Employees Database SQL and Tableau Project

Matthew Swithers

May 2023

• Introduction

This project has been broken down into two sections and is designed to display my coding experience with SQL as well as my ability to create meaningful working dashboards in Tableau. Part 1 of the project will show various SQL queries, varying in complexity, along with the problem that the query is supposed to solve. Part 2 includes more SQL queries, but will also include four Tableau charts and a dashboard that puts all of those charts together.

• Data Summary

Data for this project came from the "Employees" database from Kaggle that is commonly used for practicing data manipulation and analysis. This database does not include real-world data. The Employees database is broken down into six tables, called employees, departments, dept_manager, dept_emp, titles, and salaries. Throughout the project, other tables have been added to the database and will be referred to in the code.

• Part 1

This section will focus on SQL code and the problems that the code solves. All code is commented, but with different levels of specificity, to show that I can provide detailed or generalized comments.

- ○ Create a new table titled "departments_dup."

```sql
-- Create a duplicate table to hold department data
CREATE TABLE departments_dup (
    dept_no CHAR(4),
    dept_name VARCHAR(40)
);

-- Copy existing department data to the duplicate table
INSERT INTO departments_dup (dept_no, dept_name)
SELECT dept_no, dept_name
FROM departments;

-- Add a new department to the duplicate table using a specific department name
INSERT INTO departments_dup (dept_name)
VALUES ('Public Relations');

-- Delete a specific department from the duplicate table
DELETE FROM departments_dup
WHERE dept_no = 'd002';

-- Add a new department to the duplicate table using a specific department number
INSERT INTO departments_dup (dept_no)
VALUES ('d010');

-- Add another new department to the duplicate table using a specific department number
INSERT INTO departments_dup (dept_no)
VALUES ('d011');
```

This query is designed to create a new table called departments_dup, which will be a duplicate version of the departments table with slight modifications made to it by removing and adding rows.

- Create and call a stored procedure for average salary.

```
-- Set the database to be used
USE employees;

-- Drop the procedure if it already exists
DROP PROCEDURE IF EXISTS avg_salary;

-- Set the delimiter for the procedure definition
DELIMITER $$
CREATE PROCEDURE avg_salary()
BEGIN
    -- Select the rounded average of the salary column from the salaries table
    SELECT
        ROUND(AVG(salary), 2)
    FROM
        salaries;
END$$

-- Reset the delimiter to the default value
DELIMITER ;

-- Call the avg_salary procedure
CALL avg_salary();
```

The purpose of this code is to create a stored procedure in SQL that calculates the average salary of all employees rounded to 2 decimal places. This is a very basic stored procedure, but finding average salary is probably a very common task that can be drastically simplified for all users if made into a stored procedure.

- Create a table view to calculate the average manager salary

```
CREATE OR REPLACE VIEW v_avg_manager_salary AS
    SELECT
        ROUND(AVG(salary), 2)  -- Calculate the average salary and round to two decimal places
    FROM
        salaries s
        JOIN titles t ON s.emp_no = t.emp_no -- Join the salaries and titles tables based on
employee number
    WHERE
        t.title = 'Manager';  -- Filter for employees with the title 'Manager'
```

This query will create a table view in SQL for the average manager salary rounded to 2 decimal places. The table view can serve as a virtual table, usable in other queries to more easily write complex code in a more compact way.

- Create a trigger to automatically update hire date

```
-- Set the database to be used
USE employees;

-- Commit any pending changes in the current transaction
COMMIT;

-- Set the delimiter for the trigger definition
DELIMITER $$

-- Create a trigger named 'before_hire_date' to enforce a rule before inserting data into the
employees table
CREATE TRIGGER before_hire_date
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    -- Check if the new hire date is greater than the current date
    IF NEW.hire_date > DATE_FORMAT(SYSDATE(), '%Y-%m-%d') THEN
        -- If it is, set the new hire date to the current date
        SET NEW.hire_date = DATE_FORMAT(SYSDATE(), '%Y-%m-%d');
    END IF;
END$$

-- Reset the delimiter to the default value
DELIMITER ;
```

This is an example of a "before" trigger, which is a program that automatically modifies table entries before they are placed into the destination table. The trigger here is designed to check the hire date of a new employee against the current system date. If the new hire date is greater than the system date, the new hire date will be set to equal the system date instead. Specifically, the date format of YYYY-MM-DD is used to match the format currently in place in this database.

- Simple inner join

```
-- Retrieve employee information along with their department numbers and hire dates
SELECT
    e.emp_no, e.first_name, e.last_name, m.dept_no, e.hire_date
FROM
    employees e
    JOIN dept_manager m ON e.emp_no = m.emp_no;
```

This example is just a simple inner join that connects the employees and dept_manager tables to allow us to select the department number, which is not found in the employees table. We also use "e" as the alias for employees and "m" as the alias for dept_manager.

- Create a stored procedure with both input and output parameters

```
-- Set the database to be used
USE employees;

-- Drop the procedure if it already exists
DROP PROCEDURE IF EXISTS emp_info;

-- Set the delimiter for the procedure definition
DELIMITER $$

-- Create a new procedure named 'emp_info' with input and output parameters
CREATE PROCEDURE emp_info(
    IN p_first_name VARCHAR(255),
    IN p_last_name VARCHAR(255),
    OUT p_emp_no INTEGER
)
BEGIN
    -- Select the emp_no column and assign the result to the output parameter p_emp_no
    SELECT
        e.emp_no
    INTO p_emp_no
    FROM
        employees e
    WHERE
        -- Filter the rows by matching the first_name and last_name columns with the input
parameters
        e.first_name = p_first_name
        AND e.last_name = p_last_name;
END$$

-- Reset the delimiter to the default value
DELIMITER ;

-- Call the emp_info procedure with no arguments
CALL emp_info();
```

This stored procedure allows you to input the first and last name of an employee and return the employee number for that person. Stored procedures like this can be very helpful for someone to look up employee numbers of individuals within the company without having to write a new query each time.

- Basic query for rounded average salary

```
SELECT
    ROUND(AVG(salary), 2) -- Calculate the average salary and round to two decimal places
FROM
    salaries
WHERE
    from_date > '1997-01-01'; -- Filter for salaries with a from_date greater than '1997-01-01'
```

This simple query is to select the average salary, rounded to two decimal places, of employees who started with the company after January 1st 1997.

- Create a query for salary and the number of employees with that salary

```
SELECT
    salary, COUNT(emp_no) AS 'emps_with_same_salary' -- Select the salary and count of employees
with the same salary
FROM
    salaries
WHERE
    salary > 80000 -- Filter for salaries greater than 80000
GROUP BY
    salary -- Group the results by salary
ORDER BY
    salary; -- Order the result by salary in ascending order
```

This query uses the COUNT function to find the total number of employees who have each salary amount. Essentially, this query will figure out all of the salary values with multiple employees earning at that rate. In this example, we're only interested in salaries over $80,000 per year.

- Create a query to find duplicate first names

```
SELECT
    first_name, COUNT(first_name) AS names_count -- Select the first name and count of occurrences
FROM
    employees
WHERE
    hire_date > '1999-01-01' -- Filter for employees hired after '1999-01-01'
GROUP BY
    first_name -- Group the results by first name
HAVING
    COUNT(first_name) < 200 -- Filter for counts less than 200
ORDER BY
    first_name; -- Order the result by first name
```

In this example, we have a query that selects all of the different first names in the employees table, groups the data by first name, and displays the counts for how many employees share that first name. The results are then filtered to only show names of employees hired after January 1st 1999, and only names that occur less than 200 times. Finally, the results are ordered by first name.

- Creation of a complex query with multiple joins and unions

```
INSERT INTO emp_manager
SELECT U.*
FROM (
    -- Retrieve employee-manager relationships for emp_no <= 10020
    SELECT
        A.*
    FROM (
        -- Retrieve employee-manager relationships for emp_no <= 10020
        SELECT
            e.emp_no AS employee_ID, -- Employee ID
            MIN(de.dept_no) AS department_code, -- Department code for the employee
            (
                -- Retrieve the manager's employee ID for a specific manager
                SELECT emp_no
                FROM dept_manager
                WHERE emp_no = 110022 -- Manager's employee ID
            ) AS manager_ID
        FROM
            employees e
        JOIN dept_emp de ON e.emp_no = de.emp_no
        WHERE
            e.emp_no <= 10020 -- Limit the emp_no range
        GROUP BY e.emp_no
        ORDER BY e.emp_no -- Order the results by employee ID
    ) AS A
    UNION
    -- Retrieve employee-manager relationships for emp_no > 10020, limited to 20 results
    SELECT
        B.*
    FROM (
        -- Retrieve employee-manager relationships for emp_no > 10020, limited to 20 results
        SELECT
            e.emp_no AS employee_ID, -- Employee ID
            MIN(de.dept_no) AS department_code, -- Department code for the employee
            (
                -- Retrieve the manager's employee ID for a specific manager
                SELECT emp_no
                FROM dept_manager
                WHERE emp_no = 110039 -- Manager's employee ID
            ) AS manager_ID
        FROM
            employees e
        JOIN dept_emp de ON e.emp_no = de.emp_no
        WHERE
            e.emp_no > 10020 -- Limit the emp_no range
        GROUP BY e.emp_no
        ORDER BY e.emp_no
        LIMIT 20 -- Limit the number of results to 20
    ) AS B
    UNION
    -- Retrieve employee-manager relationship for emp_no = 110022
    SELECT
        C.*
    FROM (
        -- Retrieve employee-manager relationship for emp_no = 110022
        SELECT
            e.emp_no AS employee_ID, -- Employee ID
            MIN(de.dept_no) AS department_code, -- Department code for the employee
            (
                -- Retrieve the manager's employee ID for a specific manager
                SELECT emp_no
                FROM dept_manager
                WHERE emp_no = 110039 -- Manager's employee ID
            ) AS manager_ID
        FROM
            employees e
        JOIN dept_emp de ON e.emp_no = de.emp_no
        WHERE
            e.emp_no = 110022 -- Limit the emp_no to a specific employee
        GROUP BY e.emp_no
        ORDER BY e.emp_no -- Order the results by employee ID
    ) AS C
    UNION
    -- Retrieve employee-manager relationship for emp_no = 110039
    SELECT
        D.*
    FROM (
        -- Retrieve employee-manager relationship for emp_no = 110039
        SELECT
            e.emp_no AS employee_ID, -- Employee ID
            MIN(de.dept_no) AS department_code, -- Department code for the employee
            (
                -- Retrieve the manager's employee ID for a specific manager
                SELECT emp_no
                FROM dept_manager
                WHERE emp_no = 110022 -- Manager's employee ID
            ) AS manager_ID
        FROM
            employees e
        JOIN dept_emp de ON e.emp_no = de.emp_no
        WHERE
            e.emp_no = 110039 -- Limit the emp_no to a specific employee
        GROUP BY e.emp_no
        ORDER BY e.emp_no -- Order the results by employee ID
    ) AS D
) AS U;
```

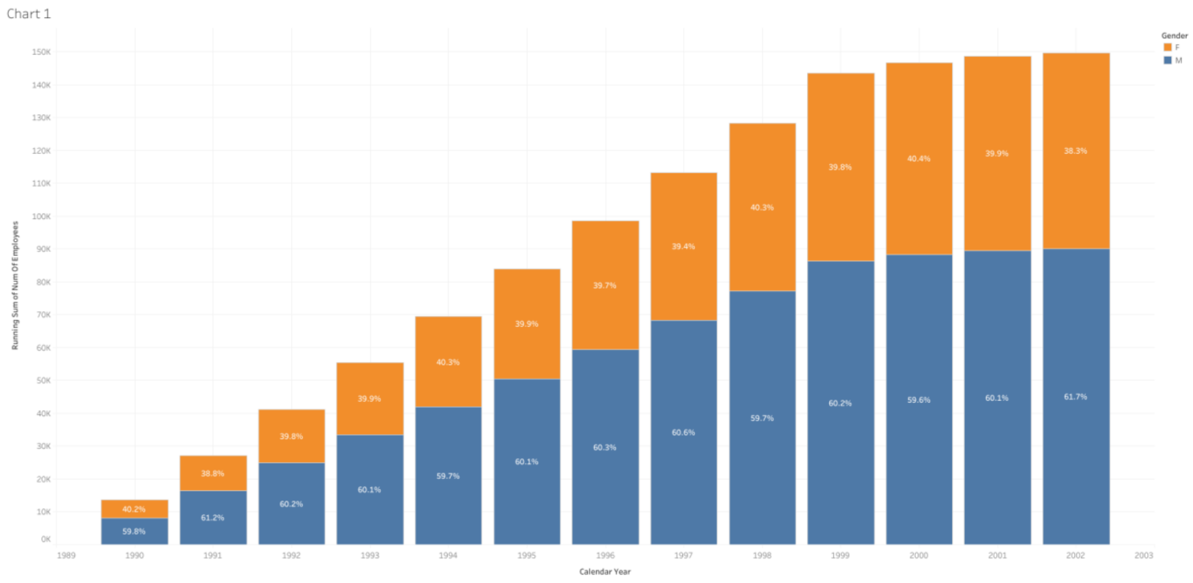
This complex query does several things at once. First note, we have a new table called emp_manager where we can assign managers to oversee certain employees. This query inserts new information into that table that assigns managers to dgroups of employees based on employee number ranges. The first section, part A, assigns the employee with the employee number of 110022 to be the manager of employees 10000 – 10020. We then union that with part B, which uses employee number 110039 and assigns them as manager over the employees with employee numbers of 10021 – 10040. After that, we union with part C, which assigns employee 110039 as manager of 110022. Part D does the opposite, where we assign 110022 to manager of 110039. With all four parts now defined, the query assigns the entire block of four sections connected with unions as a new part, called U. Then U is inserted into the emp_manager table to finish assigning the managers to the employees.

This concludes part 1. Part 2 will begin on the next page and will include more SQL code and my Tableau examples.

- Part 2 (https://public.tableau.com/app/profile/matthew.swithers)
    - Finding proportion of men and women working at the company across time

```
SELECT
    YEAR(d.from_date) AS calendar_year, -- Extract the year from 'from_date' and assign it as
'calendar_year'
    e.gender, -- Select the 'gender' column from 't_employees' table
    COUNT(e.emp_no) AS num_of_employees -- Count the number of 'emp_no' values and assign it as
'num_of_employees'
FROM
    t_employees e -- Alias for 't_employees' table
        JOIN
    t_dept_emp d ON d.emp_no = e.emp_no -- Join 't_employees' and 't_dept_emp' on matching 'emp_no'
values
GROUP BY calendar_year, e.gender -- Group the result by 'calendar_year' and 'gender'
HAVING calendar_year >= 1990; -- Filter the groups to include only those with 'calendar_year'
greater than or equal to 1990
```

This query finds the number of employees of each gender that work at the company and groups that data by year. Notice that the names of the tables have changed slightly to reflect that we have changed to a different version of the Employees database.

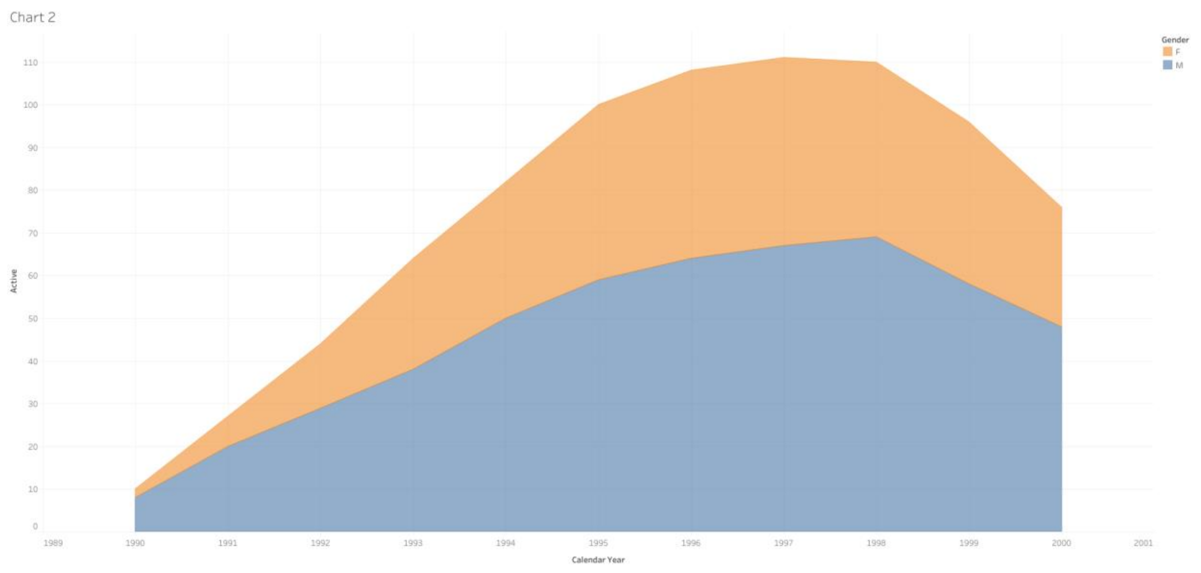


This visualization shows the results of our query as a bar chart, split by gender. This shows the cumulative number of employees working at the company each year and displays the percentage of men and women. You can see that the ratio of men to women in the company is about 6 to 4, or 60% men 40% women, and that stays roughly the same across time.

- Find the proportion of male and female active managers

```
SELECT
    d.dept_name, -- Select the 'dept_name' column from 't_departments' table
    ee.gender, -- Select the 'gender' column from 't_employees' table (aliased as 'ee')
    dm.emp_no, -- Select the 'emp_no' column from 't_dept_manager' table (aliased as 'dm')
    dm.from_date, -- Select the 'from_date' column from 't_dept_manager' table (aliased as 'dm')
    dm.to_date, -- Select the 'to_date' column from 't_dept_manager' table (aliased as 'dm')
    e.calendar_year, -- Select the 'calendar_year' column from the derived table 'e'
    CASE
        WHEN
            YEAR(dm.to_date) >= e.calendar_year -- Check if 'to_date' is greater than or equal to
'calendar_year'
                AND YEAR(dm.from_date) <= e.calendar_year -- Check if 'from_date' is less than or
equal to 'calendar_year'
        THEN
            1 -- If the condition is true, set 'active' to 1
        ELSE 0 -- Otherwise, set 'active' to 0
    END AS active -- Alias the computed column as 'active'
FROM
    (SELECT
        YEAR(hire_date) AS calendar_year -- Derive a table 'e' with 'calendar_year' extracted from
'hire_date'
    FROM
        t_employees
    GROUP BY calendar_year) e -- Group the derived table by 'calendar_year'
        CROSS JOIN
    t_dept_manager dm -- Perform a cross join with 't_dept_manager' table (aliased as 'dm')
        JOIN
    t_departments d ON dm.dept_no = d.dept_no -- Join 't_dept_manager' and 't_departments' on
matching 'dept_no' values
        JOIN
    t_employees ee ON dm.emp_no = ee.emp_no -- Join 't_dept_manager' and 't_employees' on matching
'emp_no' values (aliased as 'ee')
ORDER BY dm.emp_no, calendar_year; -- Sort the result by 'emp_no' and 'calendar_year'
```

This query is designed to find the total number of active managers in the company and break that information down by gender. First we check the from and to dates of the manager contract to check if the person is still a manager. If the system date is between those dates, then the manager is assigned a 1 to the Active column, otherwise they get a 0. Then we use that information to determine the number of managers currently active, broken down by gender.

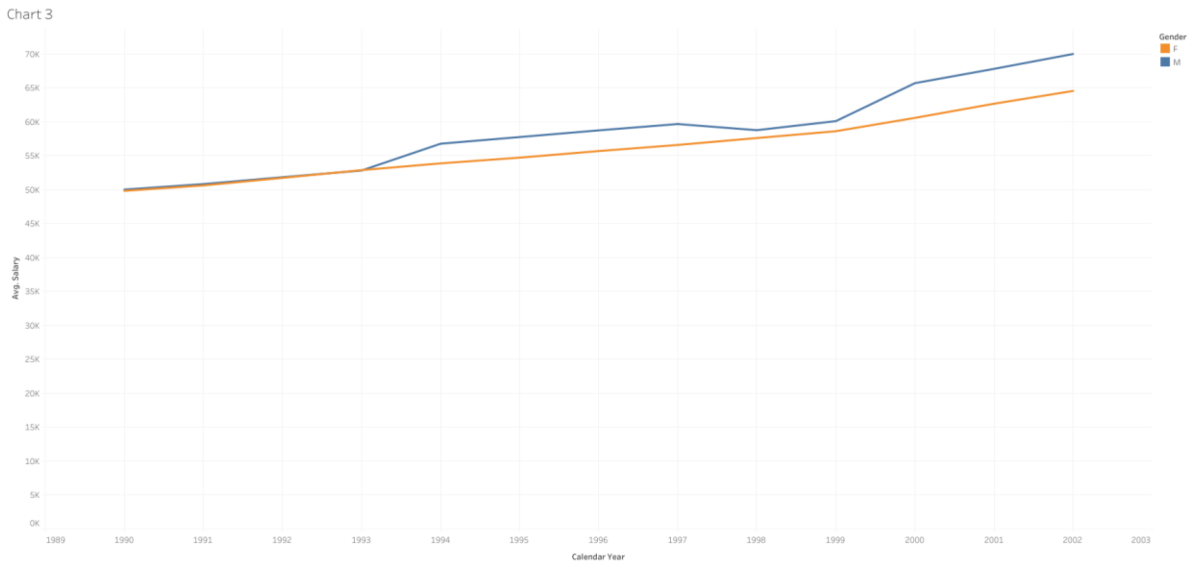


In the above graph, you can see that the ratio of male to female managers started very high in 1990 but becomes much more balanced out as time goes on. This chart can be sorted by department as well, which is possible on the actual Tableau website. You can find the link to all these charts at the beginning of Part 2 of this project.

- Finding average salary by gender across time

```
SELECT
    e.gender, -- Select the 'gender' column from 't_employees' table (aliased as 'e')
    d.dept_name, -- Select the 'dept_name' column from 't_departments' table (aliased as 'd')
    ROUND(AVG(s.salary), 2) AS salary, -- Calculate the average salary and round it to 2 decimal
places
    YEAR(s.from_date) AS calendar_year -- Extract the year from 'from_date' column and alias it as
'calendar_year'
FROM
    t_salaries s -- Select from 't_salaries' table (aliased as 's')
        JOIN
    t_employees e ON s.emp_no = e.emp_no -- Join 't_salaries' and 't_employees' on matching 'emp_no'
values (aliased as 'e')
        JOIN
    t_dept_emp de ON de.emp_no = e.emp_no -- Join 't_dept_emp' and 't_employees' on matching
'emp_no' values (aliased as 'de')
        JOIN
    t_departments d ON d.dept_no = de.dept_no -- Join 't_departments' and 't_dept_emp' on matching
'dept_no' values (aliased as 'd')
GROUP BY d.dept_no, e.gender, calendar_year -- Group the result by 'dept_no', 'gender', and
'calendar_year'
HAVING calendar_year <= 2002 -- Filter the result to include only rows where 'calendar_year' is less
than or equal to 2002
ORDER BY d.dept_no; -- Sort the result by 'dept_no'
```

This relatively basic query finds the average salary of all employees with start dates before 2002, then breaks that information down by department, gender, and year. This lets us look at the average salary trends for both genders across time and filtered by department.



As previously stated, this chart shows the trends of average salary by gender and across time. Just like the previous chart, this can be filtered by department number when viewed on the Tableau website. Link is at the beginning of Part 2 of this project.

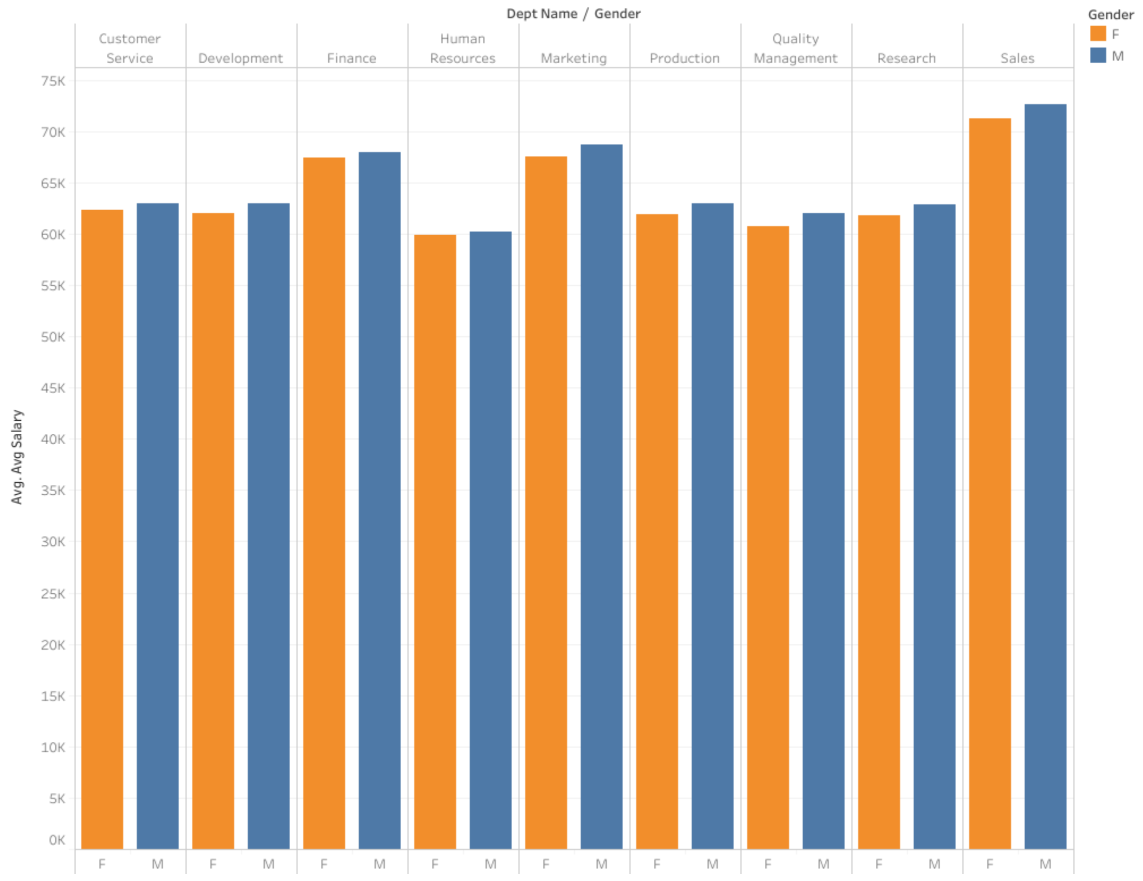
- Average salary by gender and department, excluding outliers

```
CREATE PROCEDURE filter_salary (IN p_min_salary FLOAT, IN p_max_salary FLOAT)
BEGIN
    SELECT
        e.gender, -- Select the 'gender' column from 't_employees' table (aliased as 'e')
        d.dept_name, -- Select the 'dept_name' column from 't_departments' table (aliased as 'd')
        AVG(s.salary) as avg_salary -- Calculate the average salary and alias it as 'avg_salary'
    FROM
        t_salaries s -- Select from 't_salaries' table (aliased as 's')
            JOIN
        t_employees e ON s.emp_no = e.emp_no -- Join 't_salaries' and 't_employees' on matching
'emp_no' values (aliased as 'e')
            JOIN
        t_dept_emp de ON de.emp_no = e.emp_no -- Join 't_dept_emp' and 't_employees' on matching
'emp_no' values (aliased as 'de')
            JOIN
        t_departments d ON d.dept_no = de.dept_no -- Join 't_departments' and 't_dept_emp' on
matching 'dept_no' values (aliased as 'd')
    WHERE
        s.salary BETWEEN p_min_salary AND p_max_salary -- Filter the result to include only rows
where 'salary' is between 'p_min_salary' and 'p_max_salary'
    GROUP BY
        d.dept_no, e.gender; -- Group the result by 'dept_no' and 'gender'
END$$

DELIMITER ;
```

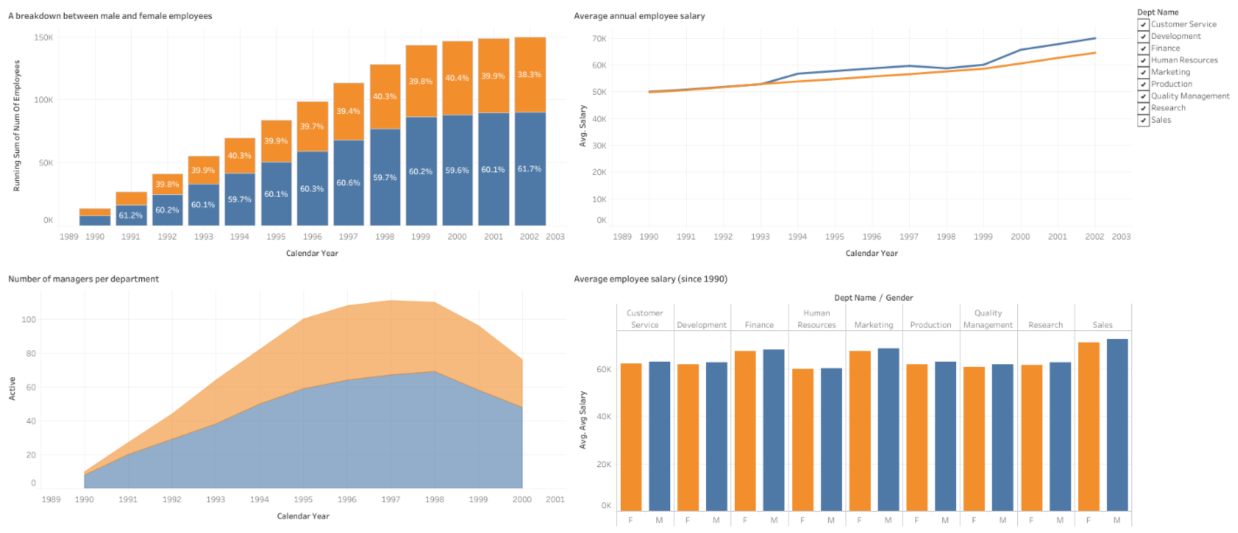
This stored procedure allows us to find the average salary information, grouped by gender and department, for all employees within a chosen salary range. Essentially, we can limit our data to fit within a certain salary range with the intent of removing outliers, while displaying the up-to-date information about average salary by gender and department.

Chart 4



The above graph shows exactly what we previously mentioned, the average salary of every employee, split by gender and department. This data was found using the stored procedure shown above using the minimum and maximum values of $50,000 and $90,000. In this hypothetical example, it was previously determined that $50,000 to $90,000 was the normal range of salaries for this population of employees.

- Putting it all together



This image shows the same four visualizations in the context of a complete Tableau dashboard. As seen on the right-hand side, this dashboard is filterable by department for 3 out of the 4 charts. All of this can be filtered and arranged in different ways for different purposes. Again, if you are interested in seeing the actual charts and complete dashboard, follow the link at the beginning of Part 2, which should link directly to my Tableau Public profile.