ACCESS SQL WORKSHOP I: INTRODUCTION TO SQL

What is a relational database?

BAS.00

BRUIN ACTUARIAL SOCIETY

What is a **database?**

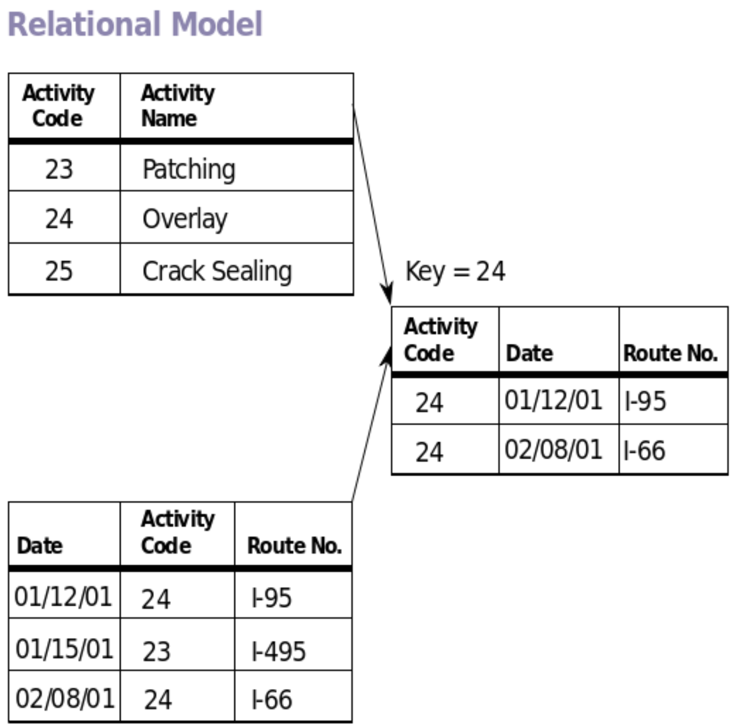
- A **database** is an organized collection of **data**, stored and accessed electronically.
- This **data** requires interpretation to become information we can analyze.

What is a **relational** database?

- A **relational database** is a digital database based on the **relational model of data.**
- Virtually all relational database systems use **SQL (Structured Query Language)** for querying and maintaining the database.
  - SQL is pronounced “S-Q-L” or “sequel”.

What is the **relational model**?

- Data is organized in **tables** of **columns (“attributes”)** and **rows (“records”)**, with a unique **key (“primary key”)** identifying each row
- Rows in a table can be **linked** to rows in other tables by adding a columns for the unique key of the linked row (“**foreign keys**”)



Relational Model

Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Key = 24

Activity Code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

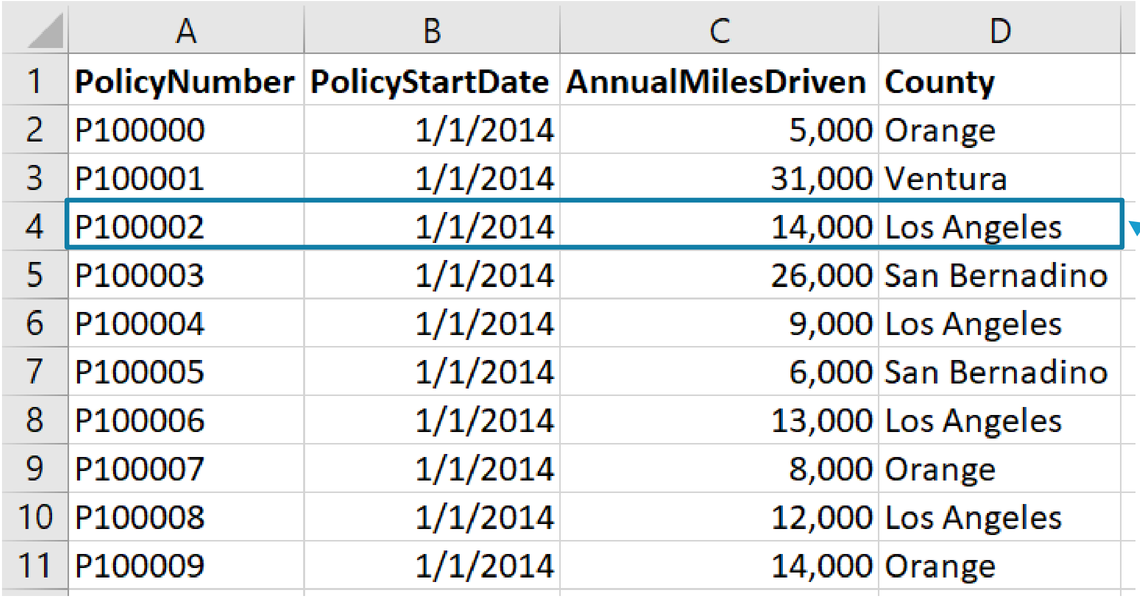
Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
02/08/01	24	I-66

I’m not a computer scientist. What does this mean for an actuary?

- This abstract data stuff has more applications to actuarial work than you may think!
- Consider the most basic example: policies and claims.

Example: Policies and Claims

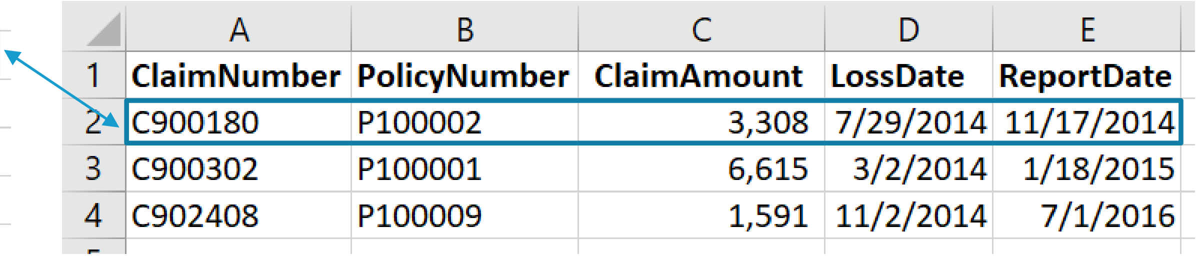
Primary Key: PolicyNumber



	A	B	C	D
1	**PolicyNumber**	**PolicyStartDate**	**AnnualMilesDriven**	**County**
2	P100000	1/1/2014	5,000	Orange
3	P100001	1/1/2014	31,000	Ventura
4	P100002	1/1/2014	14,000	Los Angeles
5	P100003	1/1/2014	26,000	San Bernadino
6	P100004	1/1/2014	9,000	Los Angeles
7	P100005	1/1/2014	6,000	San Bernadino
8	P100006	1/1/2014	13,000	Los Angeles
9	P100007	1/1/2014	8,000	Orange
10	P100008	1/1/2014	12,000	Los Angeles
11	P100009	1/1/2014	14,000	Orange

Primary Key: ClaimNumber

Foreign Key: PolicyNumber



	A	B	C	D	E
1	**ClaimNumber**	**PolicyNumber**	**ClaimAmount**	**LossDate**	**ReportDate**
2	C900180	P100002	3,308	7/29/2014	11/17/2014
3	C900302	P100001	6,615	3/2/2014	1/18/2015
4	C902408	P100009	1,591	11/2/2014	7/1/2016

BRUIN ACTUARIAL SOCIETY

Example: Policies and Claims

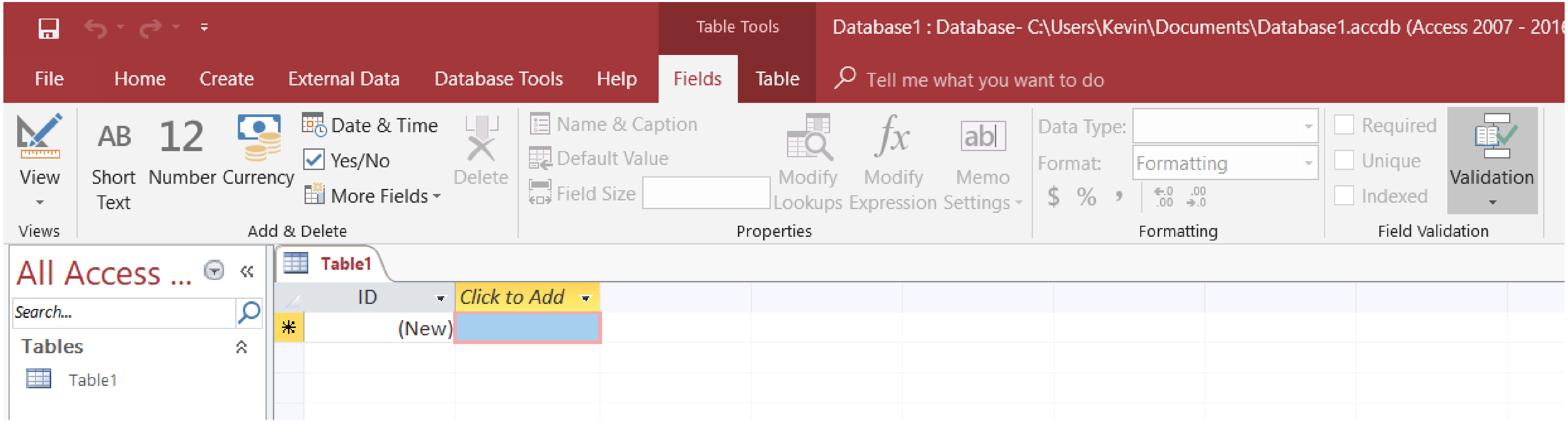
- We can see that the table for Claims can be linked to the table for Policies through the key PolicyNumber.
- That’s a brief introduction to what we’re doing and why you should care.
- In fact, this workshop will focus entirely on this seemingly-simple example of only two* tables.

*This isn’t quite true. We’ll create a few more later to play with, though that’s an issue for a later time.

Microsoft Access / SQL

- Now that you know what we're working with, let's jump in!

The Microsoft Access Environment



The Microsoft Access Environment

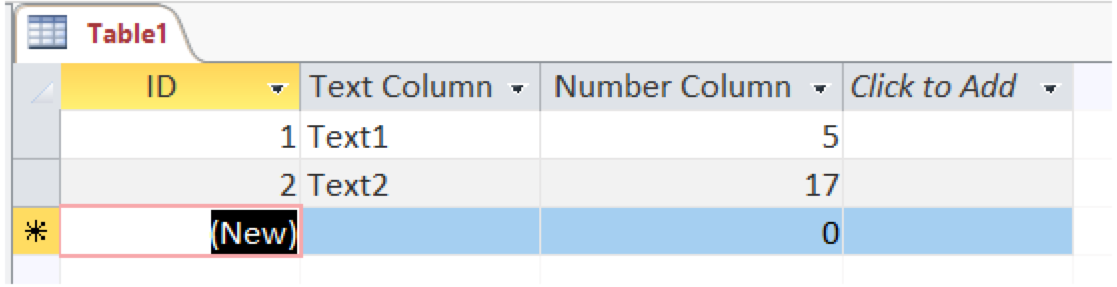
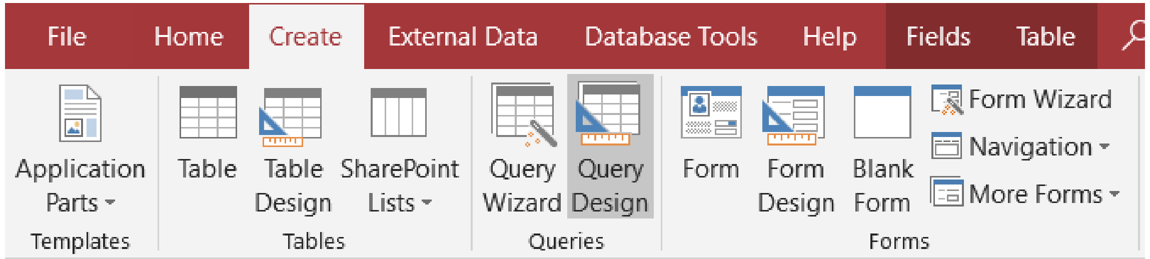


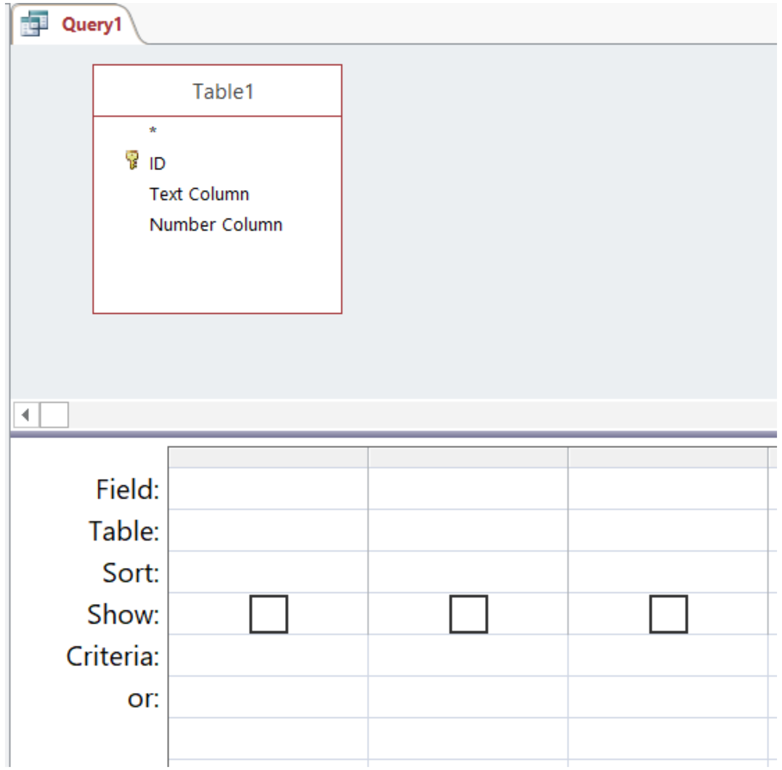
Table1

	ID	Text Column	Number Column	Click to Add
	1	Text1	5	
	2	Text2	17	
*	(New)		0	

- Data can be entered into data tables manually, and data tables can be viewed in their entirety manually

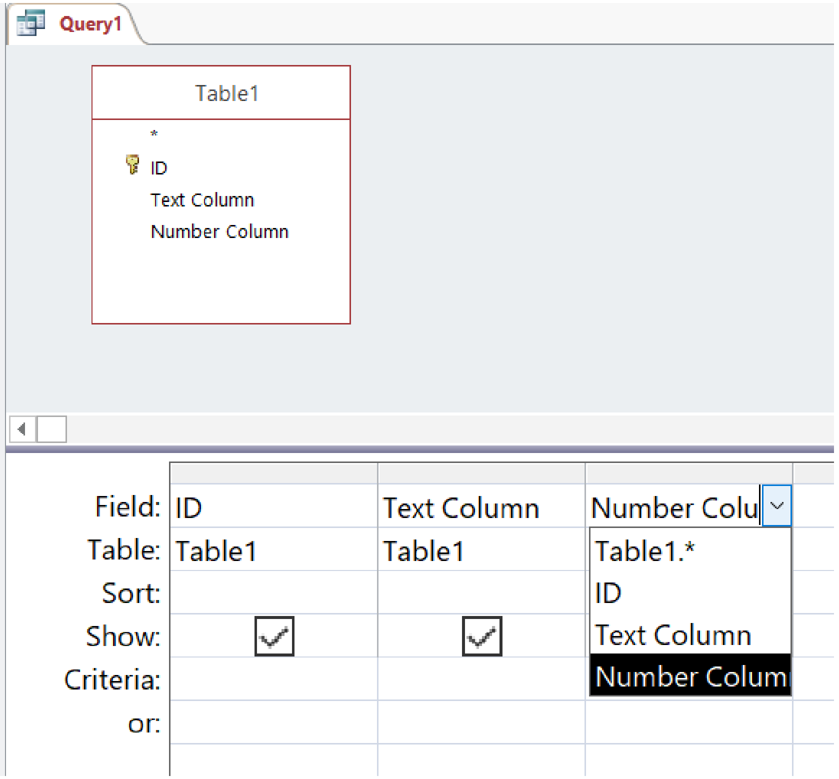
What we care about: **Queries**





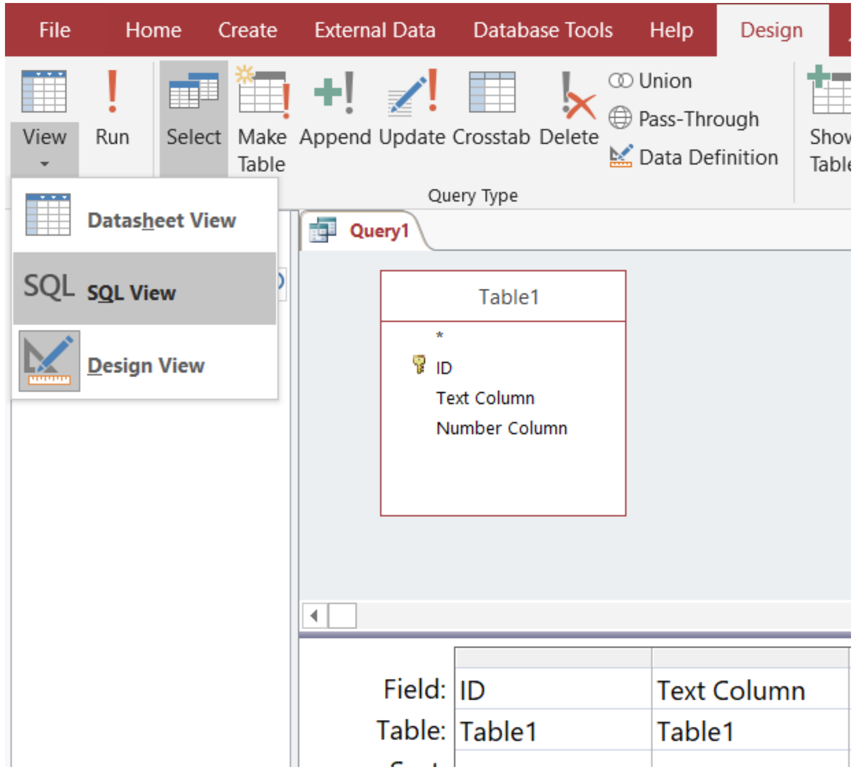
BAS.00
BRUIN ACTUARIAL SOCIETY

This user-friendly interface is good enough for some simple tasks…

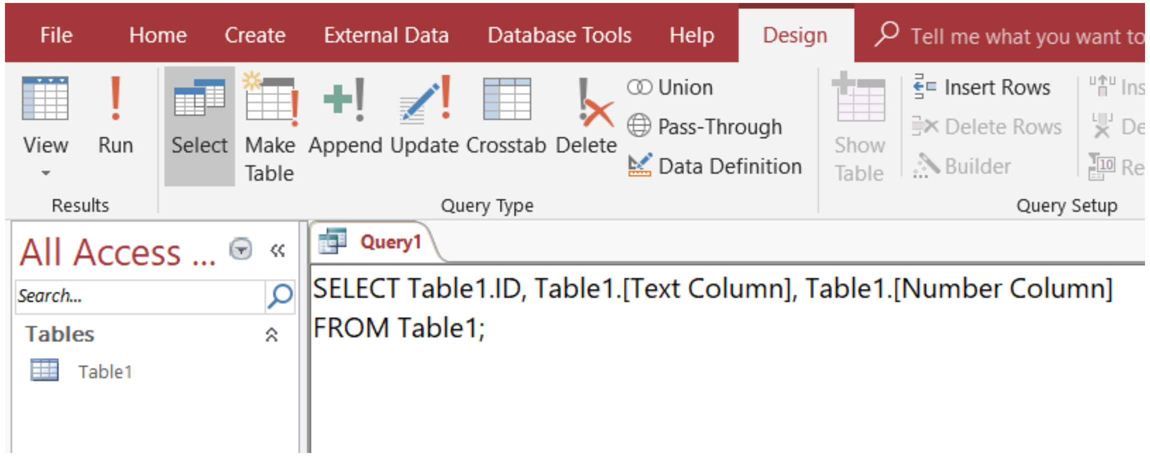


Field:	ID	Text Column	Number Colu
Table:	Table1	Table1	
Sort:			
Show:	☑	☑	
Criteria:			
or:			

…but for more advanced tasks, we'll want to use SQL.



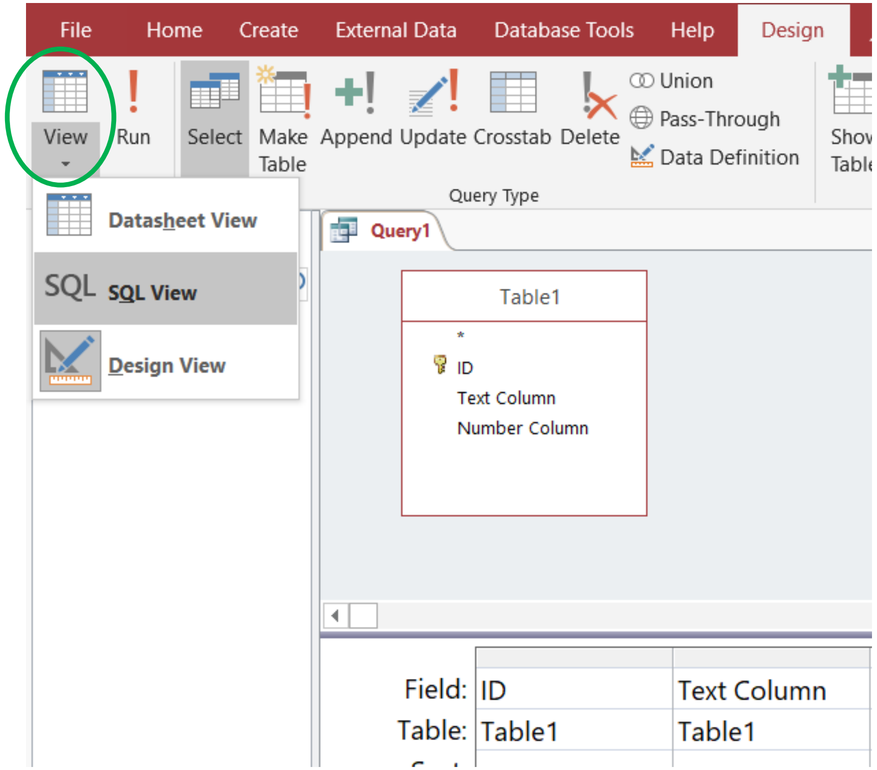
Field:	ID	Text Column
Table:	Table1	Table1



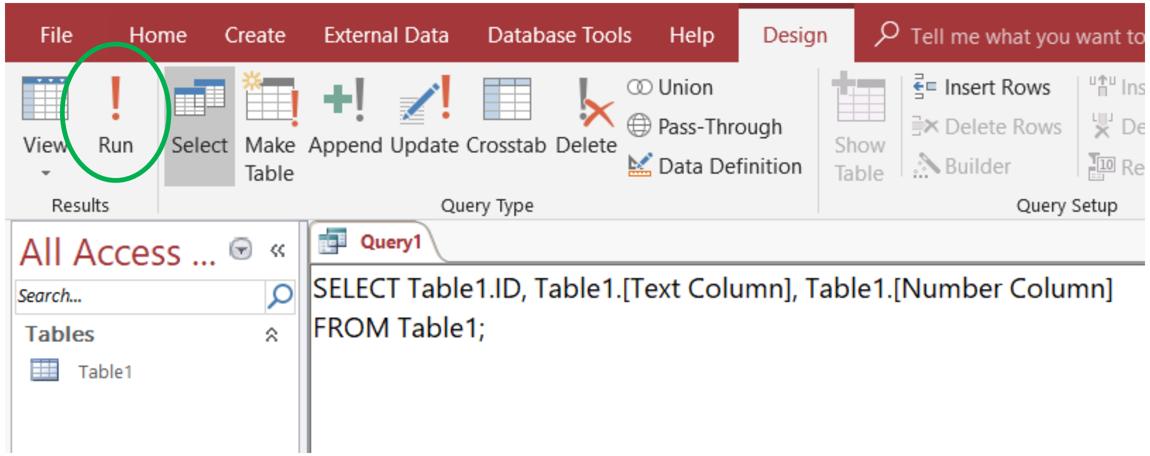
```
SELECT Table1.ID, Table1.[Text Column], Table1.[Number Column]
FROM Table1;
```

BRUIN ACTUARIAL SOCIETY

Regardless of what view you choose, click Run to run the query.



Field:	ID	Text Column
Table:	Table1	Table1

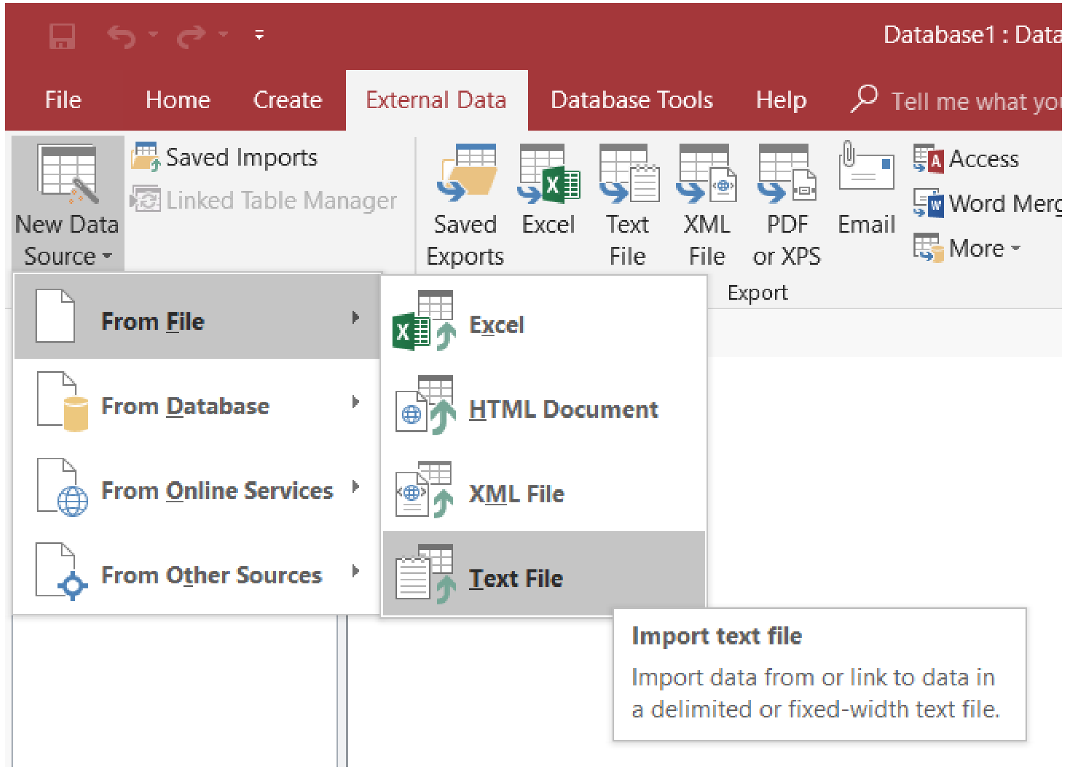


```
SELECT Table1.ID, Table1.[Text Column], Table1.[Number Column]
FROM Table1;
```

SQL View

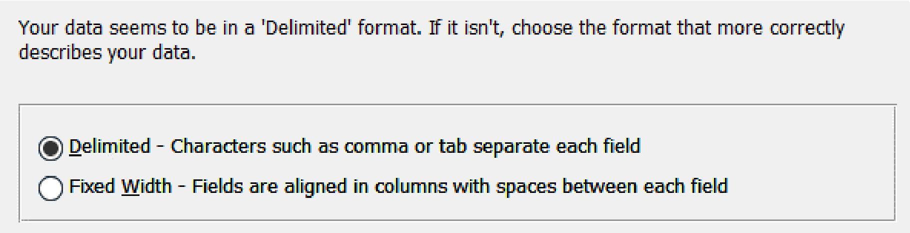
- This may not make any sense now, but that’s what this workshop is for.
- When we begin writing simple queries, you can always go back to design view to see the user-friendly depiction of what’s going on.
- That’s enough introductory stuff. Let’s begin coding!

Getting our first tables: Importing from CSVs



Import text file

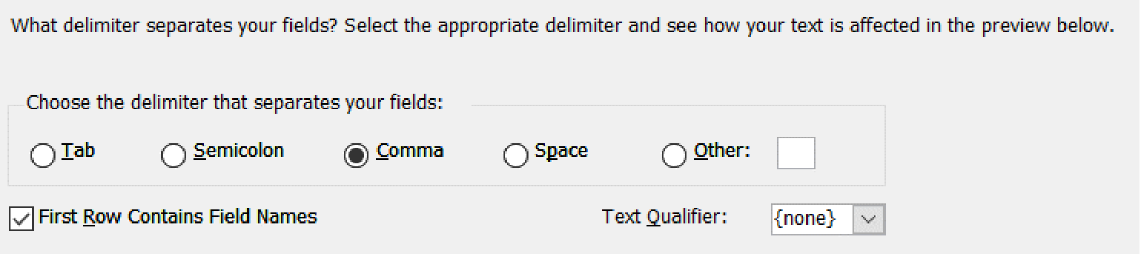
Import data from or link to data in a delimited or fixed-width text file.



Your data seems to be in a 'Delimited' format. If it isn't, choose the format that more correctly describes your data.

● Delimited - Characters such as comma or tab separate each field

○ Fixed Width - Fields are aligned in columns with spaces between each field



What delimiter separates your fields? Select the appropriate delimiter and see how your text is affected in the preview below.

Choose the delimiter that separates your fields:

○ Tab ○ Semicolon ● Comma ○ Space ○ Other: □

☑ First Row Contains Field Names

Text Qualifier: {none}

```
SELECT column1, column2, ... FROM table_name;
SELECT * FROM table_name;
```

- The first line selects all rows, but only the specified columns, from `table_name`
- The second line selects all rows and all columns from `table_name`

SELECT *column1*, *column2*, … FROM *table_name* WHERE *condition*;

- Only selects rows from *table_name* where *condition* is true (and the appropriate columns)
- *condition* can be a simple boolean:
  - WHERE *column_name* = “Value”
  - WHERE *column_name* <> 0
- Or an IN statement:
  - WHERE *column_name* IN (*value1*, *value2*, *value3*)
- Or a complex statement:
  - WHERE (*column_name* = 0 OR *column_name* = 1) AND *another_column* IN (“value1”, “value2”)

```
SELECT DISTINCT column1, column2, …
FROM table_name;
```

- Selects appropriate rows and columns, but removes duplicates

SELECT *column1*, *column2*, … FROM *table_name* ORDER BY *column1*, *column2*, … [DESC]

- Selects appropriate rows and columns, then orders them in the given order
- By default, sorts in ascending order
- Use `DESC` to sort in descending order

Exercises

- Select all policies with low driving frequency in LA or Orange county. How many are there? Note that each policy gets a new entry upon renewal—be sure to account for this.
- Select all policies with that have low or medium driving frequency or are not located in LA, Orange, or San Bernardino county.

SQL Variables

- Any “unrecognized” text (that isn’t a reserved SQL keyword) is interpreted as a variable
- Suppose our columns are “`col1`”, “`col2`”, and “`col3`”:
  - `SELECT col1, col2 FROM table WHERE col1 <> 0 AND col3 = `**`my_variable`**`;`
- Since **`my_variable`** isn’t a column name or a keyword, SQL interprets it as a variable
- You will be prompted to input a value for **`my_variable`** before the script runs

Aggregate Functions: SUM, COUNT, AVG, …

- Aggregates numerical data in groups
- Must be accompanied by a `GROUP BY` clause
- `SELECT SUM(Claim_Amount), ReportYear FROM claims GROUP BY ReportYear;`
  - This will give the total claims for each year, since the aggregate data is grouped by report year.

Exercises

- What is the total amount of claims reported in each year?
- What is the total amount of claims occurring in each year?
- Consider only claims that occurred in 2014. How much was reported in each year?

WHERE vs. HAVING

WHERE

- Used to test variables in a single "cell" of the data table
- Cannot be used with aggregate functions

HAVING

- The "equivalent" of WHERE for aggregate functions
- Comes after the GROUP BY clause
  - We need to know what to group by before we know which groups to show!

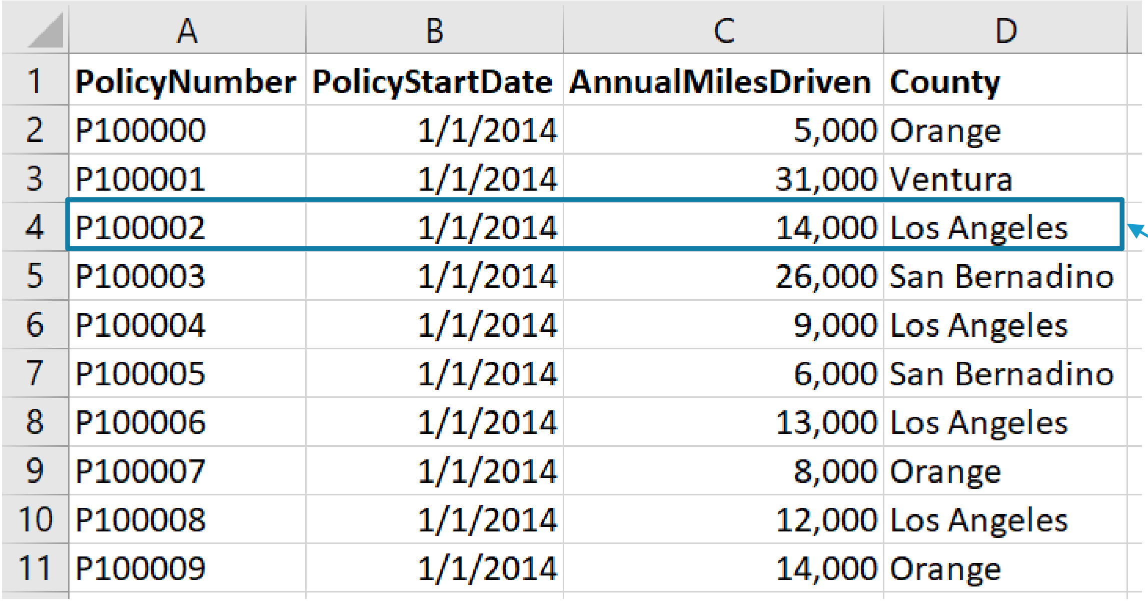
Exercises

- Use the previous example to determine how many claims each policy would have if we implemented a $1,000 policy deductible.
  - Hint: A $1,000 eliminates all policies with total claims under $1,000. For policies with claims under $1,000, their total claim amount will be reduced by $1,000.
- How many policies had total claims over $60,000? What are their policy numbers?
- Use the results from the previous part to determine how many claims each of these policies had. Which policy has the highest severity? What is it?

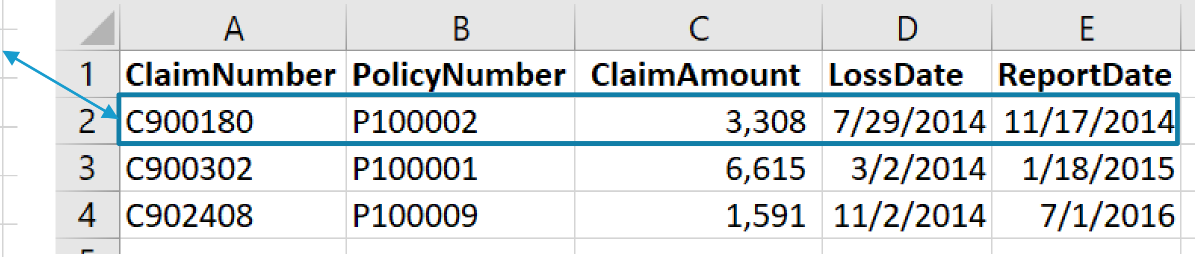
To be continued…

- A preview of next time:

Next time, we'll see how powerful SQL is by learning how to link related data.



	A	B	C	D
1	**PolicyNumber**	**PolicyStartDate**	**AnnualMilesDriven**	**County**
2	P100000	1/1/2014	5,000	Orange
3	P100001	1/1/2014	31,000	Ventura
4	P100002	1/1/2014	14,000	Los Angeles
5	P100003	1/1/2014	26,000	San Bernadino
6	P100004	1/1/2014	9,000	Los Angeles
7	P100005	1/1/2014	6,000	San Bernadino
8	P100006	1/1/2014	13,000	Los Angeles
9	P100007	1/1/2014	8,000	Orange
10	P100008	1/1/2014	12,000	Los Angeles
11	P100009	1/1/2014	14,000	Orange



	A	B	C	D	E
1	**ClaimNumber**	**PolicyNumber**	**ClaimAmount**	**LossDate**	**ReportDate**
2	C900180	P100002	3,308	7/29/2014	11/17/2014
3	C900302	P100001	6,615	3/2/2014	1/18/2015
4	C902408	P100009	1,591	11/2/2014	7/1/2016