Restaurant Inspection Data for Ottawa, Part Two

In the previous tutorial, we downloaded three city of Ottawa's restaurant inspection datasets; business, violations and inspections. The business ID column links to identical ID columns in the violations and inspections tables. In linking two, three, or even more tables, we are creating a relational database, which is discussed on pages 116 and 117 of Computer-Assisted Reporting.

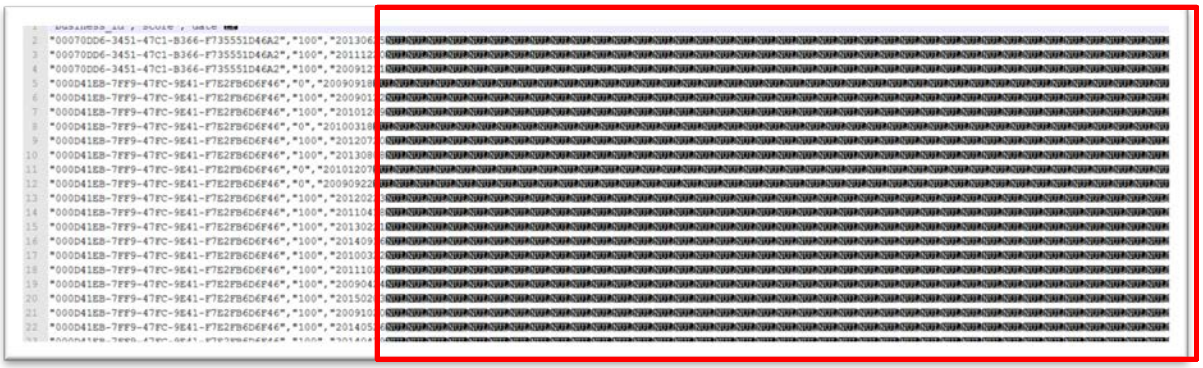
While linking two tables provides useful information about inspections, linking three of them is one step better. For instance, Tim Hortons received 382 failing grades from the beginning of 2009 to May 15, 2015, the time period covered by the dataset at the time it was downloaded for this tutorial.

There are other patterns to uncover. More questions to answer. Which establishments are receiving the highest number of failing-grade inspections? Is the number of inspections increasing or decreasing each year in Ottawa? Which parts of town have the greatest number of problematic restaurants?

NOTE: the tables in this tutorial are labelled “business_1”, “violations_1” and “inspections_1”, respectively. So the screenshots will differ slightly from your queries if you choose the original titles: “business”,”violations” and “inspections”.

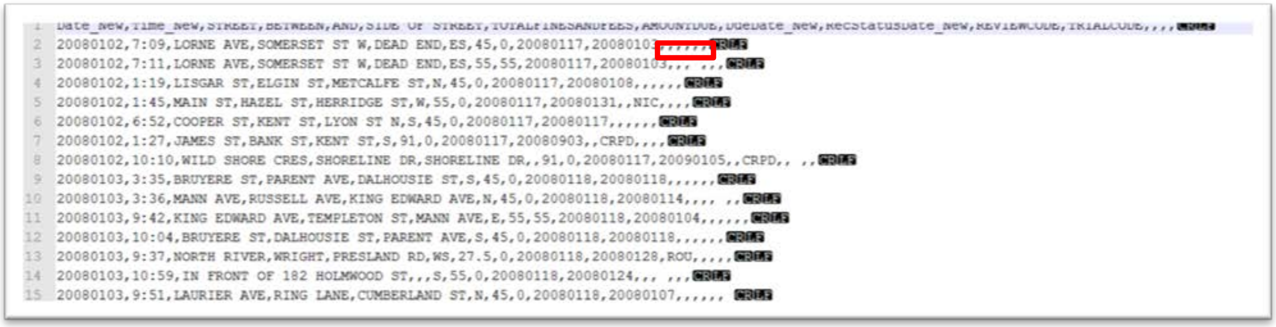
Let's get started.

1. Before joining the three tables, they must be compatible, which means all the datatypes must match. The “date” columns in the business and violations tables are out of synch.
2. In order to correct the problem, we we’ll use the “ALTER TABLE” query to modify the “date” column in the inspections table to make it compatible with the date column in the violations table.
3. Before we use the query, let’s describe the problem. In the previous tutorial, we gave the date column in the inspections table the VARCHAR(8) datatype, whereas the similar column in the “violations” table has a “datetime” data type. If these two columns preserve these datatypes, they will be impossible to join. So we have to synch them up by altering or updating the inspections date column.
4. You may ask why we just didn’t import the inspections date column as a datetime type. Here’s why.



```
"business_id","score","date"
"00070DD6-3451-47C1-B366-F735551D46A2","100","201306
"00070DD6-3451-47C1-B366-F735551D46A2","100","201112
"00070DD6-3451-47C1-B366-F735551D46A2","100","200912
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","0","20090918
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","200901
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201012
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","0","20100318
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201207
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201308
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","0","20101207
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","0","20090922
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201202
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201104
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201302
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201409
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201003
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201110
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","200904
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201502
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","200910
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201405
"000D41EB-7FF9-47FC-9E41-F7E2FB6D6F46","100","201404
```

For whatever reason, when we open the inspections file in our text editor, Notepad++ or TextWrangler, the date field contains a seemingly infinite number of “NULL” values. This means when importing this table, we used the VARCHAR(11), which deliberately truncated all the NULL values after the date. (NOTE: this is why it is always a good idea to open your data sets in a text editor before importing them into MySQL. The text editor allows you to see problem areas, which then determine the kinds of data types you use when creating your tables. For instance, in the tutorial involving the city of Ottawa fire hydrant parking fine data, we added an extra column called “Empty” to account for the extra commas at the end of each row and before the carriage returns, which we can see in this screen shot.

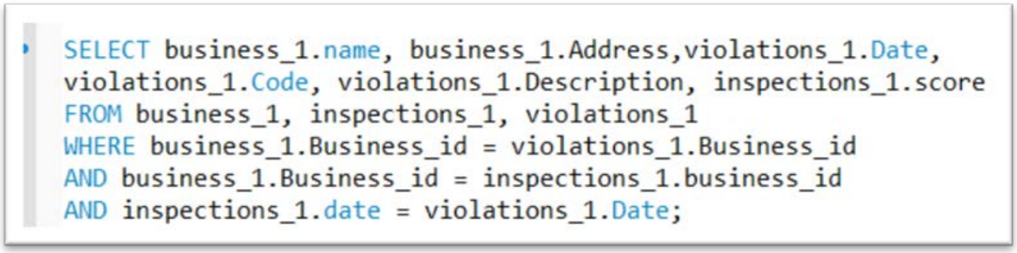


```
Date_New,Time_New,STREET,BETWEEN,AND,SIDE OF STREET,TOTALFINESANDFEES,AMOUNTDUE,DueDate_New,RecStatusDate_New,REVIEWCODE,TRIALCODE,,,,CRLF
20080102,7:09,LORNE AVE,SOMERSET ST W,DEAD END,ES,45,0,20080117,20080103,,,,,,CRLF
20080102,7:11,LORNE AVE,SOMERSET ST W,DEAD END,ES,55,55,20080117,20080103,,, ,,,CRLF
20080102,1:19,LISGAR ST,ELGIN ST,METCALFE ST,N,45,0,20080117,20080108,,,,,,CRLF
20080102,1:45,MAIN ST,HAZEL ST,HERRIDGE ST,W,55,0,20080117,20080131,,NIC,,,,CRLF
20080102,6:52,COOPER ST,KENT ST,LYON ST N,S,45,0,20080117,20080117,,,,,,CRLF
20080102,1:27,JAMES ST,BANK ST,KENT ST,S,91,0,20080117,20080903,,CRPD,,,,CRLF
20080102,10:10,WILD SHORE CRES,SHORELINE DR,SHORELINE DR,,91,0,20080117,20090105,,CRPD,, ,,CRLF
20080103,3:35,BRUYERE ST,PARENT AVE,DALHOUSIE ST,S,45,0,20080118,20080118,,,,,,CRLF
20080103,3:36,MANN AVE,RUSSELL AVE,KING EDWARD AVE,N,45,0,20080118,20080114,,,, ,,CRLF
20080103,9:42,KING EDWARD AVE,TEMPLETON ST,MANN AVE,E,55,55,20080118,20080104,,,,,,CRLF
20080103,10:04,BRUYERE ST,DALHOUSIE ST,PARENT AVE,S,45,0,20080118,20080118,,,,,,CRLF
20080103,9:37,NORTH RIVER,WRIGHT,PRESLAND RD,WS,27.5,0,20080118,20080128,ROU,,,,,CRLF
20080103,10:59,IN FRONT OF 182 HOLMWOOD ST,,,S,55,0,20080118,20080124,,, ,,,CRLF
20080103,9:51,LAURIER AVE,RING LANE,CUMBERLAND ST,N,45,0,20080118,20080107,,,,,, CRLF
```

5. Now that we have a better understanding of the problem at hand, let’s alter the inspections date column.
6. To do this, we can run this query: “ALTER TABLE inspections MODIFY COLUMN date datetime;” This query has two statements: ALTER TABLE, tells which table is in play; MODIFY COLUMN points MySQL to the column in question and the datatype to be used. You can also edit the table by changing the datatype to “datetime”. (NOTE: MySQL produces an error with the ALTER TABLE query. Just ignore it. Consversely, you can also perform the same task by editing the table and changing the datatype to “datetime” and saving the result.
7. Now we can join the tables. This tutorial will use the WHERE statement to join the business table’s ID field to the similar fields in the inspections and violations tables. For more

information on using the WHERE statement to join tables, consult pages 196 to 1977 of the textbook.

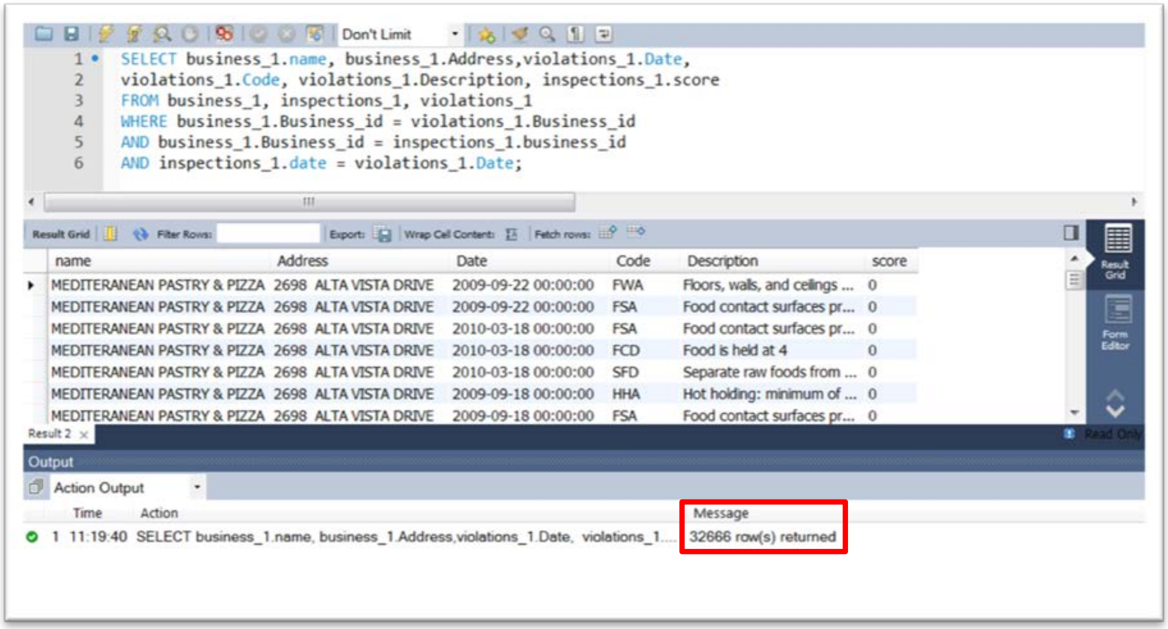
8. This is what the script looks like:



```
SELECT business_1.name, business_1.Address,violations_1.Date,
violations_1.Code, violations_1.Description, inspections_1.score
FROM business_1, inspections_1, violations_1
WHERE business_1.Business_id = violations_1.Business_id
AND business_1.Business_id = inspections_1.business_id
AND inspections_1.date = violations_1.Date;
```

Let's break down the query. In the SELECT statement, you are instructing MySQL to select the fields identified in each table. The name of each table followed by a dot (".") comes before each field name. In the FROM statement, you are telling MySQL where to get the data: in this case, the three tables. We are using the WHERE statement join the business table's "business_ID" field, which is the primary key, to the "business_ID" field in the violations table. Next, we use the "AND" operator to tell MySQL that we also want to join the business table's "business_ID" to the similar column in the inspections table. And, finally, we will use the AND operator to join the inspections "date" field that we updated in the inspections table to the date field in the violations table. (NOTE:When using three tables in a relational database, we need to ensure that each table is linked in a meaningful way. When dealing with two tables, that's only one link. When dealing with three tables, we need three links, (business to inspections, business to violations, and violations to inspections) to ensure that each row only references the row that it should.)

9. Run the query, which will take a few seconds. Your result should look like this:



```
SELECT business_1.name, business_1.Address,violations_1.Date,
violations_1.Code, violations_1.Description, inspections_1.score
FROM business_1, inspections_1, violations_1
WHERE business_1.Business_id = violations_1.Business_id
AND business_1.Business_id = inspections_1.business_id
AND inspections_1.date = violations_1.Date;
```

name	Address	Date	Code	Description	score
MEDITERANEAN PASTRY & PIZZA	2698 ALTA VISTA DRIVE	2009-09-22 00:00:00	FWA	Floors, walls, and ceilings ...	0
MEDITERANEAN PASTRY & PIZZA	2698 ALTA VISTA DRIVE	2009-09-22 00:00:00	FSA	Food contact surfaces pr...	0
MEDITERANEAN PASTRY & PIZZA	2698 ALTA VISTA DRIVE	2010-03-18 00:00:00	FSA	Food contact surfaces pr...	0
MEDITERANEAN PASTRY & PIZZA	2698 ALTA VISTA DRIVE	2010-03-18 00:00:00	FCD	Food is held at 4	0
MEDITERANEAN PASTRY & PIZZA	2698 ALTA VISTA DRIVE	2010-03-18 00:00:00	SFD	Separate raw foods from ...	0
MEDITERANEAN PASTRY & PIZZA	2698 ALTA VISTA DRIVE	2009-09-18 00:00:00	HHA	Hot holding: minimum of ...	0
MEDITERANEAN PASTRY & PIZZA	2698 ALTA VISTA DRIVE	2009-09-18 00:00:00	FSA	Food contact surfaces pr...	0

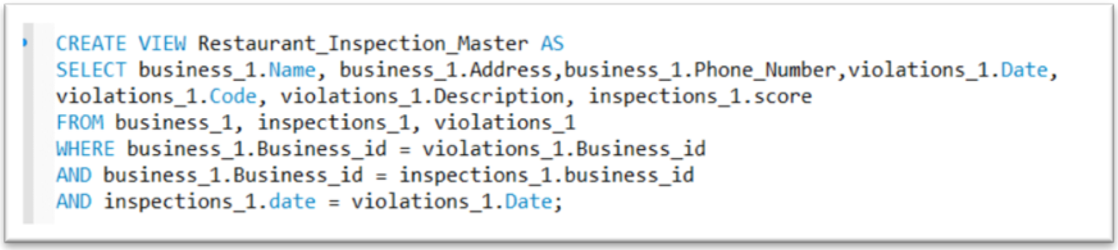
	Time	Action	Message
1	11:19:40	SELECT business_1.name, business_1.Address,violations_1.Date, violations_1....	32666 row(s) returned

10. The result returns 32,666 rows.

11. Since working with the entire table, does us little good, we want to query this table to find out which establishments are being inspected the most, and for what kinds of infractions? What areas of town contain the highest concentration of violators? Are establishments being nabbed repeatedly for the same violations? Answers to all these questions are only possible when linking the tables and using WHERE and GROUP BY statements to filter and summarize the key bits of data.

12. But before we do that, let's create a VIEW from this table. As we learned in the second query using the city of Ottawa's fire hydrant parking violation database, a VIEW is actually a query that pulls records from each of the specified tables in the SELECT statement.

13. So let’s create a VIEW from the relational database we’ve created from the three restaurant inspection tables, and then break down the query.

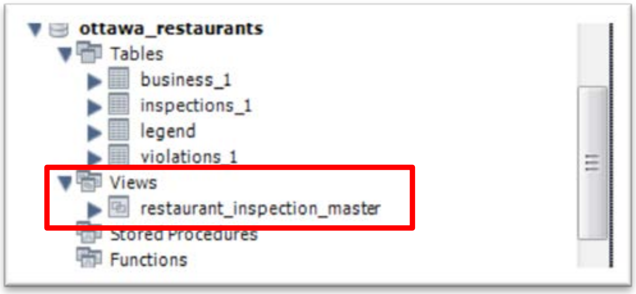


```
CREATE VIEW Restaurant_Inspection_Master AS
SELECT business_1.Name, business_1.Address,business_1.Phone_Number,violations_1.Date,
violations_1.Code, violations_1.Description, inspections_1.score
FROM business_1, inspections_1, violations_1
WHERE business_1.Business_id = violations_1.Business_id
AND business_1.Business_id = inspections_1.business_id
AND inspections_1.date = violations_1.Date;
```

- As we saw in the second tutorial using the city of Ottawa fire hydrant parking violation data, a VIEW. As Ben Forta points out on page 214 of his excellent book entitled “MySQL: CRASH COURSE”, “views are exceptionally useful for simplifying the use of calculated fields.” On page 208, under the subheading “Why Use Views”, he points to some common uses: To reuse SQL statements
- To simplify complex SQL operations. After the query is written, it can easily be reused easily, without having to know the details of the underlying query itself
- To expose parts of a table instead of complete tables
- To secure data. Users can be given access to specific subsets of tables instead of entire tables.
- To change data formatting and representation. Views can return data formatted and presented differently from their underlying tables.

14. With the restaurant-inspection database, we’re using the VIEW to make running queries easier. In this case, our view is called Restaurant_Inspection_Master.

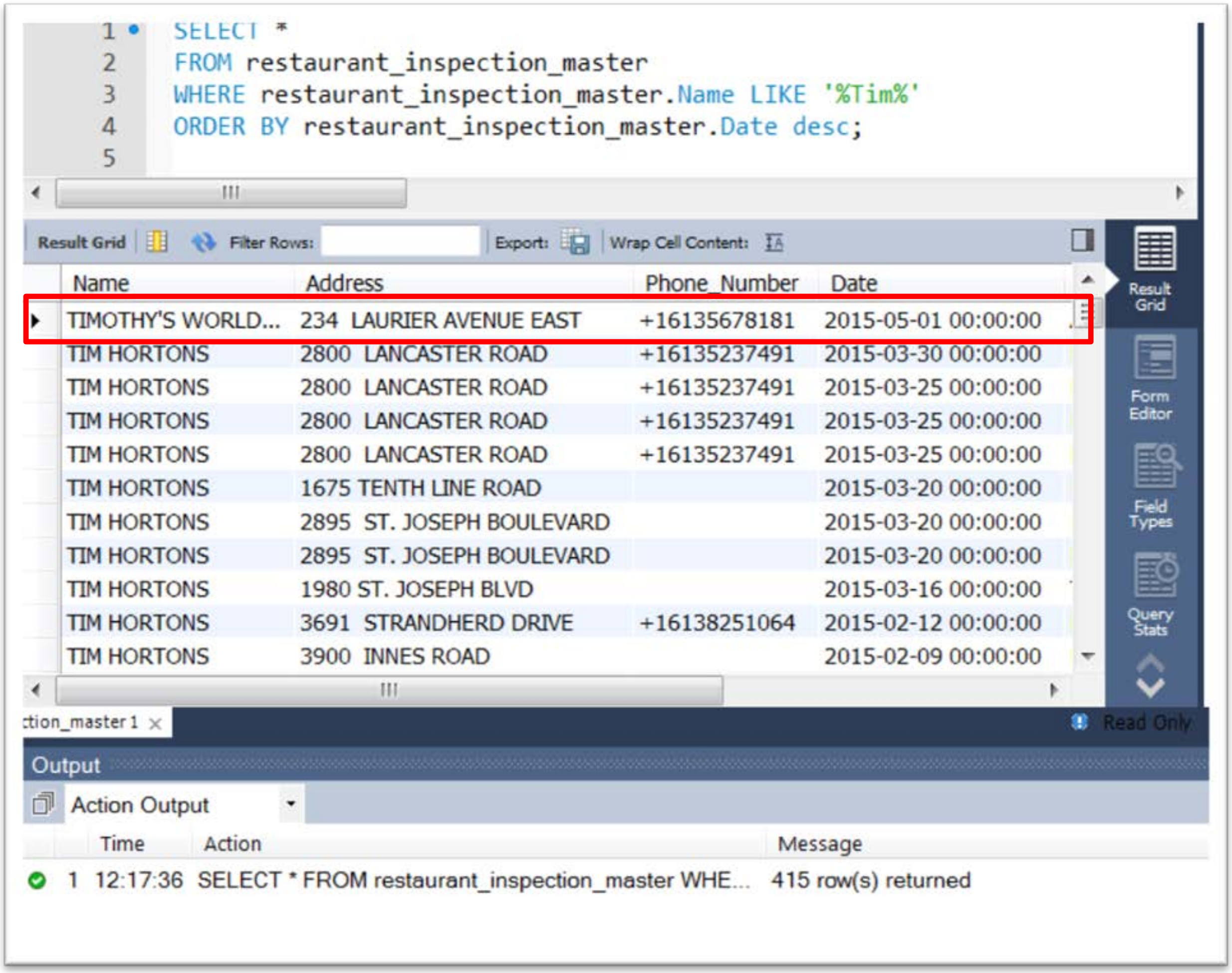
15. Run the query and refresh the menu on the left to find the view under the “Views”, section of the SCHEMA for these datasets.



If the view is not there, just refresh the menu.

16. Now we can create queries to find patterns worth pursuing. For instance, let’s check out what’s happening with Tim

Hortons restaurants.

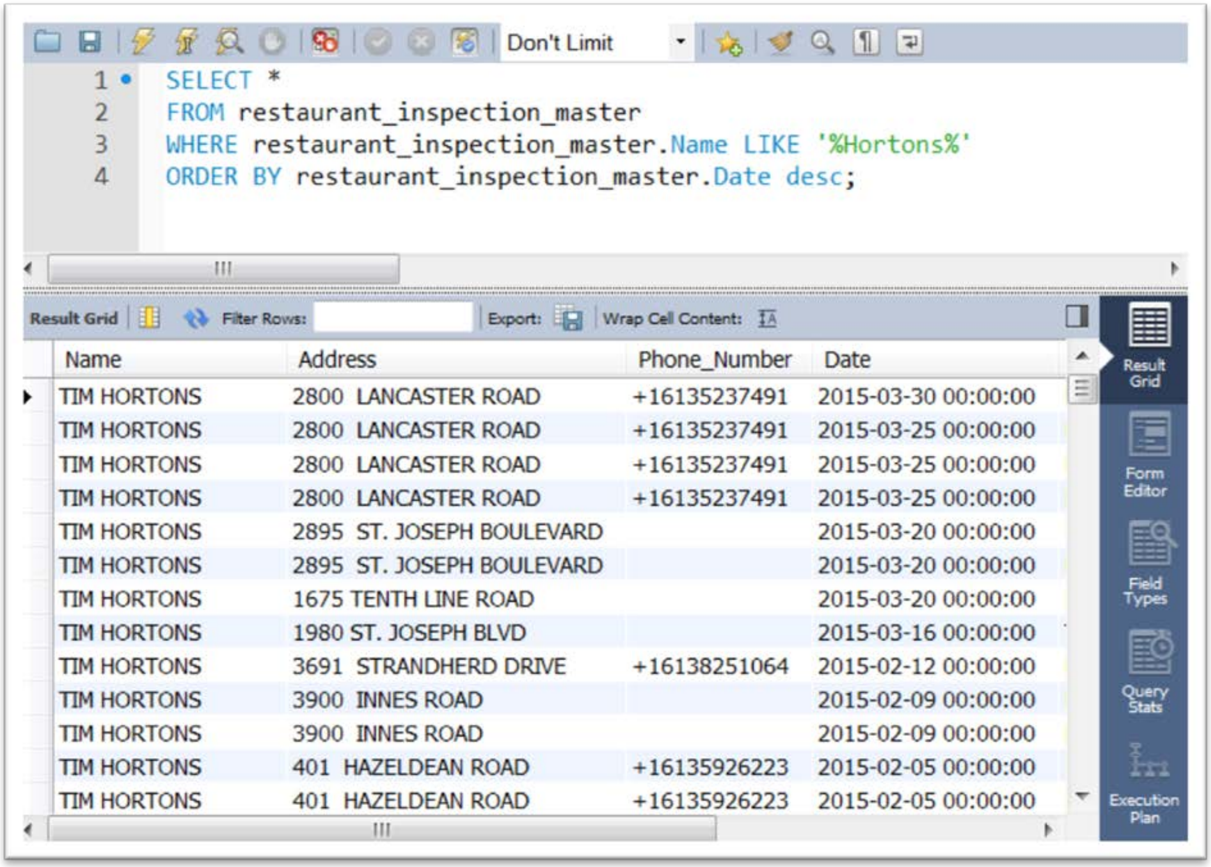


```
SELECT *
FROM restaurant_inspection_master
WHERE restaurant_inspection_master.Name LIKE '%Tim%'
ORDER BY restaurant_inspection_master.Date desc;
```

Name	Address	Phone_Number	Date
TIMOTHY'S WORLD...	234 LAURIER AVENUE EAST	+16135678181	2015-05-01 00:00:00
TIM HORTONS	2800 LANCASTER ROAD	+16135237491	2015-03-30 00:00:00
TIM HORTONS	2800 LANCASTER ROAD	+16135237491	2015-03-25 00:00:00
TIM HORTONS	2800 LANCASTER ROAD	+16135237491	2015-03-25 00:00:00
TIM HORTONS	2800 LANCASTER ROAD	+16135237491	2015-03-25 00:00:00
TIM HORTONS	1675 TENTH LINE ROAD		2015-03-20 00:00:00
TIM HORTONS	2895 ST. JOSEPH BOULEVARD		2015-03-20 00:00:00
TIM HORTONS	2895 ST. JOSEPH BOULEVARD		2015-03-20 00:00:00
TIM HORTONS	1980 ST. JOSEPH BLVD		2015-03-16 00:00:00
TIM HORTONS	3691 STRANDHERD DRIVE	+16138251064	2015-02-12 00:00:00
TIM HORTONS	3900 INNES ROAD		2015-02-09 00:00:00

	Time	Action	Message
1	12:17:36	SELECT * FROM restaurant_inspection_master WHE...	415 row(s) returned

17. You'll notice that in the "WHERE" statement, we used the wild card ("%"), which is similar to the filter "contains" that we learned in Excel. We use the wild card to get every variation on the name, such as misspellings. (NOTE: Computer-Assisted Reporting discusses wildcards on pages 191-192.) You could also use the term '%Hortons%', which would eliminate the "Timothy's World" highlighted above.



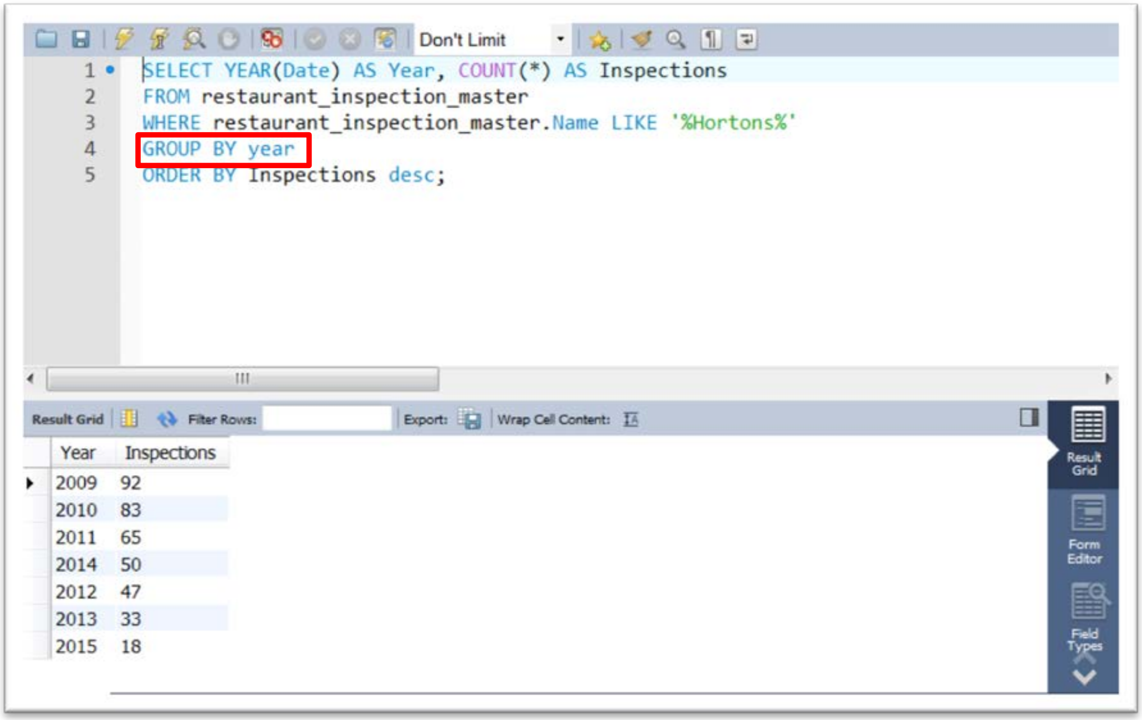
```
SELECT *
FROM restaurant_inspection_master
WHERE restaurant_inspection_master.Name LIKE '%Hortons%'
ORDER BY restaurant_inspection_master.Date desc;
```

Name	Address	Phone_Number	Date
TIM HORTONS	2800 LANCASTER ROAD	+16135237491	2015-03-30 00:00:00
TIM HORTONS	2800 LANCASTER ROAD	+16135237491	2015-03-25 00:00:00
TIM HORTONS	2800 LANCASTER ROAD	+16135237491	2015-03-25 00:00:00
TIM HORTONS	2800 LANCASTER ROAD	+16135237491	2015-03-25 00:00:00
TIM HORTONS	2895 ST. JOSEPH BOULEVARD		2015-03-20 00:00:00
TIM HORTONS	2895 ST. JOSEPH BOULEVARD		2015-03-20 00:00:00
TIM HORTONS	1675 TENTH LINE ROAD		2015-03-20 00:00:00
TIM HORTONS	1980 ST. JOSEPH BLVD		2015-03-16 00:00:00
TIM HORTONS	3691 STRANDHERD DRIVE	+16138251064	2015-02-12 00:00:00
TIM HORTONS	3900 INNES ROAD		2015-02-09 00:00:00
TIM HORTONS	3900 INNES ROAD		2015-02-09 00:00:00
TIM HORTONS	401 HAZELDEAN ROAD	+16135926223	2015-02-05 00:00:00
TIM HORTONS	401 HAZELDEAN ROAD	+16135926223	2015-02-05 00:00:00

Also notice that the wildcard -- the percentage sign operator -- is bookended by a quotation. That's because text used in the WHERE line must be surrounded by quotation marks. This is not the case for numbers.

18. To get more practice, select some other names from your business table.

19. Now let’s add the GROUP BY statement to our query to see how many times Tim Hortons has been inspected each year.



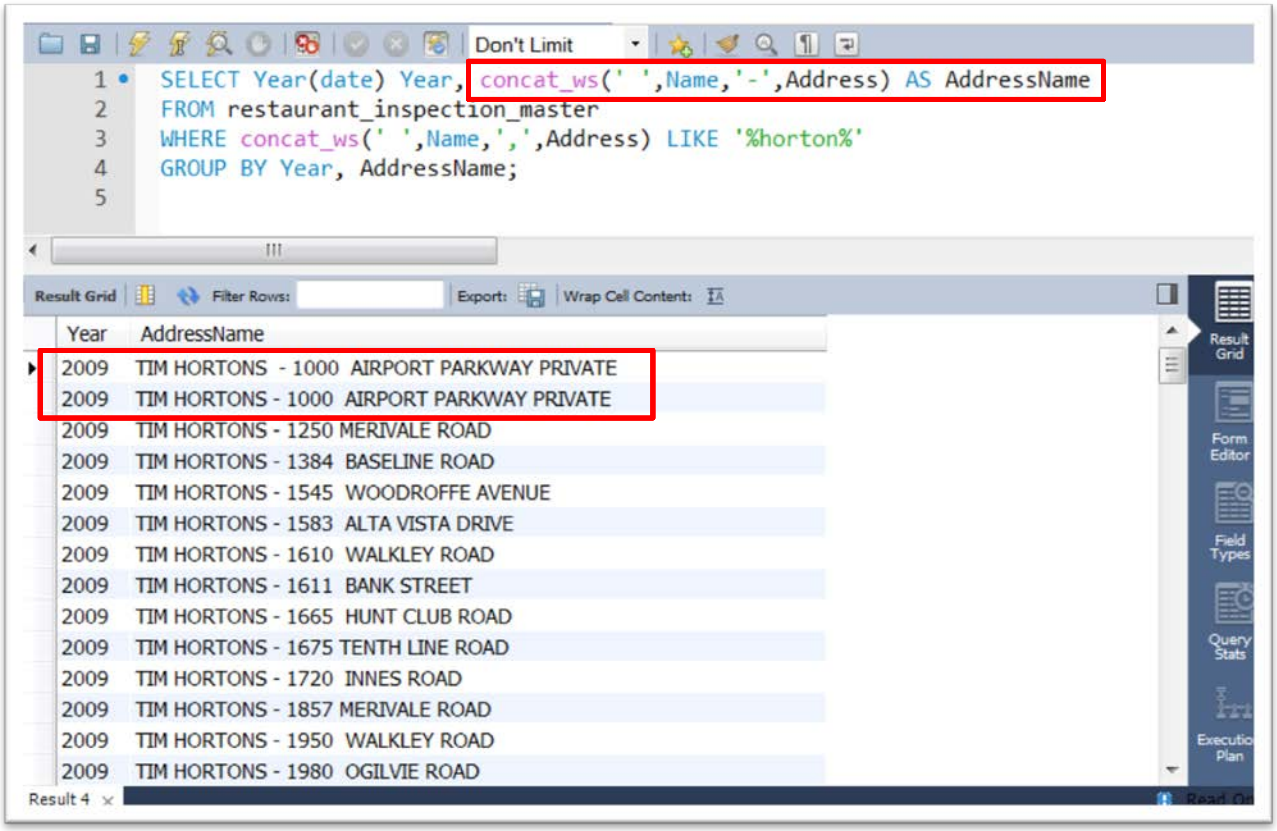
```
SELECT YEAR(Date) AS Year, COUNT(*) AS Inspections
FROM restaurant_inspection_master
WHERE restaurant_inspection_master.Name LIKE '%Hortons%'
GROUP BY year
ORDER BY Inspections desc;
```

Year	Inspections
2009	92
2010	83
2011	65
2014	50
2012	47
2013	33
2015	18

20. In this query, we have used the “Year” function to pull the year of the date column and given the column the name or alias, “Year”. We’re using COUNT to count all the inspections and have given that new column the alias, “Inspections”. In the FROM line, we’re using the VIEW we’ve created. After WHERE, we are specifying that we want all restaurants that contain the word “Tim”. The advantage of using alias is that it gives columns more appropriate names that are simpler to include your query. In this case, instead of “GROUP BY YEAR(Date)”, we can simply use the alias, “Year”. The same thing goes for the ORDER BY statement, which tells MySQL to order the inspection counts in descending order.

21. Now let’s perform one more query that will add even more power to your analysis. As we can see in the first Tim Hortons

screen shot, restaurants from some locations commit more violations than restaurants from other locations. So if we wanted to determine the Tim Horton's restaurant with the greatest number of violations, we would need to combine the "Name" column with the "Address" column. To do this, we would use the "CONCATENATION" function discussed on pages 192-194 of our textbook. This is what the query looks like:

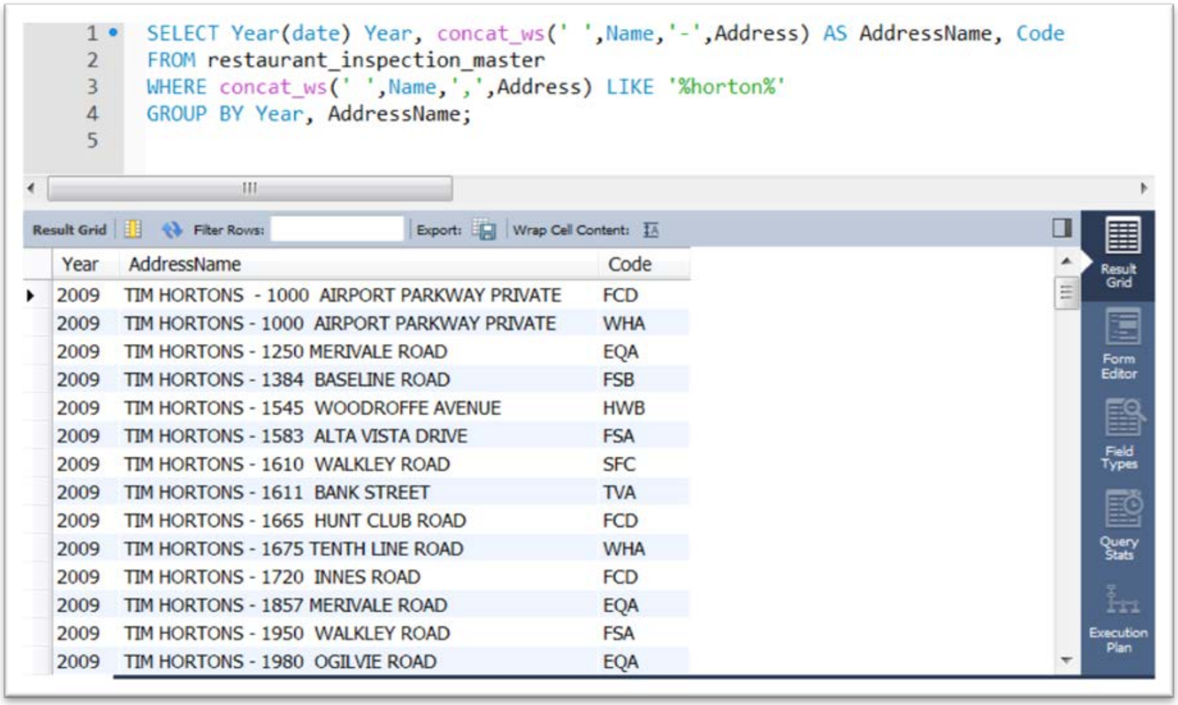


```
SELECT Year(date) Year, concat_ws(' ',Name,'-',Address) AS AddressName
FROM restaurant_inspection_master
WHERE concat_ws(' ',Name,',',Address) LIKE '%horton%'
GROUP BY Year, AddressName;
```

Year	AddressName
2009	TIM HORTONS - 1000 AIRPORT PARKWAY PRIVATE
2009	TIM HORTONS - 1000 AIRPORT PARKWAY PRIVATE
2009	TIM HORTONS - 1250 MERIVALE ROAD
2009	TIM HORTONS - 1384 BASELINE ROAD
2009	TIM HORTONS - 1545 WOODROFFE AVENUE
2009	TIM HORTONS - 1583 ALTA VISTA DRIVE
2009	TIM HORTONS - 1610 WALKLEY ROAD
2009	TIM HORTONS - 1611 BANK STREET
2009	TIM HORTONS - 1665 HUNT CLUB ROAD
2009	TIM HORTONS - 1675 TENTH LINE ROAD
2009	TIM HORTONS - 1720 INNES ROAD
2009	TIM HORTONS - 1857 MERIVALE ROAD
2009	TIM HORTONS - 1950 WALKLEY ROAD
2009	TIM HORTONS - 1980 OGILVIE ROAD

22. Okay, that's better. We can see the restaurants grouped according to their addresses. But why are there two lines for the Tim Hortons at 1000 Airport Parkway Private in 2009? Well, it could be because they were cited for two, separate code infractions. To test this theory, we need to add the "Code"

column.



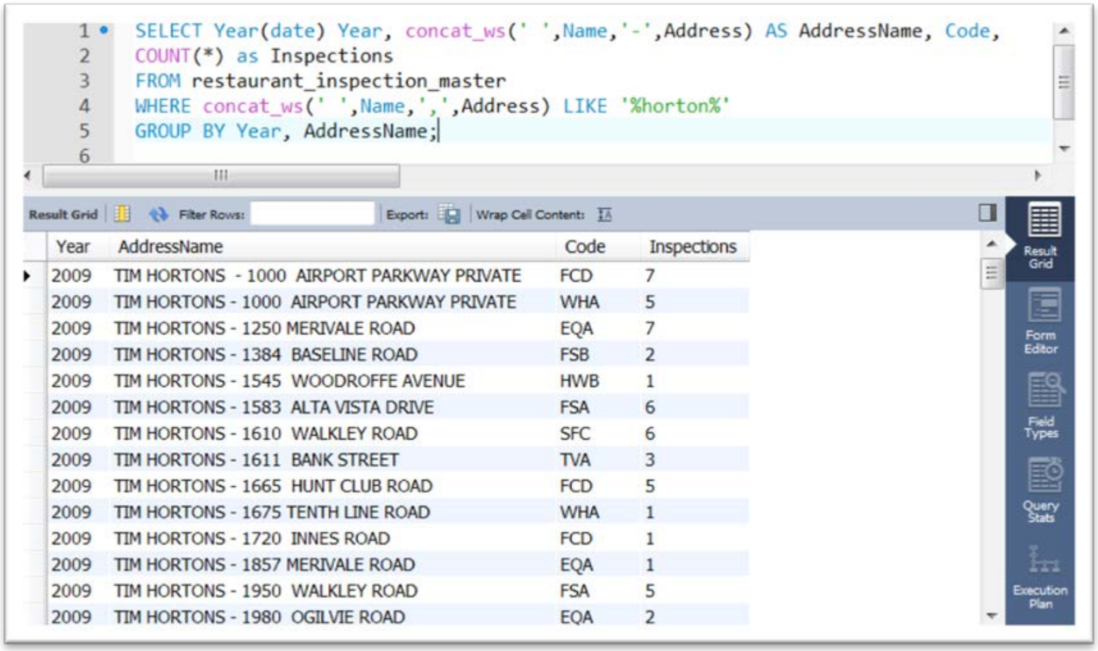
```
SELECT Year(date) Year, concat_ws(' ',Name,'-',Address) AS AddressName, Code
FROM restaurant_inspection_master
WHERE concat_ws(' ',Name,',',Address) LIKE '%horton%'
GROUP BY Year, AddressName;
```

Year	AddressName	Code
2009	TIM HORTONS - 1000 AIRPORT PARKWAY PRIVATE	FCD
2009	TIM HORTONS - 1000 AIRPORT PARKWAY PRIVATE	WHA
2009	TIM HORTONS - 1250 MERIVALE ROAD	EQA
2009	TIM HORTONS - 1384 BASELINE ROAD	FSB
2009	TIM HORTONS - 1545 WOODROFFE AVENUE	HWB
2009	TIM HORTONS - 1583 ALTA VISTA DRIVE	FSA
2009	TIM HORTONS - 1610 WALKLEY ROAD	SFC
2009	TIM HORTONS - 1611 BANK STREET	TVA
2009	TIM HORTONS - 1665 HUNT CLUB ROAD	FCD
2009	TIM HORTONS - 1675 TENTH LINE ROAD	WHA
2009	TIM HORTONS - 1720 INNES ROAD	FCD
2009	TIM HORTONS - 1857 MERIVALE ROAD	EQA
2009	TIM HORTONS - 1950 WALKLEY ROAD	FSA
2009	TIM HORTONS - 1980 OGILVIE ROAD	EQA

23. Now we have our explanation. The restaurant violated separate codes. To find out a little bit more about the inspections, you can go to the city's website that provides a bit more detail.

24. Now what if we wanted to determine how many times each of the restaurants at the particular addresses were inspected each year? To do this, we simply need to perform a simple

count.



```
SELECT Year(date) Year, concat_ws(' ',Name,'-',Address) AS AddressName, Code,
COUNT(*) as Inspections
FROM restaurant_inspection_master
WHERE concat_ws(' ',Name,',',Address) LIKE '%horton%'
GROUP BY Year, AddressName;
```

Year	AddressName	Code	Inspections
2009	TIM HORTONS - 1000 AIRPORT PARKWAY PRIVATE	FCD	7
2009	TIM HORTONS - 1000 AIRPORT PARKWAY PRIVATE	WHA	5
2009	TIM HORTONS - 1250 MERIVALE ROAD	EQA	7
2009	TIM HORTONS - 1384 BASELINE ROAD	FSB	2
2009	TIM HORTONS - 1545 WOODROFFE AVENUE	HWB	1
2009	TIM HORTONS - 1583 ALTA VISTA DRIVE	FSA	6
2009	TIM HORTONS - 1610 WALKLEY ROAD	SFC	6
2009	TIM HORTONS - 1611 BANK STREET	TVA	3
2009	TIM HORTONS - 1665 HUNT CLUB ROAD	FCD	5
2009	TIM HORTONS - 1675 TENTH LINE ROAD	WHA	1
2009	TIM HORTONS - 1720 INNES ROAD	FCD	1
2009	TIM HORTONS - 1857 MERIVALE ROAD	EQA	1
2009	TIM HORTONS - 1950 WALKLEY ROAD	FSA	5
2009	TIM HORTONS - 1980 OGILVIE ROAD	EQA	2

25. There are many ways to slice and dice this data to come up with possible story ideas. Because these tutorials build upon one another, it's important to use the previous guides as references, as well as the textbook.
26. And don't forget that after each query, you can export the table as a "csv" file to continue working with it, especially if you want display the data in a chart or map.
27. To download the queries that we've used for this tutorial, please click here right click on the file and remove the "txt" extension to leave you with the file that reads: ScriptsForSecondRestaurantInspectionTutorial.sql