Data Analytics

Introduction to SQL

GENERAL ASSEMBLY

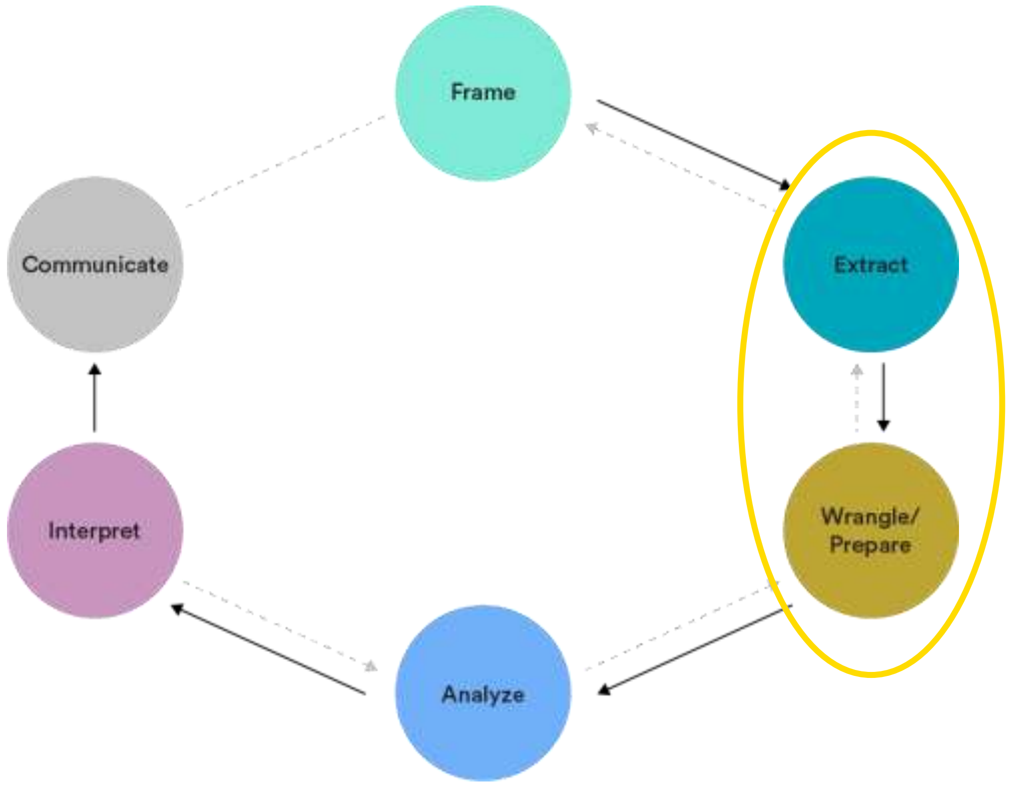
Our Learning Goals

- Navigate a relational database.
- Apply SQL commenting code.
- Practice writing and executing SQL queries, including SELECT, FROM, WHERE, and DISTINCT SELECT.
- Work with logical and comparison operators in SQL.

Where We Are in the DA Workflow

Extract: Select, import, and clean relevant data.

Wrangle/prepare: Clean and prepare relevant data.



Before We Begin...

The SQL database tool we'll use in this class is PostgreSQL because:

- It can be configured as an **object-oriented** or a **relational database** management system (RDBMS).
- It's powerful and standardized, used for enterprise databases in both public and private environments.
- It's free and open source!

Introduction to SQL

Getting Started With SQL

Discussion:

What Happens in an Internet Minute?

Within an internet minute, there are:

- 5.9 million Google searches
- 231 million Emails sent
- 3.67 million YouTube videos watched
- 6 million people shop online

All of this (and more) happens within 60 seconds!

What can we do with all that data?

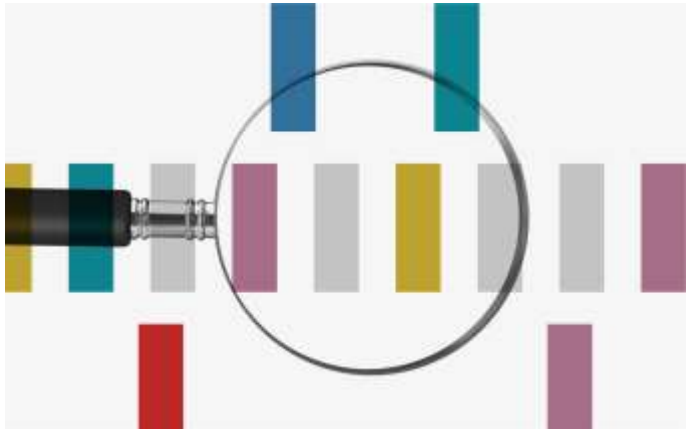
Image Source: Allaccess.com

What Is SQL?

SQL is short for **Structured Query Language**.

It's a language you can use to *query* information from your data. You can use it to ask questions and make requests such as:

- "How many users logged in this week?"
- "Show me the posts by this user from the past month."



SQL

- Can query, retrieve, and aggregate **millions** of records.
- Cloud-based data query and retrieval is not limited to your local computer system.
- Can organize data tables.
- Allows users to remotely interact with large data sets in production environments.

Excel

- Has a fixed upper **limit** of **1,048,576 rows** and **16,384 columns**.
- Is limited by your computer's available memory and system resources.
- Can analyze and visualize small amounts of data.

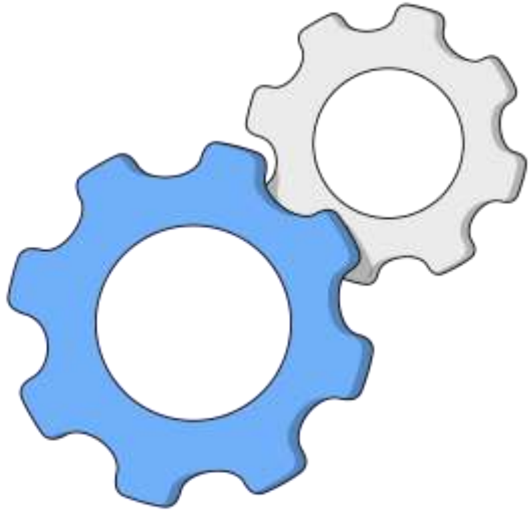
Benefits of SQL

- Simple to use.
- Industry standard for nearly all relational databases.
- Approachable syntax.
- Supportive community.
- Available in free and open versions.

Limitations of SQL

Although SQL has quite a few impressive superpowers, there are also limitations…

- SQL is **not** a data visualization tool.
- SQL can query, manipulate, organize, and analyze data.
- SQL is not a complete replacement for Excel given its lack of visualization functionality.
    - It's normally used **in conjunction with Excel, Tableau, and other tools**.



Introduction to SQL

Navigating a SQL Database

PgAdmin PostgreSQL

- PostgreSQL is an open-source SQL standard compliant RDBMS (relational database management system).
- PgAdmin is a popular and feature-rich open-source administration and development platform for PostgreSQL.
- PostgreSQL is extensible and has strong online community support.

Guided Walk-Through:

Navigating the Superstore Data Set

We'll continue working with the Superstore data set in this unit. Let's go through the following steps to get started:

1. Connect to the SQL database that we'll be using for this unit.
   a. Click [this link to] access your nearest host browser.

Username: analytics_student@generalassemb.ly
Password: analyticsga

2. Explore the functions of the client software (execute, stop, save, new query).
3. Look at the first and last 100 rows of the data from the tables using the menus.
4. Review how the column properties are defined using the menus.

Guided Walk-Through:

Superstore Data Directory Tree

- Servers (*pgAdmin can be configured for multiple server connections.*)
  - Databases → Superstore (our database for today)
    - Schemas → Public (our collection of tables)
      - Tables (This would be the Superstore Table Names.)
        - Columns (for each table)

Telling the Story of One Row

To understand our data, we need to know what's in the stored data. One way to do this is by *telling the story of one row*:

- Read across one row of your data to really see and understand the values contained in every column.

This will help you get to know what's in your data set or propel you to investigate further if you don't understand a particular column.

Group Exercise:

Exploring PGAdmin

5 minutes

- Get to know your console
- Take some time to familiarize yourself with the PG Admin console
- In your group, explore the different buttons/options available

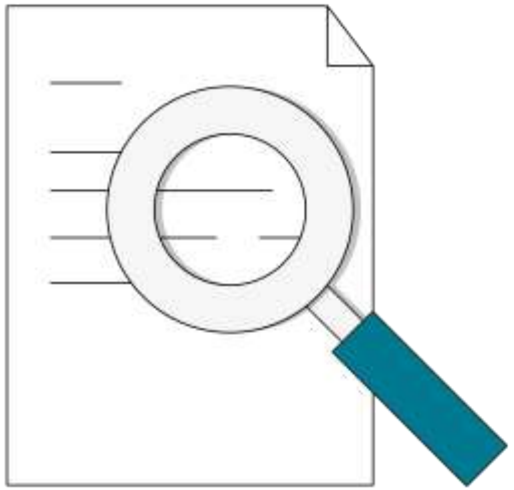
Discussion:

Previewing Your Data Set

Examining 100 rows of data may help us better understand if this is the data set we need to answer our research questions. This subset allows us to preview the data.

How does “previewing” the data connect to the data analytics workflow?

Is this useful? Why or why not?



Guided Walk-Through:

Viewing the Tables

To view the contents of a table, let's navigate to the tables in the Superstore database's public schema.

- There, you should see five tables: Customers, Orders, Products, Regions, and Returns.
- To view the data, right click on the table, go to "View Data," and select the top 100 rows.

Revealing contents of the tables and keeping the column names in view will assist you as you author queries.

Discussion:

Transactional vs. Reference Tables

Transactional tables are large and updated frequently, whereas **reference** tables are rarely modified.

Now that we've seen the five tables — Customers, Orders, Products, Regions, and Returns — which of these do you think are transactional and which ones are reference?

Partner Exercise:

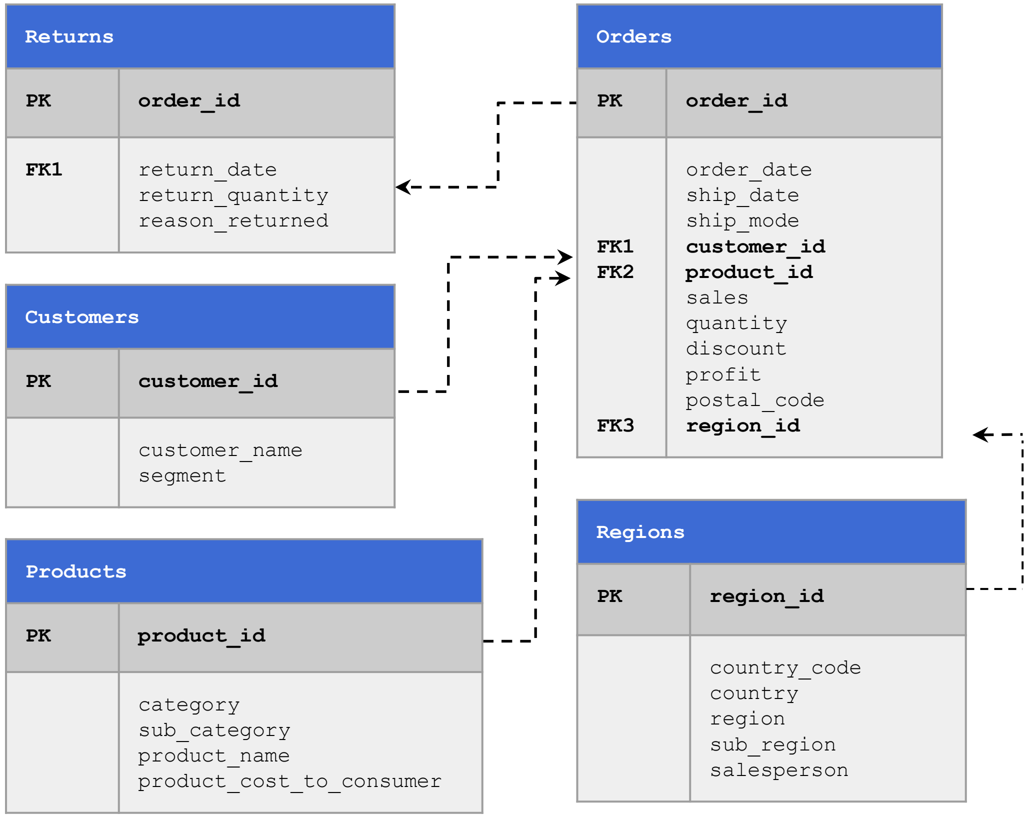
Describing Stored Data

5 minutes

Take a closer look at the tables in the Superstore data set and explore the data they contain with your partner.

- Characterize each table as either transactional or reference.
- Take five minutes and make notes about the database:
  - Write a few sentences describing the data stored in each table.
  - Note the data types assigned to each column.
  - Which columns could serve as links between tables later in our data exploration?
- Discuss with your partner.

Entity Relationship Diagram (ERD)



Returns	
PK	order_id
FK1	return_date return_quantity reason_returned

Customers	
PK	customer_id
	customer_name segment

Products	
PK	product_id
	category sub_category product_name product_cost_to_consumer

Orders	
PK	order_id
	order_date ship_date ship_mode
FK1	**customer_id**
FK2	**product_id**
	sales quantity discount profit postal_code
FK3	**region_id**

Regions	
PK	region_id
	country_code country region sub_region salesperson

An **entity** is a **component of data.**

An **entity relationship diagram** (ERD) shows the relationships of **entity sets** stored in a database.

ERDs help us illustrate the **logical structure** of databases.

PK: Primary key
FK: Foreign key, all unique identifiers

Adapted from: Guru99.com

Introduction to SQL

Writing SQL Queries

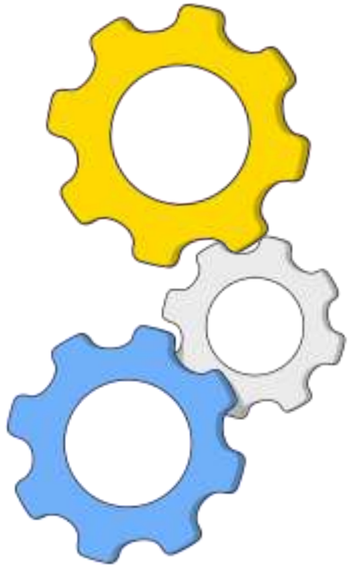
The Two Main Clauses of a SQL Query

SELECT

- Allows you to select certain *columns* from a table.
- Determines which *columns* of information are downloaded.

FROM

- Specifies the *tables* from which the query extracts data.



Did You Know...?

You ran a query already! When we selected the top 100 rows of a table using the “View Data” menu or “Query Tool” menu selections, this SQL statement ran in the background:

SELECT * FROM products LIMIT 100;

Take a closer look at the syntax above and try answering these questions:

- ❏ What does * mean?
- ❏ What does “**FROM products**” mean?
- ❏ What does the **LIMIT** do?

The SQL Query Order of Construction

SELECT picks the columns.

FROM points to the table.

WHERE puts filters on rows.

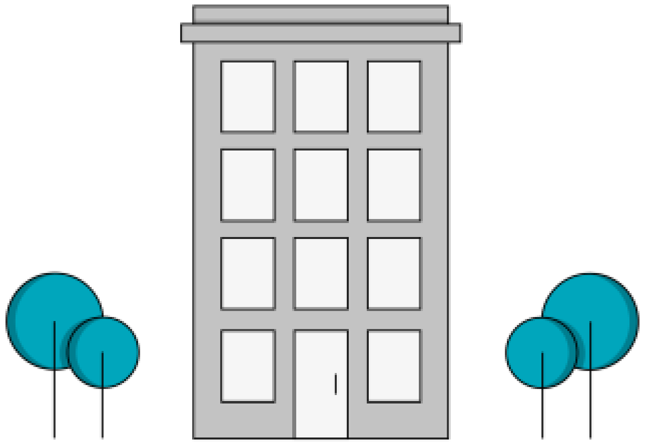
GROUP BY aggregates across values of a variable.

HAVING filters aggregated values ***after*** they have been grouped.

ORDER BY sorts the results.

LIMIT limits results to the first **n** rows.

The Query Building



Practicing Good SQL Grammar

Common Punctuation

- Signal the end of your SQL Query with a semicolon (;).
- Commas separate column names in an output list (,).
- Use single quotations around text/strings ('Nokia').

```
SELECT column1, column2
FROM table -- important note
WHERE column1 = 'Some Text';
```

Query Code Spacing

- SQL only requires a single white space to separate elements.
- Carriage returns are often used to enhance readability.

Notes Within Queries

- Always provide comments (source, revision, author, etc.).
- Comments are made after typing double dashes; or the pair /*   */.

Guided Walk-Through:

Modifying a Query

All queries can be run in the pgAdmin SQL window. For the remainder of the lesson, we'll practice modifying queries.

If we wanted to return the product ID and product name, which table would we use?

First, we can tell **SELECT** which columns or variables we want:

```
    SELECT product_id, product_name

    FROM products;
```

Group Exercise:

Modifying a Query

10 minutes

Work with your group to return all countries, regions, and salespeople.

Which table would you use and which columns do you need to return?

Group Exercise:

Modifying a Query | Solution

Your query should look like this:

```
SELECT country, region, salesperson

FROM regions;
```

Introducing SELECT DISTINCT

- **SELECT DISTINCT** returns a unique combination of values for all columns selected.
- Every row, therefore, is a unique combination of values and results in a de-duplicated table.

```
SELECT DISTINCT column1, column2
FROM table -- important note
WHERE column1 = 'Some Text';
```

Guided Walk-Through:

SELECT DISTINCT

We can add **DISTINCT** to the query statement to eliminate duplicates of categories and subcategories:

```
SELECT DISTINCT category, sub_category

FROM products;
```

The SQL Query Order of Construction

SELECT picks the columns.

FROM points to the table.

WHERE puts filters on rows.

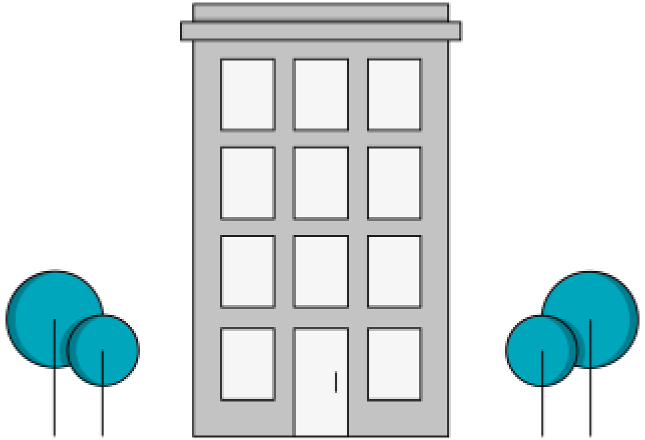
GROUP BY aggregates across values of a variable.

HAVING filters aggregated values ***after*** they have been grouped.

ORDER BY sorts the results.

LIMIT limits results to the first **n** rows.

The Query Building



Guided Walk-Through:

Sorting Results With **ORDER BY**

ORDER BY sorts results in ascending or descending order.
After **ORDER BY** is a number that indicates the column by which you're sorting.

The default sort order is ascending, but you can specify ascending (**ASC**) or descending (**DESC**) to determine the sort order.

```
SELECT *

FROM products

ORDER BY 1 ASC;
```

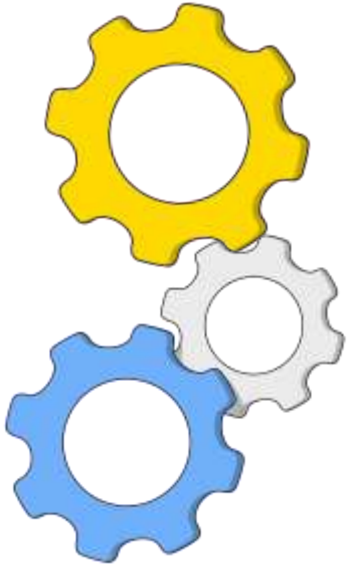
Introduction to SQL

WHERE Conditions

Introducing the WHERE Clause

WHERE

- Allows you to select certain *rows* from a table based on a single condition or multiple conditions.



The SQL Query Order of Construction

SELECT picks the columns.

FROM points to the table.

WHERE puts filters on rows.

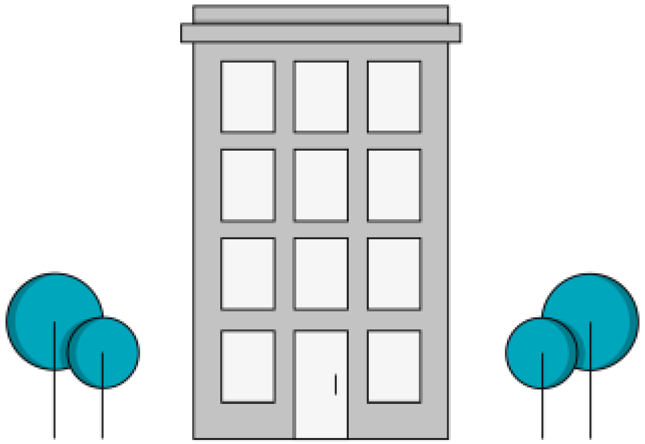
GROUP BY aggregates across values of a variable.

HAVING filters aggregated values ***after*** they have been grouped.

ORDER BY sorts the results.

LIMIT limits results to the first **n** rows.

The Query Building



Guided Walk-Through:
Filtering With **WHERE**

The **WHERE** clause filters rows by setting a criteria:

```
SELECT *

FROM products

WHERE sub_category = 'Furnishings';
```

Guided Walk-Through:

Validating Data With **COUNT**

COUNT is a basic aggregation function that counts the number of rows returned.

Here are two popular use cases:

- **COUNT(*)** - Counts all rows returned by the query.
- **COUNT(column_name)** - Counts all rows where the field is not **NULL**.

```
SELECT COUNT(*)
FROM orders
WHERE sales > 100;
```

Counts all rows of orders over $100.

Guided Walk-Through:

Basic Operators for WHERE

<>, !=	Not equal to.
>, >=	Greater than; greater than or equal to.
<, <=	Less than; less than or equal to.

Which countries are not in the “EMEA” region?

```
    SELECT DISTINCT country
    FROM regions
WHERE region != 'EMEA';
```

Guided Walk-Through:

Basic Operators for WHERE (Cont.)

Which orders have more than $90 in profit?

```
SELECT *
FROM orders
WHERE profit > 90;
```

Guided Walk-Through:

Basic Operators for WHERE (Cont.)

Which returns include more than one quantity of an item?

```
SELECT *
FROM returns
WHERE return_quantity > 1;
```

Guided Walk-Through:

Filtering With **AND**, **OR**, and **()**

Some additional query methods to try in the WHERE clause:

- **AND**: Returns if both conditions are true.
- **OR**: Returns if either condition is true (FALSE if neither is true).
- **( ):** Parentheses group conditions to set all equal to true.

```
SELECT *
FROM regions
WHERE (region = ‘Americas’
    OR region = ‘APAC’)
    AND salesperson <> ‘Anna Andreadi’;
```

Comparison Operators

IN ( )	Found in list of items.
BETWEEN	Within the range of, including boundaries.
NOT	Negates a condition.
LIKE, ILIKE	Contains item. ILIKE disregards case.
%	Wildcard, none to many characters.
_	Wildcard, single character.

Guided Walk-Through:

Filtering With **IN**

IN ( ) allows you to specify multiple values in WHERE, while **NOT IN ( )** allows you to negate the list of values.

Which products are **either** furnishings or technology (based on category name)?

```
SELECT *
FROM products
WHERE category IN (‘Furniture’, ‘Technology’);
```

Which products are **neither** furnishings nor technology?

```
SELECT *
FROM products
WHERE category NOT IN (‘Furniture’, ‘Technology’);
```

Guided Walk-Through:

Filtering With **BETWEEN**

BETWEEN ( ) allows you to select values within a given range.

1. Which products have a cost to consumers between $25 and $100?

```
SELECT *
FROM products
WHERE product_cost_to_consumer BETWEEN 25 AND 100;
```

2. Which orders have sales between $50 and $100?

```
SELECT *
FROM orders
WHERE sales BETWEEN 50 AND 100;
```

Guided Walk-Through:

Filtering **LIKE** and **ILIKE**

LIKE is used for pattern matching in SQL, while NOT LIKE negates the match.

1. Which products have ‘Calculator’ in the product name?

```
SELECT *
FROM products
WHERE product_name LIKE '%Calculator%’;
```

1. Which products have ‘Printer’ in the category name?

```
SELECT *
FROM products
WHERE product_name ILIKE '%PRINTER%’;
```

Guided Walk-Through:

Filtering With Wildcards

PostgreSQL provides two wildcard characters to work with LIKE:

- **percent (%)** for matching any sequence of characters.
- **underscore (_)** for matching any single character.

1. Which products have ‘Clock’ in the product name?

```
SELECT *
FROM products
WHERE product_name LIKE '%Clock%’;
```

1. Which customers have names that start with “A” and third letter of “r”?

```
SELECT *
FROM customers
WHERE customer_name LIKE 'A_r%’;
```

Introduction to SQL

From Stakeholder Questions to Efficient Queries

From Stakeholder Questions to SQL Queries

Example Stakeholder Question 1:

Who are the salespeople in the United States?

Resulting SQL query:

```
SELECT DISTINCT salesperson
FROM regions
WHERE country ILIKE ‘%United States%’;
```

From Stakeholder Questions to SQL Queries

Example Stakeholder Question 2:

What were the top five sales in 2019?

Resulting SQL query:

```
SELECT DISTINCT order_id, sales
FROM orders
WHERE DATE_PART('year', order_date) = 2019
ORDER BY 2 DESC
LIMIT 5;
```

From Stakeholder Questions to SQL Queries

Example Stakeholder Question 3:

What was the highest discount we gave in 2019?

Resulting SQL query:

```
SELECT DISTINCT discount
FROM orders
WHERE DATE_PART(‘year’, order_date) = 2019
ORDER BY 2 DESC
LIMIT 1;
```

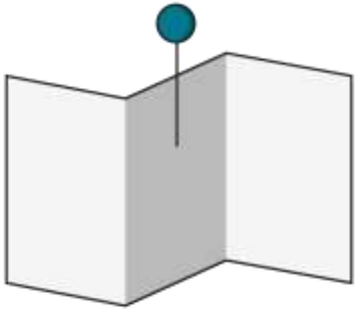
Partner Exercise:

From Questions to Queries

10 minutes

Together with your partner, write SQL queries for the following questions.

1. How many countries are in the “Americas” region?
2. Which technology products are sold to consumers(product_cost_to_consumer) for more than $1,000?
3. How many product items were returned in 2019 for unknown reasons?



Partner Exercise:

How Did It Go? | **Solutions**

1. How many countries are in the “Americas” region?

```
SELECT COUNT(country)FROM regions
WHERE region = ‘Americas’;
```

2. Which technology products are sold to consumers for more than $1,000?

```
SELECT * FROM products WHERE category = ‘Technology’
AND product_cost_to_consumer > 1000;
```

3. How many product items were returned in 2019 for unknown reasons?

```
SELECT COUNT(order_id) FROM returns
WHERE DATE_PART(‘year’, return_date) = 2019
AND reason_returned = ‘Not Given’;
```

Validating Your Output

How do you know your results are correct?

- Check that the table and column names are correct.
- Check that your single quotes (‘) or double quotes (“) are closed.
- Check operators such as >, <. =. !=, etc.
- Use EXPLAIN before SELECT to check how your SQL statement will access your database.
- Review error messages when your query does not run.
- Apply business context or industry knowledge (of similar data) to check order of magnitude.

Introduction to SQL

Additional Practice With SQL Queries and Operators

Partner Exercise:

Writing and Executing SQL Queries

Here are the questions that we want to ask of our data. With your partner, practice writing and executing queries that'll help us answer these questions.

1. How many countries are in the APAC sales region?
2. Which furniture products are sold for more than $50 and less than $500?
3. Which products are recycled material and are sold for less than $25?
4. How many customers are in our consumer segment with a first name that starts with the letter "A"?
5. What's the total profit (dollars) for orders made in 2019 with 2–3 product items in the order?

Partner Exercise:

Writing and Executing SQL Queries | **Solutions**

```
1. SELECT COUNT(country) FROM regions WHERE region = ‘APAC’;
```

29 countries

```
2. SELECT * FROM products WHERE category = ‘Furniture’ AND product_cost_to_consumer
BETWEEN 50 AND 500;
```

1461 rows

```
3. SELECT * FROM products WHERE product_name LIKE ‘%Recycled%’ AND
product_cost_to_consumer < 25;
```

309 rows

```
4. SELECT COUNT(customer_id) FROM customers WHERE segment = ‘Consumer’ AND customer_name
LIKE ‘A%’;
```

66 customers

```
5. SELECT SUM(profit) FROM orders WHERE DATE_PART(‘year’, order_date) = ‘2019’ AND
quantity BETWEEN 2 and 3;
```

$195,634.91

Grouping in SQL

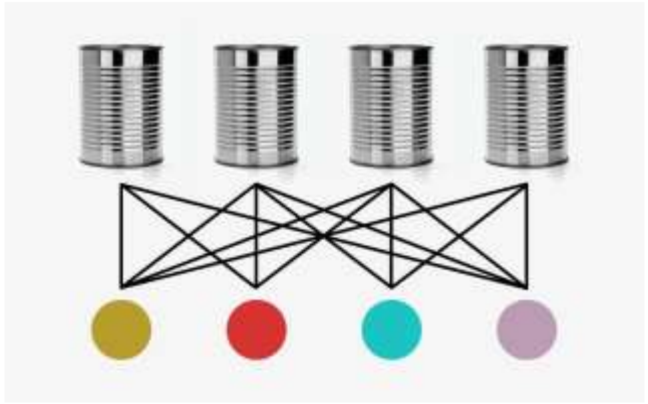
Aggregate Functions in SQL

Aggregate Functions

In SQL, aggregate functions help summarize large quantities of data and...

- Produce a single value from a defined group.
- Operate on sets of rows and return results based on **groups of data**.

The most commonly used aggregate functions are **MIN**, **MAX**, **SUM**, **AVG**, and **COUNT**.



Guided Walk-Through:

Adding Aggregates to SELECT

Aggregate functions fit into the SELECT statement just like unaggregated columns:

```
SELECT SUM(col1)
    FROM table;
```

Partner Exercise:

Aggregate Functions Matchmaking

5 minutes

With your partner, refer to the scenarios on the left and connect them to the aggregate functions on the right that will return the results you need.

1. The total number of orders placed on a specific date.
2. The highest profit margin in the consumer segment.
3. The typical quantity of copiers sold in the central region.
4. The number of customers placing orders from Madhya Pradesh, India(without duplicate customers).
5. The lowest discount given in the furniture category.

COUNT: Counts how many values are in a particular column.

COUNT DISTINCT: Counts how many unique values are in a particular column.

SUM: Adds together all the values in a particular column.

AVG: Calculates the average of a group of selected values.

MIN/MAX: Return the lowest and highest values in a particular column, respectively.

Partner Exercise:

Let's Try It!

5 minutes

Now that you have a good sense of what each aggregate function does, select 2–3 prompts from the previous slide and try writing out the query for each.

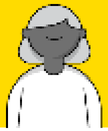
At a minimum, your queries should include SELECT and FROM and an aggregate function. For example:

```sql
SELECT SUM(sales)
FROM orders;
```

Ready, set, go!

Introduction to SQL

Wrapping Up



Solo Exercise:

Optional Homework

To help Superstore better understand their data, you will write queries that answer the questions below:

1. Which products are made by Xerox(hint: product_name contains xerox)?
2. How many countries are in our Western Europe sub-region?
3. How many customers in our consumer segment have names that start with the letter “S”?
4. What is the total number of items (not orders) returned with some reason given?
5. How many orders used Standard Shipping and a discount code?

Recap

In today's class, we...

- Navigated a relational database.
- Practiced writing and executing SQL queries, including SELECT, FROM, WHERE, and DISTINCT SELECT.
- Worked with logical and comparison operators in SQL.

Looking Ahead

Homework:

- Review todays class
- Optional myGA lessons: **Beginner SQL** (unit) —
  - Using Aggregate Functions to Summarize and Compare Data
  - Using CASE to Make New Fields

Up Next: Grouping in SQL.

Additional Resources

- What's the Difference Between a Primary and Unique Key?
- Combining the AND and OR Conditions
- Logical Operators (Transact-SQL)
- Get Ready to Learn SQL Server: 4. Query Results Using Boolean Logic