Exploratory Data Analysis in SQL

As part of the SQL journey to explore a database and the data in it, I accessed and built on basic functions of joining tables, grouping data, and using subqueries. Using data from Stack Overflow, Fortune 500 companies, and 311 help requests from Evanston, IL, I first got familiar with numeric, character, and date/time data types. I used functions to aggregate, summarize, and analyze data without leaving the database. Errors and inconsistencies in the data are common problems, so I developed strategies to clean up messy data. I explored this project using the PostgreSQL database.

1. Summarize all columns of a table in SQL:

```
SELECT *
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'stackoverflow';
```

2. Combine two columns even with blanks (NULL values) - COALESCE

```
SELECT coalesce(industry, sector, 'Unknown') AS industry2,
      -- Don't forget to count!
      count(*)
 FROM fortune500
-- Group by what? (What are you counting by?)
GROUP BY industry2
-- Order results to see most common first
ORDER BY count DESC
-- Limit results to get just the one value you want
Limit 1;
```

OR

```
SELECT company_original.name, title, rank
 -- Start with original company information
 FROM company AS company_original
```

```
      -- Join to another copy of company with parent
      -- company information
    LEFT JOIN company AS company_parent
      ON company_original.parent_id = company_parent.id
      -- Join to fortune500, only keep rows that match
      INNER JOIN fortune500
      -- Use parent ticker if there is one,
      -- otherwise original ticker
      ON coalesce(company_original.ticker,
                  company_parent.ticker) =
           fortune500.ticker
-- For clarity, order by rank
ORDER BY rank;
```

3. INNER JOIN (to select all records from the two tables where there is a match in both tables)

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

JOIN or INNER JOIN

`JOIN` and `INNER JOIN` will return the same result.

`INNER` is the default join type for `JOIN`, so when you write `JOIN` the parser actually writes `INNER JOIN`.

4. RIGHT JOIN (*to select all the records from the Customers table plus all the matches in the Orders table)*

```
SELECT *
FROM Orders
RIGHT JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

5. Compute the average revenue per employee for Fortune 500 companies by sector:

```
-- Select average revenue per employee by sector
SELECT sector,
       avg(revenues/employees::numeric) AS avg_rev_employee
 FROM fortune500
GROUP BY sector
-- Use the column alias to order the results
ORDER BY avg_rev_employee;
```

6. Compare columns with NOT equal to 0:

```
-- Divide unanswered_count by question_count
SELECT unanswered_count/question_count::numeric AS
computed_pct,
       -- What are you comparing the above quantity to?
       unanswered_pct
 FROM stackoverflow
-- Select rows where question_count is not 0
WHERE question_count != 0
Limit 10;
```

7. Summarize numeric columns:

```
-- Select min, avg, max, and stddev of fortune500 profits
SELECT min(profits),
       avg(profits),
       max(profits),
       stddev(profits)
 FROM fortune500;
```

8. Summarize profits by “sector”.
   And order the results by the “average” profits for each sector.

```
-- Select sector and summary measures of fortune500 profits
SELECT min(profits),
       avg(profits),
```

```
      max(profits),
      stddev(profits)
 FROM fortune500
-- What to group by?
GROUP BY sector
-- Order by the average profits
ORDER BY avg;
```

9. Summarize the distribution of the number of questions with the tag "dropbox" on Stack Overflow per day by binning the data:

```
-- Select the min and max of question_count
SELECT min(question_count), max(question_count)
 -- From what table?
 FROM stackoverflow
-- For tag dropbox
WHERE tag = 'dropbox';
```

10. Correlation:

```
-- Correlation between revenues and profit
SELECT corr(revenues, profits) AS rev_profits,
       -- Correlation between revenues and assets
      corr(revenues, assets) AS rev_assets,
      -- Correlation between revenues and equity
      corr(revenues, equity) AS rev_equity
 FROM fortune500;
```

11. Create temporary tables:
    Find the Fortune 500 companies that have profits in the top 20% for their sector (compared to other Fortune 500 companies).

12. Select the count of each level of priority

(*How many rows does each priority level have?*)

```
SELECT priority, count(*)
FROM evanston311
GROUP BY priority;
```

```
-- Find values of zip that appear in at least 100 rows
-- Also get the count of each value
SELECT zip, count(*)
FROM evanston311
GROUP BY zip
HAVING count(*) >= 100;
-- Find values of source that appear in at least 100 rows
-- Also get the count of each value
SELECT source, count(*) AS count
FROM evanston311
GROUP BY source
HAVING count(*) >= 100
ORDER BY count DESC;
```

13. Select the five most common values of street and the count of each.

```
-- Find the 5 most common values of street and the count of
each
SELECT street, count(*)
FROM evanston311
GROUP BY street
ORDER BY count(*) DESC
LIMIT 5;
```

14. Trimming:
(Some of the street values in evanston311 include house numbers with # or / in them. In addition, some street values end with a .)

```
SELECT distinct street,
 -- Trim off unwanted characters from street
      trim(street, '0123456789 #/.') AS cleaned_street
 FROM evanston311
```

```
ORDER BY cleaned_street;
```

15. Count rows where the description includes 'trash' or 'garbage' but the category does not.

```
-- Count rows
SELECT count(*)
 FROM evanston311
-- description contains trash or garbage (any case)
WHERE (description ILIKE '%trash%'
   OR description ILIKE '%garbage%')
-- category does not contain Trash or Garbage
  AND category NOT LIKE '%Trash%'
  AND category NOT LIKE '%Garbage%';
```

16. Find the most common categories for rows with a description about trash that don't have a trash-related category.

```
-- Count rows with each category
SELECT category, count(*)
 FROM evanston311
WHERE (description ILIKE '%trash%'
   OR description ILIKE '%garbage%')
  AND category NOT LIKE '%Trash%'
  AND category NOT LIKE '%Garbage%'
-- What are you counting?
GROUP BY category
--- order by most frequent values
ORDER BY count DESC
LIMIT 10;
```

17. Split strings on a delimiter

```
-- Select the first word of the street value
SELECT split_part(street, ' ', 1) AS street_name,
       count(*)
 FROM evanston311
GROUP BY street_name
```

```
ORDER BY count DESC
LIMIT 20;
```

18. Shorten long strings:

```
-- Select the first 50 chars when length is greater than 50
SELECT CASE WHEN length(description) > 50
           THEN left(description, 50) || '...'
      -- otherwise just select description
      ELSE description
      END
 FROM evanston311
-- limit to descriptions that start with the word I
WHERE description LIKE 'I %'
ORDER BY description;
```

19. Group and recode values:

```
-- Create a temporary table and some additional cleanup
DROP TABLE IF EXISTS recode;

CREATE TEMP TABLE recode AS
 SELECT DISTINCT category,
        rtrim(split_part(category, '-', 1)) AS standardized
   FROM evanston311;

-- Update to group trash cart values
UPDATE recode
  SET standardized='Trash Cart'
WHERE standardized LIKE 'Trash%Cart';

-- Update to group snow removal values
UPDATE recode
  SET standardized='Snow Removal'
WHERE standardized LIKE 'Snow%Removal%';

-- Examine effect of updates
SELECT DISTINCT standardized
 FROM recode
```

```
WHERE standardized LIKE 'Trash%Cart'
   OR standardized LIKE 'Snow%Removal%';
```

20. Create a table with indicator variables

```
-- To clear table if it already exists
DROP TABLE IF EXISTS indicators;

-- Create the temp table
CREATE TEMP TABLE indicators AS
 SELECT id,
        CAST (description LIKE '%@%' AS integer) AS email,
        CAST (description LIKE '%___-___-____%' AS integer)
AS phone
   FROM evanston311;
 -- Select the column you'll group by
SELECT priority,
      -- Compute the proportion of rows with each indicator
      sum(email)/count(*)::numeric AS email_prop,
      sum(phone)/count(*)::numeric AS phone_prop
 -- Tables to select from
 FROM evanston311
      LEFT JOIN indicators
      -- Joining condition
      ON evanston311.id=indicators.id
-- What are you grouping by?
GROUP BY priority;
```

21. Date/time types and formats

```
-- Count requests created on January 31, 2017
SELECT count(*)
 FROM evanston311
   WHERE date_created :: date = '2017-01-31';
```

```
-- Count requests created on February 29, 2016
SELECT count(*)
 FROM evanston311
WHERE date_created >= '2016-02-29'
```

```
  AND date_created < '2016-03-01';
```

```
-- Count requests created on March 13, 2017
SELECT count(*)
 FROM evanston311
WHERE date_created >= '2017-03-13'
  AND date_created < '2017-03-13'::date + 1;
```

```
-- Subtract the min date_created from the max
SELECT max(date_created) - min(date_created), count(*)
 FROM evanston311;
```

```
-- Add 100 days to the current timestamp
SELECT now() + '100 days' :: interval;
```

22. Completion time by category:

```
-- Select the category and the average completion time by
category
SELECT category,
      AVG(date_completed - date_created) AS completion_time
FROM evanston311
GROUP BY category
-- Order the results
ORDER BY completion_time DESC;
```

23. How many requests are created in each of the 12 months during 2016-2017?

```
-- Extract the month from date_created and count requests
SELECT date_part('month', date_created) AS month,
      count(*)
 FROM evanston311
-- Limit the date range
WHERE date_created >= '2016-01-01'
  AND date_created < '2018-01-01'
-- Group by what to get monthly counts?
GROUP BY month
ORDER BY month DESC;
```

24. What is the most common hour of the day for requests to be created?

```
-- Get the hour and count requests
SELECT date_part('hour', date_created) AS hour,
       count(*)
 FROM evanston311
GROUP BY hour
-- Order results to select most common
ORDER BY count(*) DESC
LIMIT 1;
```

25. During what hours are requests usually completed?

```
-- Count requests completed by hour
SELECT date_part('hour', date_completed) AS hour,
       count(*)
 FROM evanston311
GROUP BY hour
ORDER BY count(*) DESC;
```

26. Variation by day of week:

```
-- Select name of the day of the week the request was
created
SELECT to_char(date_created, 'day') AS day,
       -- Select avg time between request creation and
completion
       AVG(date_completed - date_created) AS duration
 FROM evanston311
-- Group by the name of the day of the week and
-- integer value of day of week the request was created
GROUP BY day, EXTRACT(DOW FROM date_created)
-- Order by integer value of the day of the week
-- the request was created
ORDER BY EXTRACT(DOW FROM date_created);
```

27. Subquery to count the number of requests created per day:

```
-- Aggregate daily counts by month
SELECT date_trunc('month', day) AS month,
      avg(count)
 -- Subquery to compute daily counts
 FROM (SELECT date_trunc('day', date_created) AS day,
              count(*) AS count
         FROM evanston311
        GROUP BY day) AS daily_count
GROUP BY month
ORDER BY month;
```

28. Custom Aggregation periods:

```
-- Bins from Step 1
WITH bins AS (
     SELECT generate_series('2016-01-01',
                            '2018-01-01',
                            '6 months'::interval) AS lower,
            generate_series('2016-07-01',
                            '2018-07-01',
                            '6 months'::interval) AS upper),
-- Daily counts from Step 2
    daily_counts AS (
    SELECT day, count(date_created) AS count
      FROM (SELECT generate_series('2016-01-01',
                                   '2018-06-30',
                                   '1 day'::interval)::date
AS day) AS daily_series
           LEFT JOIN evanston311
           ON day = date_created::date
     GROUP BY day)
-- Select bin bounds
SELECT lower,
      upper,
      -- Compute median of count for each bin
```

```
      percentile_disc(0.5) WITHIN GROUP (ORDER BY count) AS
median
 -- Join bins and daily_counts
 FROM bins
      LEFT JOIN daily_counts
      -- Where the day is between the bin bounds
      ON day >= lower
         AND day < upper
-- Group by bin bounds
GROUP BY lower, upper
ORDER BY lower;
```

29. Monthly average with missing dates:

```
-- generate series with all days from 2016-01-01 to
2018-06-30
WITH all_days AS
    (SELECT generate_series('2016-01-01',
                            '2018-06-30',
                            '1 day'::interval) AS date),
    -- Subquery to compute daily counts
    daily_count AS
    (SELECT date_trunc('day', date_created) AS day,
            count(*) AS count
       FROM evanston311
      GROUP BY day)
-- Aggregate daily counts by month using date_trunc
SELECT date_trunc('month', date) AS month,
      -- Use coalesce to replace NULL count values with 0
      avg(coalesce(count, 0)) AS average
 FROM all_days
      LEFT JOIN daily_count
      -- Joining condition
      ON all_days.date=daily_count.day
GROUP BY month
ORDER BY month;
```

30. Time Gaps:

```
-- Compute the gaps
WITH request_gaps AS (
       SELECT date_created,
              -- lead or lag
              lag(date_created) OVER (ORDER BY
date_created) AS previous,
              -- compute gap as date_created minus lead or
lag
              date_created - lag(date_created) OVER (ORDER
BY date_created) AS gap
         FROM evanston311)
-- Select the row with the maximum gap
SELECT *
 FROM request_gaps
-- Subquery to select maximum gap from request_gaps
WHERE gap = (SELECT max(gap)
               FROM request_gaps);
```

31. Examine the distribution of request completion times by number of days:

```
-- Truncate the time to complete requests to the day
SELECT date_trunc('day', date_completed - date_created) AS
completion_time,
-- Count requests with each truncated time
      count(*)
 FROM evanston311
-- Where category is rats
WHERE category = 'Rodents- Rats'
-- Group and order by the variable of interest
GROUP BY completion_time
ORDER BY completion_time;
```

32. Compute average completion time per category excluding the longest 5% of requests (outliers).

```
SELECT category,
      -- Compute average completion time per category
```

```
      avg(date_completed - date_created) AS
avg_completion_time
 FROM evanston311
-- Where completion time is less than the 95th percentile
value
WHERE date_completed - date_created <
-- Compute the 95th percentile of completion time in a
subquery
        (SELECT percentile_disc(0.95) WITHIN GROUP (ORDER
BY date_completed - date_created)
           FROM evanston311)
GROUP BY category
-- Order the results
ORDER BY avg_completion_time DESC;
```

33. Get corr() between avg. completion time and monthly requests. EXTRACT(epoch FROM interval) returns seconds in interval.

```
-- Compute correlation (corr) between
-- avg_completion time and count from the subquery
SELECT corr(avg_completion, count)
 -- Convert date_created to its month with date_trunc
 FROM (SELECT date_trunc('month', date_created) AS month,
              -- Compute average completion time in number
of seconds
              avg(EXTRACT(epoch FROM date_completed -
date_created)) AS avg_completion,
              -- Count requests per month
              count(*) AS count
         FROM evanston311
        -- Limit to rodents
        WHERE category='Rodents- Rats'
        -- Group by month, created above
        GROUP BY month)
        -- Required alias for subquery
        AS monthly_avgs;
```

34. Select the number of requests created and number of requests completed per month.

```sql
-- Compute monthly counts of requests created
WITH created AS (
      SELECT date_trunc('month', date_created) AS month,
             count(*) AS created_count
        FROM evanston311
       WHERE category='Rodents- Rats'
       GROUP BY month),
-- Compute monthly counts of requests completed
     completed AS (
      SELECT date_trunc('month', date_completed) AS month,
             count(*) AS completed_count
        FROM evanston311
       WHERE category='Rodents- Rats'
       GROUP BY month)
-- Join monthly created and completed counts
SELECT created.month,
      created_count,
      completed_count
 FROM created
      INNER JOIN completed
      ON created.month=completed.month
ORDER BY created.month;
```