ACCESS SQL WORKSHOP II: INTERMEDIATE SQL

Working with multiple tables

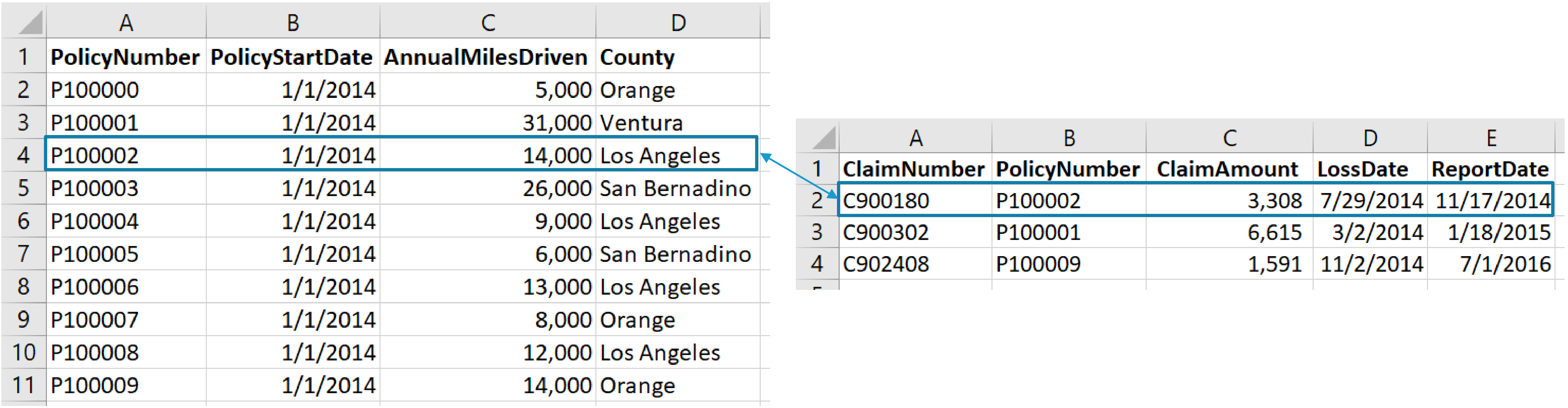
BAS.00

BRUIN ACTUARIAL SOCIETY

A quick recap of last time

- What is a relational database, and what is the relational model?
- How does the Microsoft Access environment work, and how does one write and execute queries?
- The `SELECT` statement and all its modifications (`FROM`, `WHERE`, `ORDER BY`, `DISTINCT`)
- SQL Variables
- Aggregate functions (`SUM`, `COUNT`, `AVG`, …) and how to condition on them (`WHERE` vs. `HAVING`)

Last time, we left off at this slide. This is what we're going to cover today.



	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

Our goal for today

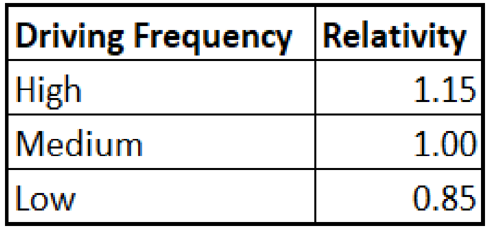
- To calculate 2018 loss ratio. We will assume that claims information includes loss adjustment expenses. The following formulas will be useful:

$$\text{Loss Ratio} = \frac{\text{Total Losses Paid}}{\text{Earned Premiums}}$$

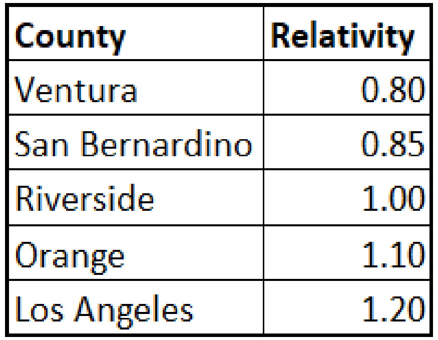
$$\text{Earned Premiums} = \text{Written Premiums } \cdot (\text{Proportion of policy period elapsed})$$

Our goal for today

- Proportion of policy period elapsed, in red, is the hard part.
- You may assume that for the entire duration a policy is inforce, driving frequency and county do not change.
- The annual base rate is **$2,200**, and rate relativities for driving frequency and county are below.



Driving Frequency	Relativity
High	1.15
Medium	1.00
Low	0.85



County	Relativity
Ventura	0.80
San Bernardino	0.85
Riverside	1.00
Orange	1.10
Los Angeles	1.20

But first…

- Sometimes, the data isn’t given to you in nice files. Sometimes, it’s given as a screenshot of an Excel worksheet.
- (Hopefully, this will only happen in learning demonstrations.)

CREATE TABLE *table_name* (*column1 datatype*, *column2 datatype*, …);

- Creates a table with the given column names and datatypes
- Access data types:
  - Text (text/numbers, 255 characters max)
  - Integer (whole numbers between -32,768 and 32,767)
  - Long (whole numbers between -2,147,483,648 and 2,147,483,647)
  - Double (double precision floating-point)
  - Date/Time (dates and times)

INSERT INTO *table_name* (*column1*, *column2*, …) VALUES (*value1*, *value2*, …);

- Inserts values into a table
- If a column is omitted, the value of that column for the row inserted will be null
  - “Null” is **not** the same thing as “0” or the empty (zero-length) string “”.

```
UPDATE table_name SET column1 = value1,
column2 = value2, ... WHERE condition;
```

- Changes **all** row(s) of `table_name` where `condition` is true.
- Changes the columns specified into the values specified.
- Do **not** omit the `WHERE` clause! Otherwise, all rows will be changed!

DELETE FROM *table_name* WHERE *condition*;

- Deletes **all** row(s) of *table_name* where *condition* is true.
- Do **not** omit the WHERE clause! Otherwise, all rows will be deleted!

DROP TABLE *table_name;*

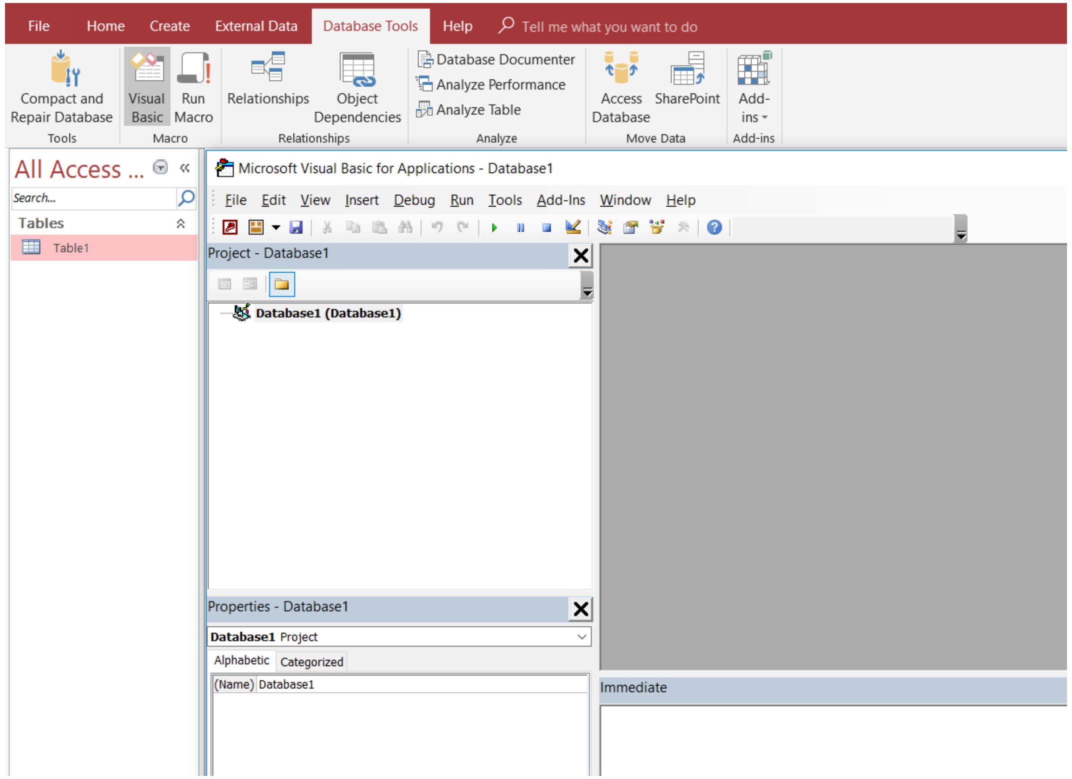
- Drops a table from the database
- Generally, it is a bad idea to run this command on your company's databases*

*Unless it's a table you created in your own schema or something like that…

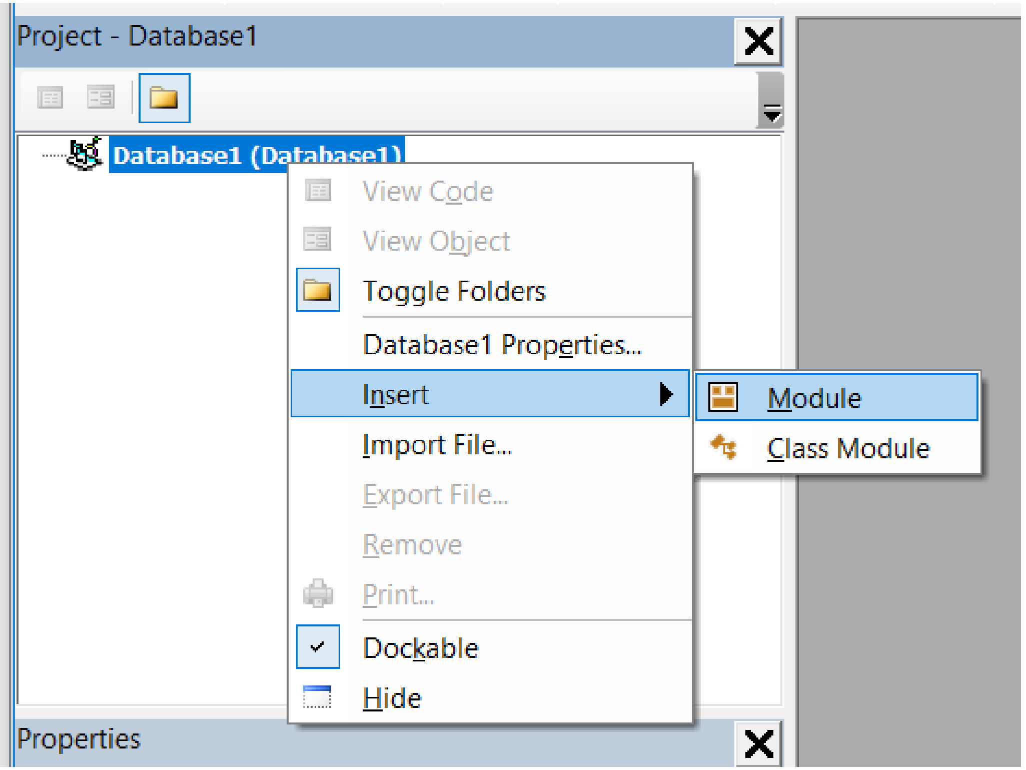
```
SELECT column1, column2, … INTO new_table
FROM old_table WHERE condition;
```

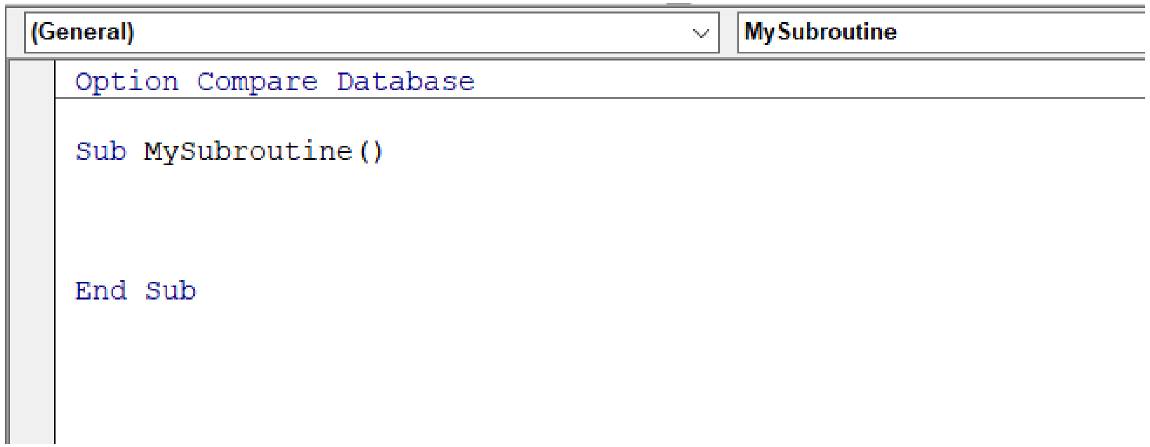
- Any query will do here, but the general idea is the same
- Instead of outputting the results of a query, saves it into a new table named `new_table`

Speeding it up: Doing this in VBA



Speeding it up: Doing this in VBA





```
Option Compare Database

Sub MySubroutine()


End Sub
```

DoCmd.RunSQL *sql*

- *sql* can be a string variable

```
Dim sql as string
sql = “my code here”
```

- Or it can be a string literal

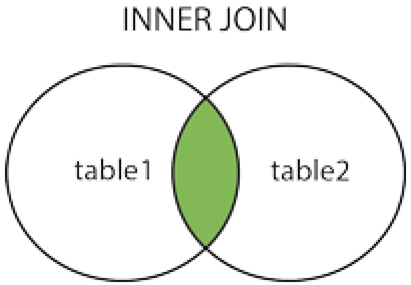
```
DoCmd.RunSQL “my code here”
```

- This is the command to use to quickly create/update/delete tables and entries (instead of clicking “run” every time)

Why you came: **Joins**

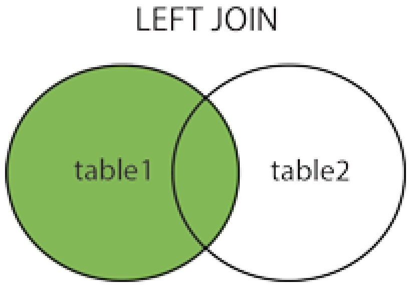
Inner Join

- Selects only the records that have matching values in both tables.



Left (Outer) Join

- Returns all records from the left table, and the matched records from the right table.
- The result is NULL from the right side, if there is no match.



SELECT *column_name(s)* FROM *table1* INNER JOIN *table2* ON *table1.col_name* = *table2.col_name*;

- Joins *table1* and *table2* together on the variable *col_name*
  - Note that the columns don’t HAVE to be named the same thing
- To select columns from the tables, the column name should be preceded by **table1**.col_name or **table2**.col_name, to identify which table to select from
  - SELECT *table1.col1*, *table2.col2*, *table2.col3* FROM *table1* INNER JOIN *table2* ON *table1.col4* = *table2.col4*;
- Records will only be listed if they are in both tables

`SELECT column_name(s) FROM table1 LEFT JOIN table2 ON table1.col_name = table2.col_name;`

- Left joins `table1` and `table2` together on the variable `col_name`
- All records from `table1` will be returned, with the potential for additional detail from `table2`, if applicable
- If the table names are long, we can shorten them by `SELECT`ing `FROM table1 AS (alias)`
  - `SELECT t1.col1, t2.col2, t2.col3 FROM table1 AS t1 LEFT JOIN table2 AS t2 ON t1.col4 = t2.col4;`

Exercises

- Select the initial policy start date (renewals do not count as an initial policy start date) and date of first reported claim for all policies with claims.
  - Hint: you may find the aggregate function MIN useful.
- For accident year 2016, what is the severity in each county? For each driving frequency?
  - Accident year 2016 refers to claims that occurred between 1/1/2016 and 12/31/2016.
- For report year 2016, what is the severity in each county? For each driving frequency?
  - Report year 2016 refers to claims that were reported between 1/1/2016 and 12/31/2016.
- Modify a previous example to select all policies that had no claims. How many are there?

Subqueries: Feeding the output of one query as a table for another

- Queries return a table (or a single value)
- Regardless of which it is, we can enclose a query in parentheses and use that table/value in another query.
-
```
SELECT SUM(c2) FROM
 (SELECT t1.c1 AS c1, t1.c2 AS c2 FROM t1
  LEFT JOIN t2 ON t1.key = t2.key);
```
- The highlighted portion is a perfectly valid standalone query

Exercises

- Refer to the previous exercise where we selected all policies that had no claims. How many are there in each county? In each driving frequency?
- Note that a subquery can return a single result. Use this fact to obtain the claim data for the policy with the largest single loss.
  - Hint: You may find the aggregate function MAX useful.
  - Hint: If a subquery returns a single value, you can use that value in WHERE statements to compare.
- Modify the previous exercise to also obtain the county and driving frequency for this policy. Note that this could be done with joins or with subqueries.

SWITCH(*expression1*, *value1*, *expression2*, *value2*, ... *expression_n*, *value_n*)

- Kind of like a case statement / if statement
- If `expression1` is true, returns `value1`, and so on
- Only the first `expression` that is true will be evaluated to the `value`