INTRODUCTION TO SQL

Baruch AIS – Data Science

AGENDA

1. Theory
2. Basics
3. Special Topics
4. Kahoot!

PREPARATION

1. For this workshop, we will be using **PostgreSQL 15**, please follow instructions on this page to install the software to your machine: https://www.datacamp.com/tutorial/installing-postgresql-windows-macosx

2. Additionally, we will run basic queries on a **dataset of fortune 500** companies provided by DataCamp – you can download it using this link: https://www.dropbox.com/sh/f5zt3pmxom53xq3/AADv81MiFx4vGoe8cpjliKXWa?dl=0

THEORY

Let’s get started with introductory concepts!

DATABASE MANAGEMENT SYSTEMS

A database management system (DBMS) is a software package for creating and managing databases. Zhao A. (2021) provides a comparison of different DBMS:

RDBMS	Owner	Highlights
Microsoft SQL Server	Microsoft	- Popular proprietary RDBMS - Often used alongside other Microsoft products including Microsoft Azure and the .NET framework - Common on the Windows platform - Also referred to as *MSSQL* or *SQL Server*
MySQL	Open Source	- Popular open source RDBMS - Often used alongside web development languages like HTML/CSS/Javascript - Acquired by Oracle, though still open source
Oracle Database	Oracle	- Popular proprietary RDBMS - Often used at large corporations given the amount of features, tools, and support available - Also referred to as simply as *Oracle*
PostgreSQL	Open Source	- Quickly growing in popularity - Often used alongside open source technologies like Docker and Kubernetes - Efficient and great for large datasets
SQLite	Open Source	- World's most used database engine - Common on iOS and Android platforms - Lightweight and great for a small database

STRUCTURED QUERY LANGUAGE

SQL stands for Structured Query Language and is used to communicate with relational databases to store, manipulate, and retrieve data

- Still widely used due to its convenience
- Example: PostgreSQL – a light-weight and open-source RDBMS

SQL databases are relational databases and require predefined schema (structure of tables)

- **Data Definition Language** (**DDL**) defines the schema of the database
- i.e., the characteristics of each record, the fields' datatype and length, the relationship among records

Data Manipulation Language (**DML**) provides the data manipulation techniques like selection, insertion, deletion, updating, modification, replacement, retrieval, sorting and display of records

DATA TYPES

To import our dataset to Postgres, we first need to create an empty table, name its columns and define each field's datatype. Tanimura C. (2021) provides an overview of different data types:

Type	Name	Description
String	CHAR / VARCHAR	Holds strings. A CHAR is always of fixed length, whereas a VARCHAR is of variable length, up to some maximum size (256 characters, for example).
	TEXT / BLOB	Holds longer strings that don't fit in a VARCHAR. Descriptions or free text entered by survey respondents might be held in these fields.
Numeric	INT / SMALLINT / BIGINT	Holds integers (whole numbers). Some databases have SMALLINT and/or BIGINT. SMALLINT can be used when the field will only hold values with a small number of digits. SMALLINT takes less memory than a regular INT. BIGINT is capable of holding numbers with more digits than an INT, but it takes up more space than an INT.
	FLOAT / DOUBLE / DECIMAL	Holds decimal numbers, sometimes with the number of decimal places specified.
Logical	BOOLEAN	Holds values of TRUE or FALSE.
	DATETIME / TIMESTAMP	Holds dates with times. Typically in a YYYY-MM-DD hh:mi:ss format, where YYYY is the four-digit year, MM is the two-digit month number, DD is the two-digit day, hh is the two-digit hour (usually 24-hour time, or values of 0 to 23), mi is the two-digit minutes, and ss is the two-digit seconds. Some databases store only timestamps without time zone, while others have specific types for timestamps with and without time zones.
	TIME	Holds times.

QUERY OPTIMIZATION

When you are executing a SQL command , the RDBMS determines the best way to carry out your request and the SQL engine figures out how to interpret the task.

- Metrics used to determine query efficiency: CPU cycles and number of I/O operations

An execution plan may vary in:

- Order of operations
- Algorithms used for table joins and other operations (e.g., nested loops, hash join)
- Data retrieval methods (e.g., indexes usage, full scan)

In this simple query, execution logic may look like this:

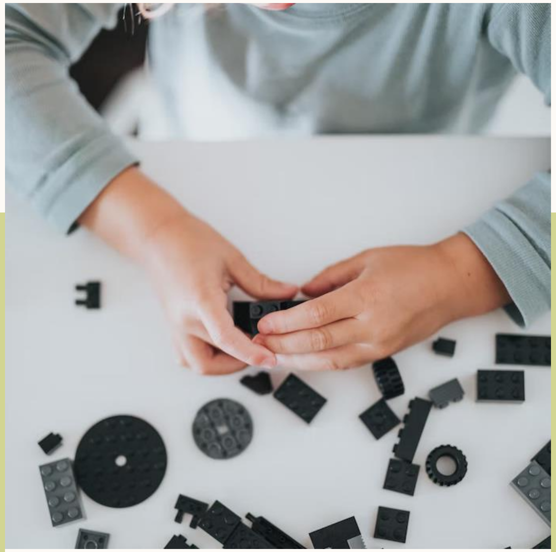
1. Identify tables to pull data from
2. Filter rows based on specified conditions
3. Display specified column(s)

```sql
SELECT title, ticker, sector, revenues
FROM fortune500
WHERE sector = 'Financials';
```

BASICS

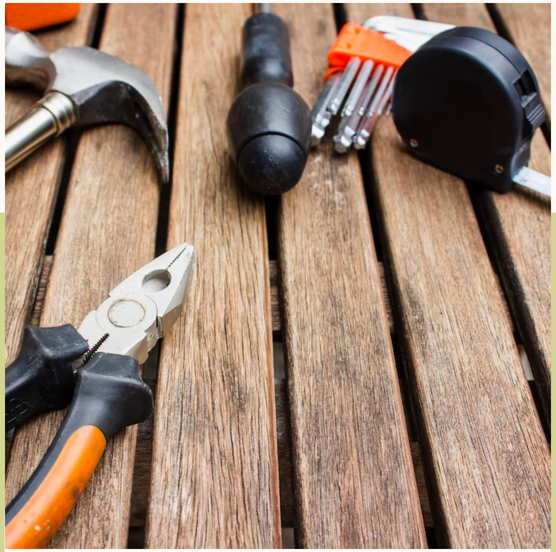
Let’s dive into the SQL syntax!

C.R.U.D. OPERATIONS



CREATE

READ



UPDATE

DELETE

GETTING STARTED WITH POSTGRES

Let's first connect to a server and define the table structure for the fortune500 dataset

Here is an example of creating a table schema before importing a CSV file into the table

Note: highlighted words are key words!

```
CREATE TABLE fortune500(rank_position INT NOT NULL,
                        title VARCHAR PRIMARY KEY,
                        company_name VARCHAR NOT NULL,
                        ticker VARCHAR(5),
                        url VARCHAR,
                        hq VARCHAR,
                        sector VARCHAR NOT NULL,
                        industry VARCHAR,
                        employees INT,
                        revenues INT,
                        revenues_change REAL,
                        profits NUMERIC,
                        profits_change REAL,
                        assets NUMERIC,
                        equity NUMERIC
                       );
```

Next, we import our dataset to fill the table:

Right-click our table -> import data

Let's print out the first 5 rows to test, if the data has loaded successfully

```
SELECT * FROM fortune500 LIMIT 5;
```

LOGICAL OPERATORS AND CLAUSES

Logical Operators	Comparison Operators (Symbols)	Comparison Operators (Keywords)	Math Operators
AND	=	BETWEEN	+
OR	!=, <>	EXISTS	-
NOT	<	IN	*
	<=	IS NULL	/
	>	LIKE	%
	V=		

```
SELECT       -- columns to display
FROM         -- table(s) to pull from
WHERE        -- filter rows
GROUP BY     -- split rows into groups
HAVING       -- filter grouped rows
ORDER BY     -- columns to sort
```

RETRIEVE DATA

```
SELECT * FROM fortune500;
```

- From fortune500 table, retrieve all rows

```
SELECT title, ticker, hq, sector, revenues, profits
FROM fortune500;
```

- From fortune500, retrieve all rows from fields: title, ticker, hq, sector, revenues, and profits

```
SELECT title, revenues, sector FROM fortune500
WHERE sector = 'Technology';
```

- From fortune500, find rows that meet condition: sector = Technology, retrieve title and revenue columns for filtered data

```
SELECT title, ticker, industry, FROM fortune500
WHERE revenues<=10000 OR profits<0;
```

- From fortune500, find observations that have negative profits or revenues no more than 10,000 Million USD. Retrieve the title, ticker and industry of matching companies

```
SELECT title, profits/revenues AS gross_profit_margin
FROM fortune500 WHERE profits/revenues>0.1;
```

- From fortune500, find rows that meet condition: profits/revenues>0.1, retrieve title and profits/revenues as gross_profit_margin columns

CASTING

You can change the displayed datatype of a column using the CAST() function. Let's use the last query as an example.

- Real is a variable precision decimal
- The following query uses a previous query as a subquery inside the FROM statement

```
SELECT title, CAST(gross_profit_margin AS REAL)
FROM
    (SELECT title, profits/revenues AS gross_profit_margin
     FROM fortune500 WHERE profits/revenues>0.1)
AS subquery;
```

Casting is also possible with ::

```
SELECT title, gross_profit_margin::REAL
FROM
    (SELECT title, profits/revenues AS gross_profit_margin
     FROM fortune500 WHERE profits/revenues>0.1)
AS subquery;
```

SUMMARY FUNCTIONS

- Sometimes, you might want to perform computations on existing columns. Postgres has built-in functions for that!

```
SELECT MIN(profits) AS least_profit, MAX(revenues) AS most_revenue, AVG(equity) AS mean_equity,
        SUM(assets) AS fortune500_total_assets, COUNT(ticker) AS number_of_tickers
FROM fortune500;
```

- When we are using aggregate functions, we must add a GROUP BY statement to the query if we additionally select regular columns as well:

get minimum, mean, and maximum profit values for each sector in the fortune500 >

```
SELECT  sector,
        MIN(profits),
        AVG(profits),
        MAX(profits)
FROM fortune500
GROUP BY sector;
```

Data Output Messages Notifications

	sector character varying	min numeric	avg numeric	max numeric
1	Food & Drug Stor...	-502.2	1217.428571	4173
2	Health Care	-1721	2773.260526	16540
3	Business Services	57.2	1155.355000	5991
4	Wholesalers	-199.4	391.2793103	2258
5	Food, Beverages ...	-677	2346.183333	14239

GENERAL-PURPOSE AGGREGATE FUNCTIONS

Function	Argument Type	Return Type	Description
`avg(expression)`	`smallint`, `int`, `bigint`, `real`, `double precision`, `numeric`, or `interval`	`numeric` for any integer type argument, `double precision` for a floating-point argument, otherwise the same as the argument data type	the average (arithmetic mean) of all input values
`bit_and(expression)`	`smallint`, `int`, `bigint`, or `bit`	same as argument data type	the bitwise AND of all non-null input values, or null if none
`bit_or(expression)`	`smallint`, `int`, `bigint`, or `bit`	same as argument data type	the bitwise OR of all non-null input values, or null if none
`bool_and(expression)`	`bool`	`bool`	true if all input values are true, otherwise false
`bool_or(expression)`	`bool`	`bool`	true if at least one input value is true, otherwise false
`count(*)`		`bigint`	number of input rows
`count(expression)`	any	`bigint`	number of input rows for which the value of *expression* is not null
`every(expression)`	`bool`	`bool`	equivalent to `bool_and`
`max(expression)`	any array, numeric, string, or date/time type	same as argument type	maximum value of *expression* across all input values
`min(expression)`	any array, numeric, string, or date/time type	same as argument type	minimum value of *expression* across all input values
`sum(expression)`	`smallint`, `int`, `bigint`, `real`, `double precision`, `numeric`, or `interval`	`bigint` for `smallint` or `int` arguments, `numeric` for `bigint` arguments, `double precision` for floating-point arguments, otherwise the same as the argument data type	sum of *expression* across all input values

https://www.postgresql.org/docs/8.2/functions-aggregate.html

SORT AND SEARCH

- PostgreSQL allows you to count observations, order them by the most frequent values or in alphabetical order – similar to Excel. Let's explore these capabilities!

Determine how many fortune500 companies belong to each sector?

```
SELECT sector, COUNT(*) FROM fortune500
GROUP BY sector ORDER BY COUNT DESC;
```

How many are headquartered in the same location?

```
SELECT hq, COUNT(*) FROM fortune500
GROUP BY hq ORDER BY COUNT DESC;
```

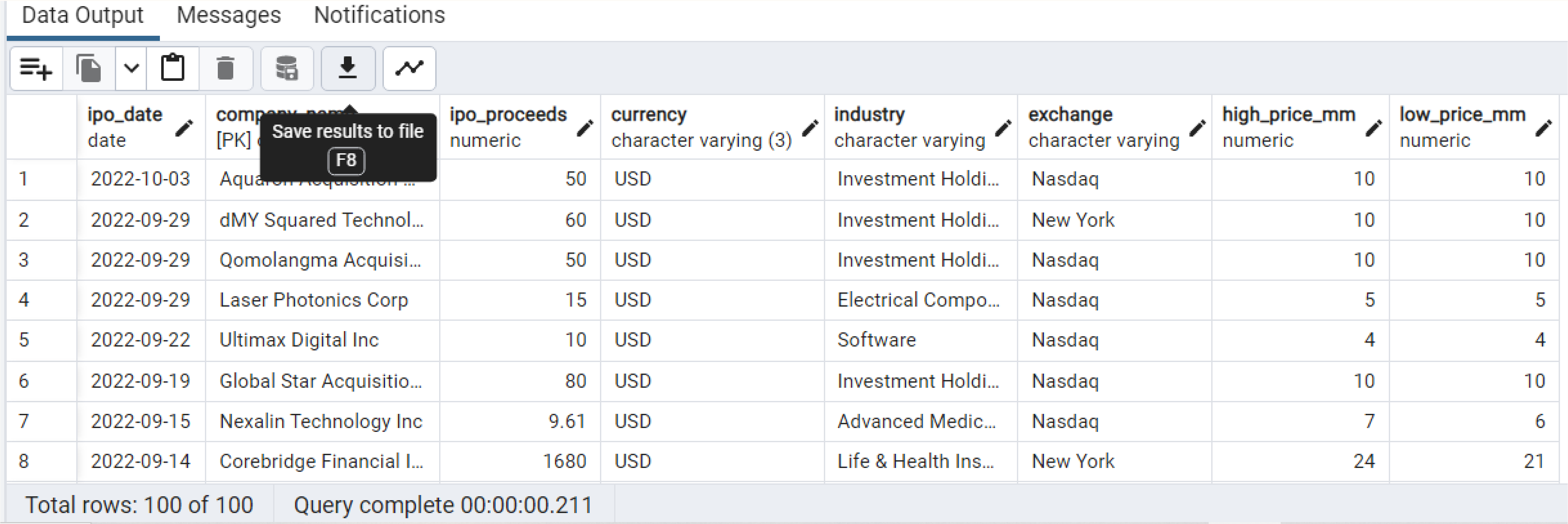
- Case-insensitive search with ILIKE() keyword:

```
SELECT * FROM fortune500 WHERE company_name ILIKE('%bank%');

SELECT * FROM fortune500 WHERE company_name ILIKE('%apple%');
```

EXPORT DATA

At the bottom of the Query Tool, you will find a downward arrow icon.
Click that icon or press F8 to download the output as a CSV file!



	ipo_date date	comp[illegible] [PK] [illegible]	ipo_proceeds numeric	currency character varying (3)	industry character varying	exchange character varying	high_price_mm numeric	low_price_mm numeric
1	2022-10-03	Aqua[illegible]...	50	USD	Investment Holdi...	Nasdaq	10	10
2	2022-09-29	dMY Squared Technol...	60	USD	Investment Holdi...	New York	10	10
3	2022-09-29	Qomolangma Acquisi...	50	USD	Investment Holdi...	Nasdaq	10	10
4	2022-09-29	Laser Photonics Corp	15	USD	Electrical Compo...	Nasdaq	5	5
5	2022-09-22	Ultimax Digital Inc	10	USD	Software	Nasdaq	4	4
6	2022-09-19	Global Star Acquisitio...	80	USD	Investment Holdi...	Nasdaq	10	10
7	2022-09-15	Nexalin Technology Inc	9.61	USD	Advanced Medic...	Nasdaq	7	6
8	2022-09-14	Corebridge Financial I...	1680	USD	Life & Health Ins...	New York	24	21

SPECIAL TOPICS

You are welcome to use queries from this part of the workshop for your personal research and data analysis!

TIMESERIES

IPO_dataset is a timeseries dataset –
Let's write the following queries and discuss the results!

```
--extracting and summarizing by month
SELECT date_part('month', IPO_date) AS IPO_month, SUM(IPO_proceeds) AS total_proceeds
FROM IPO_dataset GROUP BY IPO_month ORDER BY IPO_month;

--truncate to keep larger units: months
SELECT date_trunc('month', IPO_date) AS IPO_month, SUM(IPO_proceeds) AS total_proceeds
FROM IPO_dataset GROUP BY IPO_month ORDER BY IPO_month;

--grouping by fiscal quarter
SELECT date_part('quarter', IPO_date) AS IPO_quarter, SUM(IPO_proceeds) AS total_proceeds
FROM IPO_dataset GROUP BY IPO_quarter ORDER BY IPO_quarter;

--zooming in on a year
SELECT * FROM IPO_dataset WHERE IPO_date BETWEEN '2021-01-01' AND '2021-12-31';
```

SQL JOINS

INNER JOIN	LEFT JOIN	RIGHT JOIN	CROSS JOIN	FULL JOIN
Returns the intersection of two tables. No missing values!	Keeps all the data from the left table, adds some data from the right table	Keeps all the data from the right table, adds some data from the left table	Generates a paired combination of each row of the first table with each row of the second table.	Keeps all data in both tables!

TIME SERIES & JOINS

Let’s do an inner join between two datasets: IPO data and Macro trends

```
--inner join with macro trends
CREATE TEMP TABLE proceeds_each_month AS
    SELECT date_trunc('month', IPO_date) AS IPO_month,
    SUM(IPO_proceeds) AS total_proceeds_MM, COUNT(company_name) AS IPO_num
        FROM IPO_dataset GROUP BY IPO_month ORDER BY IPO_month;

CREATE TABLE macro_trends(  release_month DATE NOT NULL,
                            CPI_pct_change NUMERIC,
                            M2_pct_change NUMERIC,
                            House_price_pct_change NUMERIC,
                            Unemployment_pct_change NUMERIC);

SELECT * FROM proceeds_each_month INNER JOIN macro_trends
ON proceeds_each_month.IPO_month=macro_trends.release_month;
```

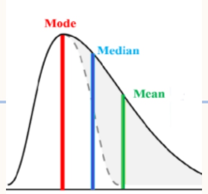
LAG & LEAD COLUMNS

When we have a time series dataset, we can create lag and lead columns to compute percentage changes!

```sql
-- create lag columns of IPO proceeds
SELECT IPO_month, total_proceeds_MM,
LAG(total_proceeds_MM) OVER (ORDER BY IPO_month),
(total_proceeds_MM - LAG(total_proceeds_MM) OVER (ORDER BY IPO_month))/100 AS pct_change
FROM proceeds_each_month ORDER BY IPO_month;
```

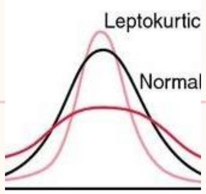
```sql
-- create lead columns of IPO proceeds
SELECT IPO_month, total_proceeds_MM,
LEAD(total_proceeds_MM) OVER (ORDER BY IPO_month),
(total_proceeds_MM - LEAD(total_proceeds_MM) OVER (ORDER BY IPO_month))/100 AS pct_change
FROM proceeds_each_month ORDER BY IPO_month;
```

ANOMALY DETECTION



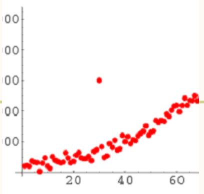
SKEWNESS

- High skewness is usually not a big deal
- May result in an inaccurate model from the data



KURTOSIS

- High kurtosis might indicate an increased risk of getting either extremely high or low returns



EXTREME OUTLIERS

- In the past few years, many events have led to markets behaving abnormally
- might exist as a result of a bad data point

SKEWNESS

The formula for skewness is

$$Skew = \frac{n}{(n-1)*(n-2)} * \sum_{i=1}^{n} \left(\frac{v_i - \mu}{\sigma}\right)^3$$

Dejan Sarka (2017) teaches how to compute skewness and kurtosis for statistical analysis using SQL.

Source: https://learnsql.com/blog/high-performance-statistical-queries-skewness-kurtosis/

```
WITH SkewCTE AS
(SELECT SUM(1.0*total_proceeds_MM) AS rx,
 SUM(POWER(1.0*total_proceeds_MM,2)) AS rx2,
 SUM(POWER(1.0*total_proceeds_MM,3)) AS rx3,
 COUNT(1.0*total_proceeds_MM) AS rn,
 STDDEV_SAMP(1.0*total_proceeds_MM) AS stdv,
 AVG(1.0*total_proceeds_MM) AS av
FROM proceeds_each_month
)
SELECT
    (rx3 - 3*rx2*av + 3*rx*av*av - rn*av*av*av)
    / (stdv*stdv*stdv) * rn / (rn-1) / (rn-2) AS Skewness
FROM SkewCTE;
```

Output Messages Notifications

skewness numeric
2.257766638

KURTOSIS

$$Kurt = \frac{n*(n+1)}{(n-1)*(n-2)*(n-3)} * \sum_{i=1}^{n} \left(\frac{v_i - \mu}{\sigma}\right)^4 - \frac{3*(n-1)^2}{(n-2)*(n-3)}$$

Dejan Sarka (2017) teaches how to compute skewness and kurtosis for statistical analysis using SQL.

Source: https://learnsql.com/blog/high-performance-statistical-queries-skewness-kurtosis/

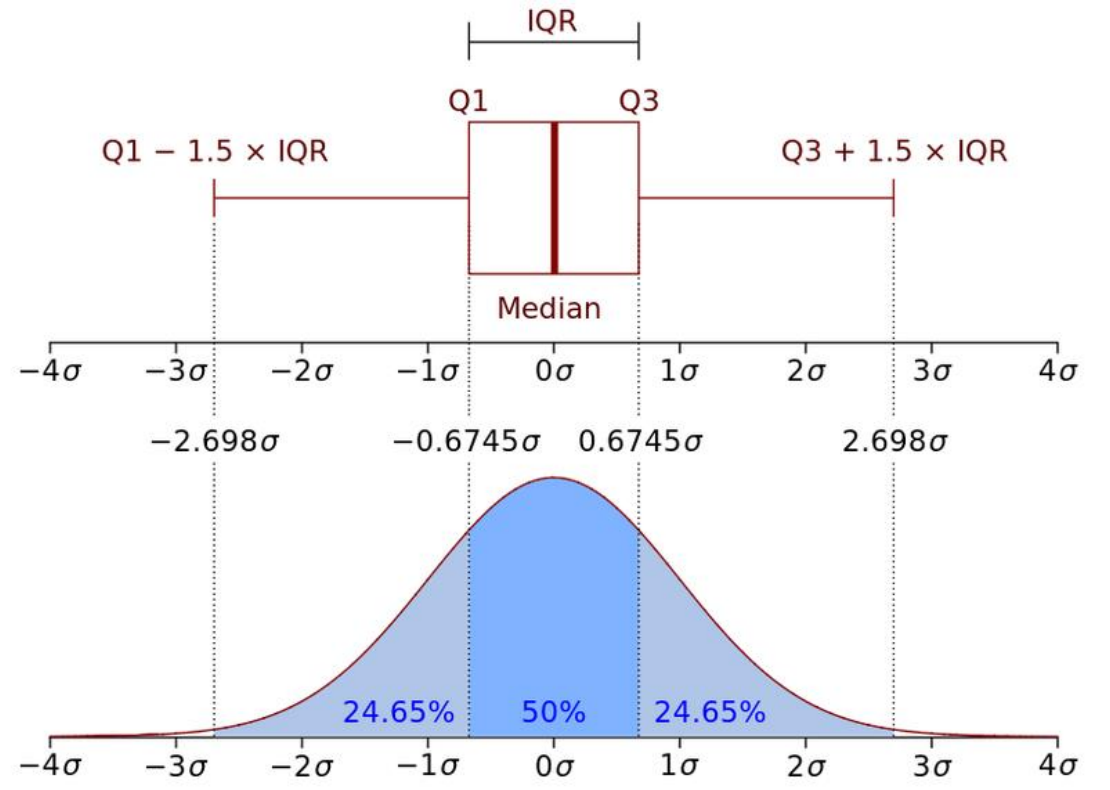
```
WITH KurtCTE AS
(
SELECT SUM(1.0*total_proceeds_MM) AS rx,
 SUM(POWER(1.0*total_proceeds_MM,2)) AS rx2,
 SUM(POWER(1.0*total_proceeds_MM,3)) AS rx3,
 SUM(POWER(1.0*total_proceeds_MM,4)) AS rx4,
 COUNT(1.0*total_proceeds_MM) AS rn,
 STDDEV_SAMP(1.0*total_proceeds_MM) AS stdv,
 AVG(1.*total_proceeds_MM) AS av
FROM proceeds_each_month
)
SELECT
   (rx4 - 4*rx3*av + 6*rx2*av*av - 4*rx*av*av*av + rn*av*av*av*av)
   / (stdv*stdv*stdv*stdv) * rn * (rn+1) / (rn-1) / (rn-2) / (rn-3)
   - 3.0 * (rn-1) * (rn-1) / (rn-2) / (rn-3) AS Kurtosis
FROM KurtCTE;
```

Output Messages Notifications

kurtosis numeric
5.560862626

OUTLIERS

Positive Outlier = 75th Percentile + 1.5 * (75th percentile - 25th percentile)
Negative Outlier = 25th Percentile - 1.5 * (75th percentile - 25th percentile)



Source: Salgado, R. (2019) https://towardsdatascience.com/anomaly-detection-with-sql-7700c7516d1d

FINDING OUTLIERS

To separate the outliers from inliers, let's create bins!

```
WITH iqr_table AS (
SELECT pct_25, pct_75, (pct_75 - pct_25) AS iqr
FROM (SELECT
      percentile_disc(0.25) WITHIN GROUP (ORDER BY total_proceeds_MM) AS pct_25,
      percentile_disc(0.75) WITHIN GROUP (ORDER BY total_proceeds_MM) AS pct_75
      FROM proceeds_each_month) AS iqr_proceeds
)
SELECT IPO_month, total_proceeds_MM, IPO_num,
CASE
    WHEN total_proceeds_MM>=pct_75 + iqr*1.5 THEN 'positive_outlier'
    WHEN total_proceeds_MM<=pct_75 - iqr*1.5 THEN 'negative_outlier'
    ELSE 'inlier'
END AS outlier_type
FROM proceeds_each_month, iqr_table;
```

Output Messages Notifications

ipo_month timestamp with time zone	total_proceeds_mm numeric	ipo_num bigint	outlier_type text
2021-03-01 00:00:00-05	48822.99	132	positive_outlier
2021-04-01 00:00:00-04	14098.58	36	inlier
2021-05-01 00:00:00-04	7864.50	32	inlier

AGGREGATE FUNCTIONS FOR STATISTICS

`stddev(`***`expression`***`)`	**`smallint`**, **`int`**, **`bigint`**, **`real`**, **`double precision`**, or **`numeric`**	**`double precision`** for floating-point arguments, otherwise **`numeric`**	historical alias for `stddev_samp`
`stddev_pop(`***`expression`***`)`	**`smallint`**, **`int`**, **`bigint`**, **`real`**, **`double precision`**, or **`numeric`**	**`double precision`** for floating-point arguments, otherwise **`numeric`**	population standard deviation of the input values
`stddev_samp(`***`expression`***`)`	**`smallint`**, **`int`**, **`bigint`**, **`real`**, **`double precision`**, or **`numeric`**	**`double precision`** for floating-point arguments, otherwise **`numeric`**	sample standard deviation of the input values
`variance(`***`expression`***`)`	**`smallint`**, **`int`**, **`bigint`**, **`real`**, **`double precision`**, or **`numeric`**	**`double precision`** for floating-point arguments, otherwise **`numeric`**	historical alias for `var_samp`
`var_pop(`***`expression`***`)`	**`smallint`**, **`int`**, **`bigint`**, **`real`**, **`double precision`**, or **`numeric`**	**`double precision`** for floating-point arguments, otherwise **`numeric`**	population variance of the input values (square of the population standard deviation)
`var_samp(`***`expression`***`)`	**`smallint`**, **`int`**, **`bigint`**, **`real`**, **`double precision`**, or **`numeric`**	**`double precision`** for floating-point arguments, otherwise **`numeric`**	sample variance of the input values (square of the sample standard deviation)

EVEN MORE FUNCTIONS :]

Function	Argument Type	Return Type	Description
`corr(Y, X)`	`double precision`	`double precision`	correlation coefficient
`covar_pop(Y, X)`	`double precision`	`double precision`	population covariance
`covar_samp(Y, X)`	`double precision`	`double precision`	sample covariance
`regr_avgx(Y, X)`	`double precision`	`double precision`	average of the independent variable (`sum(X)/N`)
`regr_avgy(Y, X)`	`double precision`	`double precision`	average of the dependent variable (`sum(Y)/N`)
`regr_count(Y, X)`	`double precision`	`bigint`	number of input rows in which both expressions are nonnull
`regr_intercept(Y, X)`	`double precision`	`double precision`	y-intercept of the least-squares-fit linear equation determined by the (*X*, *Y*) pairs
`regr_r2(Y, X)`	`double precision`	`double precision`	square of the correlation coefficient
`regr_slope(Y, X)`	`double precision`	`double precision`	slope of the least-squares-fit linear equation determined by the (*X*, *Y*) pairs
`regr_sxx(Y, X)`	`double precision`	`double precision`	`sum(X^2) - sum(X)^2/N` ("sum of squares" of the independent variable)
`regr_sxy(Y, X)`	`double precision`	`double precision`	`sum(X*Y) - sum(X) * sum(Y)/N` ("sum of products" of independent times dependent variable)
`regr_syy(Y, X)`	`double precision`	`double precision`	`sum(Y^2) - sum(Y)^2/N` ("sum of squares" of the dependent variable)

SUMMARY

WE LOVE SQL!

```
SELECT 'I LOVE SQL' AS FEEDBACK;
```

- SQL is a powerful tool – very useful for retrieving data from a database as well as gathering initial insights and preparing data for further visualization and analysis

THANK YOU

If this workshop has sparked your interest in learning more about SQL, I encourage you to take a deeper look into the resources on the next page :)

RECOMMENDED RESOURCES

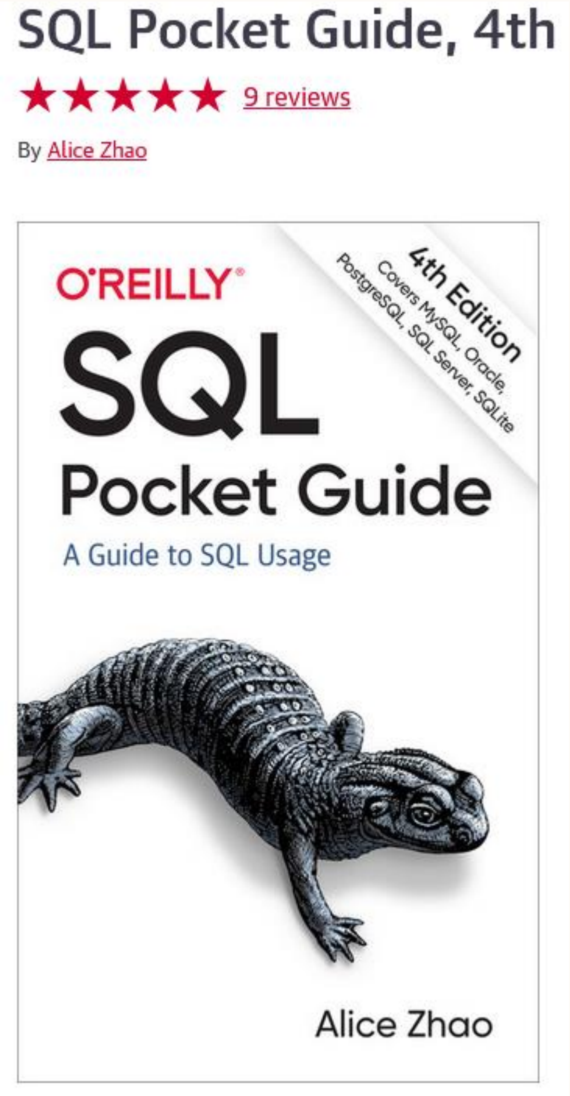
DataCamp:

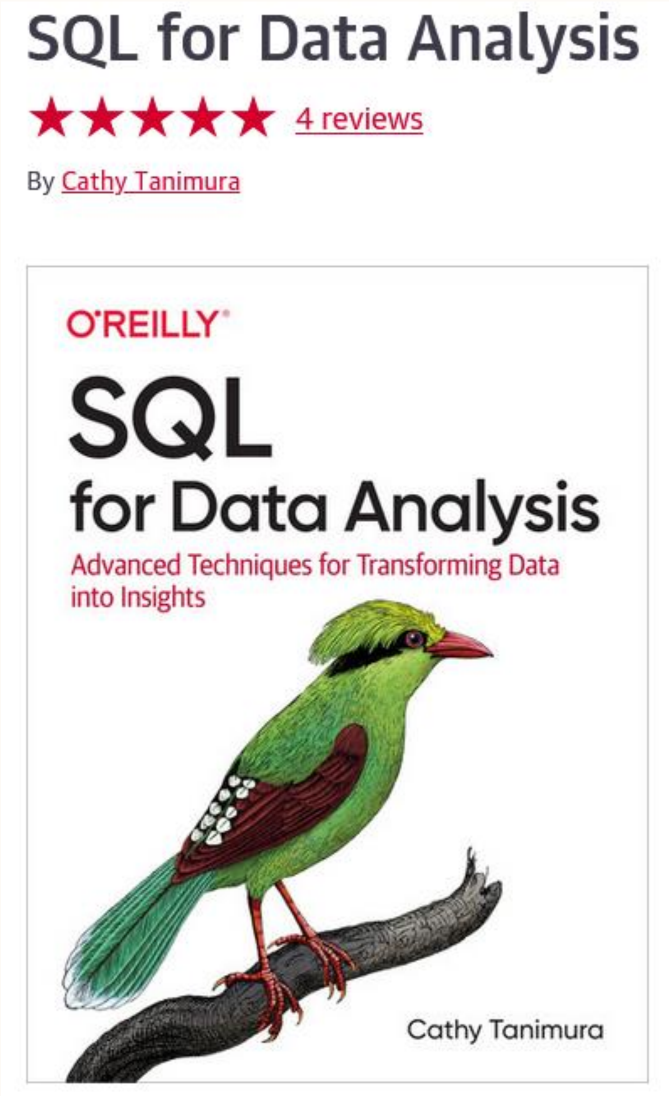
- ✓ Intermediate SQL Queries: https://www.datacamp.com/courses/intermediate-sql-queries
- ✓ Exploratory Data Analysis in SQL: https://www.datacamp.com/courses/exploratory-data-analysis-in-sql

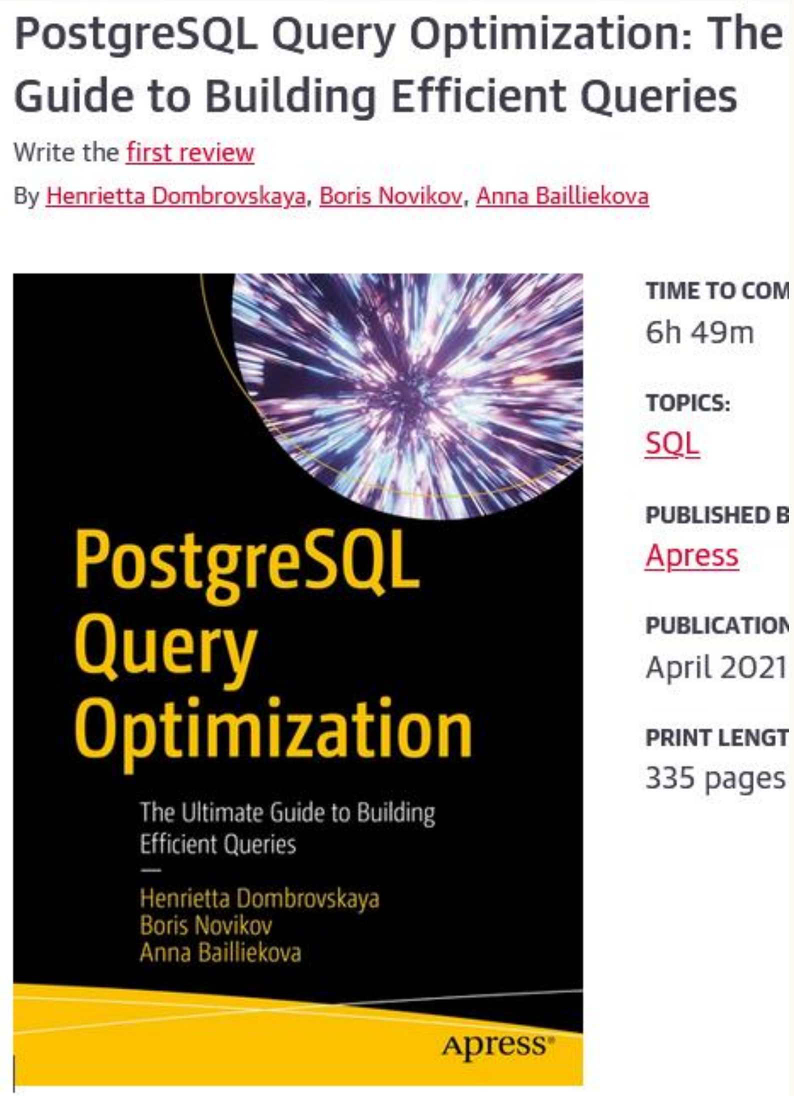
Udacity:

- ✓ SQL for Data Analysis: https://www.udacity.com/course/sql-for-data-analysis--ud198
- ✓ Intro to Relational Databases: https://www.udacity.com/course/intro-to-relational-databases--ud197

WORKS CITED


SQL Pocket Guide, 4th
9 reviews
By Alice Zhao
O'REILLY
4th Edition
Covers MySQL, Oracle, PostgreSQL, SQL Server, SQLite
SQL
Pocket Guide
A Guide to SQL Usage
Alice Zhao


SQL for Data Analysis
4 reviews
By Cathy Tanimura
O'REILLY
SQL
for Data Analysis
Advanced Techniques for Transforming Data into Insights
Cathy Tanimura


PostgreSQL Query Optimization: The Guide to Building Efficient Queries
Write the first review
By Henrietta Dombrovskaya, Boris Novikov, Anna Bailliekova
PostgreSQL
Query
Optimization
The Ultimate Guide to Building Efficient Queries
Henrietta Dombrovskaya
Boris Novikov
Anna Bailliekova
Apress
6h 49m
TOPICS:
SQL
Apress
April 2021
335 pages