

# Numeric Data Types and Summary Functions

EXPLORATORY DATA ANALYSIS IN SQL



Christina Maimone  
Data Scientist

# Numeric types: integer

Name	Storage Size	Description	Range
<code>integer</code> <i>or</i> <code>int</code> <i>or</i> <code>int4</code>	4 bytes	typical choice	-2147483648 to +2147483647

# Numeric types: integer

Name	Storage Size	Description	Range
<code>integer</code> <i>or</i> <code>int</code> <i>or</i> <code>int4</code>	4 bytes	typical choice	-2147483648 to +2147483647
<code>smallint</code> <i>or</i> <code>int2</code>	2 bytes	small-range	-32768 to +32767
<code>bigint</code> <i>or</i> <code>int8</code>	8 bytes	large-range	-9223372036854775808 to +9223372036854775807

# Numeric types: integer

Name	Storage Size	Description	Range
<code>integer</code> <i>or</i> <code>int</code> <i>or</i> <code>int4</code>	4 bytes	typical choice	-2147483648 to +2147483647
<code>smallint</code> <i>or</i> <code>int2</code>	2 bytes	small-range	-32768 to +32767
<code>bigint</code> <i>or</i> <code>int8</code>	8 bytes	large-range	-9223372036854775808 to +9223372036854775807
<code>serial</code>	4 bytes	auto-increment	1 to 2147483647
<code>smallserial</code>	2 bytes	small auto-increment	1 to 32767
<code>bigserial</code>	8 bytes	large auto-increment	1 to 9223372036854775807

# Numeric types: decimal

Name	Storage Size	Description	Range
<code>decimal</code> <i>or</i> <code>numeric</code>	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point

# Numeric types: decimal

Name	Storage Size	Description	Range
<code>decimal</code> <i>or</i> <code>numeric</code>	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
<code>real</code>	4 bytes	variable-precision, inexact	6 decimal digits precision
<code>double precision</code>	8 bytes	variable-precision, inexact	15 decimal digits precision

# Division

```
-- integer division  
SELECT 10/4;
```

2

```
-- numeric division  
SELECT 10/4.0;
```

2.5000000000

# Range: min and max

```
SELECT min(question_pct)
FROM stackoverflow;
```

```
min
----
0
(1 row)
```

```
SELECT max(question_pct)
FROM stackoverflow;
```

```
max
-----
0.071957428
(1 row)
```



# Average or mean

```
SELECT avg(question_pct)
FROM stackoverflow;
```

```
      avg
-----
0.00379494620059319
(1 row)
```

# Variance

## Population Variance

```
SELECT var_pop(question_pct)
FROM stackoverflow;
```

```
var_pop
-----
0.000140268640974167
(1 row)
```

## Sample Variance

```
SELECT var_samp(question_pct)
FROM stackoverflow;
```

```
var_samp
-----
0.000140271571051059
(1 row)
```

```
SELECT variance(question_pct)
FROM stackoverflow;
```

```
variance
-----
0.000140271571051059
(1 row)
```

# Standard deviation

## Sample Standard Deviation

```
SELECT stddev_samp(question_pct)
FROM stackoverflow;
```

```
stddev_samp
-----
0.0118436299778007
(1 row)
```

```
SELECT stddev(question_pct)
FROM stackoverflow;
```

```
stddev
-----
0.0118436299778007
(1 row)
```

## Population Standard Deviation

```
SELECT stddev_pop(question_pct)
FROM stackoverflow;
```

```
stddev_pop
-----
0.0118435062787237
(1 row)
```

# Round

```
SELECT round(42.1256, 2);
```

```
42.13
```

# Summarize by group

```
-- Summarize by group with GROUP BY
SELECT tag,
       min(question_pct),
       avg(question_pct),
       max(question_pct)
FROM   stackoverflow
GROUP BY tag;
```

tag	min	avg	max
amazon-sqs	6.91e-05	8.08328877005347e-05	9.6e-05
amazon-kinesis	2.1e-05	3.3924064171123e-05	4.64e-05
android-pay	2.97e-05	3.16712477396022e-05	3.29e-05
amazon-cloudformation	4.8e-05	9.34518997326204e-05	0.00015246
citrix	3.6e-05	3.95804407713499e-05	4.39e-05
amazon-ec2	0.001058039	0.00122817236730946	0.001378872
actionsript	0.000551486	0.00067589990909091	0.000856132
amazon-ecs	1.17e-05	3.40544117647059e-05	6.51e-05
mongodb	0.0049625	0.00577465885069125	0.00631164
amazon-redshift	0.000117294	0.000160832181818182	0.000212208
...			

# Let's work with numbers!

EXPLORATORY DATA ANALYSIS IN SQL

# Exploring distributions

EXPLORATORY DATA ANALYSIS IN SQL



**Christina Maimone**  
Data Scientist

# Count values

```
SELECT unanswered_count, count(*)  
  FROM stackoverflow  
 WHERE tag='amazon-ebs'  
 GROUP BY unanswered_count  
 ORDER BY unanswered_count;
```

unanswered_count	count
37	12
38	40
...	
43	10
44	8
45	17
46	4
47	1
...	
54	131
55	34
56	1

(20 rows)



# Truncate

```
SELECT trunc(42.1256, 2);
```

42.12

```
SELECT trunc(12345, -3);
```

12000

# Truncating and grouping

```
SELECT trunc(unanswered_count, -1) AS trunc_uo,  
       count(*)  
FROM stackoverflow  
WHERE tag='amazon-ebs'  
GROUP BY trunc_uo  -- column alias  
ORDER BY trunc_uo; -- column alias
```

trunc_uo	count
30	74
40	194
50	480

(3 rows)

# Generate series

```
SELECT generate_series(start, end, step);
```

# Generate series

```
SELECT generate_series(1, 10, 2);
```

```
generate_series
-----
          1
          3
          5
          7
          9
(5 rows)
```

```
SELECT generate_series(0, 1, .1);
```

```
generate_series
-----
          0
         0.1
         0.2
         0.3
         0.4
         0.5
         0.6
         0.7
         0.8
         0.9
        1.0
(11 rows)
```

# Create bins: output

lower	upper	count
30	35	0
35	40	74
40	45	155
45	50	39
50	55	445
55	60	35
60	65	0

(7 rows)

# Create bins: query

```
-- Create bins
WITH bins AS (
  SELECT generate_series(30,60,5) AS lower,
         generate_series(35,65,5) AS upper),

;
```

# Create bins: query

```
-- Create bins
WITH bins AS (
  SELECT generate_series(30,60,5) AS lower,
         generate_series(35,65,5) AS upper),
-- Subset data to tag of interest
ebs AS (
  SELECT unanswered_count
  FROM stackoverflow
  WHERE tag='amazon-ebs')

;
```

# Create bins: query

```
-- Create bins
WITH bins AS (
  SELECT generate_series(30,60,5) AS lower,
         generate_series(35,65,5) AS upper),
-- Subset data to tag of interest
ebs AS (
  SELECT unanswered_count
  FROM stackoverflow
  WHERE tag='amazon-ebs')
-- Count values in each bin
SELECT lower, upper, count(unanswered_count)
-- left join keeps all bins
FROM bins
  LEFT JOIN ebs
    ON unanswered_count >= lower
   AND unanswered_count < upper

;
```



# Create bins: query

```
-- Create bins
WITH bins AS (
    SELECT generate_series(30,60,5) AS lower,
           generate_series(35,65,5) AS upper),
-- Subset data to tag of interest
ebs AS (
    SELECT unanswered_count
    FROM stackoverflow
    WHERE tag='amazon-ebs')
-- Count values in each bin
SELECT lower, upper, count(unanswered_count)
-- left join keeps all bins
FROM bins
    LEFT JOIN ebs
        ON unanswered_count >= lower
        AND unanswered_count < upper
-- Group by bin bounds to create the groups
GROUP BY lower, upper
ORDER BY lower;
```

# Create bins: output

lower	upper	count
30	35	0
35	40	74
40	45	155
45	50	39
50	55	445
55	60	35
60	65	0

(7 rows)

# Time to explore some distributions!

EXPLORATORY DATA ANALYSIS IN SQL

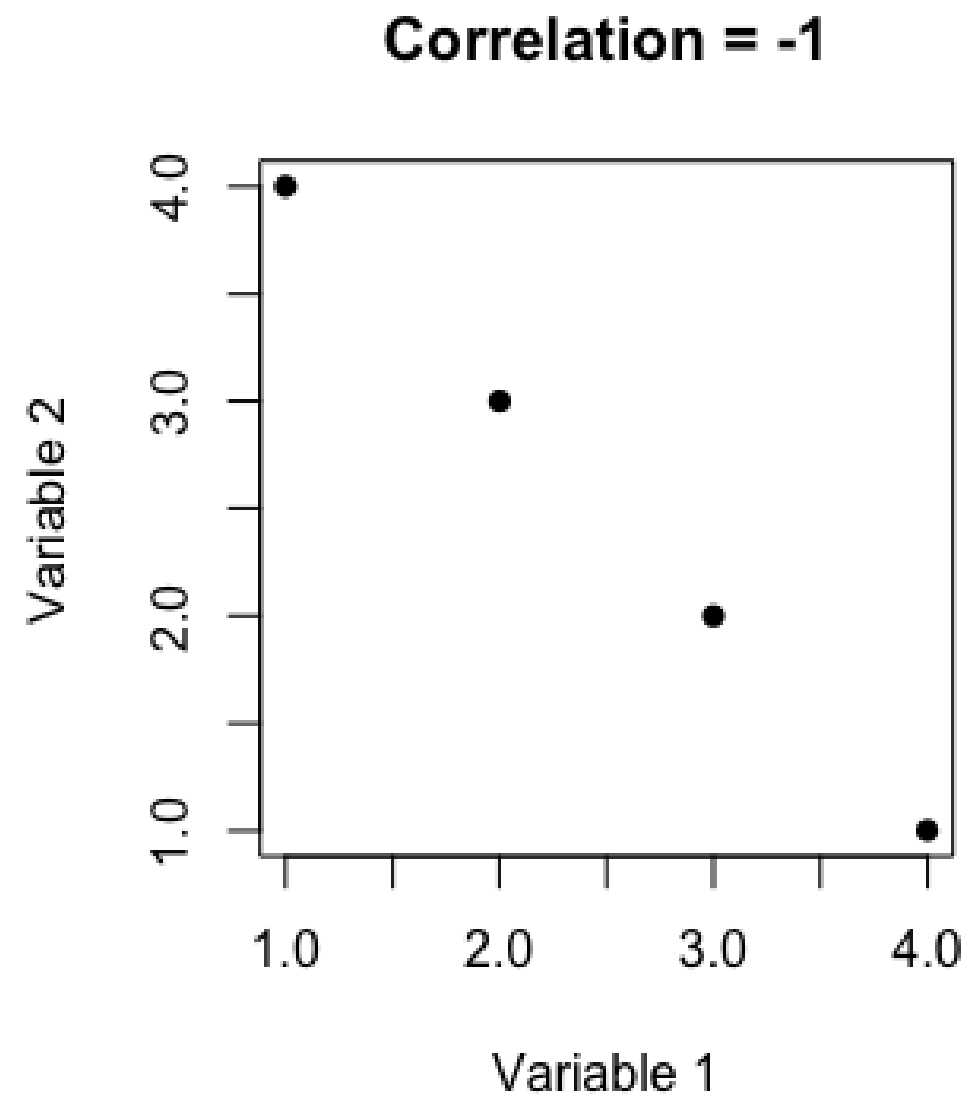
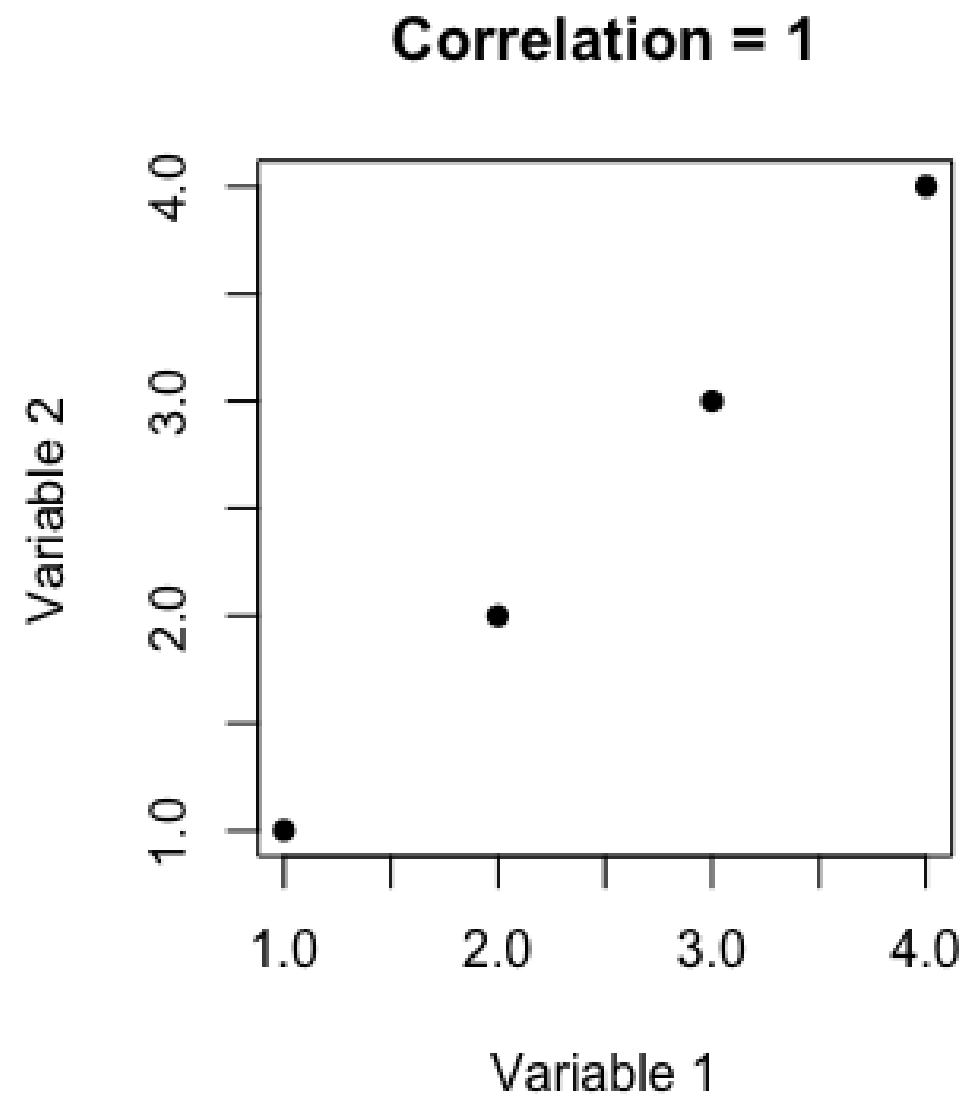
# More Summary Functions

EXPLORATORY DATA ANALYSIS IN SQL



**Christina Maimone**  
Data Scientist

# Correlation



# Correlation function

```
SELECT corr(assets, equity)
FROM fortune500;
```

corr

-----  
0.637710143588615

(1 row)

# Median

```
1 1 4 4 4 5 6 7 13 19 20 20 21 21 22
      ^
    median
  50th percentile

^                               ^
0th percentile      100th percentile
```

# Percentile functions

```
SELECT percentile_disc(percentile) WITHIN GROUP (ORDER BY column_name)
FROM table;

-- percentile between 0 and 1
```

- Returns a value from column

```
SELECT percentile_cont(percentile) WITHIN GROUP (ORDER BY column_name)
FROM table;
```

- Interpolates between values



# Percentile examples

```
SELECT val
FROM nums;
```

```
val
----
1
3
4
5
(4 rows)
```

```
SELECT percentile_disc(.5) WITHIN GROUP (ORDER BY val),
       percentile_cont(.5) WITHIN GROUP (ORDER BY val)
FROM nums;
```

```
percentile_disc | percentile_cont
-----+-----
3 | 3.5
```

# Common issues

- Error codes
  - Examples: 9, 99, -99
- Missing value codes
  - NA, NaN, N/A, #N/A
  - 0 = missing or 0?
- Outlier (extreme) values
  - Really high or low?
  - Negative values?
- Not really a number
  - Examples: zip codes, survey response categories

# Let's practice!

EXPLORATORY DATA ANALYSIS IN SQL

# Creating Temporary Tables

EXPLORATORY DATA ANALYSIS IN SQL



**Christina Maimone**  
Data Scientist

# Syntax

## Create Temp Table Syntax

```
-- Create table as  
CREATE TEMP TABLE new_tablename AS  
-- Query results to store in the table  
SELECT column1, column2  
FROM table;
```

## Select Into Syntax

```
-- Select existing columns  
SELECT column1, column2  
-- Clause to direct results to a new temp table  
INTO TEMP TABLE new_tablename  
-- Existing table with existing columns  
FROM table;
```

# Create a table

```
CREATE TEMP TABLE top_companies AS
SELECT rank,
       title
FROM fortune500
WHERE rank <= 10;
```

```
SELECT *
FROM top_companies;
```

```
rank | title
-----+-----
1 | Walmart
2 | Berkshire Hathaway
3 | Apple
4 | Exxon Mobil
5 | McKesson
6 | UnitedHealth Group
7 | CVS Health
8 | General Motors
9 | AT&T
10 | Ford Motor
(10 rows)
```

# Insert into table

```
INSERT INTO top_companies
SELECT rank, title
FROM fortune500
WHERE rank BETWEEN 11 AND 20;
```

```
SELECT * FROM top_companies;
```

```
rank | title
-----+-----
1 | Walmart
2 | Berkshire Hathaway
3 | Apple
...
9 | AT&T
10 | Ford Motor
11 | AmerisourceBergen
12 | Amazon.com
13 | General Electric
14 | Verizon
15 | Cardinal Health
16 | Costco
17 | Walgreens Boots Alliance
18 | Kroger
19 | Chevron
20 | Fannie Mae
(20 rows)
```

# Delete (drop) table

```
DROP TABLE top_companies;
```

```
DROP TABLE IF EXISTS top_companies;
```



# Time to create some tables!

EXPLORATORY DATA ANALYSIS IN SQL