SQL Bootcamp

Wifi GA_GUEST

Pass: yellowpencil

Head to:

Github.com/morrisdata

Download

SQL bootcamp repository

Follow directions to install PGADMIN4

***note work devices may not work**

Value of Data

- **Objectives**
  - Set up Environment
  - Pgadmin4
  - Postgres DB
  - Connect to AWS
  - Connect to Local Database
  - Data Flow & SQL
  - Retrieving and filtering data with SQL
  - Aggregations with SQL
  - Dynamic data referencing with SQL (joins)
  - Demo 3 Value Logic/Case Statements/Subselects
- **Next steps**

GENERAL ASSEMBLY

Matthew Morris

Advanced Analytics Data Analyst, Costco
Data Analytics Course Instructor, General Assembly

Let’s get started… INSTALLAPOLOOZA!!!

Installations/Config

- Load PGADMIN and connect to AWS

- Load Postgres(bonus material not needed for 99% of todays class)

- Help fellow students connect if you finish

- As a class lets load a table together

- If you cannot get connected or configured you can group up with another stude
Or reschedule for another class time. Installs and configuration troubleshooting
Can depend on many factors and we wont have time to troubleshoot everyones
Devices

GENERAL ASSEMBLY

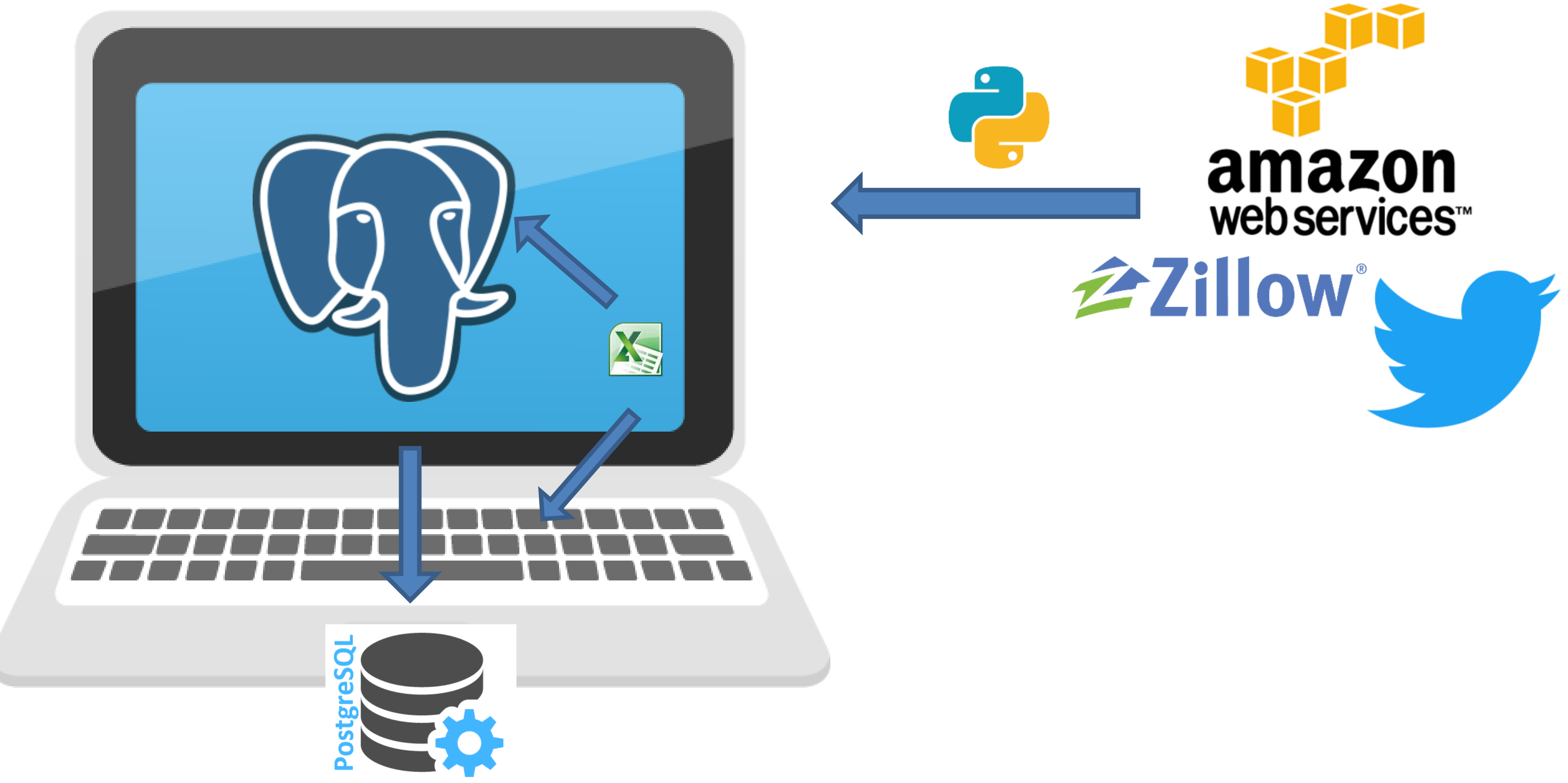
CREATING A LOCAL DATABASE

POSTGRES AND PGADMIN INSTALL
BRIEF OVERVIEW DATA MODELING

Matthew Morris
Git: Morrisdata
MatthewMorris.DA@gmail.com

OBTAINING AND MODELING DATA

Obtaining Data



Obtaining Data

Id	Release date	Record Label	Artist	Song	Album	sales
1						
2						
3						
4						
5						

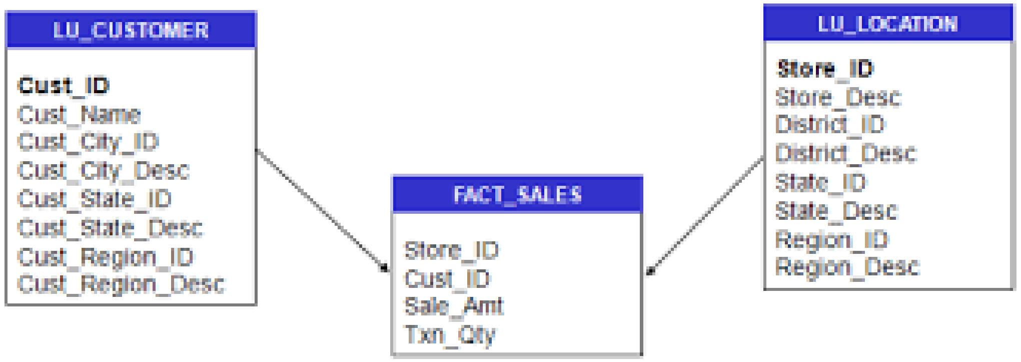
PostgreSQL

Id	Artist	Song	Album	sales
1				
2				
3				
4				
5				

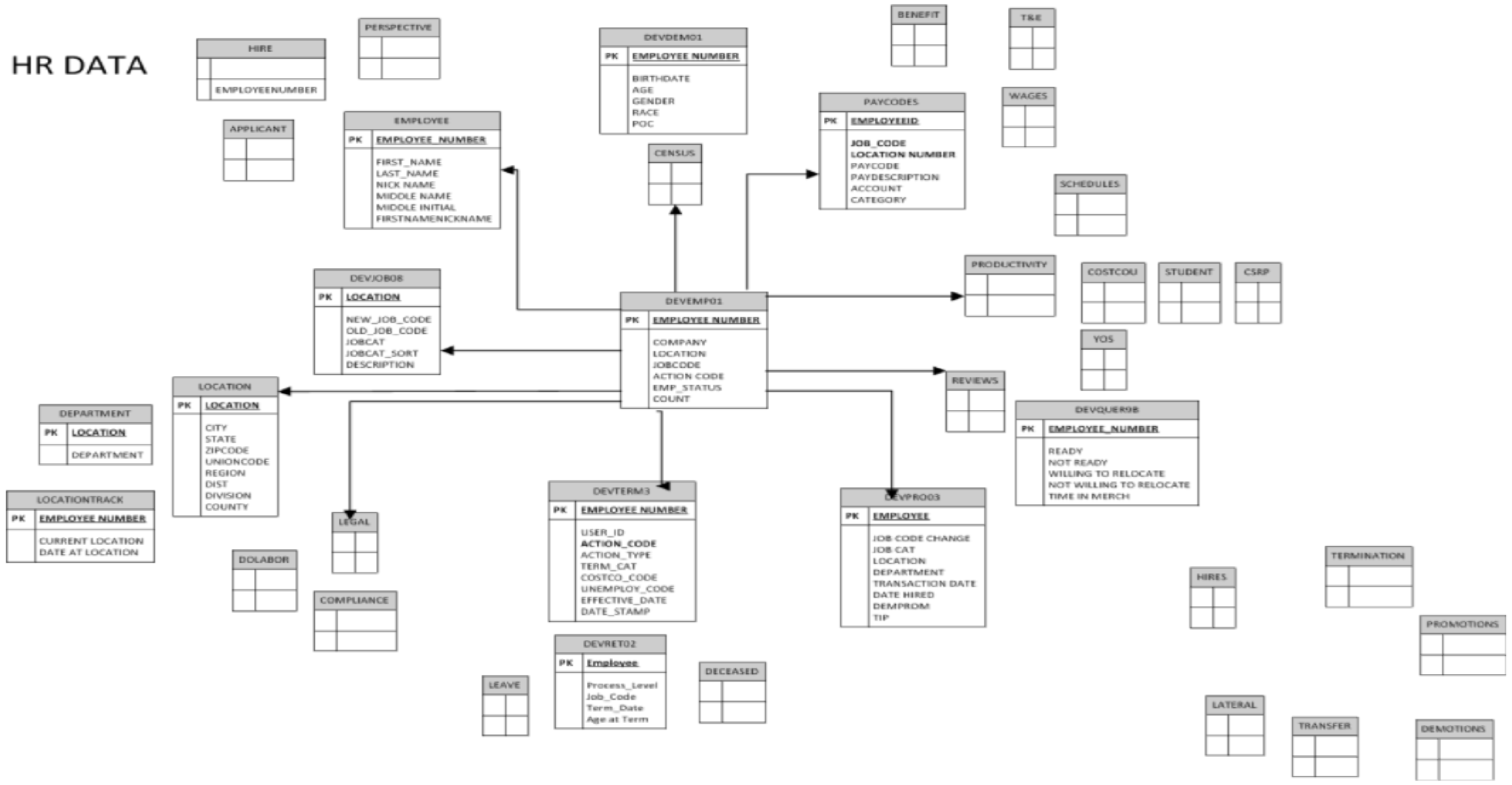
Obtaining Data

SALES

FIELD	TYPE	LENGTH
ID	PK	1
ARTIST	Char	25
SONG	Char	225
ALBUM	Char	225



Obtaining Data



Creating Table and loading Data

```
CREATE TABLE
schema.tablename
(
      field1    integer,
      field2    numeric,
      field3    character(30),
     field4    money
);
```

Creating Table and Loading Data

COPY table FROM ‘datasource’ Delimiter ‘,’ filetype Header;

GENERAL ASSEMBLY

FUNDAMENTALS OF DATAFLOW AND SQL

Explain where SQL fits in the dataflow
Retrieve and filter data with basic SQL
Navigate a Relational Database

SELECT

FROM

WHERE

ORDERBY

LIMIT

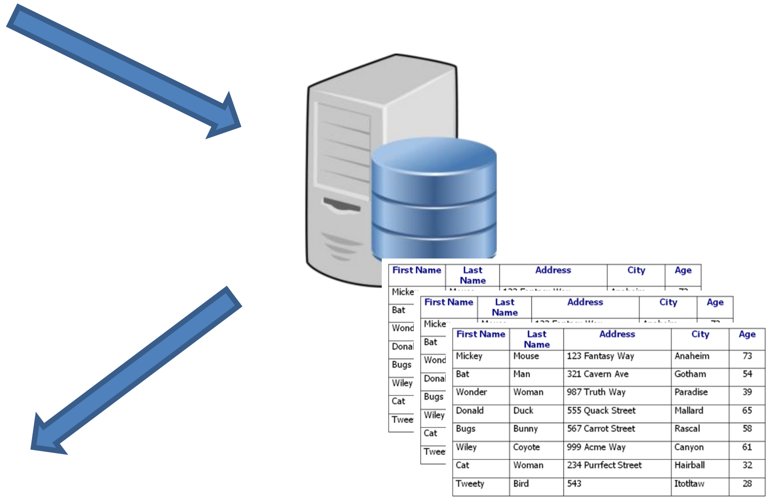
Matthew Morris
Git: Morrisdata
MatthewMorris.DA@gmail.com

FUNDAMENTALS OF DATAFLOW AND SQL

GATHERING REQUIREMENTS

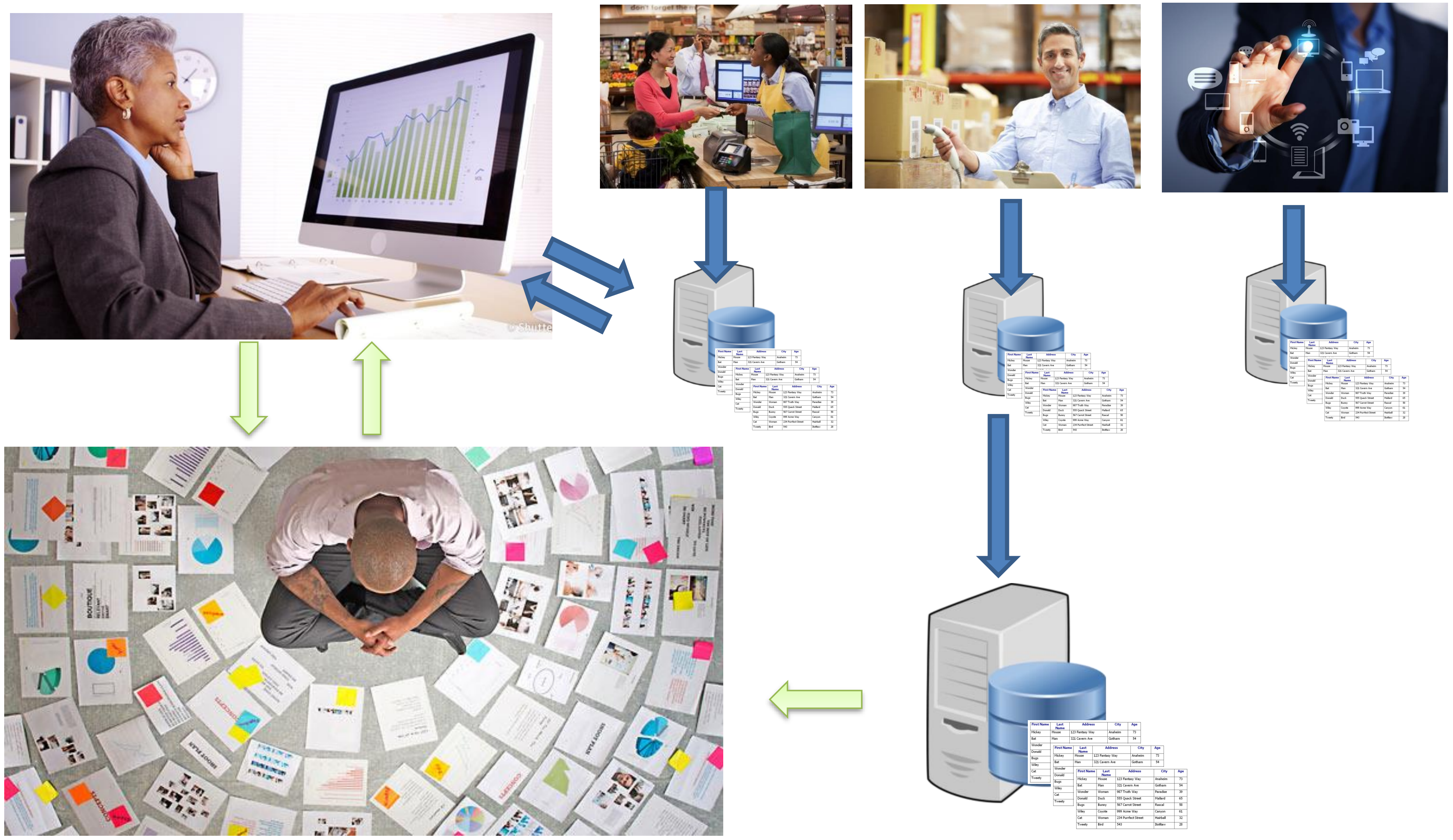
GATHERING REQUIREMENTS

GATHERING REQUIREMENTS

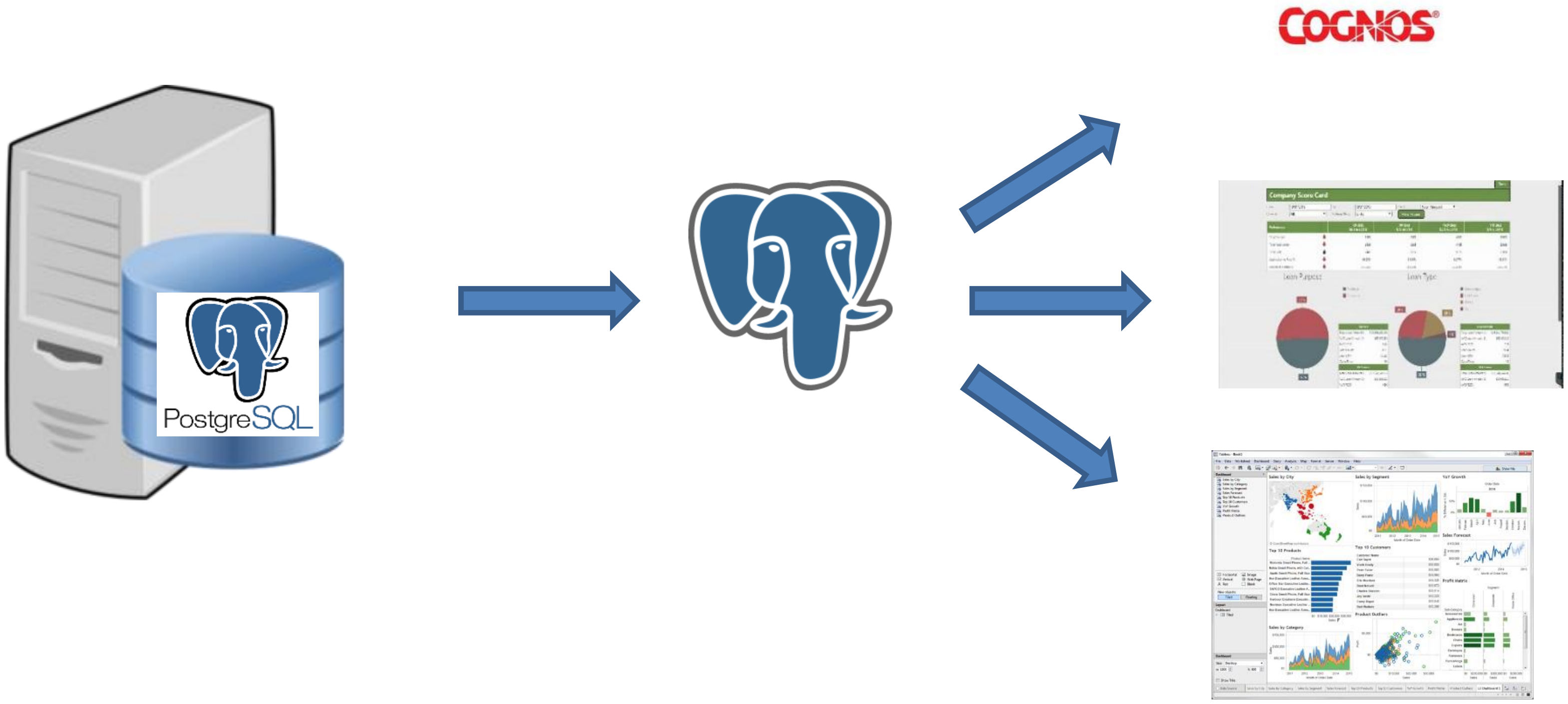


First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

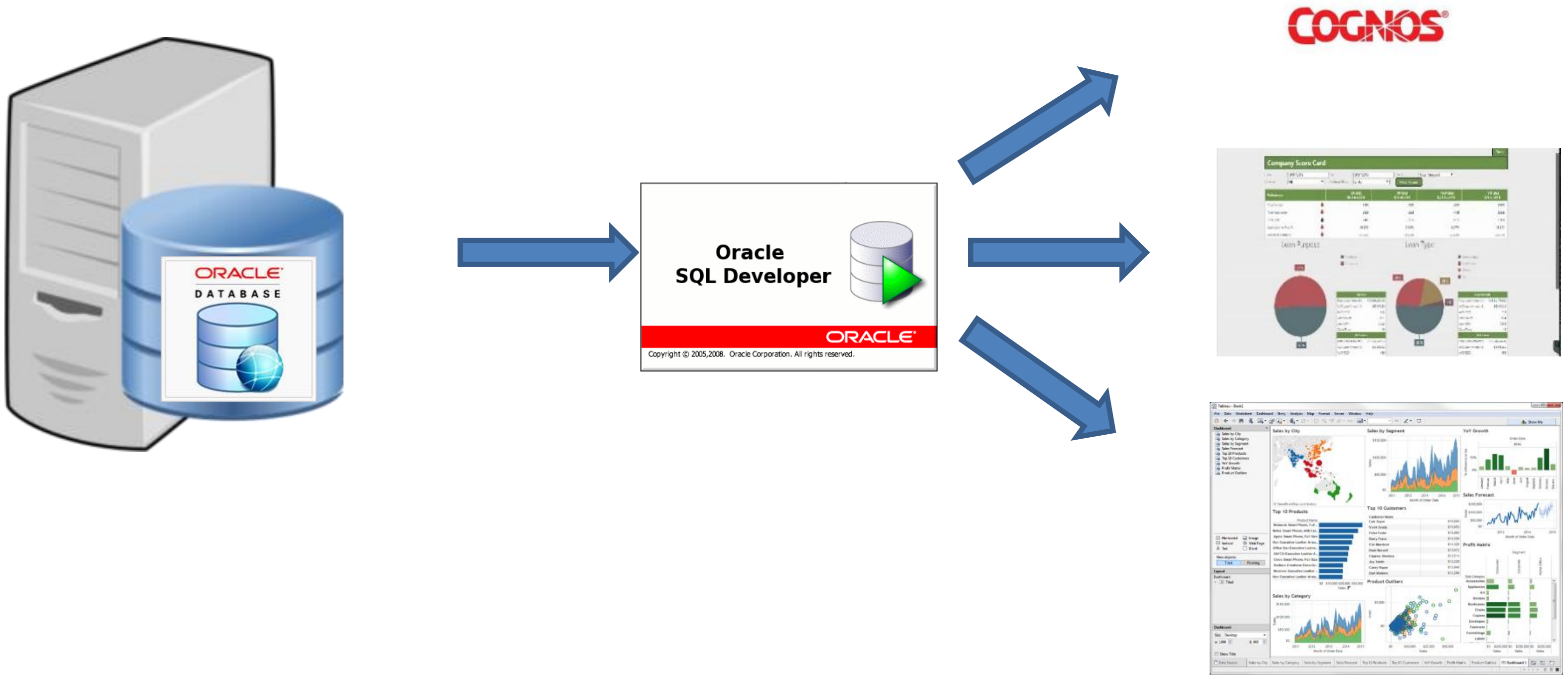
GATHERING REQUIREMENTS



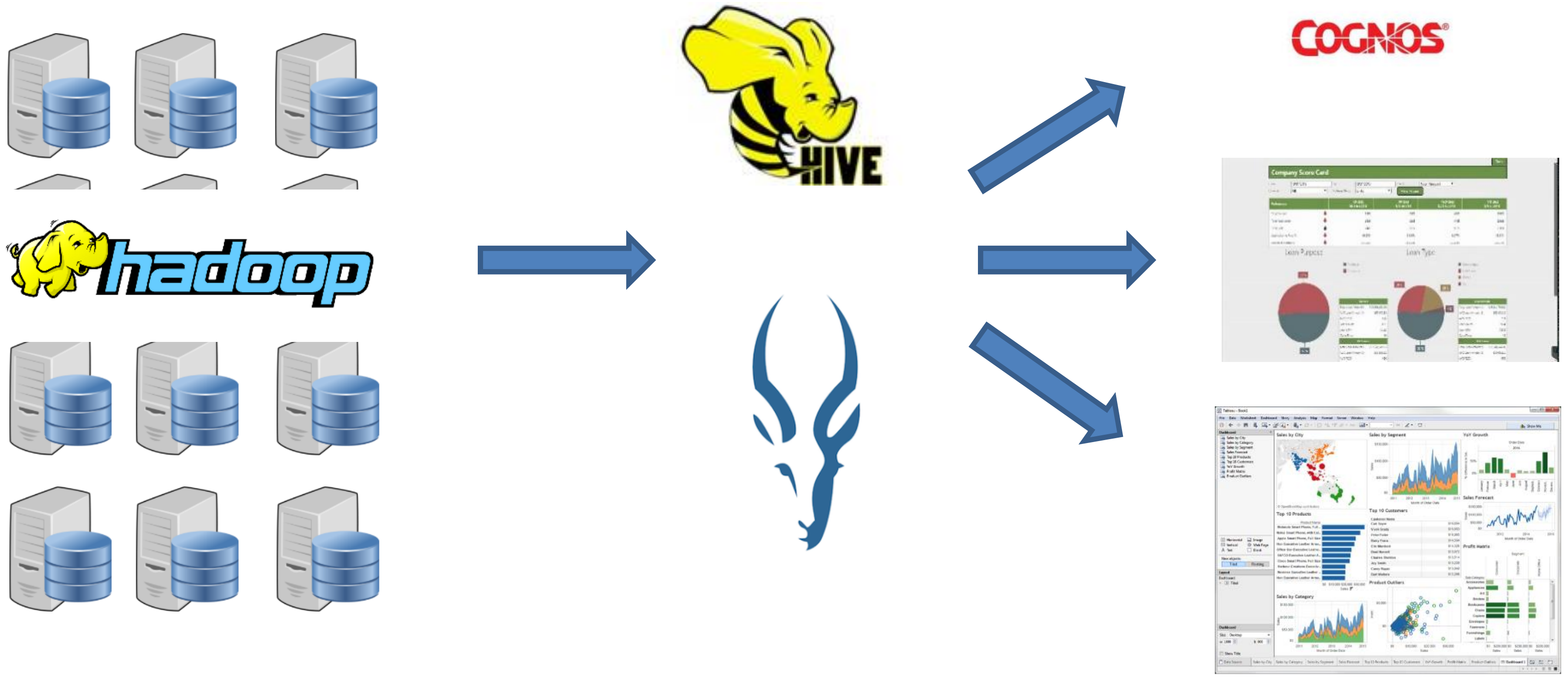
Basic Ecosystem



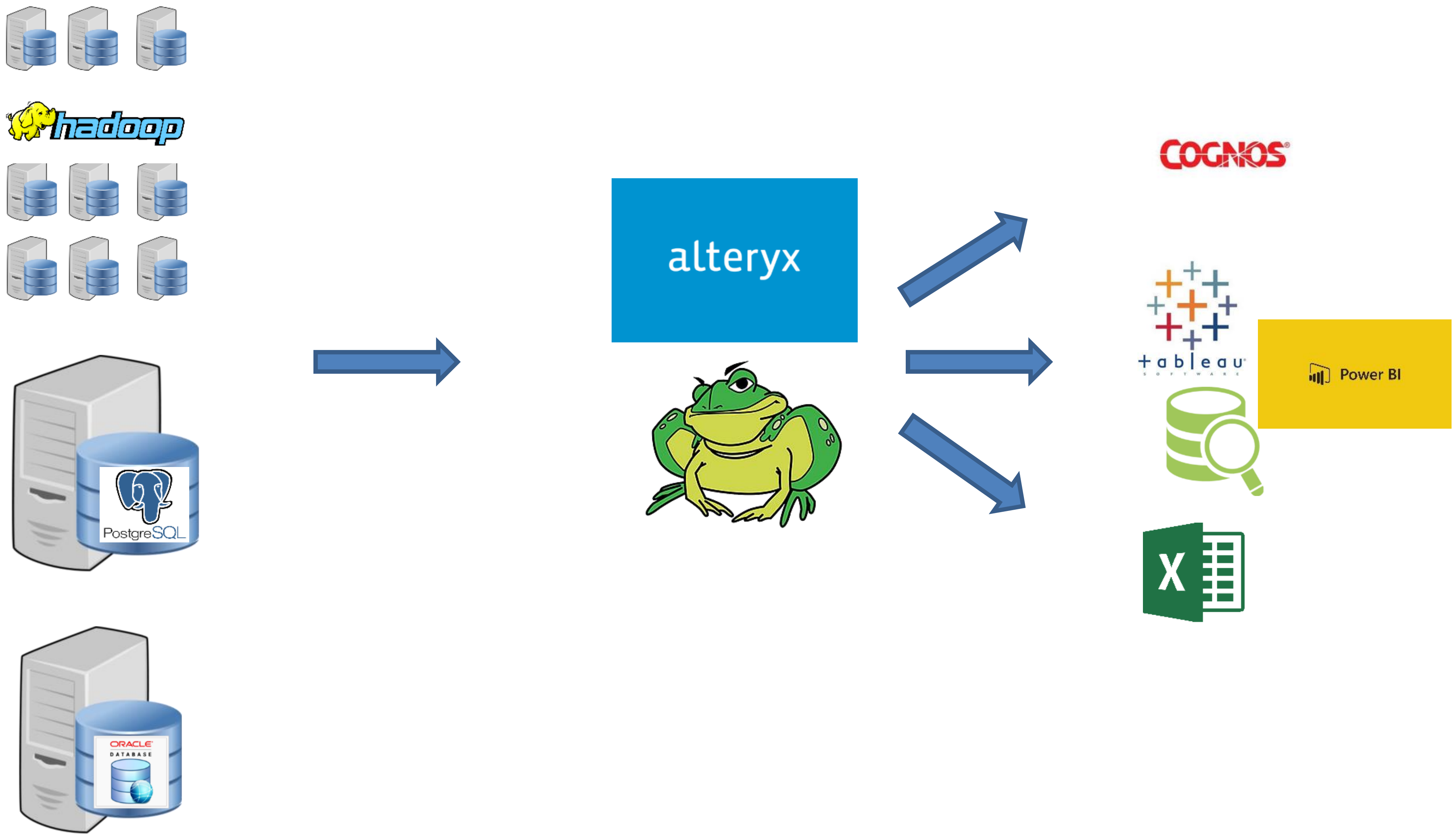
Basic Ecosystem


ORACLE
DATABASE
Oracle
SQL Developer
ORACLE
Copyright © 2005,2008. Oracle Corporation. All rights reserved.
COGNOS
Company Score Card

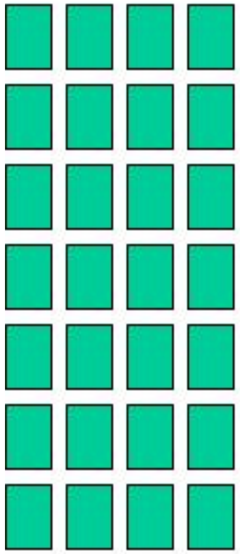
Basic Ecosystem



Basic Ecosystem



Relational DB



Libraries/Collections

Schema

Tables/Files/Objects

Members/Partitions in files

Review

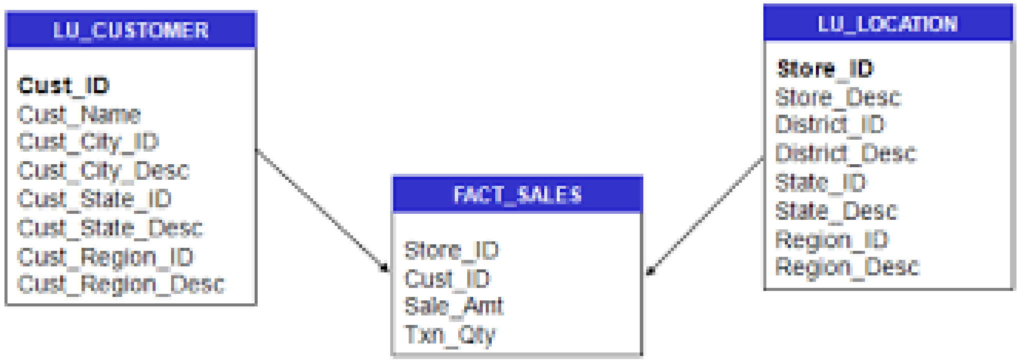
Id	Release date	Record Label	Artist	Song	Album	sales
1						
2						
3						
4						
5						

Id	Artist	Song	Album	sales
1				
2				
3				
4				
5				

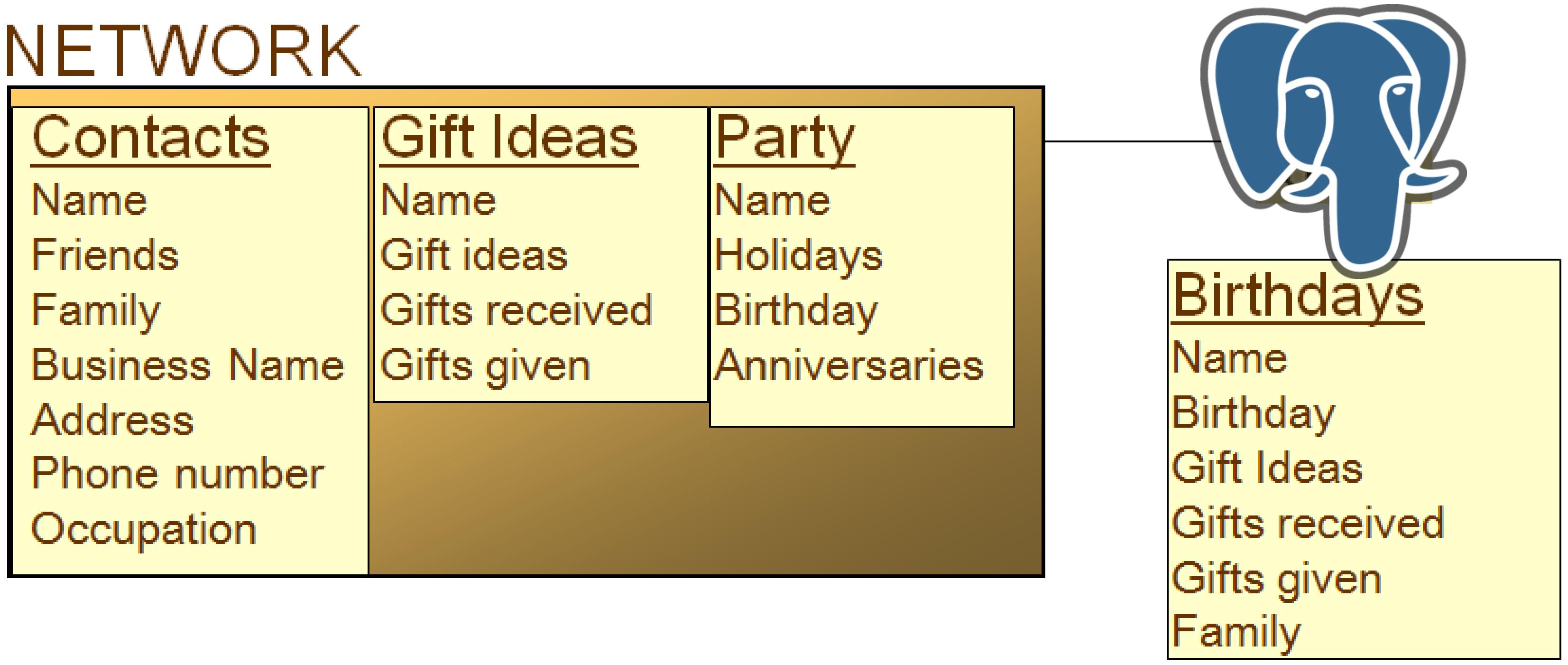
Review

SALES

FIELD	TYPE	LENGTH
ID	PK	1
ARTIST	Char	25
SONG	Char	225
ALBUM	Char	225



Relational DB

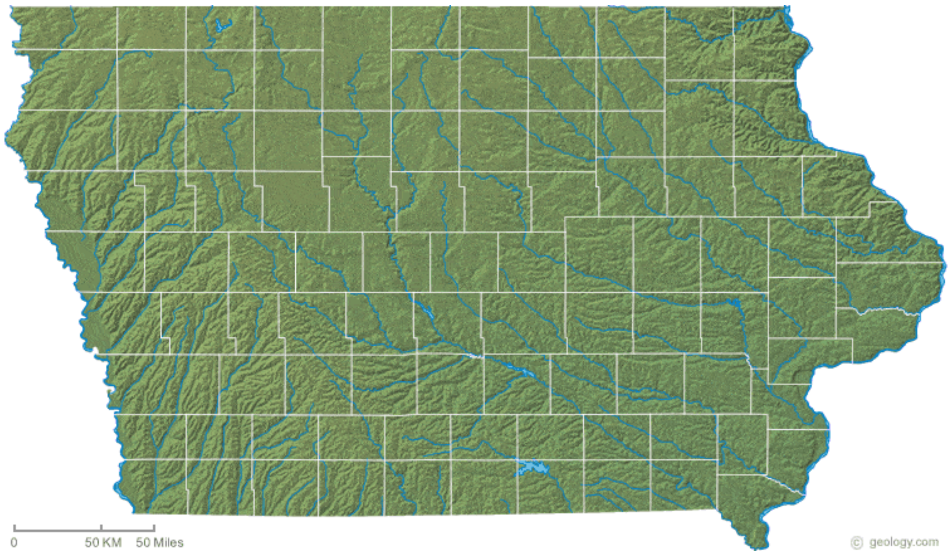


NETWORK

Contacts	Gift Ideas	Party
Name	Name	Name
Friends	Gift ideas	Holidays
Family	Gifts received	Birthday
Business Name	Gifts given	Anniversaries
Address		
Phone number		
Occupation		

Birthdays
Name
Birthday
Gift Ideas
Gifts received
Gifts given
Family

Relational DB



Understanding your data.
What are the tables?
What are the fields?
How might you link the tables?

Basic SQL

SELECT

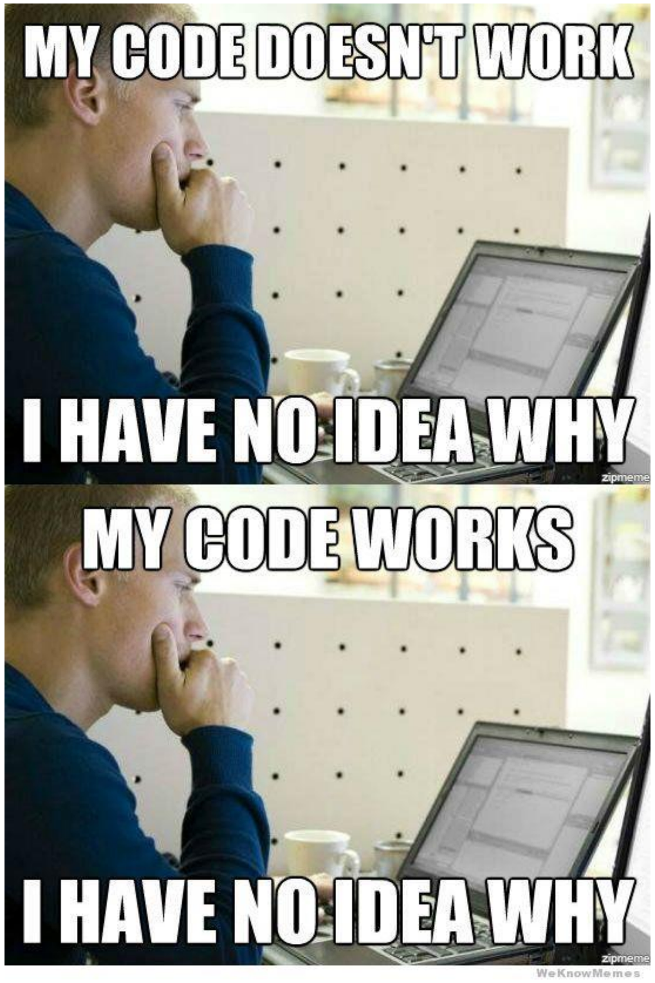
FROM

WHERE

ORDER BY

LIMIT

Commenting



```
-- Basic commenting

/* Multiple line
comment


End of Multiple line
comment*/
```

Basic SQL

SELECT *

SELECT FIELD1, FIELD2 …

SELECT (FIELD1+FIELD2), FIELD 3…

SELECT SUM(FIELD1), FIELD2

Basic SQL

CAST(field AS type)

A helpful String function

Basic SQL

Is Case Cost a discount for stores or for customers? Use SQL and field exploration and math in your statement to answer this question.

Basic SQL

```
SELECT DISTINCT  Location, NumberOfSales
```

```
Location     NumberOfSales
Seattle         101
Seattle          40
Tacoma           72
```

Basic SQL

SELECT DISTINCT Location, NumberOfSales, Date

Location	NumberOfSales	Date
Seattle	101	10/28/17
Seattle	101	10/27/17
Seattle	40	10/26/17
Tacoma	72	10/28/17
Tacoma	72	10/27/17

Basic SQL

```
WHERE COUNTRY = US

WHERE COUNTRY = US
AND STATE = WA

WHERE COUNTRY = US
AND STATE = WA
OR SALES > 100
```

Basic SQL

```
ORDER BY 1

ORDER BY 1,2 DESC
```

Basic SQL

LIMIT 1000

ROWNUM <= 1000

Basic SQL

1. Select various fields from the SALES table that interest you.

   *BE sure to use LIMIT 1000

2. Practice using filters.

3. Use AND to apply multiple filters

Change the sort.

4. Save your Query

Use your new skills to review Iowa Liquor Sales

GENERAL ASSEMBLY

Filters and Aggregations

FILTERS = , !=, >, <
IN, NOT IN, BETWEEN, LIKE, NOT LIKE
SUM, MIN, MAX, COUNT
GROUP BY, HAVING
COMMENTING

Matthew Morris
Git: Morrisdata
MatthewMorris.DA@gmail.com

Aggregations

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT

Filters

WHERE

- =, !=, >,<
- IS NULL, IS NOT NULL
- IN, NOT IN
- BETWEEN
- LIKE
- OR

Aggregate functions

- MIN
- MAX
- SUM
- COUNT

Aggregations

GROUP BY store, item

GROUP BY 1,2

Aggregations

```
HAVING AVG(sales)>100
AND COUNT(customers)>20
```

Aggregations

WHERE – filter for dimensions and measures

GROUP BY –groups dimensions when a measure is aggregated

HAVING – filter for aggregated measure

Aggregations

SELECT	- Fields you want to see in your results
FROM	- Table where fields come from
WHERE	- Filters for your results
GROUP BY	- Groups dimensions when using an aggregate
HAVING	- Filters aggregations
ORDER BY	- How you can sort your results
LIMIT	- Limits number of records returned

Fix the Code

```
SELECT Store, (cost –sell price),   SUM(sales),
FROM sales
WHERE Category = ‘Tequila’
AND units purchased >2
GROUP BY Store
HAVING SUM(sales) > 30.00
ORDER BY 3
```

Workshop

Which products have a case cost of more than $100?

Which tequilas have a case cost of more than $100?

Which tequilas or scotch whiskies have a case cost of more than $100?

Which tequilas or scotch whiskies have a cast cost between $100 and $120?

Which whiskies of any kind cost more than $100?'

Which whiskies of any kind cost between $100 and $150?

Which products except tequilas cost between $100 and $120?

GENERAL ASSEMBLY

Querying Relational Database

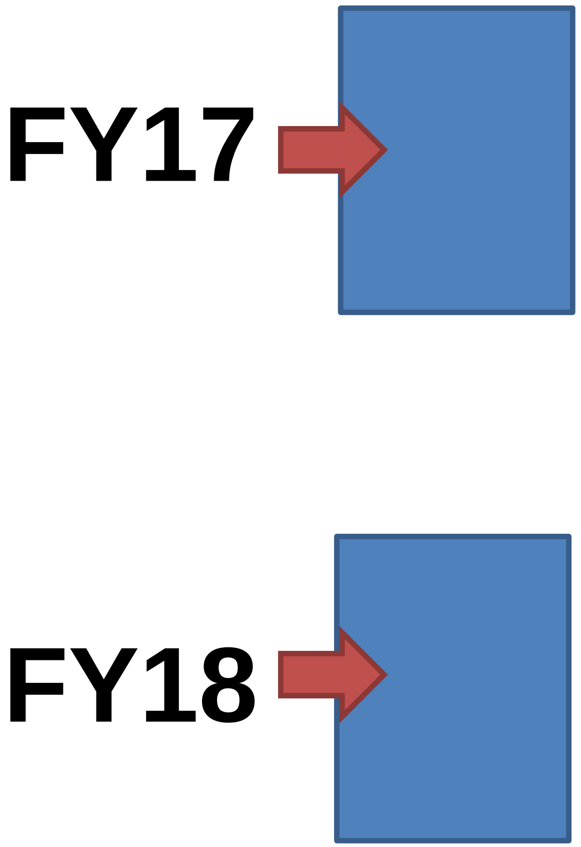
UNION
JOIN 2 Tables
JOIN Multiple Tables

Matthew Morris
Git: Morrisdata
MatthewMorris.DA@gmail.com

Query a Relational Database

SELECT
FROM
JOIN
ON
WHERE
GROUP BY
HAVING
UNION
ORDER BY
LIMIT

Unions

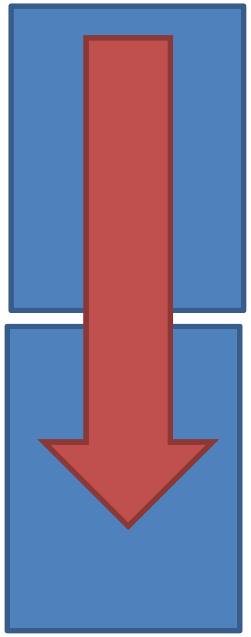


```
SELECT fy, pd, store_name, week1,
week2, week3 week4
FROM  FY17

UNION

SELECT fy, pd, store_name, week1,
week2, week3 week4
FROM  FY18
```

Unions

FY17

FY18



```
SELECT fy, pd, store_name, week1,
week2, week3 week4
FROM  FY17
UNION
SELECT fy, pd, store_name, week1,
week2, week3 week4
FROM  FY18
```

Unions

COLUMNS
CONDITIONS
UNION and UNION ALL
ORDER BY

QUERY A RELATIONAL DATABASE

JOIN 1 Table

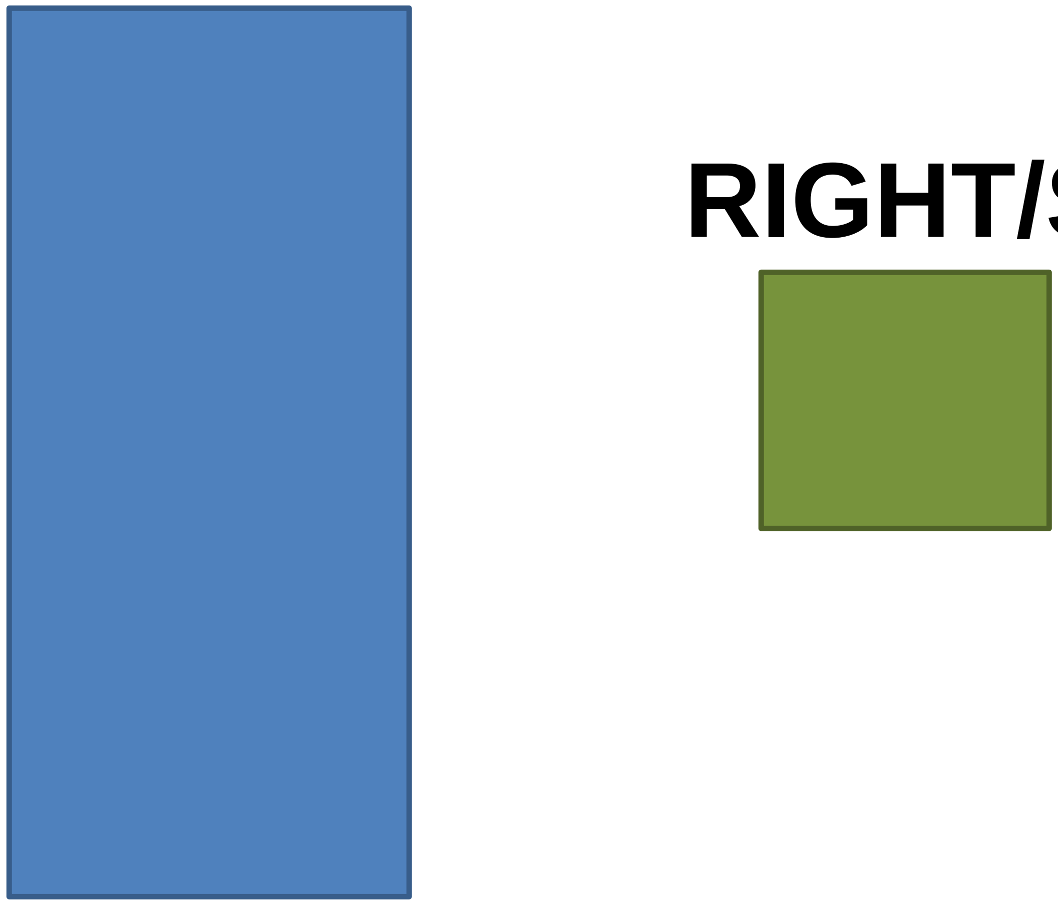
Joins

Joins

Joins

LEFT/PRIMARY

RIGHT/SECONDARY



Joins

What table is the transaction table?

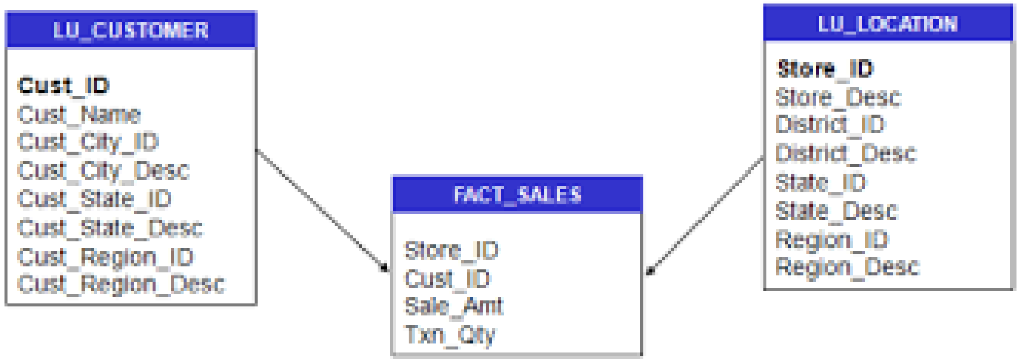
If you wanted to link on the lowest level of detail to the other tables what fields would you use?

Joins

SALES

FIELD	TYPE	LENGTH
ID	PK	1
ARTIST	Char	25
SONG	Char	225
ALBUM	Char	225

Create a rough sketch with how The Iowa Liquor Sales Database Would JOIN



Joins

```
SELECT a.item, b.description, a.sales

FROM sales a
JOIN products b
ON a.item=b.item
```

Joins

1. Create separate queries to join each table to Sales
    a. Products to Sales
    b. County to Sales
    c. Stores to Sales
2. Use this as an opportunity to bring fields in from both tables.
3. Try out some aggregations or Wild card searches. Stretch with an Aggregate and a Group by

Joins

```
SELECT c.field, a.field, b.field, a.field,c.field

FROM table1 a

JOIN table2 b
ON a.field=b.field

JOIN table3 c
ON a.field=c.field
```

GENERAL ASSEMBLY

Join types

LEFT OUTER
RIGHT OUTER
LEFT EXCEPTION
RIGHT EXCEPTION
CROSS
COALESCE

Matthew Morris

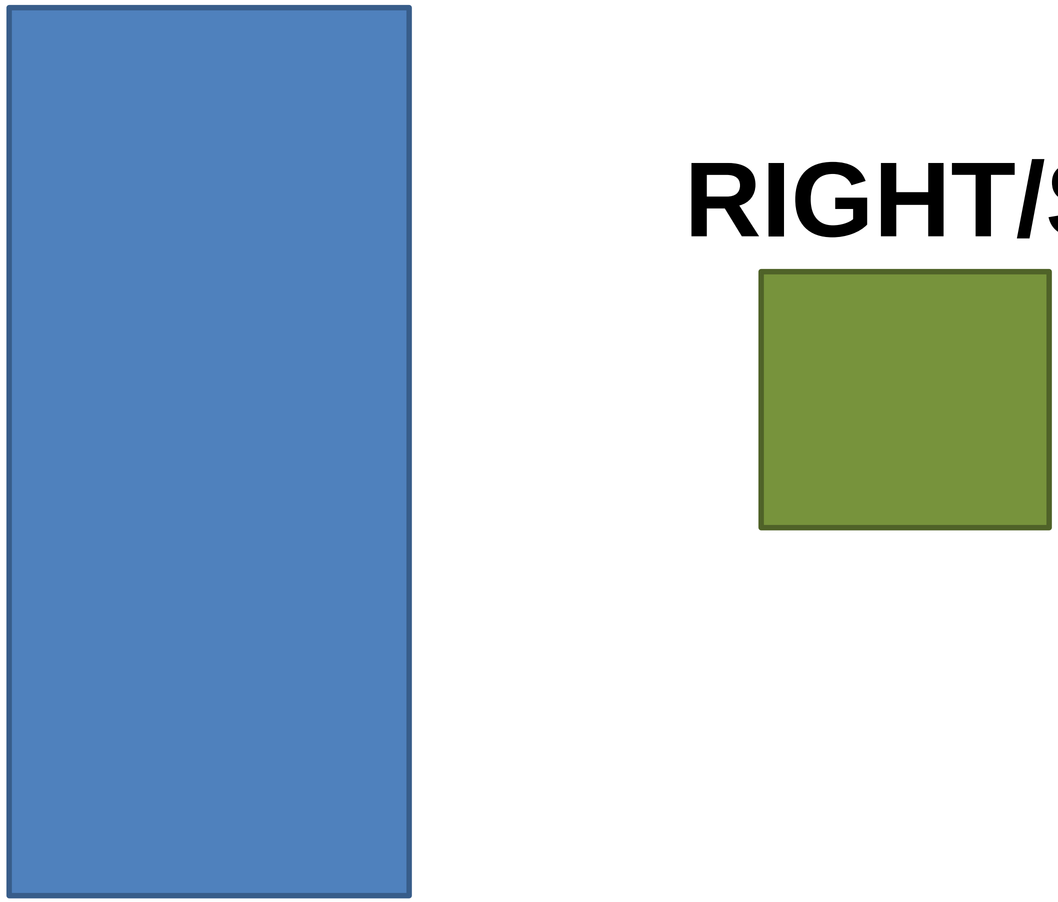
Git: Morrisdata

MatthewMorris.DA@gmail.com

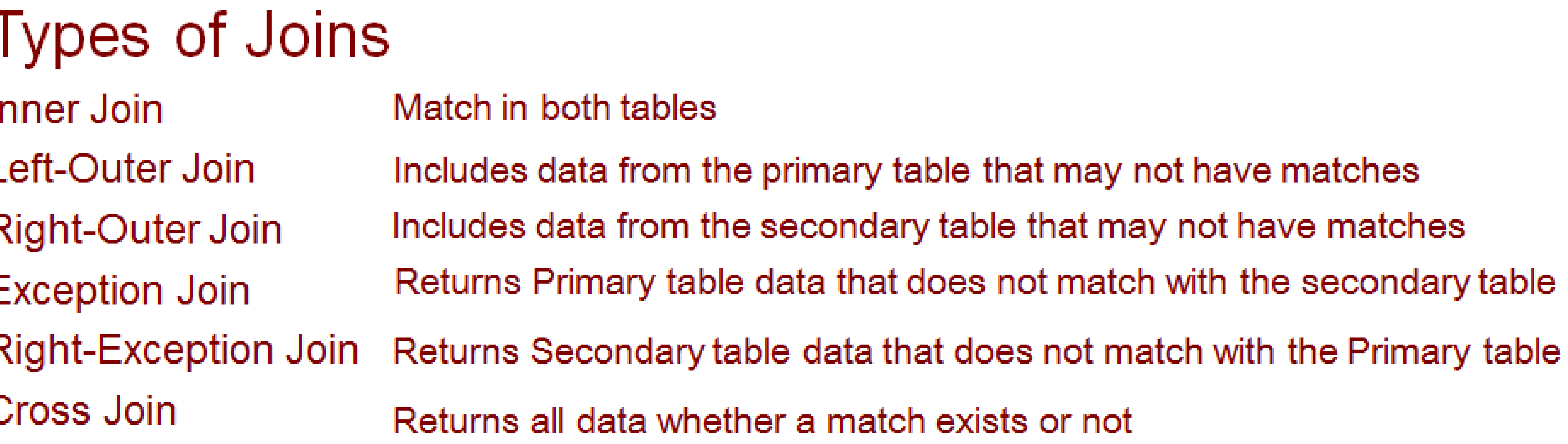
Joins

LEFT/PRIMARY

RIGHT/SECONDARY



Joins



Types of Joins

nner Join	Match in both tables
eft-Outer Join	Includes data from the primary table that may not have matches
Right-Outer Join	Includes data from the secondary table that may not have matches
Exception Join	Returns Primary table data that does not match with the secondary table
Right-Exception Join	Returns Secondary table data that does not match with the Primary table
Cross Join	Returns all data whether a match exists or not

Join Types

OUTER, EXCEPTION, AND CARTESIAN JOINS

Employees

id	first_name	last_name
2	Gabe	Moore
3	Doreen	Mandeville
5	Simone	MacDonald
7	Madisen	Flateman
11	Ian	Paasche
13	Mimi	St. Felix

Salaries

id	current_salary
2	50000
3	60000
7	55000
11	75000
13	120000
17	70000

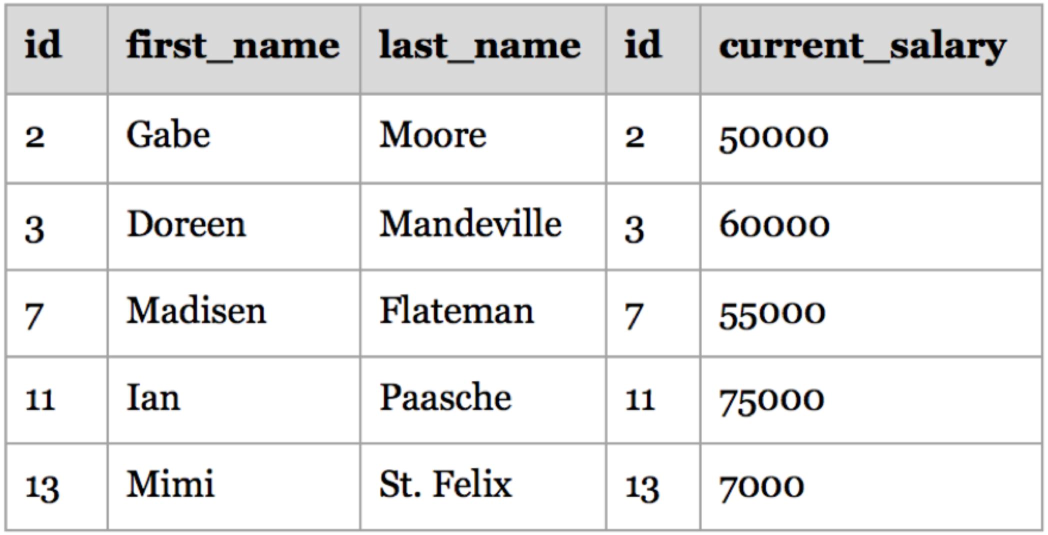
Join Types

OUTER, EXCEPTION, AND CARTESIAN JOINS

- An INNER JOIN (also called a direct join) displays only the rows that have a match in both joined tables.

- An INNER JOIN would yield this table:

SELECT * FROM employees INNER JOIN salaries ON employees.ID = salaries.ID;



id	first_name	last_name	id	current_salary
2	Gabe	Moore	2	50000
3	Doreen	Mandeville	3	60000
7	Madisen	Flateman	7	55000
11	Ian	Paasche	11	75000
13	Mimi	St. Felix	13	7000

Join Types

OUTER, EXCEPTION, AND CARTESIAN JOINS

A LEFT OUTER JOIN returns both:

- Data that both tables have in common.
- Data from the **primary** table selected, which does not have matching data to join to in the secondary table.

- A LEFT OUTER JOIN would yield this table:

```
SELECT * FROM employees LEFT OUTER JOIN salaries ON employees.ID = salaries.ID;
```

id	first_name	last_name	id	current_salary
2	Gabe	Moore	2	50000
3	Doreen	Mandeville	3	60000
5	Simone	MacDonald	NULL	NULL
7	Madisen	Flateman	7	55000
11	Ian	Paasche	11	75000
13	Mimi	St. Felix	13	120000

Join Types

OUTER, EXCEPTION, AND CARTESIAN JOINS

- A RIGHT OUTER JOIN returns both:
  - Data that two tables have in common.
  - Data from the **secondary** table selected, which does not have matching data to join to in the primary table.
- A RIGHT OUTER JOIN would yield this table:

```
SELECT * FROM employees RIGHT OUTER
JOIN salaries ON employees.ID = salaries.ID;
```

id	first_name	last_name	id	current_salary
2	Gabe	Moore	2	50000
3	Doreen	Mandeville	3	60000
7	Madisen	Flateman	7	55000
11	Ian	Paasche	11	75000
13	Mimi	St. Felix	13	120000
NULL	NULL	NULL	17	70000

Join Types

OUTER, EXCEPTION, AND CARTESIAN JOINS

- A FULL OUTER JOIN returns all data from each table, regardless of whether it has matching data in the other table.

- A FULL OUTER JOIN would yield this table:

SELECT * FROM employees FULL OUTER JOIN salaries ON employees.ID = salaries.ID;

id	first_name	last_name	id	current_salary
2	Gabe	Moore	2	50000
3	Doreen	Mandeville	3	60000
5	Simone	MacDonald	NULL	NULL
7	Madisen	Flateman	7	55000
11	Ian	Paasche	11	75000
13	Mimi	St. Felix	13	120000
NULL	NULL	NULL	17	70000

Join Types

OUTER, EXCEPTION, AND CARTESIAN JOINS

- An EXCEPTION JOIN returns only the data from the primary, or first table selected, which does not have matching data to join to in the secondary table.
- An EXCEPTION JOIN would yield this table:

```
SELECT * FROM employees LEFT OUTER
JOIN salaries ON employees.ID = salaries.ID
WHERE salaries.ID IS NULL;
```

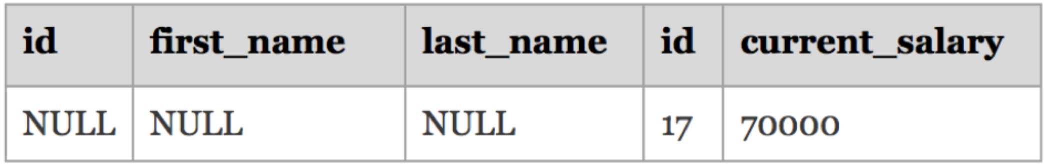
id	first_name	last_name	id	current_salary
5	Simone	MacDonald	NULL	NULL

Join Types

OUTER, EXCEPTION, AND CARTESIAN JOINS

- A RIGHT EXCEPTION JOIN returns only the data from the secondary, or second table selected, which does not have matching data to join to in the primary table.
- A RIGHT EXCEPTION JOIN would yield this table:

```
SELECT * FROM employees RIGHT OUTER
JOIN salaries ON employees.ID = salaries.ID
WHERE employees.ID IS NULL;
```



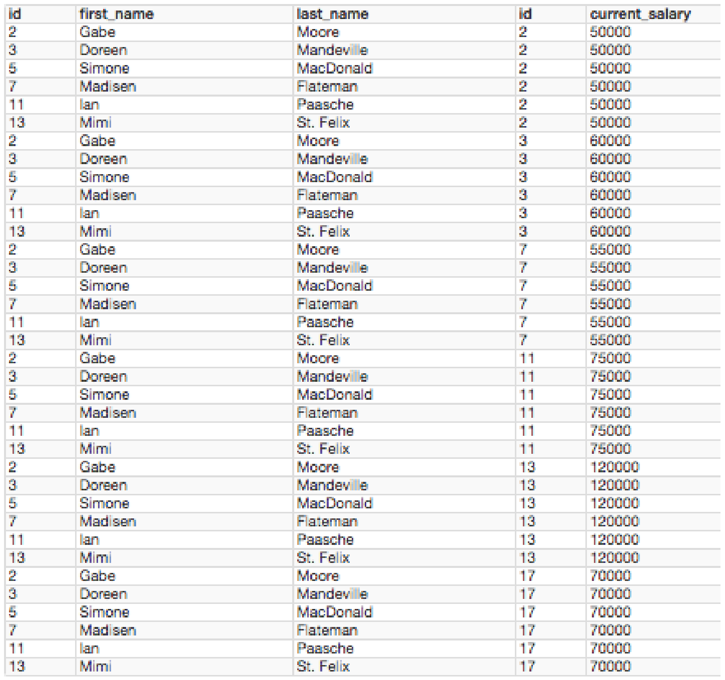
id	first_name	last_name	id	current_salary
NULL	NULL	NULL	17	70000

Join Types

OUTER, EXCEPTION, AND CARTESIAN JOINS

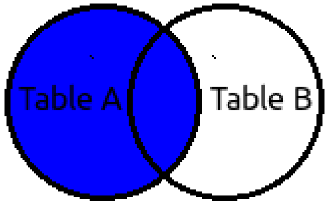
- A CROSS JOIN matches every row of the primary table with every row of the secondary table.
- This type of join results in a Cartesian product of the tables, is generally detrimental to slow performance, and is not desired.
- A CROSS JOIN would yield this table:

```
SELECT * FROM employees CROSS JOIN salaries;
```

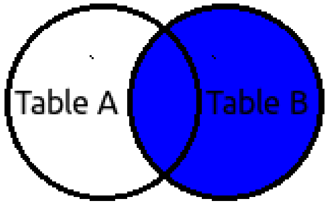


id	first_name	last_name	id	current_salary
2	Gabe	Moore	2	50000
3	Doreen	Mandeville	2	50000
5	Simone	MacDonald	2	50000
7	Madisen	Flateman	2	50000
11	Ian	Paasche	2	50000
13	Mimi	St. Felix	2	50000
2	Gabe	Moore	3	60000
3	Doreen	Mandeville	3	60000
5	Simone	MacDonald	3	60000
7	Madisen	Flateman	3	60000
11	Ian	Paasche	3	60000
13	Mimi	St. Felix	3	60000
2	Gabe	Moore	7	55000
3	Doreen	Mandeville	7	55000
5	Simone	MacDonald	7	55000
7	Madisen	Flateman	7	55000
11	Ian	Paasche	7	55000
13	Mimi	St. Felix	7	55000
2	Gabe	Moore	11	75000
3	Doreen	Mandeville	11	75000
5	Simone	MacDonald	11	75000
7	Madisen	Flateman	11	75000
11	Ian	Paasche	11	75000
13	Mimi	St. Felix	11	75000
2	Gabe	Moore	13	120000
3	Doreen	Mandeville	13	120000
5	Simone	MacDonald	13	120000
7	Madisen	Flateman	13	120000
11	Ian	Paasche	13	120000
13	Mimi	St. Felix	13	120000
2	Gabe	Moore	17	70000
3	Doreen	Mandeville	17	70000
5	Simone	MacDonald	17	70000
7	Madisen	Flateman	17	70000
11	Ian	Paasche	17	70000
13	Mimi	St. Felix	17	70000

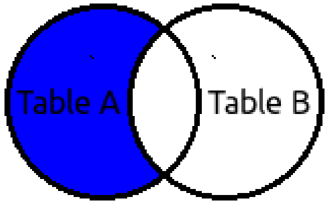
Join Types



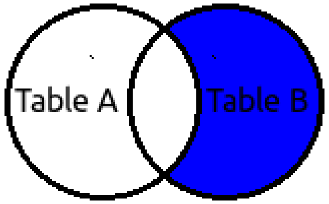
```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
```



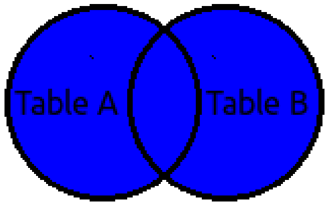
```
SELECT [list] FROM
    [Table A] A
RIGHT JOIN
    [Table B] B
ON A.Value = B.Value
```



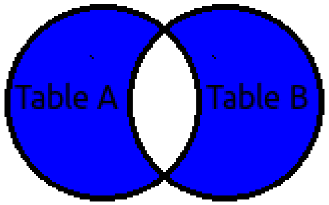
```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
WHERE B.Value IS NULL
```



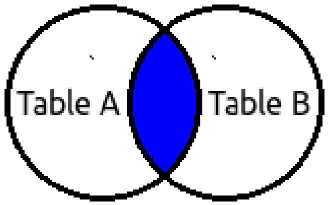
```
SELECT [list] FROM
    [Table A] A
RIGHT JOIN
    [Table B] B
ON A.Value = B.Value
WHERE A.Value IS NULL
```



```
SELECT [list] FROM
    [Table A] A
FULL OUTER JOIN
    [Table B] B
ON A.Value = B.Value
```



```
SELECT [list] FROM
    [Table A] A
FULL OUTER JOIN
    [Table B] B
ON A.Value = B.Value
WHERE A.Value IS NULL
OR B.Value IS NULL
```



```
SELECT [list] FROM
    [Table A] A
INNER JOIN
    [Table B] B
ON A.Value = B.Value
```

Join Types

"I want to see all of the information we can get on inactive stores for sales, if there are any, and their addresses."

Process to pick the right join (be methodical)

GENERAL ASSEMBLY

Common Table Expressions/Subselect

WITH
SELECT INTO

Matthew Morris
Git: Morrisdata
MatthewMorris.DA@gmail.com

GENERAL ASSEMBLY

Previously in Data Analytics

LEFT OUTER
RIGHT OUTER
LEFT EXCEPTION
RIGHT EXCEPTION
CROSS
COALESCE

Cte

WITH

SELECT INTO

FROM

JOIN

ON

WHERE

GROUP BY

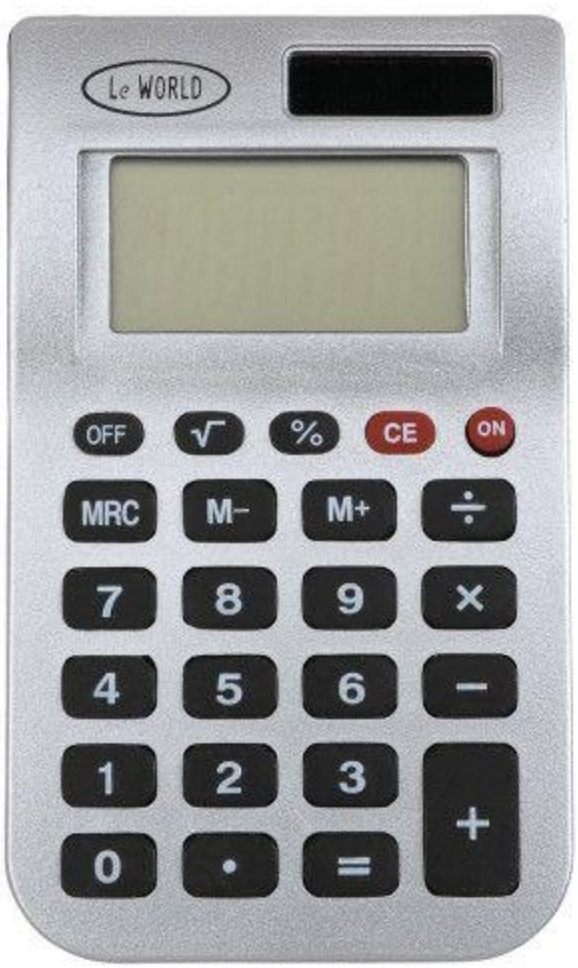
HAVING

UNION

ORDER BY

LIMIT

Cte

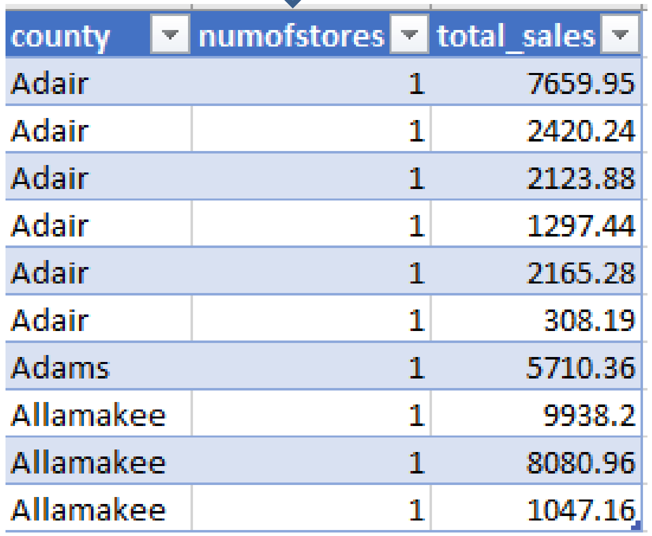


A common table expression (CTE) can be thought of as a temporary result set that is defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement. A CTE is similar to a derived table in that it is not stored as an object and lasts only for the duration of the query. Unlike a derived table, a CTE can be self-referencing and can be referenced multiple times in the same query.

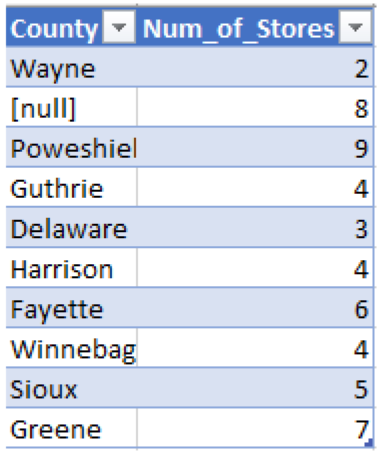
Cte

```
SELECT county, COUNT(DISTINCT store) AS NumOfStores, SUM(total) AS total_sales
FROM sales
WHERE category_name LIKE '%TEQUILA%'
GROUP BY county, store
HAVING count(store)>2 AND sum(total)<10000
```

```
SELECT county, SUM(NumOfStores)
FROM county_sales
GROUP BY county
```



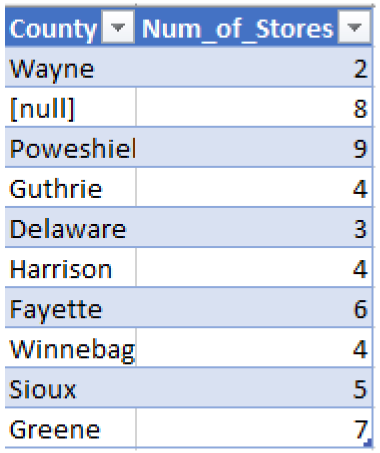
county	numofstores	total_sales
Adair	1	7659.95
Adair	1	2420.24
Adair	1	2123.88
Adair	1	1297.44
Adair	1	2165.28
Adair	1	308.19
Adams	1	5710.36
Allamakee	1	9938.2
Allamakee	1	8080.96
Allamakee	1	1047.16



County	Num_of_Stores
Wayne	2
[null]	8
Poweshiel	9
Guthrie	4
Delaware	3
Harrison	4
Fayette	6
Winnebag	4
Sioux	5
Greene	7

Cte

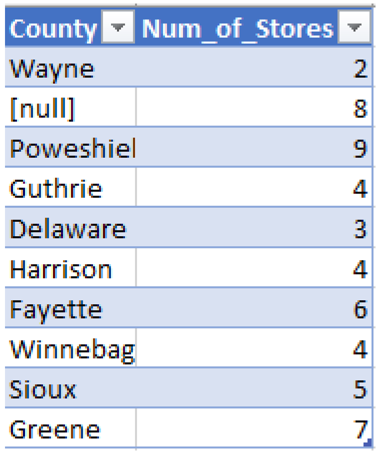
```
 WITH county_sales AS (
                  SELECT county, COUNT(DISTINCT store) AS NumOfStores, SUM(total) AS total_sales
                   FROM sales
                   WHERE category_name LIKE '%TEQUILA%'
                   GROUP BY county, store
                   HAVING count(store)>2 AND sum(total)<10000
                   )
SELECT county,
      SUM(NumOfStores)
FROM county_sales
GROUP BY county
```



County	Num_of_Stores
Wayne	2
[null]	8
Poweshiel	9
Guthrie	4
Delaware	3
Harrison	4
Fayette	6
Winnebag	4
Sioux	5
Greene	7

Cte

```
SELECT county, SUM(NumOfStores)
FROM
    (SELECT county, store, COUNT(DISTINCT store) AS NumOfStores, SUM(total) AS total_sales
     FROM sales
     WHERE Category_name LIKE '%TEQUILA%'
     GROUP BY county, store
     HAVING count(store)>20 AND sum(total)>500000 ) AS county_sales
GROUP BY county
```



County	Num_of_Stores
Wayne	2
[null]	8
Poweshiel	9
Guthrie	4
Delaware	3
Harrison	4
Fayette	6
Winnebag	4
Sioux	5
Greene	7

DEMO(aggregate over an aggregate)

ACTIVITY: AGGREGATE OVER AN AGGREGATE

EXERCISE

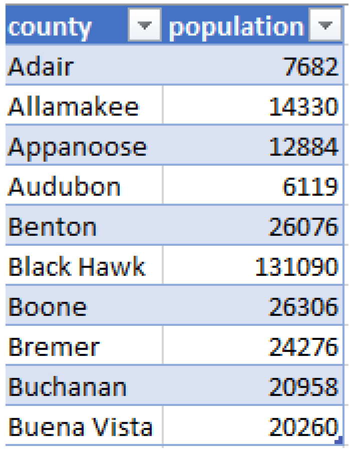
What is the total sum of stores by county?

Cte

```
WITH
a AS (SELECT county, population
FROM counties
WHERE population >5000),

b AS (SELECT county, sum(total) as total_sales
FROM sales
GROUP BY  county)
```

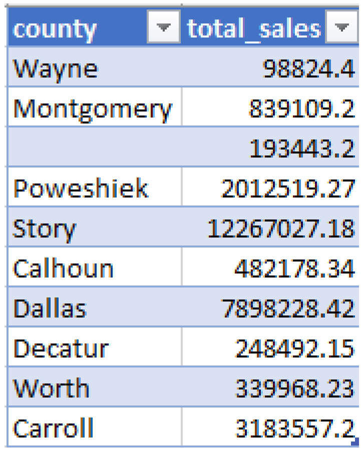
```
SELECT *
FROM a
JOIN b
ON a.county = b.county
```

a

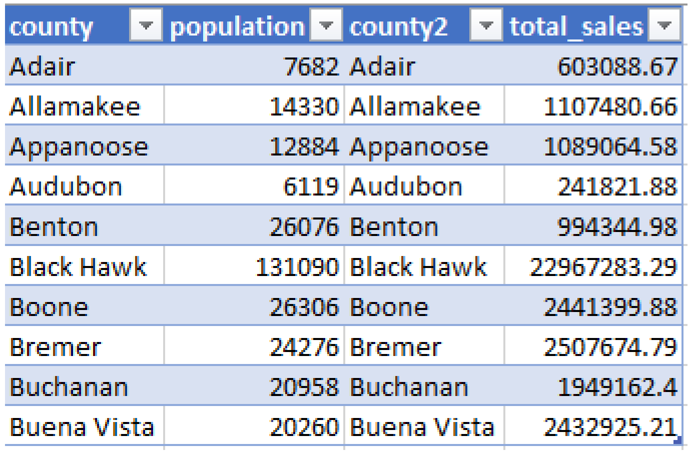


county	population
Adair	7682
Allamakee	14330
Appanoose	12884
Audubon	6119
Benton	26076
Black Hawk	131090
Boone	26306
Bremer	24276
Buchanan	20958
Buena Vista	20260

b



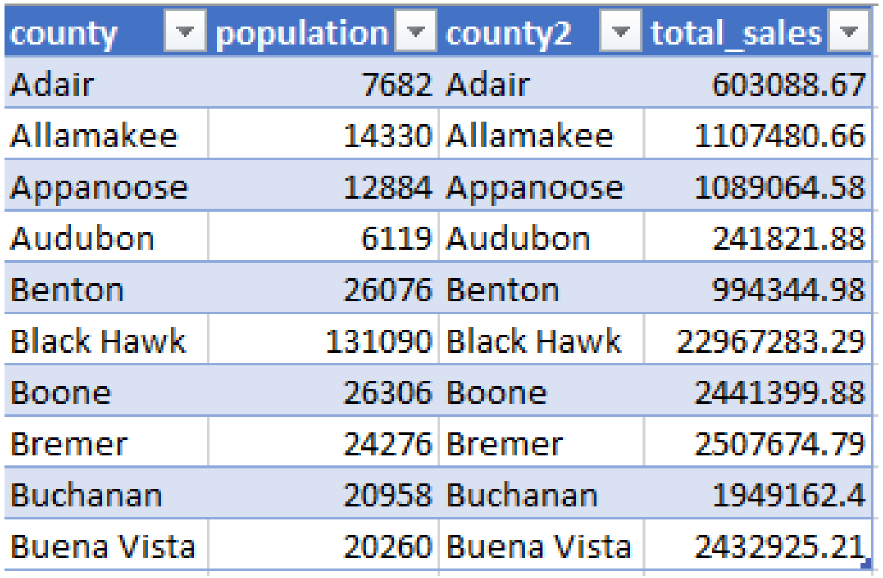
county	total_sales
Wayne	98824.4
Montgomery	839109.2
	193443.2
Poweshiek	2012519.27
Story	12267027.18
Calhoun	482178.34
Dallas	7898228.42
Decatur	248492.15
Worth	339968.23
Carroll	3183557.2



county	population	county2	total_sales
Adair	7682	Adair	603088.67
Allamakee	14330	Allamakee	1107480.66
Appanoose	12884	Appanoose	1089064.58
Audubon	6119	Audubon	241821.88
Benton	26076	Benton	994344.98
Black Hawk	131090	Black Hawk	22967283.29
Boone	26306	Boone	2441399.88
Bremer	24276	Bremer	2507674.79
Buchanan	20958	Buchanan	1949162.4
Buena Vista	20260	Buena Vista	2432925.21

Cte

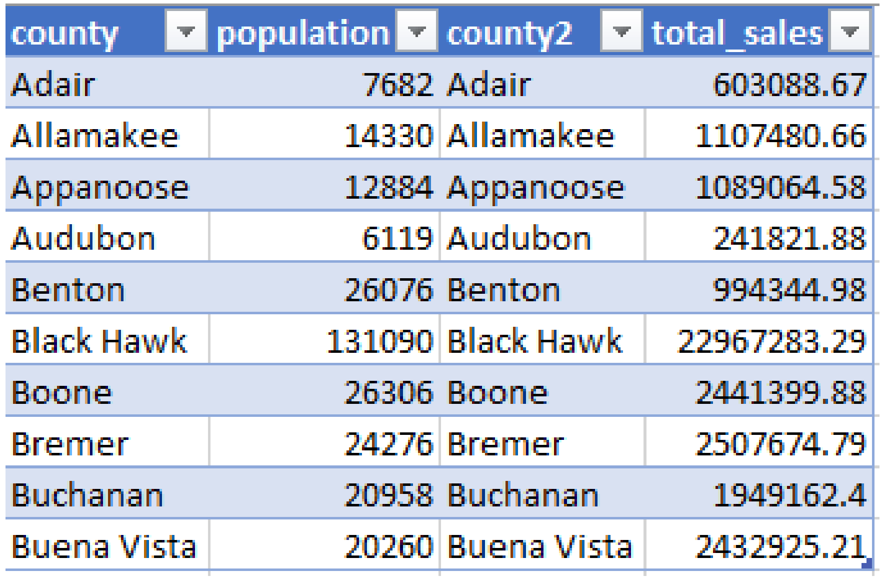
```
WITH
a AS (SELECT county, population
FROM counties
WHERE population >5000),

b AS (SELECT county, sum(total) as total_sales
FROM sales
GROUP BY  county)

SELECT *
FROM a
JOIN b
ON a.county = b.county
```



county	population	county2	total_sales
Adair	7682	Adair	603088.67
Allamakee	14330	Allamakee	1107480.66
Appanoose	12884	Appanoose	1089064.58
Audubon	6119	Audubon	241821.88
Benton	26076	Benton	994344.98
Black Hawk	131090	Black Hawk	22967283.29
Boone	26306	Boone	2441399.88
Bremer	24276	Bremer	2507674.79
Buchanan	20958	Buchanan	1949162.4
Buena Vista	20260	Buena Vista	2432925.21

Cte

```
SELECT *
FROM

(SELECT county, population
FROM counties
WHERE population >5000) as a

JOIN

(SELECT county, sum(total) as total_sales
FROM sales
GROUP BY  county) as b

ON a.county = b.county
```



county	population	county2	total_sales
Adair	7682	Adair	603088.67
Allamakee	14330	Allamakee	1107480.66
Appanoose	12884	Appanoose	1089064.58
Audubon	6119	Audubon	241821.88
Benton	26076	Benton	994344.98
Black Hawk	131090	Black Hawk	22967283.29
Boone	26306	Boone	2441399.88
Bremer	24276	Bremer	2507674.79
Buchanan	20958	Buchanan	1949162.4
Buena Vista	20260	Buena Vista	2432925.21

DEMO(joining temp tables)

ACTIVITY: JOINING temp Tables

EXERCISE

Create a process that looks at store and store count
LIMIT to 5 for a sample data set
We also want to review a sum of sales for anything like tequila
USE CTE’s for this as we may want some more stats to add later