Querying Databases with SQL

Spring 2020 workshop

Dominic Bordelon, djb190@pitt.edu

Digital Scholarship Services | University Library System

University of Pittsburgh

Objectives

By the end of this workshop, you will be able to...

- Understand basic features and structures of relational databases
- Query a database table to explore information as well as answer specific questions
- Combine and query multiple tables at once using joins
- Understand common data types and null values
- Use functions to transform query inputs and results
- Aggregate results (e.g., identify and count categories in the data)
- Investigate a database to learn about what information it contains
- Save queries and results

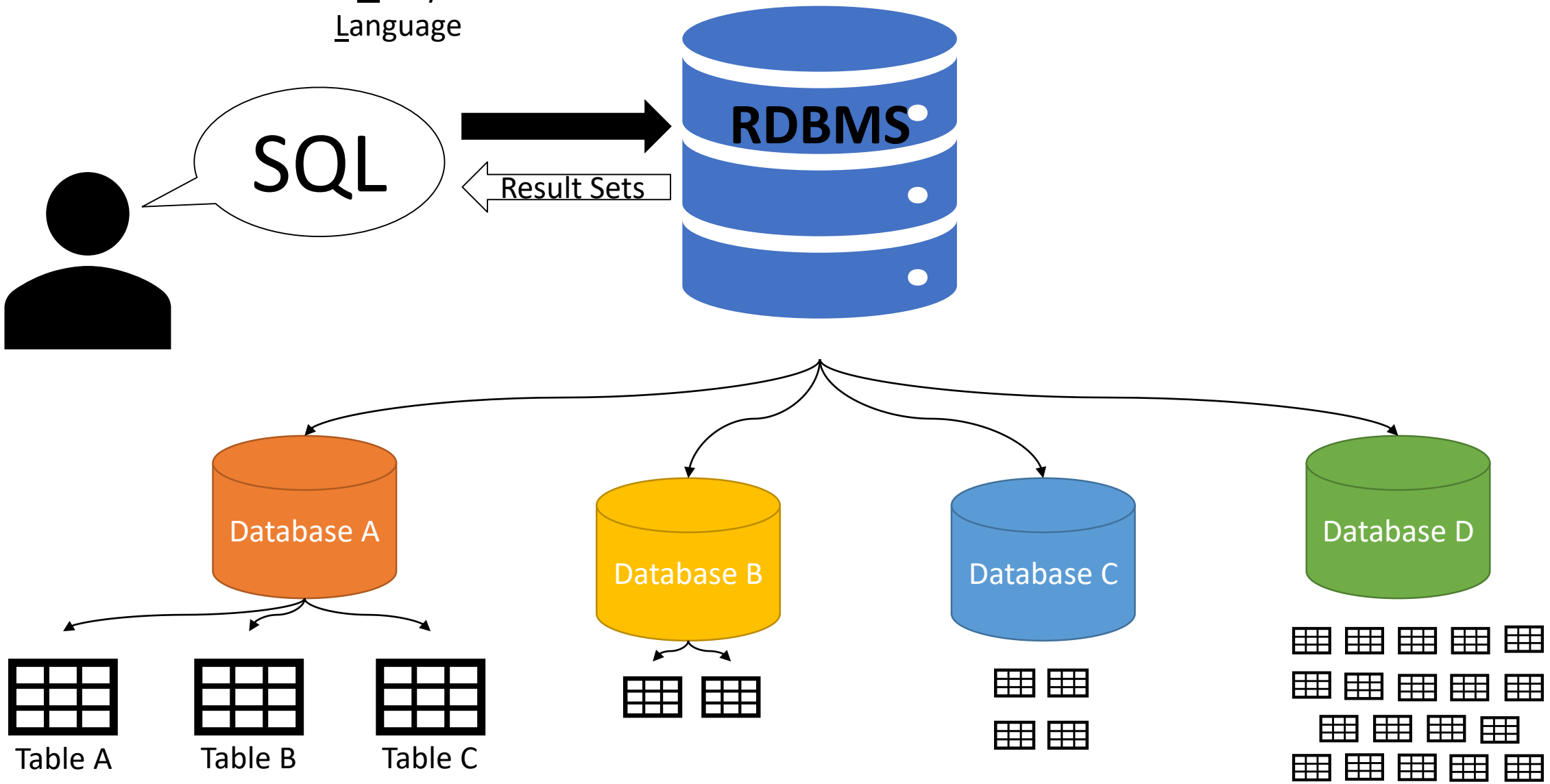
Setup

- Visit: http://pi.tt/dss-sql2020
- Download and run DB Browser for SQLite
- Download the Chinook database (`Chinook_Sqlite.sqlite`)
- Download the “portal all” database from Data Carpentry for Biologists (`portal_mammals.sqlite`)

What is SQL? What is an RDBMS*?

* Relational DataBase Management System

Structured Query Language



Data is stored in tables

id	lastName	department
1	Smith	HR
2	Jones	IT
3	Harris	HR

id	movieTitle	year
1	Jaws	1975
2	Toy Story	1995

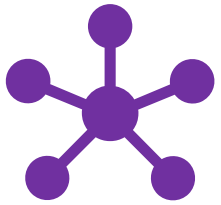
id	screen_name	location	description
17874544	TwitterSupport	Twitter HQ	Your official source for Twitter Support. …

- Table: describes a category of thing
  - Person, publication, invoice, product, observation, event…
- Column: an attribute/property of the thing
  - Eye color, publication date, invoice total, product name, geolocation…
  - There is usually an “ID” column, also called a **primary key**
- Row: a thing (or more precisely, record of a thing)

Some advantages of RDBMSes (or: why people use them)

Built-in data integrity measures

Changes can be carefully controlled through reversible **transactions**



RDBMS servers: single data store + job management

Scalability for complex and large amounts of data

SQL works on sets

Example:
Given a list of scores, which ones are in the range of 80-89?

In python

```
>>> scores = [73, 82, 95, 84, 91]
>>> [score for score in scores if 80 <=
score <= 89]
[82, 84]
```

For each score in the list, return the score if it is between 80 and 89.

→ **imperative** language

how to get the results

In SQL

```
SELECT score FROM scores
WHERE score BETWEEN 80 AND 89;
```

Give me the **set** of items such that their score is between 80 and 89.

→ **declarative** language

what results you want (the engine figures out how)

The result of a query is always a set

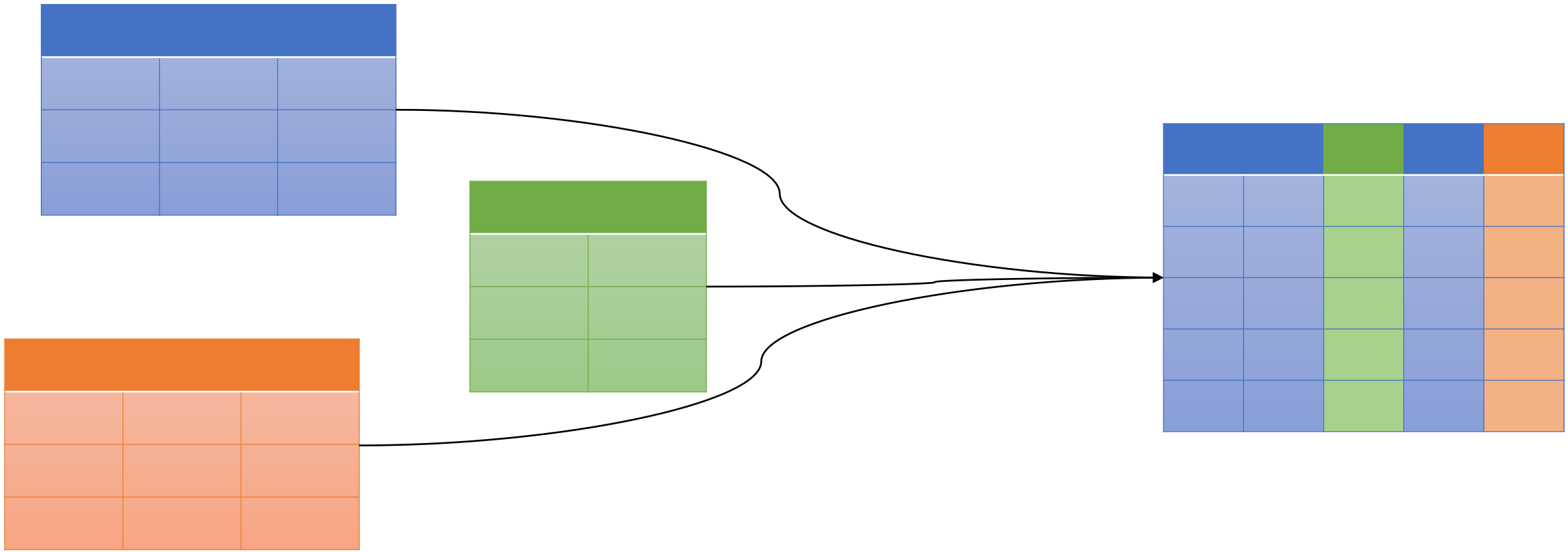
(even if it has only one member)

Give me the last name, first name, and email of the employee whose Employee ID is 7.

	EmployeeId	LastName	FirstName	Email
1	7	King	Robert	robert@chinookcorp.com

Result: 1 rows returned in 6ms

Tables are often combined to ask more complex questions— “joining”



What can you do with SQL? (types of statements)

- `SELECT`: read from the database
- `INSERT`: create new row(s) in a table
- `UPDATE`: update existing row(s) of a table
- `DELETE`: delete row(s) from a table




Today's focus

- `CREATE TABLE, ALTER TABLE, RENAME TABLE, DROP TABLE…`
- `GRANT, REVOKE` for controlling user access
- `BEGIN TRANSACTION, COMMIT, ROLL BACK`

- SQL keywords are not case-sensitive (`SELECT == select`)—but entities like tables and columns **are** case-sensitive

Anatomy of a SELECT query

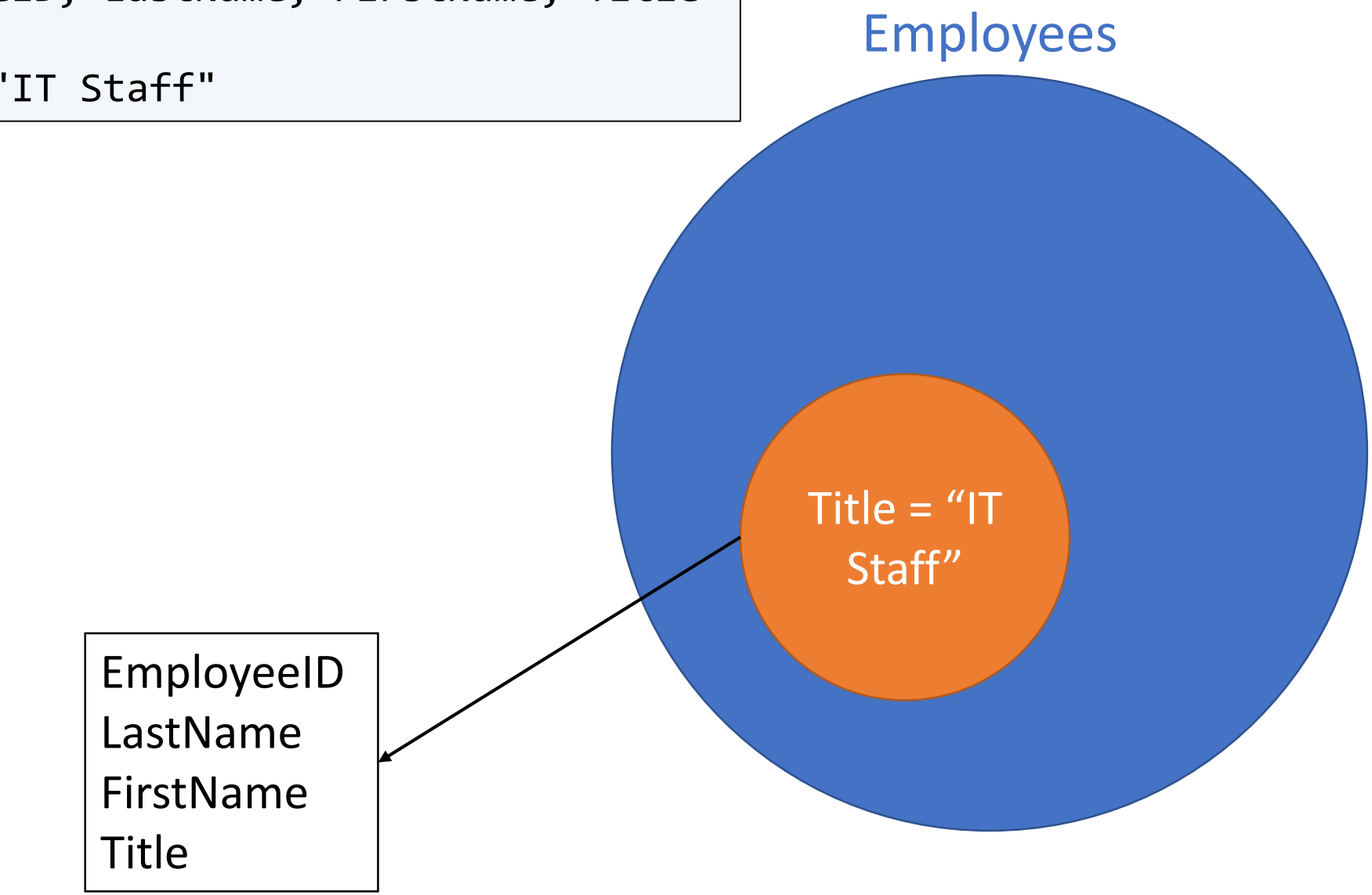
(using the Chinook database)

"What are the names of the IT staff?"

clauses

```
SELECT EmployeeID, LastName, FirstName, Title  ← columns
FROM Employee  ← table
WHERE Title = "IT Staff"  ← filtering based on some conditions
```

```
SELECT EmployeeID, LastName, FirstName, Title
FROM Employee
WHERE Title = "IT Staff"
```



Anatomy of a SELECT query

(using the Chinook database)

"What are the names of the IT staff?"

clauses

```
SELECT EmployeeID, LastName, FirstName, Title   ← columns
FROM Employee   ← table
WHERE Title = "IT Staff"   ← filtering based on some conditions
```

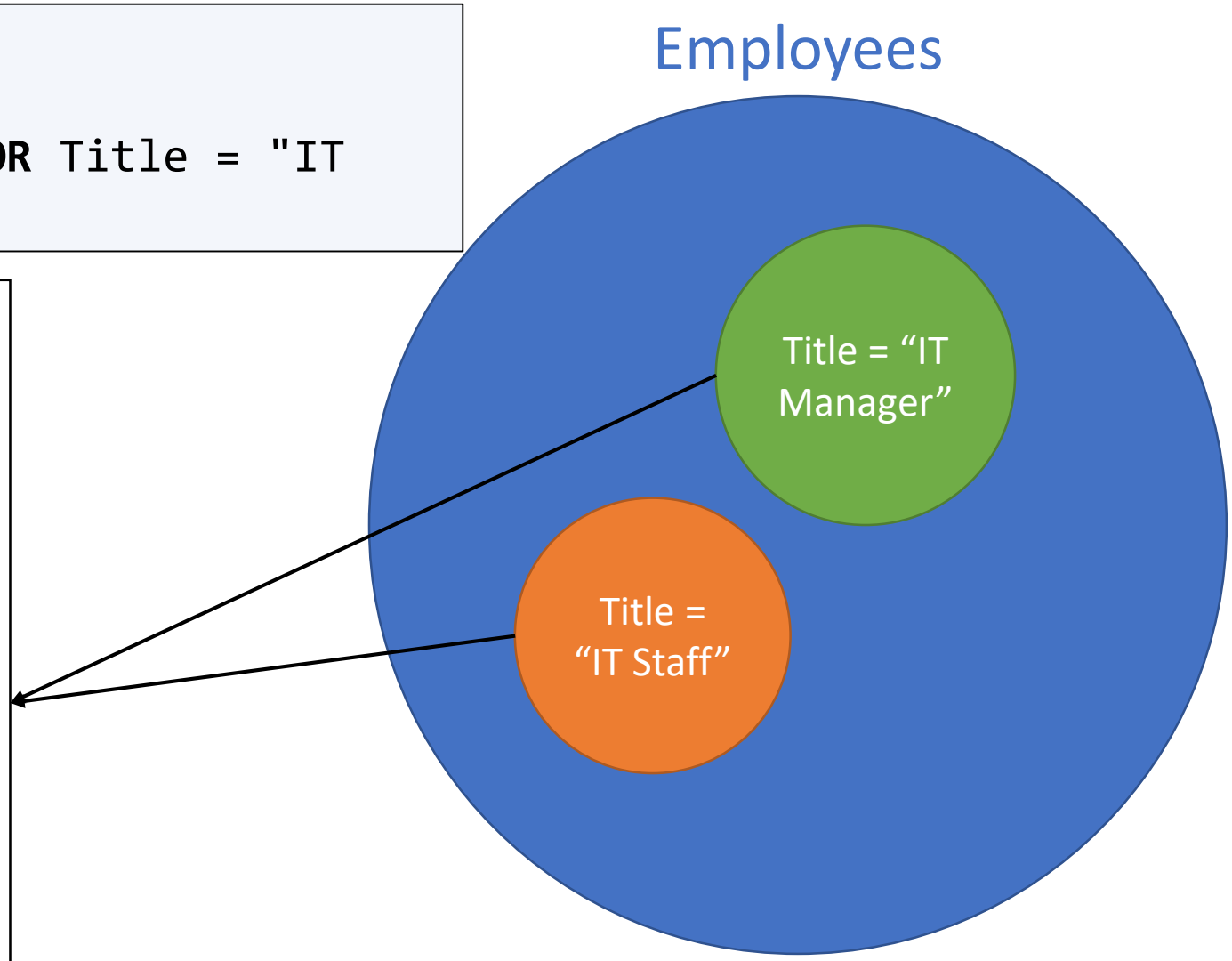
Try also:

```
SELECT * FROM Employee WHERE Title = "IT Staff" OR Title = "IT Manager"

…WHERE Title IN("IT Staff", "IT Manager")
…WHERE Title LIKE "IT%"
```

```
SELECT *
FROM Employee
WHERE Title = "IT Staff" OR Title = "IT
Manager"
```

All attributes!
(EmployeeID,
LastName,
FirstName,
Title,
ReportsTo,
BirthDate,
HireDate,
Address,
City,
State,
Country,
PostalCode,
Phone,
Fax,
Email)



Sorting your results

```
SELECT Country, LastName, FirstName FROM Customer
ORDER BY Country, LastName
```

```
SELECT InvoiceId, InvoiceDate, Total FROM Invoice
ORDER BY InvoiceDate DESC
```

What is the “default” sort?

If there is no ORDER BY clause, relational DBs don’t guarantee any ordering in particular.

…but usually, it will be ascending by the ID column.

WHERE conditions: comparators

```
SELECT InvoiceId, InvoiceDate, Total FROM Invoice
WHERE Total > 10.00
```

>=, <, <= also work

```
…WHERE Total BETWEEN 5.00 AND 10.00
```

Comparisons also work with dates:

```
…WHERE InvoiceDate > '2009-01-01'
…WHERE InvoiceDate BETWEEN '2009-01-01' AND '2009-12-
31'
```

What is NULL?

- NULL means *no value*
  - It is **not** the same as `0` (zero)—NULL is not an integer, can't be used for math, and does not exist on the number line
  - It is **not** the same as "" (an empty string)—NULL is not a string
- What does it represent? It depends on the context
  - Could be: column not applicable for this row; value unknown; value to be filled later; a mistake…
- In any case, ⚠ beware NULL when doing (in)equality or comparisons!

- The proper way to test for it is `WHERE columnName IS NULL`

  or `WHERE columnName IS NOT NULL`

WHERE conditions: functions

Functions can transform a column's contents.

"What are the invoices whose total rounds to $9?"

```
SELECT * FROM Invoice
WHERE round(Total) = 9
```

⚠ **NOTE**: Functions vary greatly from one database vendor to another!

Data types

Generic type	Common variants	In SQLite
Integer	INT, BIT, TINYINT, BIGINT	INTEGER
Real/decimal	FLOAT, REAL, DOUBLE	REAL
String	CHAR, VARCHAR	TEXT
Text (long)	TEXT	TEXT
Boolean (true or false)	BOOLEAN, BIT	INTEGER
Date / datetime	DATE, TIME, DATETIME	TEXT
BLOB (Binary Large Object)	BLOB	BLOB

A column's data type determines…

- What kind of values can be stored in it
  - A decimal or a string can't be stored in an INT column
- The maximum size that can be stored in it (size of number, or length of text), e.g. `VARCHAR(50)` or `BIT`
- How much disk space is allocated for it
- How efficiently the DB engine can use it (shorter text is faster than long)
- What functions can be used on it
  - In stricter engines than SQLite, converting (also called "casting") is a very common operation

Functions work in the SELECT clause, too

```
SELECT Total, round(Total) FROM Invoice
WHERE round(Total) = 9
```

“What are the minimum, maximum, and average invoice totals?”

```
SELECT min(Total), max(Total), avg(Total)
FROM Invoice
```

Combine two string columns using the concatenate operator ||

```
SELECT FirstName || ' ' || LastName AS FullName
FROM Employee
```

Querying more than one table: joins!

Oftentimes, the information we want is in more than one table…

“Who is Frank Harris’s customer support rep?”

1. Query Customer to get SupportRepId (4)
2. Query Employee using SupportRepId to see Employee #4’s name (Margaret Park)

Since we know that there is a **relationship** between the Employee and Customer tables—that SupportRepId corresponds to EmployeeId—we can **join** these tables into a single view.

How to join

"Connect the Customer and Employee tables, such that Customer.SupportRepId is the same as Employee.EmployeeId."

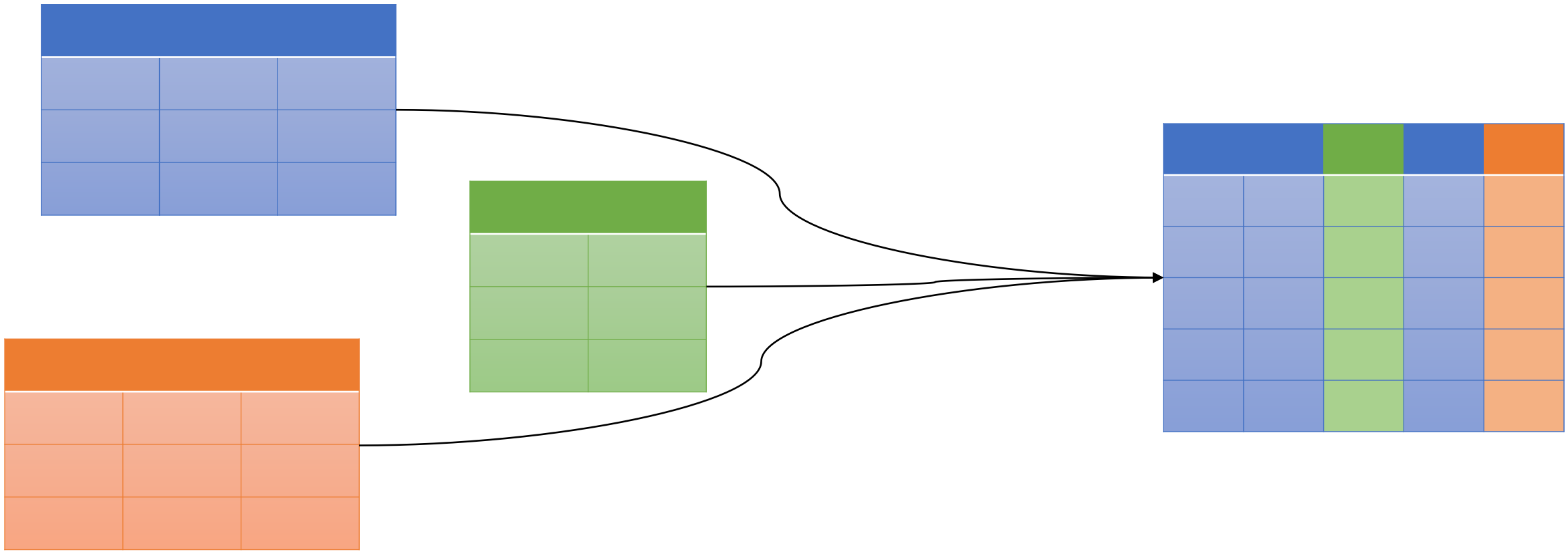
```
SELECT C.LastName, C.FirstName, SupportRepId,
E.EmployeeId, E.LastName, E.FirstName

FROM Customer AS C
JOIN Employee AS E
ON C.SupportRepId = E.EmployeeId

WHERE C.LastName = "Harris"
```

⚠ **NOTE:** This is the most common kind of join ("inner join"). There are others, with important differences.

Tables are often combined to ask more complex questions— “joining”



Why is it like this?

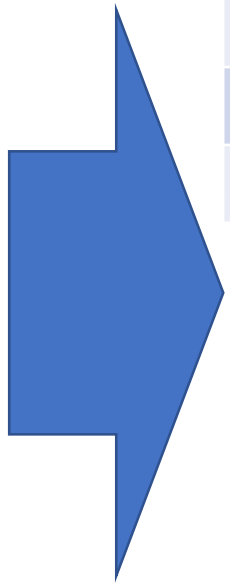
Why do we need to check multiple tables? Why not put everything in one?

When a DB designer sees the same value (or a small variety of values) in a column—for example, music genre—they try to **normalize** the data, by separating the values into their own table, and using the new ID ("**foreign key**") to refer to the value.

This is good for disk space, data integrity, and database efficiency.

Before and after normalization

Title	Genre
For Those About To Rock (We Salute You)	Rock
Fast As a Shark	Rock
Restless and Wild	Rock
Princess of the Dawn	Rock
Put The Finger On You	Rock
Let's Get It Up	Rock
Inject The Venom	Rock
Snowballed	Rock
Evil Walks	Rock
C.O.D.	Rock
Breaking The Rules	Rock
Night Of The Long Knives	Rock
Spellbound	Rock



Title	GenreId
For Those About To Rock (We Salute You)	1
Fast As a Shark	1
Restless and Wild	1
Princess of the Dawn	1
Put The Finger On You	1
…	…

Foreign key relationship

GenreId	Name
1	Rock
2	Jazz
3	Metal
…	…

Investigating a database

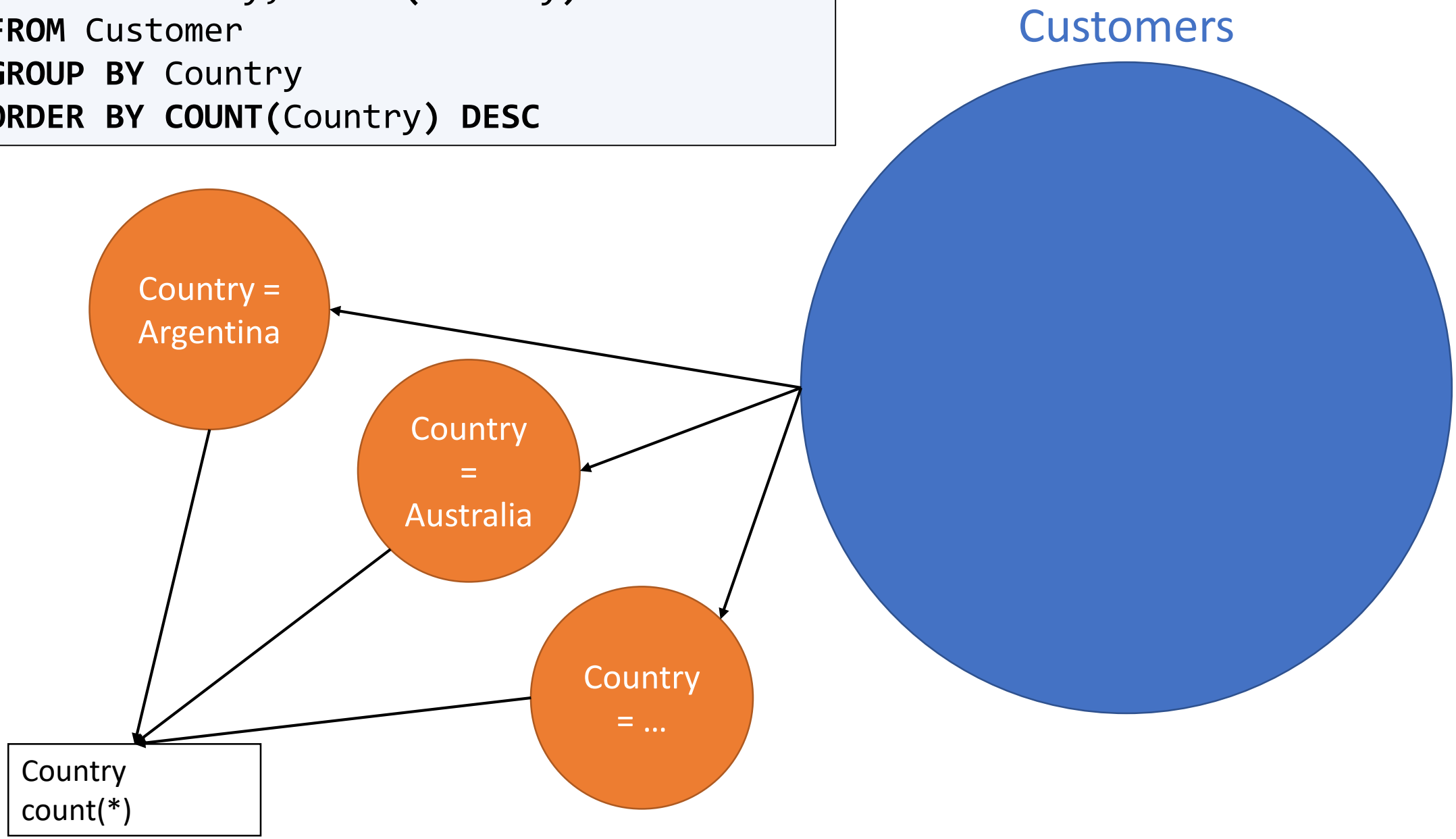
- What tables are there, and what relationships do they have?
  - Tells us (maybe) what entities are described in the DB
  - GUI: usually an “explorer” feature
- What are a table’s column definitions?
  - Tells us how an entity is described
  - Note data types, NULLable, DEFAULT
  - Foreign keys with other tables! Easiest is an Entity-Relationship (ER) diagram
- Using `SELECT DISTINCT` to see the range of recorded values

Grouping results

- “How many customers do we have in each country? How many tracks do we have in each genre? What are the min, max, and average lengths of track on each album?”
- These can all be answered with the GROUP BY clause + aggregating functions

```
SELECT Country, COUNT(Country)
FROM Customer
GROUP BY Country
ORDER BY COUNT(Country) DESC
```

```
SELECT Country, COUNT(Country)
FROM Customer
GROUP BY Country
ORDER BY COUNT(Country) DESC
```



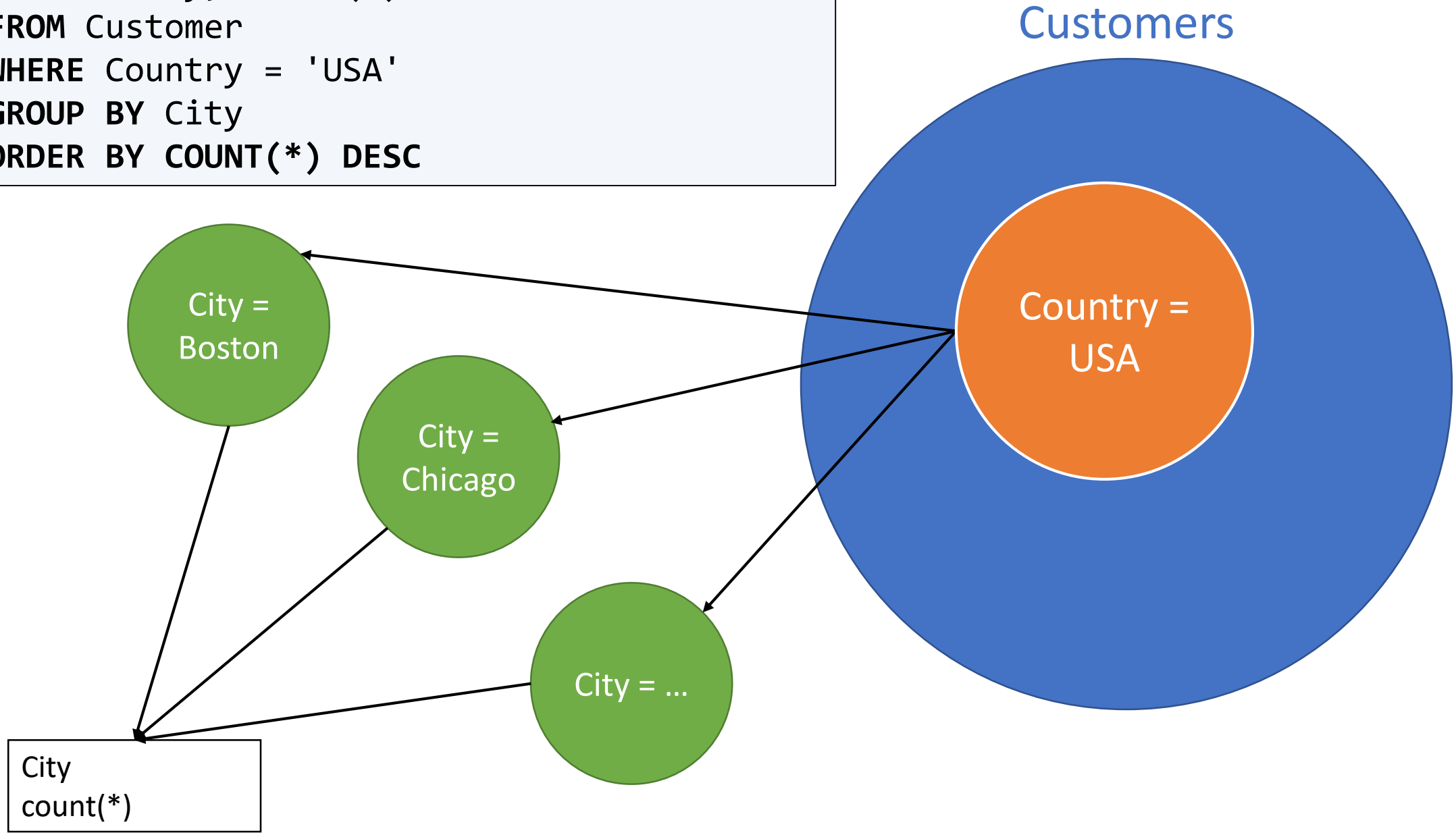
Using WHERE and GROUP BY together

WHERE filters rows *before* they are grouped.

The WHERE clause is always written before the GROUP BY clause.

```
SELECT City, COUNT(*)
FROM Customer
WHERE Country = 'USA'
GROUP BY City
ORDER BY COUNT(*) DESC
```

```
SELECT City, COUNT(*)
FROM Customer
WHERE Country = 'USA'
GROUP BY City
ORDER BY COUNT(*) DESC
```



Common aggregate functions

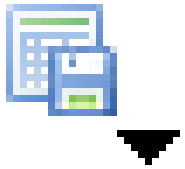
`COUNT(), MIN(), MAX(), AVG(), SUM()`

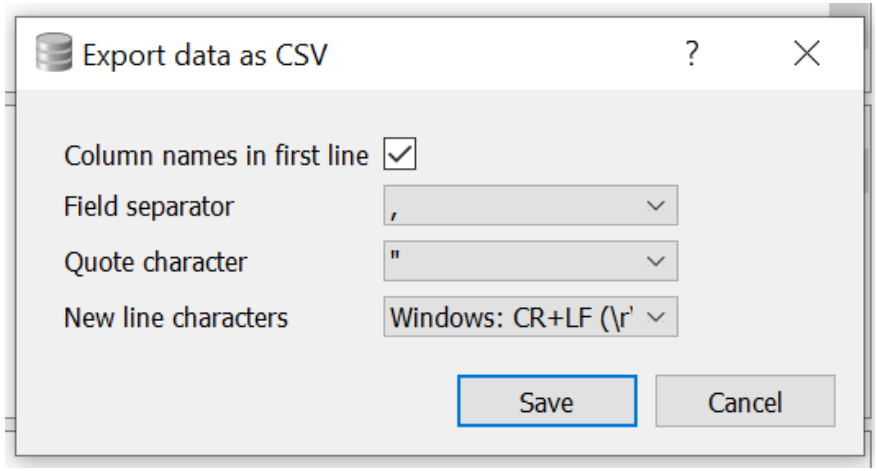
In the “portal all” database:

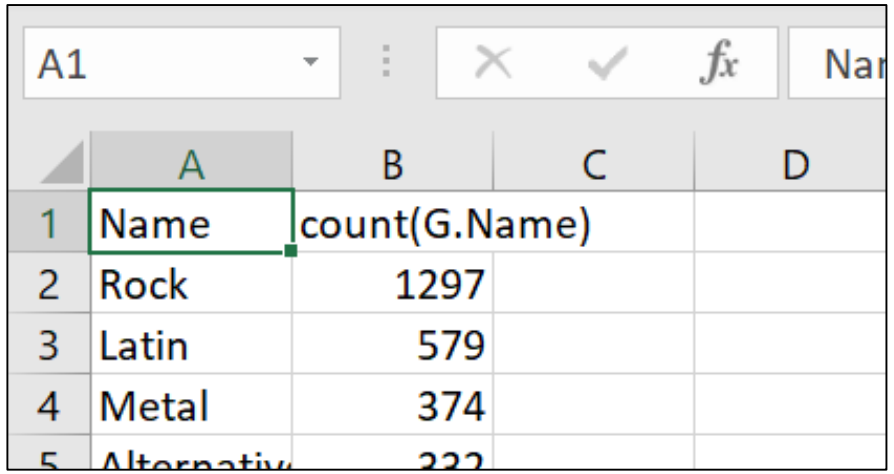
```
SELECT species_id, COUNT(*), MIN(weight),
MAX(weight), AVG(weight)
FROM surveys
GROUP BY species_id
ORDER BY species_id ASC
```

Saving queries and results

- SQL *queries* can be saved as a plaintext .sql file
  - Comments (marked with --) can be very helpful reminders
- SQL *result sets* can be exported as .csv files
  - Can be opened (and further manipulated) and visualized in Excel and many other applications
  - Can be used more programmatically with languages like python or R







SQL + other programming languages

```
>>> import sqlite3
>>> conn = sqlite3.connect('Chinook_Sqlite.sqlite')
>>> c = conn.cursor()
>>> c.execute('select LastName, FirstName from Employee')
<sqlite3.Cursor object at 0x0000025C42A9B8F0>
>>> print(c.fetchall())
[('Adams', 'Andrew'), ('Edwards', 'Nancy'), ('Peacock', 'Jane'), ('
Park', 'Margaret'), ('Johnson', 'Steve'), ('Mitchell', 'Michael'),
('King', 'Robert'), ('Callahan', 'Laura')]
```

Many (most?) modern programming languages (python, R, javascript, Java, C#...) have libraries for connecting to a variety of RDBMSes.

If you want more…

- RCE@Pitt workshops by SCI
  - 8 weeks of data-centric workshops; 2 of them are about SQL

- Videos and courses at LinkedIn Learning
  - Free for Pitt students, faculty, and staff (via CSSD)
  - (formerly Lynda.com)

- Ebooks and videos at O'Reilly
  - Free for Pitt students, faculty, and staff (library subscription)
  - (formerly Safari Ebooks)

How did we do?

- Please complete our survey!

- Check out our other workshops: https://pi.tt/ds-workshops
- Follow us on twitter: @DSSatPitt

- Contact me: Dominic Bordelon, djb190@pitt.edu

Image credits

- Cover slide: Photo by Ugur Akdemir on Unsplash