Operators and Aggregates.

Objectives:

By the end of this chapter, you should be able to:

- Use built in operators to write more complex queries
- Use `ORDER BY` to categorize results
- Use built in aggregate functions to calculate data
- Use `GROUP BY` to aggregate data into sub groups
- Use `CASE` statements for custom conditional output

Let's start with the following SQL in `psql`:

```
DROP DATABASE IF EXISTS sports; -- in case you copy this whole set of
commands multple times
CREATE DATABASE sports;
```

`\c sports` to connect to newly-created sports database.

Now to seed the data:

```
CREATE TABLE players (id SERIAL PRIMARY KEY, name TEXT, sport TEXT, team
TEXT, jersey_number INTEGER, is_rookie BOOLEAN);

INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('Martin', 'hockey', 'devils', 12, false);
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('David', 'baseball', 'mets', 2, true);
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('David', 'soccer', 'galaxy', 17, false);
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('Elie', 'baseball', 'cobras', 8, false);
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('Lisa', 'basketball', 'sparks', 44, false);
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('Seabass', 'frisbee', 'jumbos', 44, false);
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('Sue', 'basketball', 'lynx', 54, false);
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('Candace', 'sparks', 'cobras', 49, false);
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('Swin', 'basketball', 'shock', 1, true);
```

```
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('Gilbert', 'basketball', 'warriors', 0, false);
INSERT INTO players (name, sport, team, jersey_number, is_rookie) VALUES
('Maria', 'tennis', 'none', 24, false);
```

WHERE

The building block for these operators is the `WHERE` clause which comes after any operation like `SELECT`, `UPDATE` or `DELETE`

=

```
SELECT * FROM players WHERE sport = 'basketball';

/*
 id |  name   |   sport    |   team   | jersey_number | is_rookie
----+---------+------------+----------+---------------+-----------
  5 | Lisa    | basketball | sparks   |            44 | f
  7 | Sue     | basketball | lynx     |            54 | f
  9 | Swin    | basketball | shock    |             1 | t
 10 | Gilbert | basketball | warriors |             0 | f
*/
```

IN

To find multiple records we can search for multiple terms using `IN`

```
SELECT * FROM players WHERE jersey_number IN (0,1);
/*
 id |  name   |   sport    |   team   | jersey_number | is_rookie
----+---------+------------+----------+---------------+-----------
  9 | Swin    | basketball | shock    |             1 | t
 10 | Gilbert | basketball | warriors |             0 | f

*/
```

NOT IN

We can do the exact inverse of `IN` using `NOT IN`

```
SELECT * FROM players WHERE jersey_number IN (0,1);
/*
 id |  name   |   sport    |  team  | jersey_number | is_rookie
----+---------+------------+--------+---------------+-----------
  1 | Martin  | hockey     | devils |            12 | f
  2 | David   | baseball   | mets   |             2 | t
```

```
  3 | David   | soccer     | galaxy |            17 | f
  4 | Elie    | baseball   | cobras |             8 | f
  5 | Lisa    | basketball | sparks |            44 | f
  6 | Seabass | frisbee    | jumbos |            44 | f
  7 | Sue     | basketball | lynx   |            54 | f
  8 | Candace | sparks     | cobras |            49 | f
 11 | Maria   | tennis     | none   |            24 | f

*/
```

BETWEEN

To search for a field in a range, we can use `BETWEEN x AND y`.

```
SELECT * FROM players WHERE jersey_number BETWEEN 0 AND 25;
/*
 id |  name   |   sport    |   team   | jersey_number | is_rookie
----+---------+------------+----------+---------------+-----------
  1 | Martin  | hockey     | devils   |            12 | f
  2 | David   | baseball   | mets     |             2 | t
  3 | David   | soccer     | galaxy   |            17 | f
  4 | Elie    | baseball   | cobras   |             8 | f
  9 | Swin    | basketball | shock    |             1 | t
 10 | Gilbert | basketball | warriors |             0 | f
 11 | Maria   | tennis     | none     |            24 | f

*/
```

Arithmetic

SQL supports all kinds of arithmetic operators like <, <=, >=, > and `!=`.

```
SELECT * FROM players WHERE jersey_number > 25;
/*
 id |  name   |   sport    |  team  | jersey_number | is_rookie
----+---------+------------+--------+---------------+-----------
  5 | Lisa    | basketball | sparks |            44 | f
  6 | Seabass | frisbee    | jumbos |            44 | f
  7 | Sue     | basketball | lynx   |            54 | f
  8 | Candace | sparks     | cobras |            49 | f

*/
```

AND

To check that multiple conditions are satisfied we can use the `AND` command.

```
SELECT * FROM players WHERE jersey_number > 25 and id < 6;
/*
 id | name |   sport    |  team  | jersey_number | is_rookie
----+------+------------+--------+---------------+-----------
  5 | Lisa | basketball | sparks |            44 | f

*/
```

OR

To check that at least one condition is satisfied we can use the `OR` command.

```
SELECT * FROM players WHERE jersey_number > 25 or id < 6;
/*
 id |  name   |   sport    |  team  | jersey_number | is_rookie
----+---------+------------+--------+---------------+-----------
  1 | Martin  | hockey     | devils |            12 | f
  2 | David   | baseball   | mets   |             2 | t
  3 | David   | soccer     | galaxy |            17 | f
  4 | Elie    | baseball   | cobras |             8 | f
  5 | Lisa    | basketball | sparks |            44 | f
  6 | Seabass | frisbee    | jumbos |            44 | f
  7 | Sue     | basketball | lynx   |            54 | f
  8 | Candace | sparks     | cobras |            49 | f

*/
```

LIKE

To search for a term we can use the `LIKE` command. The `%` denotes any possible character. `LIKE` **is** case sensitive.

```
--find all players whose name starts with a capital S--
SELECT * FROM players WHERE name LIKE 'S%';
/*
 id |  name   |   sport    |  team  | jersey_number | is_rookie
----+---------+------------+--------+---------------+-----------
  6 | Seabass | frisbee    | jumbos |            44 | f
  7 | Sue     | basketball | lynx   |            54 | f
  9 | Swin    | basketball | shock  |             1 | t
*/
```

ILIKE

`ILIKE` is similar to the `LIKE` command, but it is **case insensitive**.

```
SELECT * FROM players WHERE name ILIKE 's%';
/*
 id |  name   |   sport    |  team  | jersey_number | is_rookie
----+---------+------------+--------+---------------+-----------
  6 | Seabass | frisbee    | jumbos |            44 | f
  7 | Sue     | basketball | lynx   |            54 | f
  9 | Swin    | basketball | shock  |             1 | t
*/
```

ORDER BY

If we want to order results in ascending or descending order we use the `ORDER BY ASC_or_DESC` command.

```
SELECT * FROM players ORDER BY jersey_number DESC;
/*
 id |  name   |   sport    |   team   | jersey_number | is_rookie
----+---------+------------+----------+---------------+-----------
  7 | Sue     | basketball | lynx     |            54 | f
  8 | Candace | sparks     | cobras   |            49 | f
  5 | Lisa    | basketball | sparks   |            44 | f
  6 | Seabass | frisbee    | jumbos   |            44 | f
 11 | Maria   | tennis     | none     |            24 | f
  3 | David   | soccer     | galaxy   |            17 | f
  1 | Martin  | hockey     | devils   |            12 | f
  4 | Elie    | baseball   | cobras   |             8 | f
  2 | David   | baseball   | mets     |             2 | t
  9 | Swin    | basketball | shock    |             1 | t
 10 | Gilbert | basketball | warriors |             0 | f
*/
```

Changing Data Types

Commonly in SQL, we will want to output a certain data type for an operation, to convert one data type to another we can use the `CAST` command or use `::data_type`

```
SELECT round((SUM(id) / COUNT(jersey_number))::numeric, 2)::float
```

Very commonly, we will want to take multiple values in a table and group them into sub-categories or a single category based on an aggregate function. Let's look at a few aggregate functions, but first - some sample data:

```
CREATE TABLE sales (id SERIAL PRIMARY KEY, product TEXT, customer_name TEXT,
price REAL, quantity SMALLINT);
```

```
INSERT INTO sales (product, customer_name, price, quantity) VALUES ('Chair', 'Elie', 99.99, 1);
INSERT INTO sales (product, customer_name, price, quantity) VALUES ('Table', 'Tim', 250.00, 1);
INSERT INTO sales (product, customer_name, price, quantity) VALUES ('Chair', 'Matt', 49.99, 3);
INSERT INTO sales (product, customer_name, price, quantity) VALUES ('Table', 'Janey', 1000.00, 2);
INSERT INTO sales (product, customer_name, price, quantity) VALUES ('Chair', 'Janey', 300.00, 2);
INSERT INTO sales (product, customer_name, price, quantity) VALUES ('Table', 'Tim', 2200.00, 2);
INSERT INTO sales (product, customer_name, price, quantity) VALUES ('Bookshelf', 'Elie', 1200.00, 2);
```

You can read more about these data types here.

Now that we have some sample data, let's examine a few common aggregate functions which collect multiple pieces of data and return a single value.

COUNT

To count the number of occurances we use the `COUNT` function.

```
SELECT COUNT(*) FROM sales;
SELECT COUNT(*) FROM sales WHERE product = 'Chair';
```

SUM

To figure out the sum we can use the `SUM` function and even round numbers using the `ROUND` function as well.

```
SELECT SUM(price) FROM sales;
SELECT ROUND(SUM(price)) FROM sales;
```

MIN

To find the minimum value in a data set we use the `MIN` function.

```
SELECT MIN(price) FROM sales;
```

MAX

To find the maximum value in a data set we use the `MAX` function.

```
SELECT MAX(price) FROM sales;
```

AVG

To find the average value in a data set we use the `AVG` function. We can attach the `AS` command to alias the column name.

```
SELECT AVG(price) AS max_count FROM sales;
```

GROUP BY

Now that we have seen a couple aggregate functions, lets take some information.

```
SELECT product, COUNT(product) FROM sales GROUP BY product;
```

HAVING

When using a `GROUP BY` clause, we can not attach a `WHERE` if we want to be more selective. Instead we use the `HAVING` keyword to place condition on our `GROUP BY` command.

```
SELECT product, COUNT(product) FROM sales GROUP BY product HAVING
COUNT(product) > 2;
```

DISTINCT

If we only want to find unique values in a column, we can use `DISTINCT`, we can also do this for pairs of columns separated by a comma.

```
SELECT DISTINCT customer_name FROM sales;
```

CASE

In SQL, we can use conditional logic to query our data and display custom results based off of the condition.

```
SELECT product, price,
    CASE WHEN price < 50 THEN 'inexpensive'
         WHEN price > 50 AND price < 100 THEN 'reasonable'
         WHEN price < 50 AND price < 400 THEN 'expensive'
         ELSE 'very expensive' END AS how_expensive
    FROM sales;
```