SQL Query:

```
SELECT E.FirstName, E.LastName, D.DeptName, E.Salary
FROM Employees E
JOIN Departments D
ON E.DepartmentID = D.DepartmentID;
```

Explanation:
This query joins the `Employees` and `Departments` tables using an **INNER JOIN** on `DepartmentID`. It selects the employee's first name, last name, department name, and salary.

Results / Insights:
Combines employee and department data in a readable format. Useful for reporting and HR dashboards showing employee details with their associated departments.

Task 14: Show all departments even if they have no employees (LEFT JOIN)

SQL Query:

```
SELECT D.DeptName, E.FirstName, E.LastName, E.Salary
FROM Departments D
LEFT JOIN Employees E
ON D.DepartmentID = E.DepartmentID;
```

Explanation:
This query uses a **LEFT JOIN** to show all departments, including those that do **not currently have any employees**. Departments without employees will have `NULL` in the employee-related columns.

Results / Insights:
Helpful to identify vacant or newly created departments with no assigned staff. Useful in workforce planning or department setup reviews.

Task 15: Show employee details for those in the IT department

SQL Query:

```sql
SELECT E.*, D.DeptName
FROM Employees E
JOIN Departments D
 ON E.DepartmentID = D.DepartmentID
 Where DeptName = 'IT';
```

Explanation:
This query joins `Employees` and `Departments`, then filters for records where the department name is **'IT'**. It returns full employee details plus the department name.

Results / Insights:
Provides a complete list of all employees in the IT department. Useful for analyzing technical team members or preparing team-specific reports.

Task 16: Add a column showing max salary across all employees next to each row

SQL Query:

```sql
SELECT FirstName, LastName, DepartmentID, Salary,
MAX(Salary) OVER () AS Max_Salary
FROM Employees;
```

With department name included:

```sql
select E.FirstName,
E.LastName,E.DepartmentID,E.Salary,D.DeptName,Max(E.Salary) Over() AS
Max_Salary
FROM Employees E
Join Departments D
ON E.DepartmentID = D.DepartmentID;
```

Explanation:
This query uses the `MAX()` **window function without a PARTITION clause** to show the **maximum salary in the entire company** next to each employee row.

Results / Insights:
Each employee row includes the highest salary value in the dataset, making it easy to compare individual salaries to the company maximum.

Task 17: For each department, show each employee with their department's max salary

SQL Query:

```
SELECT FirstName, LastName, DepartmentID, Salary, MAX(Salary) OVER
(PARTITION by (DepartmentID)) AS Max_Department_Salary
From Employees;
```

Explanation:
This query partitions the data by `DepartmentID` and uses `MAX()` as a **window function** to return the **maximum salary for each department** alongside every employee row.

Results / Insights:
Allows side-by-side comparison of an employee's salary with the top earner in their department. Useful for salary benchmarking and equity analysis.

Task 18: Add a column to show employee rank by salary within each department

SQL Query:

```
SELECT
   E.*,
   D.DeptName,
   rank() OVER (
   Partition by (E.DepartmentID)
    oRDER BY E.Salary DESC
   ) AS SALARY_RANK
FROM Employees E
Join Departments D
ON E.DepartmentID=D.DepartmentID;
```

Explanation:
This query uses the `RANK()` window function to rank employees by their salary **within each department**. Higher salaries get lower ranks (1 = highest paid).

Results / Insights:
Reveals the salary hierarchy inside departments. Useful for identifying top performers, structuring promotions, or targeting reviews.

Task 19: Show the top 2 highest paid employees in each department

SQL Query:

```
SELECT *
FROM
(
   SELECT
   E.FirstName,
   E.LastName,
   E.Salary,
   D.DeptName,
   rank() OVER (
```

```
   Partition by (E.DepartmentID)
    oRDER BY E.Salary DESC
   ) AS SALARY_RANK
FROM Employees E
Join Departments D
ON E.DepartmentID=D.DepartmentID
) AS RANKED_EMPLOYEES
WHERE Salary_Rank<=2;
```

Explanation:
This query uses a **window function with RANK()** to assign salary ranks within each department. Then it filters to return only the **top 2 earners** in each department.

Results / Insights:
Displays the highest and second-highest paid employees in every department. Useful for performance reviews, incentive planning, or identifying key talent.

Task 20: Find employees with duplicate first names

Option A – Just show duplicate names and how many times they occur:

```
SELECT FirstName, COUNT(*) AS NameCount
FROM Employees
GROUP BY FirstName
HAVING COUNT(*) > 1;
```

Option B – Show full details of employees with duplicate first names:

```
SELECT *,COUNT(*) OVER (PARTITION BY FirstName) AS NameCount
FROM Employees
WHERE FirstName IN (
   SELECT FirstName
   FROM Employees
   GROUP BY FirstName
   HAVING COUNT(*) > 1
);
```

Explanation:
The first query counts how many times each first name appears and filters those that occur more than once.
The second query shows **full employee records** for all such names using a subquery and window function.

Results / Insights:
Useful for identifying naming collisions or preparing systems that require unique identifiers.

Task 21: Create a new column that shows the year only from the HireDate

SQL Query:

```sql
SELECT *, YEAR(HireDate) AS Year_Hired
FROM Employees;
```

Explanation:
This query uses the `YEAR()` function to extract just the **year** part from the `HireDate`, adding it as a new column named `Year_Hired`.

Results / Insights:
Helpful for summarizing hiring trends over years or building year-based visualizations and filters.