* Intro to SQL
  + At its core, a database is an organized collection of data
  + Intro to Database
    - Databases are organized according to a schema, as defined by the user
    - A database schema comprises declarations for the relations (“tables”) of the database
    - A database schema is a roadmap of tables and fields within each database. The schema is a blueprint of how a database is constructed and the relationship between tables.
    - There are 2 types of databases:
      * Relational databases
        + Is composed of “rows and columns<” similar to an Excel spreadsheet
        + Each entry is a row
        + Offers specific structure of tables and columns
        + Is strictly organized (only dates can go in the date column, only numbers can go in the numbers columns, ect)
      * Document-based (non-relational) databases
        + Each entry is a document
        + Is not strictly structured as a relational database
        + Accesses each record / document with a key or ID
        + Is general, the difference between a relational and document-based database is similar to the difference between tabular data formats (CSV, TXT) and markup-based formats (XML, JSON)
  + Intro to SQL
    - Why would you need to use SQL when you have Excel?
      * Excel and SQL both use many of the same functions (COUNT, IF THEN ELSE, SUM, ect)
      * There is also a limit of 1,048,576 rows and 16,384 columns
      * Finally, while VLOOKUP or INDEX / MATCH work well in Excel, they are not designed to work on a larger scale. This is where SQL comes in
    - SQL more easily relates data together across databases and distinguishes itself from Excel in terms of scalability. SQL can navigate databases robustly and retrieve or aggregate millions of records. SQL is also more adept than Excel at creating data flows for cleaning and preparing data at higher volumes.
    - SQL also has connections to other steps in the workflow:
      * Identify the problem: Before accessing a database, you should already have an idea of what you need in order to solve your problem
      * Understand the data: There are various ways of understanding what data is stored in a database, which you should leverage after you’ve used SQL to obtain it
      * Prepare the data: SQL can query, structure, clean, and aggregate data, and we can also use it to calculate useful statistics and visualtions once we’ve exported our data
    - SQL is the languge used to access data in a relational database
    - You need to make sure the data you’re looking at in Excel and SQL is stored as the correct data type, otherwise your functions may not work
* Basic Queries
  + The SELECT Statement
    - SQL lets you ask specific-but-complicated questions of your data
    - The most common and powerful question or query you will use is the SELECT statement
    - SELECT is a SQL keyword with a simple function: It retrieves information from a database
    - A SELECT statement always starts with the SELECT keyword
    - It is followed by the columns you’d like to return by the query
      * This line is called a **clause**
    - In SQL, clauses are made up of a keyword and a reference to the data
      * SELECT column\_one, column\_two
    - The next clause in a SELECT statement starts with the FROM keyword, followed by the column name

SELECT column\_one, column\_two

FROM table\_name

* + - Finally, every SELECT statement ends with a semicolon:

SELECT column\_one, column\_two

FROM table\_name

;

* + - * It acts like a period at the end of a sentence – your statements won’t be complete without a final semicolon
  + Commenting Code in SQL
    - Headers are a great way to keep a history of why SQL has been built and who requested the changes that have been made

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* NAME: Name of report

\*\* DESC: Description of report

\*\* AUTH: Name of author

\*\* REQ: Name of requester

\*\* DATE: Date report published

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*Change History

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Version| Date | Author |Description

\*\*------------------------------------------

\*\* 1.1 |10/15/16|Pat Doe| Description of change

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* + Let’s say we have a table called “flights,” but we don’t know enough about this table to SELECT specific columns
    - To see all of the columns in this table, we’d type:

SELECT \*

FROM flights;

* + - In SQL, the star (\*) is shorthand for all columns
  + Queries can be scary if they SELECT hundreds of thouslands of rows of data
    - They can slow down or crash you system
    - So, it’s smart to LIMIT the number of results your query returns

SELECT \*

FROM flights LIMIT 3;

* + SQL also has a helpful clause that sorts your data output called ORDER BY
    - When you ORDER BY numerical data, it will automatically arrange the values from smallest to largest
    - When you apply ORDER BY to text, it will automatically sort from A-Z
    - NOTE: you can ORDER BY multiple columns and just like sorting in Excel the sequence of the ORDER BY execution is determines by how the order you write them

SELECT \*

FROM flights

ORDER BY price DESC

* WHERE and Conditionals
  + We can filter our data using a set of criteria (just like Advanced Filtering in Excel)
    - For example, now that we know more about the “flights” table we could ask for:
      * Flights on Virgin Atlantic, leaving from either Gatwick or Heathrow airport, that cost less than or equal to $600
        + Each of these criteria can be thought of as a **condition**, which means that it evaluates as either true or false
      * Lets’ rewrite our request as a series of conditions:
        + The airline is Virgin Atlancit
        + The flight leaves from either Gatwick or Heathrow airport
        + The flight costs less than or equal to $600
        + If all of these conditions are true, then we want our query to return that flight information
  + One keyword that takes conditions is WHERE
    - WHERE returns records from our dataset where the condition is true
* SELECT origin, destination airline, price
* FROM flights
* WHERE airline = 'Virgin Atlantic';
  + - The equals (=) sign here is known as a **comparison operator**.
      * A comparison operator tests or defines the relationship between two values
      * Comparison Operator:
        + = equal to
        + <> not equal to
        + < less than
        + > greater than
        + <= less than or equal to
        + >= greater than or equal to
  + If we weren’t 100% sure of the name of the airline we want, we could use the LIKE operator or the negative version NOT LIKE
    - LIKE is used in place of the equals sign.
    - For example, the query below would return all airlines that start with the letter V:

SELECT origin, destination airline, price

FROM flights

WHERE airline = 'V%';

* + - The % sign is a wildcard and fills in for the letters we don’t know.
  + Wildcards
    - Wildcards come in handy for situaitons like this, where we want to search for a string that follows a specific pattern
      * For example, to include the misspelled values in our search for Virgin Atlantic flights, we could search for words that start with a “V,” followed by two unknown letters, and ending in “gin:”

SELECT origin, destination airline, price

FROM flights

WHERE airline = 'V\_gin%';

* + - * The underscore (\_) wildcard character fills in for one character, whether it’s a letter, number, or symbol
      * Here, we used two of them to fill in for the two letters between “V” and “g”
* String Functions and CASE
  + There are several different categories of functions used with SQL. We already covered many different types:
    - Numeric functions
    - Aggregate functions
    - Distinct functions
    - Joint operators
    - Predicate operators (such as WHERE or HAVING)
    - Boolean operators
  + In this lesson we will cover two more important categories:
    - String functions
      * String functions make it possible for you to find and alter text values, such as VARCHAR and CHAR datatypes
      * Using these functions you can alter a text value such as changing “Smith, Joe” to “Joe Smith”
    - Time and date functions
      * Time and date functions perform operations on date and time inputs values and return string, numeric, or date and time values
      * The most difficult part of working with dates is to be sure the format of the date you are trying to insert matches the format of the date column in the database
  + String Functions
    - The IF function is a popular function in Excel — it’s commonly used for creating a column that categorizes another column.
    - You can do something similar in SQL using the CASE statement. It uses the same IF…THEN…ELSE, and ELSEIF logic.
    - The syntax in SQL is slightly different, but the outcome is the same.
    - Using CASE statements allows you to to map decision trees into SQL, and then you can use them for other functions or grouping.
    - CASE syntax looks like:

SELECT dimensions

CASE

WHEN condition THEN result

WHEN condition THEN result

ELSE

END

AS output name

FROM table;

* + - Concatenate
      * The CONCAT function combines two fields or expressions together. Here’s the formal syntax for most SQL programs:
        + CONCAT(field1, field2, field3…)
      * It’s comparable to the CONCATENATE or CONCAT functions in Excel, or using the “&” in Excel
      * In SQLite which we will be learning about in the next unit, || is used instead of CONCAT(). So in SQLite the syntax is field1 || field2 || field3…
      * When concatenating fields, we need to add in spaces, dashes, or other delimiters that we want to appear on the screen.

SELECT CONCAT(item\_description, ‘- ‘, bottle\_size)

AS our\_offerings

FROM products

* + In SQLite:

SELECT item\_description || '- ' || bottle\_size

AS our\_offerings

FROM products

* + - Letter Case
      * Since SQL is case sensitive, sometimes you want to transform all values in a column to appear upper case or lower case so you can remove duplicates or group them. There are two functions for this:
        + LOWER: converts a filed or expression to lower case:

Syntax: LOWER(field1)

* + - * + UPPER: Converts a field or expression to upper case

Syntax: UPPER(field1)

* + - * LEFT / RIGHT Selection
        + A different way of combining the characters in a field is using the left and right selection fuctions
        + LEFT: Selects a given number of characters from the left side

Syntax: LEFT(field1, length)

* + - * + RIGHT: Selects a given number of characters from the right side

Syntax: RIGHT (field1, length)

* + - SUBSTRING
      * This function allows you to isolate a section of characters within a field to retrieve
        + Syntax: SUBSTRING (field1, starting position, number of characters to retrieve from starting positions)
    - LENGTH
      * The LENGTH function counts the length of characters in a field
        + Syntax: LENGTH (field1)
    - Left / Right Trim
      * These functions trim blanks from the given side
        + LTRIM: Trims all blanks from the left side.

Syntax: LTRIM(field1)

* + - * + RTRIM: Trims all blanks from the right side.

Syntax: RTRIM(field1)

* + - TRIM
      * This function removes specified characters from start of field (leading characters), end of field (trailing characters), or both.
        + Syntax:

TRIM(leading ‘characters’, from field1)

TRIM(trailing ‘characters’, from field1)

TRIM(both ‘characters’, from field1)

* + - REPLACE
      * This function is similar to the Excel function substitute; it allows you to replace a value in a field with another value.
        + Syntax: REPLACE(field\_to\_change, content\_to\_replace, new\_content)
  + Time and Date Functions
    - * It can be difficult to get into learning about time and date functions.
      * Syntax can be vastly different depending on the SQL platform.
      * For example, IBM’s current query tool uses a TIMESTAMPDIFF function, whereas pgAdmin uses a simple AGE function to the same effect.
      * The best approach is to have an overall understanding of what DATES can do, and have your particular vendor's DATE syntax documentation (usually found online) close by.
      * Having said that, we will take a look at CURRENT\_DATE and AGE. These two date functions tend to be the most popular.
    - CURRRENT\_DATE brings back the current date from the system.
      * Syntax: CURRENT\_DATE
    - AGE or DATEDIFF (in MySQL) returns the difference between two dates in days.
      * Syntax: AGE( date1, date2)
      * DATEDIFF( date1, date2) in MySQL
    - DATEDIFF in SQL Server calculates the difference between two dates in units specified in the query.
      * Syntax: DATEDIFF (datepart, expression1, expression2)
      * Example:
        + SELECT DATEDIFF (month, '2017-03-31 23:59:59.000','2017-04-01 00:00:00.000');

The result is: 1

Even though the actual difference between the two date/time values is 1 second, the DATEDIFF function returns 1 when we ask for the difference in months.

This is because SQL Server only compares the year and month fields. All other fields (day, hour, minute, second, millisecond) are ignored.

DATEDIFF returns an integer. It only compares the values down to the lowest level of the given datepart and does not look at anything that is lower.