

SQL For Data Science

Feb 7th - Josiah Baker, Steve Mortimer

Why Learn SQL?

Fundamental Data Science Skills

1. **Ubiquitous** - It's everywhere in the data community
2. SQL teaches you **Relational Algebra** and **Set Theory**
3. Expands Your **Computing Power**
4. Ease of **Access** - You can get started today

What is a Relational Database?

Key Terminology and Concepts

- **Rows** represent single items (“Josiah”)
- **Columns** are labeled attributes of an Item (“Occupation: Data Scientist”, “Employer: ForRent”)
- **Tables** are sets of rows that share the same attributes (“Employees”)
- **Views** are sets of any rows from different tables in response to a query (“Get me all Data Scientists in DE”)
- Proposed in 1970 by E. F. Codd while working at IBM

Why Relational Databases?

Key Terminology and Concepts

Relational data models:

- Are very easy to extend
- Improve and maintain data integrity
- Allow for re-assembling data many different ways without reorganizing the underlying structure

“Normalization” is the extent to which the data replicated into multiple places within the database.

Highly Normalized = Little Redundancy = More Joins

Denormalized = More Redundancy = Less Joins

Getting Started

The AdventureWorks Database

Adventure Works Cycles is a fictitious bicycle manufacturer

Scenarios include:

- Manufacturing
- Sales
- Purchasing
- Product Management
- Contact Management
- Human Resources

Data needs to be hosted on a running database!



Getting Started

Connecting to the Database

Connecting to the database server requires a set of credentials

You will need:

- Host - *ec2-23-21-219-105.compute-1.amazonaws.com*
- Port - *5432*
- Username - *dcppkvqqofzvp*
- Password - *12709a63bbb19e91b950008dab2d0301df1d48a6853d43cdf43964fcc863b6db*
- Database Name - *d6bkkvg5ahrdo*

Check the database is running: (open your command window)

```
telnet ec2-23-21-219-105.compute-1.amazonaws.com 5432
```

HOST

PORT

Getting Started

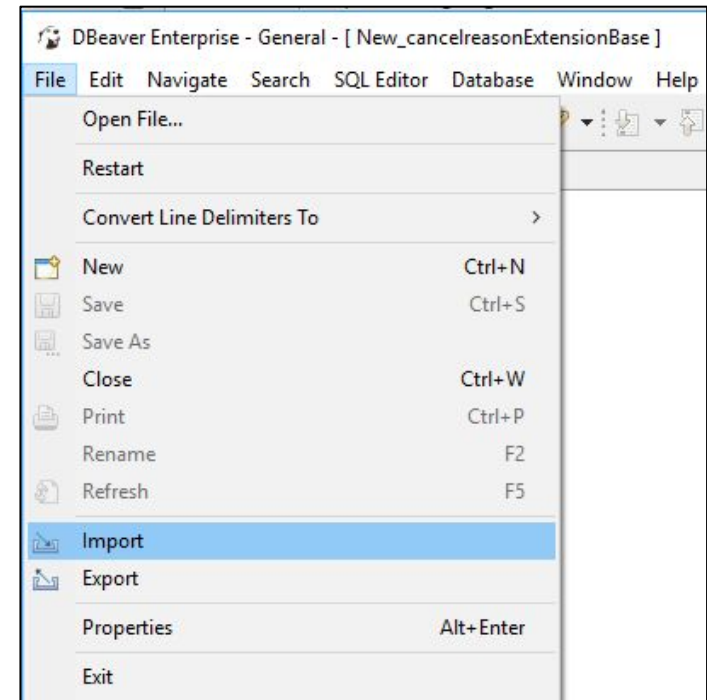
DBeaver - A SQL Client

A SQL Client is a program to query databases

First, install DBeaver - <http://dbeaver.jkiss.org/download/>

Second, download profile (contains connection and SQL scripts)

Third, import the profile into DBeaver
Open up DBeaver -> File -> Import



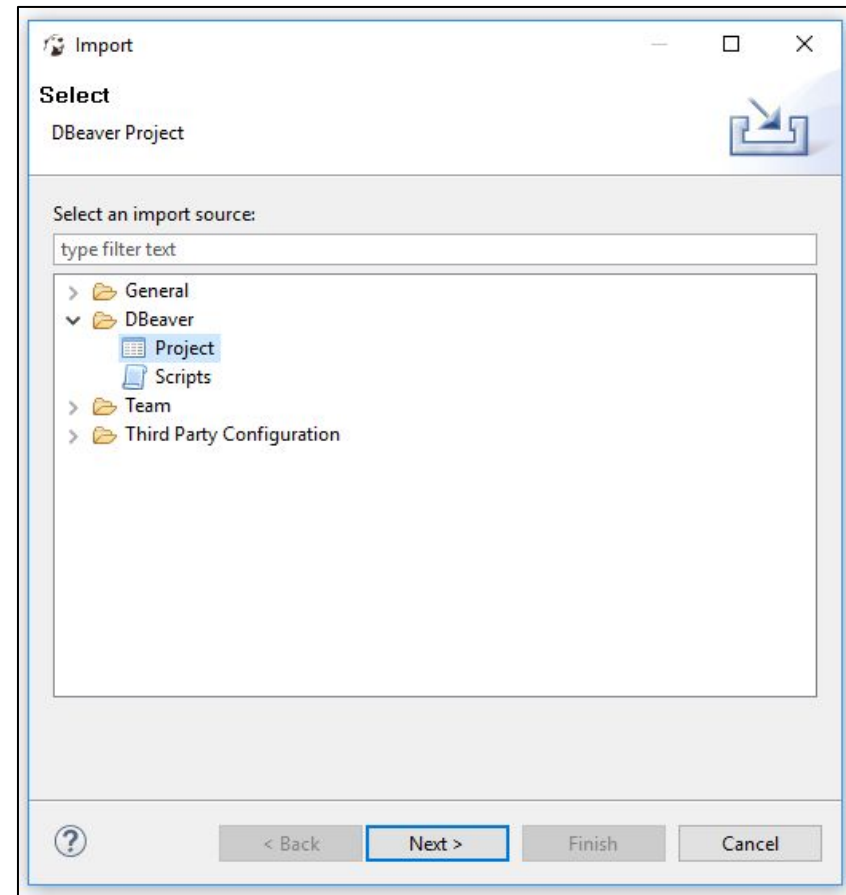
Getting Started

DBeaver - A SQL Client (continued)

Next,

Select “Project” as the file type you’d like to import

Click “Next”



Getting Started

DBeaver - A SQL Client (continued)

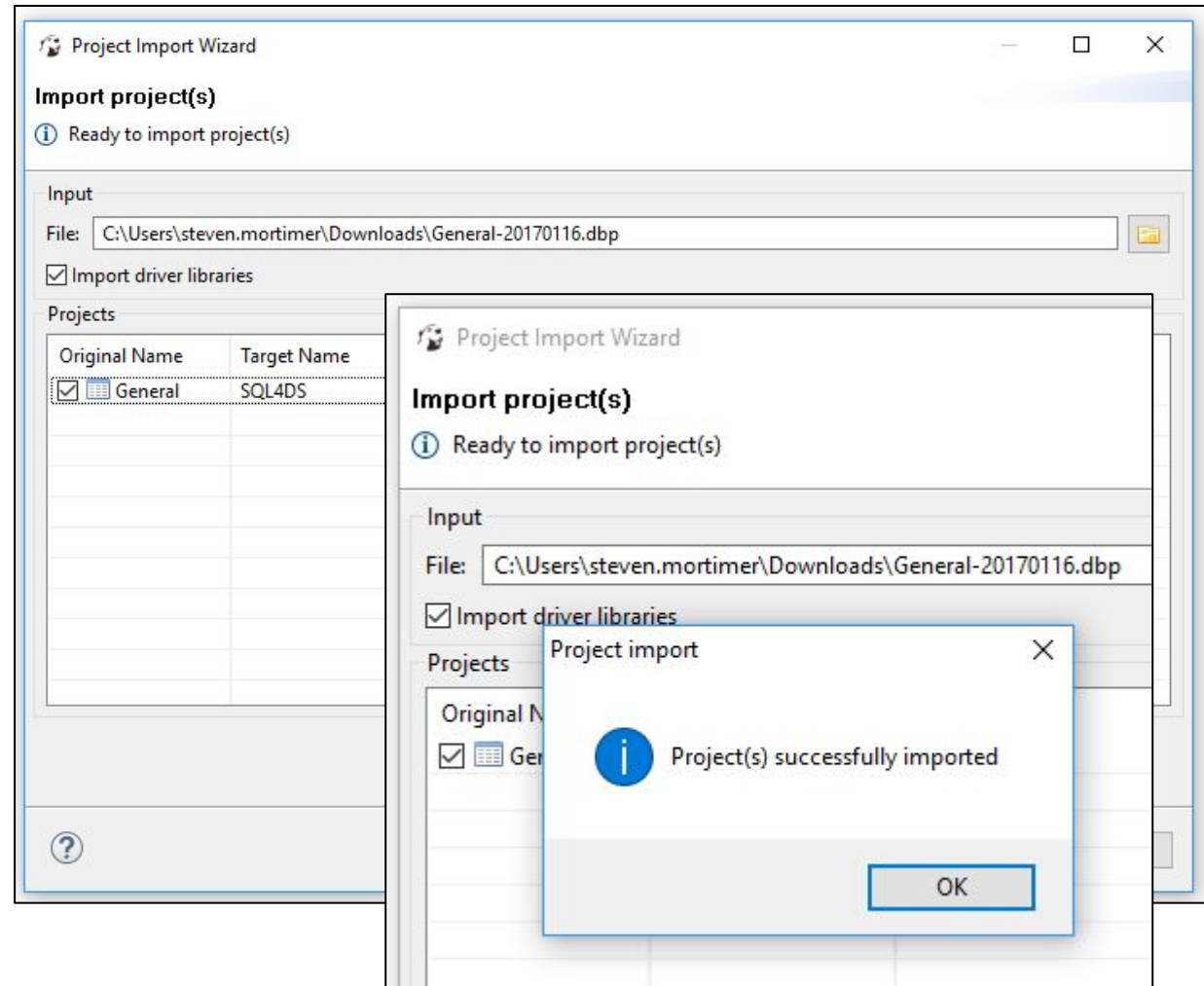
Next,

Select the .dbp file
that you downloaded

Ensure that “Import
driver libraries” is
checked

Change the Target
Name to “SQL4DS”

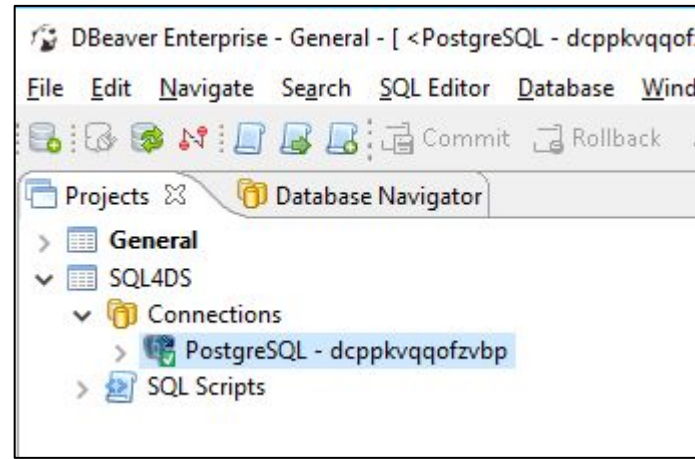
Click “Finish”



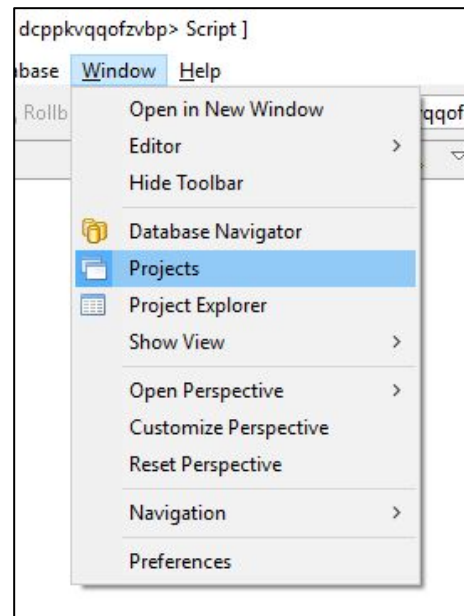
Getting Started

DBeaver - A SQL Client (continued)

Click on SQL4DS Project. You should see a connection “PostgreSQL” with green check.



If you do not see a “Projects” tab, go to “Window” -> “Projects”

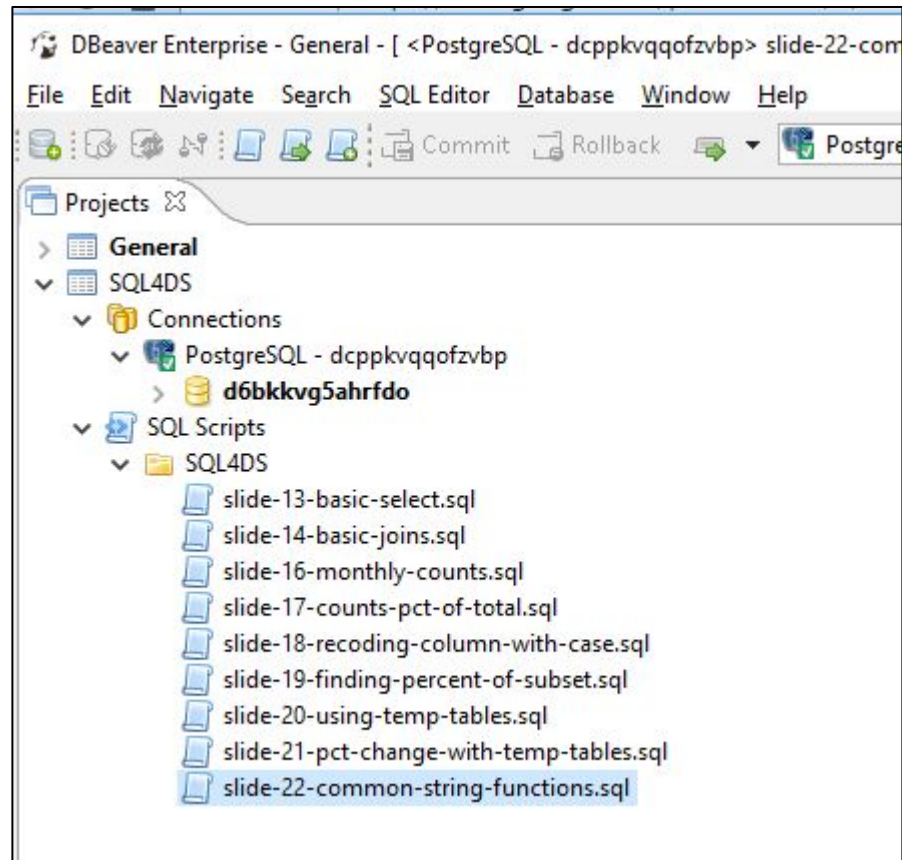


Getting Started

DBeaver - A SQL Client (continued)

If you click on arrow to expand the “SQL Scripts” you will see scripts and folders of scripts that are part of your profile.

There is a folder called “SQL4DS” with all scripts for this presentation



PostgreSQL (Database)

- Do you have connection credentials?
 - Can you “telnet”?
-

DBeaver (SQL Client)

- Installed?
- Do you have project or connection profile?
- Can you connect to database?
- Can you see scripts? Do they execute?

The basic outline of a SQL query is SELECT, FROM, WHERE

```
SELECT
    productid
  , name
  , listprice
FROM production.product
WHERE productnumber = 'LO-C100'
;
```

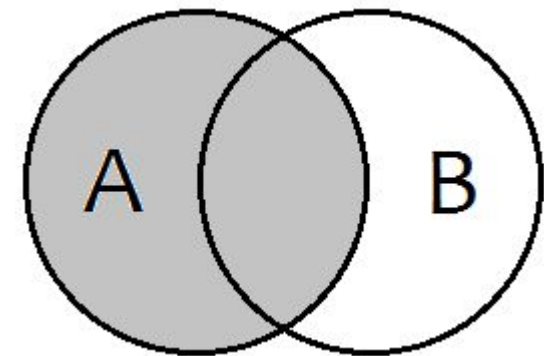
SQL is declarative, just tell the database what you want! But... it can get messy so format your queries to be easy to read.

[See the SQL Style Guidelines Cheatsheet](#)

A “JOIN” connects data from 2 or more tables (similar to VLOOKUP)

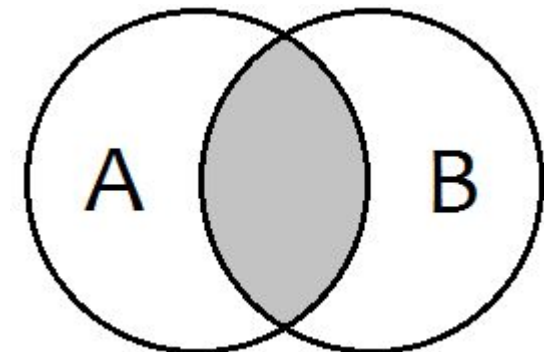
LEFT JOIN

```
SELECT COUNT(*) -- 504 records
FROM production.product p
LEFT JOIN production.productcategory as pc
  ON p.productsubcategoryid = pc.productcategoryid
;
```



INNER JOIN

```
SELECT COUNT(*) -- 105 records
FROM production.product p
INNER JOIN production.productcategory as pc
  ON p.productsubcategoryid = pc.productcategoryid
;
```

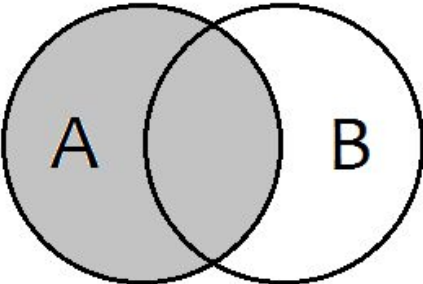
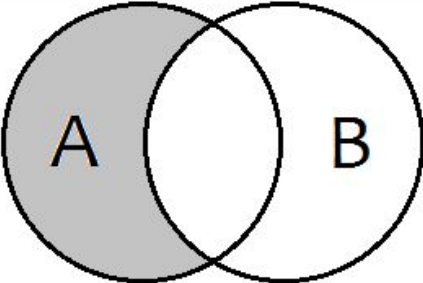
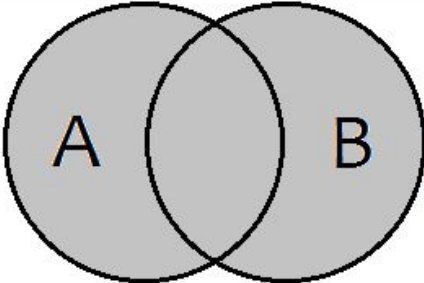
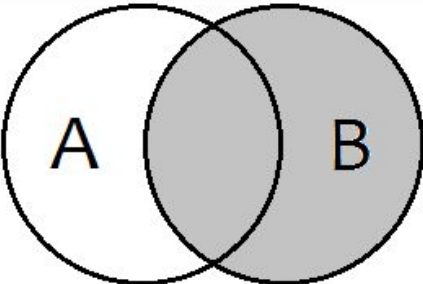
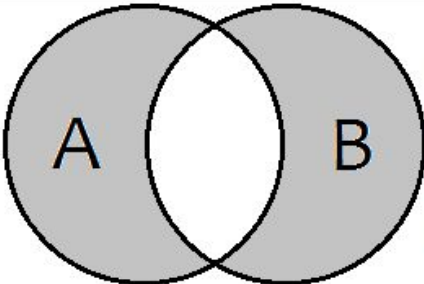
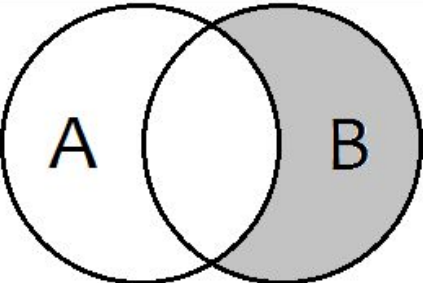
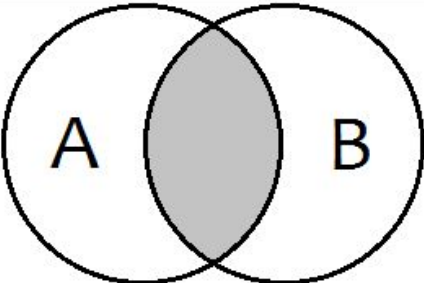


The SQL Basics

The JOINS Cheatsheet



Data Science

 <pre>SELECT * FROM TableA a LEFT JOIN TableB b ON a.Key = b.Key</pre>	<h1>SQL JOINS</h1>
 <pre>SELECT * FROM TableA a LEFT JOIN TableB b ON a.Key = b.Key WHERE b.Key IS NULL</pre>	 <pre>SELECT * FROM TableA a FULL OUTER JOIN TableB b ON a.Key = b.Key</pre>
 <pre>SELECT * FROM TableA a RIGHT JOIN TableB b ON a.Key = b.Key</pre>	 <pre>SELECT * FROM TableA a FULL OUTER JOIN TableB b ON a.Key = b.Key WHERE a.Key IS NULL OR b.Key IS NULL</pre>
 <pre>SELECT * FROM TableA a RIGHT JOIN TableB b ON a.Key = b.Key WHERE a.Key IS NULL</pre>	 <pre>SELECT * FROM TableA a INNER JOIN TableB b ON a.Key = b.Key</pre>

Find Out Count of Transactions per Month

The GROUP BY Statement

```
SELECT  p.name
        , COUNT(*) AS total
FROM production.transactionhistory th
INNER JOIN production.product p
    ON th.productid = p.productid
WHERE TO_CHAR(transactiondate, 'YYYY-MM') = '2014-01'
GROUP BY p.name
;
```

Count by Month for All Months in 2014.
Don't Copy/Paste! Group it!

```
SELECT  TO_CHAR(th.transactiondate, 'YYYY-MM') AS month
        , p.name
        , COUNT(*) AS total
FROM production.transactionhistory th
INNER JOIN production.product p
    ON th.productid = p.productid
WHERE DATE_PART('year', transactiondate) = '2014'
GROUP BY month, p.name
;
```


Find Out Distribution of Products by Category

Divide by Total Count to get Proportion of Total

```
SELECT    pc.name AS category
          , COUNT(p.productid)
          , CAST(COUNT(p.productid) AS decimal) / (
              SELECT COUNT(*) FROM production.product) AS pct
FROM production.product p
INNER JOIN production.productcategory pc
    ON p.productssubcategoryid = pc.productcategoryid
GROUP BY category
;
```

Be careful! Always inspect your results!!

```
SELECT    pc.name AS category
          , COUNT(p.productid)
          , CAST(COUNT(p.productid) AS decimal) / (
              SELECT COUNT(*) FROM production.product) AS pct
FROM production.product p
LEFT JOIN production.productcategory pc
    ON p.productssubcategoryid = pc.productcategoryid
GROUP BY category
;
```

Recoding a Column with CASE

The CASE Statement

“CASE” follows *WHEN-THEN-ELSE* logic to change values, here we rename bikes and components to “Core”.

```
SELECT
  p.name
, CASE
    WHEN pc.name = 'Bikes' OR pc.name = 'Components' THEN 'Core'
    ELSE 'Non-Core'
  END AS finance_category
FROM production.product p
LEFT JOIN production.productcategory pc
  ON p.productsubcategoryid = pc.productcategoryid
;
```

Find Out What Percent of Products are “Core”

Convert to 0 or 1 and the Average = the Proportion

```
SELECT
    AVG(core_indicator) AS pct_core
FROM production.product p
LEFT JOIN (
    SELECT
        productcategoryid
        , CASE
            WHEN name = 'Bikes' OR name = 'Components' THEN 1
            ELSE 0
        END AS core_indicator
    FROM production.productcategory
) pc
ON p.productsubcategoryid = pc.productcategoryid
;
```

Note: The product table is LEFT JOIN'ed to a “subquery”. A subquery is its own query that is part of a larger query.

Using Temp Tables

Smaller, Separate Tables to Make it Easier

```
WITH t AS (  
    SELECT    productid  
              , TO_CHAR(transactiondate, 'YYYY-MM') AS month  
    FROM production.transactionhistory  
    WHERE DATE_PART('year', transactiondate) = '2014'  
)  
  
    , t2 AS (  
    SELECT t.*, p.name  
    FROM t  
    INNER JOIN production.product p  
        ON t.productid = p.productid  
)  
  
SELECT month, name, COUNT(*) AS total  
FROM t2  
GROUP BY month, name  
ORDER BY month, name  
;
```

Find Out the Percent Change Month over Month

The LAG Statement

```
, monthly AS (  
  SELECT month, COUNT(*) AS total  
  FROM t2  
  GROUP BY month  
  ORDER BY month  
)  
, lagged AS (  
  SELECT  
    month  
    , total  
    , LAG(total) OVER (ORDER BY month) AS prior_month  
  FROM monthly  
)  
  
SELECT month, total, prior_month,  
       (total::float - prior_month) / prior_month AS pct_diff  
FROM lagged  
;
```

Manipulate Strings and Find via Fuzzy Match

Text Columns can be Modified and Searched as Needed

```
SELECT
    name AS original
  , LENGTH(name) AS str_length
  , UPPER(name) AS str_upper
  , SUBSTRING(name, 1, 3) AS str_sub
  , REPLACE(name, 'a', 'zzz') AS str_replace
  , CONCAT('PREFIX_', name) AS str_concat
FROM production.product
WHERE productnumber LIKE 'LO%'
;
```

LENGTH, UPPER, SUBSTRING, REPLACE, CONCAT will modify a column. LIKE will perform a fuzzy match

W3Schools - <http://www.w3schools.com/sql/>

Hacker Rank - <https://www.hackerrank.com/domains/sql/select>

SQL Zoo - <http://sqlzoo.net/wiki/AdventureWorks>

Codecademy - <https://www.codecademy.com/learn/learn-sql>

Other Norfolk Data Science Members!

Experienced SQL users are all around you

Questions

We're Here to Help

In Person - Ask Now!

Via Chat - Join the Norfolk Data Science Community on Slack

<http://norfolkdatsci.herokuapp.com/>

Via Email

Steve - reportmort@gmail.com

Josiah - josiahfbaker@gmail.com

