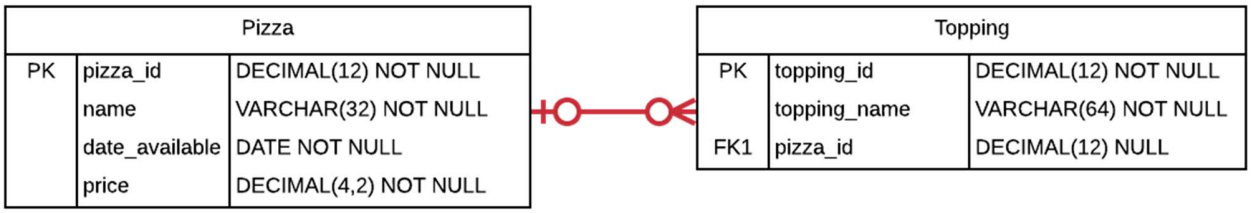
Section One – Relating Data

Section Background

To practice relating data, you will be working with the following simplified Pizza and Toppings schema.



In this schema, the Pizza table contains a primary key, the name of the pizza (for example, “Veggie” or “Meat Lovers”), the date when the pizza became available to order, and the price of the pizza. The Topping table contains a primary key, the name of the topping (such as “Sausage” or “Peppers”), and a foreign key that references the Pizza the topping is put on. The foreign key enforces the relationship between Pizza and Topping so that many toppings can be a part of a pizza. The foreign key is nullable since a particular topping may not have been assigned to a pizza (for example, perhaps a topping is available as an add-on but not part of the standard ingredients). There can also exist a pizza that has no toppings, namely, a plain pizza that only has tomato sauce and spice.

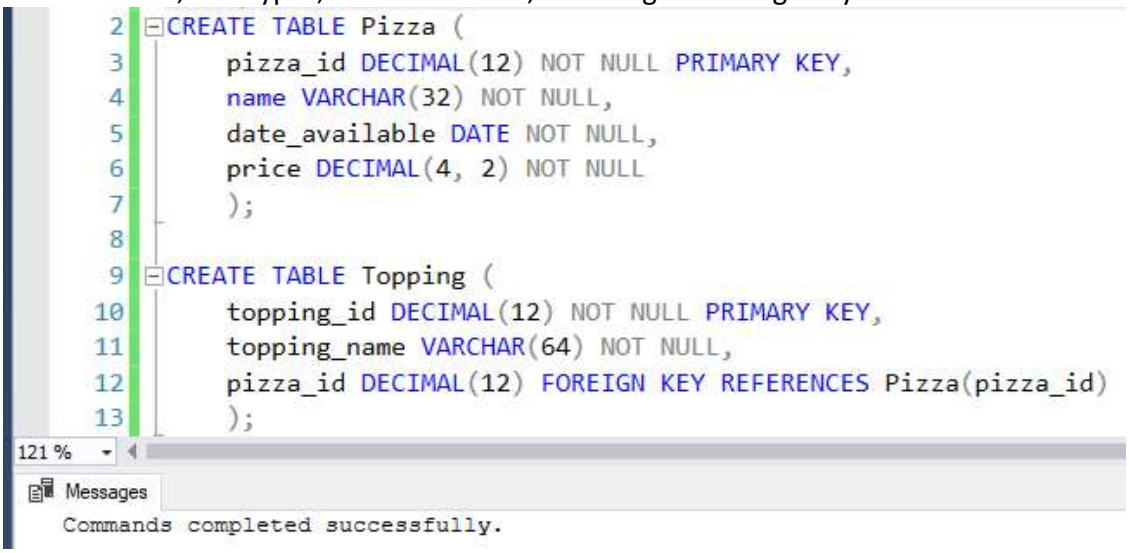
The schema is intentionally simplified when compared to what you might see in a real-world production schema. The schema only allows a particular topping to be a standard ingredient for one Pizza. The schema does not record a history of price changes as the price changes, nor does it support special pricing during special events. Many other attributes that would exist in a production database are not present. The current complexity is sufficient; additional complexity in the schema would not aid your learning at this point.

Do not worry if you don’t yet fully understand foreign keys and relationships. The Lab 2 explanations document gives you the information you need to complete the steps in this lab.

As a reminder, for each step that requires SQL, make sure to capture a screenshot of the command and the results of its execution.

Section Steps

1. *Creating the Table Structure* – Create the Pizza and Toppings tables, including all of their columns, datatypes, and constraints, including the foreign key constraint.



2. *Populating the Tables* – Insert at least four rows into the Pizza table. Two of the rows should have the values given below.

Pizza 1
name = Plain
date_available = 6/13/2020
price = $9.89
toppings: This pizza has no toppings.

Pizza 2
name = Downtown Masterpiece
date_available = 9/23/2020
price = $10.79
- toppings: You choose at least two veggie toppings for this pizza.
- For the other rows, you insert the ids, names, dates, prices, and toppings of your choosing (maybe you have some favorite pizzas?). Ensure that these other pizzas you create have at least two toppings.
- Lastly, insert an extra topping that is not associated with any pizza, that is, the topping should be an “add-on” which is not included in any pizza’s standard toppings.

```sql
-- Pizzas
INSERT INTO Pizza (pizza_id, name, date_available, price)
VALUES (1, 'Plain', '6/13/2020', 9.89);

INSERT INTO Pizza (pizza_id, name, date_available, price)
VALUES (2, 'Downtown Masterpiece', '9/23/2020', 10.79);

INSERT INTO Pizza (pizza_id, name, date_available, price)
VALUES (3, 'Baby Spinach and Onions', '6/13/2021', 12.89);

INSERT INTO Pizza (pizza_id, name, date_available, price)
VALUES (4, 'Bell Pepper and Mushroom', '3/14/2020', 8.79);
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

```sql
-- Toppings
--- Pizza 1 has no toppings.
--- Pizza 2 has two toppings.
INSERT INTO Topping (topping_id, topping_name, pizza_id)
VALUES (1, 'Baby Spinach', 2);

INSERT INTO Topping (topping_id, topping_name, pizza_id)
VALUES (2, 'Mushroom', 2);

--- Pizza 3 has two topping.
INSERT INTO Topping (topping_id, topping_name, pizza_id)
VALUES (3, 'Baby Spinach', 3);

INSERT INTO Topping (topping_id, topping_name, pizza_id)
VALUES (4, 'Onions', 3);

--- Pizza 4 has two topping.
INSERT INTO Topping (topping_id, topping_name, pizza_id)
VALUES (5, 'Bell Pepper', 4);

INSERT INTO Topping (topping_id, topping_name, pizza_id)
VALUES (6, 'Mushroom', 4);

-- Add-on topping - not included in any pizza's standard toppings.
INSERT INTO Topping (topping_id, topping_name, pizza_id)
VALUES (7, 'Artichoke', NULL);
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

Select all rows in both tables to view what you inserted.

```
82 SELECT *
83 FROM Pizza
```

121 %

Results | Messages

	pizza_id	name	date_available	price
1	1	Plain	2020-06-13	9.89
2	2	Downtown Masterpiece	2020-09-23	10.79
3	3	Baby Spinach and Onions	2021-06-13	12.89
4	4	Bell Pepper and Mushroom	2020-03-14	8.79

```
85 SELECT *
86 FROM Topping;
```

121 %

Results | Messages

	topping_id	topping_name	pizza_id
1	1	Baby Spinach	2
2	2	Mushroom	2
3	3	Baby Spinach	3
4	4	Onions	3
5	5	Bell Pepper	4
6	6	Mushroom	4
7	7	Artichoke	NULL

3. *Invalid Reference Attempt* – As an exercise, attempt to insert a topping that references a pizza that doesn't exist.

```
89 INSERT INTO Topping (topping_id, topping_name, pizza_id)
90 VALUES (8, 'Baby Spinach', 5);
```

121 %

Messages

```
Msg 547, Level 16, State 0, Line 89
The INSERT statement conflicted with the FOREIGN KEY constraint "FK__Topping__pizza_i__1940BAED".
The conflict occurred in database "master", table "dbo.Pizza", column 'pizza_id'.
The statement has been terminated.
```

Summarize:

a. why the insertion failed, and
   The insertion failed because the foreign key constraint was violated. Here the foreign key constraint allows NULL to be inserted as the value for the foreign key or a value that exists in the referenced column pizza_id in the Pizza table. However, since the Pizza table does not contain any pizza with the pizza_id 5, the above error is shown.
b. how you would interpret the error message from your RDBMS so that you know that the error indicates the Pizza reference is invalid.
   The error message says that the 'INSERT statement conflicted with the

FOREIGN KEY constraint' named 'FK__Topping__pizza_i__1940BAED'; so the problem is the violation of the foreign key constraint.
Since we know that the foreign key is applied to the pizza_id column in the table Topping, which references the pizza_id column in the table Pizza, we can assume that the Pizza reference is invalid. This interpretation is further supported by the error message saying exactly where the conflict occurred: in the column pizza_id in the table Pizza.

4. *Listing Pizzas with Toppings* – With a single SQL query, fulfill the following request:

List the names of the pizzas that have toppings, and the names of all of the toppings that go with each pizza.

```
93  SELECT name, topping_name
94  FROM Pizza
95  INNER JOIN Topping ON Pizza.pizza_id = Topping.pizza_id;
```

121 %

Results | Messages

	name	topping_name
1	Downtown Masterpiece	Baby Spinach
2	Downtown Masterpiece	Mushroom
3	Baby Spinach and Onions	Baby Spinach
4	Baby Spinach and Onions	Onions
5	Bell Pepper and Mushroom	Bell Pepper
6	Bell Pepper and Mushroom	Mushroom

From a technical SQL perspective, explain why some rows in the Pizza table and some rows in the Toppings table were not listed.
First, the Cartesian product of all rows in both the Pizza and Topping tables is determined (every row in the Pizza table is combined with every row in the Topping table). Which will look like this (see table below):

	pizza_id	name	date_available	price	topping_id	topping_name	pizza_id
1	1	Plain	2020-06-13	9.89	1	Baby Spinach	2
2	1	Plain	2020-06-13	9.89	2	Mushroom	2
3	1	Plain	2020-06-13	9.89	3	Baby Spinach	3
4	1	Plain	2020-06-13	9.89	4	Onions	3
5	1	Plain	2020-06-13	9.89	5	Bell Pepper	4
6	1	Plain	2020-06-13	9.89	6	Mushroom	4
7	1	Plain	2020-06-13	9.89	7	Artichoke	NULL
8	2	Downtown Masterpiece	2020-09-23	10.79	1	Baby Spinach	2
9	2	Downtown Masterpiece	2020-09-23	10.79	2	Mushroom	2
10	2	Downtown Masterpiece	2020-09-23	10.79	3	Baby Spinach	3
11	2	Downtown Masterpiece	2020-09-23	10.79	4	Onions	3
12	2	Downtown Masterpiece	2020-09-23	10.79	5	Bell Pepper	4
13	2	Downtown Masterpiece	2020-09-23	10.79	6	Mushroom	4
14	2	Downtown Masterpiece	2020-09-23	10.79	7	Artichoke	NULL
15	3	Baby Spinach and On...	2021-06-13	12.89	1	Baby Spinach	2
16	3	Baby Spinach and On...	2021-06-13	12.89	2	Mushroom	2
17	3	Baby Spinach and On...	2021-06-13	12.89	3	Baby Spinach	3
18	3	Baby Spinach and On...	2021-06-13	12.89	4	Onions	3
19	3	Baby Spinach and On...	2021-06-13	12.89	5	Bell Pepper	4
20	3	Baby Spinach and On...	2021-06-13	12.89	6	Mushroom	4
21	3	Baby Spinach and On...	2021-06-13	12.89	7	Artichoke	NULL
22	4	Bell Pepper and Mush...	2020-03-14	8.79	1	Baby Spinach	2
23	4	Bell Pepper and Mush...	2020-03-14	8.79	2	Mushroom	2
24	4	Bell Pepper and Mush...	2020-03-14	8.79	3	Baby Spinach	3
25	4	Bell Pepper and Mush...	2020-03-14	8.79	4	Onions	3
26	4	Bell Pepper and Mush...	2020-03-14	8.79	5	Bell Pepper	4
27	4	Bell Pepper and Mush...	2020-03-14	8.79	6	Mushroom	4
28	4	Bell Pepper and Mush...	2020-03-14	8.79	7	Artichoke	NULL

And in the second step, the join condition is then applied. Since inner join defines that the resulting table contains all the rows that have matching values for the two pizza_id columns and as can be seen above, not every row has matching values in pizza_id columns, this means that not all columns are included in the resulting output, and this is why some rows in the Pizza table and some rows in the Toppings table were not listed. Below the table above after the inner join has been applied (here with all columns):

	pizza_id	name	date_available	price	topping_id	topping_name	pizza_id
1	2	Downtown Masterpiece	2020-09-23	10.79	1	Baby Spinach	2
2	2	Downtown Masterpiece	2020-09-23	10.79	2	Mushroom	2
3	3	Baby Spinach and Onions	2021-06-13	12.89	3	Baby Spinach	3
4	3	Baby Spinach and Onions	2021-06-13	12.89	4	Onions	3
5	4	Bell Pepper and Mushroom	2020-03-14	8.79	5	Bell Pepper	4
6	4	Bell Pepper and Mushroom	2020-03-14	8.79	6	Mushroom	4

5. *Listing All Pizzas* – Fulfill the following request:

List the names and availability date of all pizzas whether or not they have toppings. For the pizzas that have toppings, list the names of the toppings that go with each of those pizzas. Order the list by the availability date, oldest to newest.
There are two kinds of joins that can be used to satisfy this request. Write two queries using each type of join to satisfy this request:

```
98  SELECT name, date_available, topping_name
99  FROM Pizza
100 LEFT JOIN Topping ON Pizza.pizza_id = Topping.pizza_id
101 ORDER BY date_available ASC;
102
```

121 %

Results | Messages

	name	date_available	topping_name
1	Bell Pepper and Mushroom	2020-03-14	Bell Pepper
2	Bell Pepper and Mushroom	2020-03-14	Mushroom
3	Plain	2020-06-13	NULL
4	Downtown Masterpiece	2020-09-23	Baby Spinach
5	Downtown Masterpiece	2020-09-23	Mushroom
6	Baby Spinach and Onions	2021-06-13	Baby Spinach
7	Baby Spinach and Onions	2021-06-13	Onions

```
104 SELECT name, date_available, topping_name
105 FROM Pizza
106 FULL JOIN Topping ON Pizza.pizza_id = Topping.pizza_id
107 WHERE name IS NOT NULL
108 ORDER BY date_available ASC;
```

121 %

Results | Messages

	name	date_available	topping_name
1	Bell Pepper and Mushroom	2020-03-14	Bell Pepper
2	Bell Pepper and Mushroom	2020-03-14	Mushroom
3	Plain	2020-06-13	NULL
4	Downtown Masterpiece	2020-09-23	Baby Spinach
5	Downtown Masterpiece	2020-09-23	Mushroom
6	Baby Spinach and Onions	2021-06-13	Baby Spinach
7	Baby Spinach and Onions	2021-06-13	Onions

6. *Listing All Toppings* – Fulfill the following request:

List the names of all toppings whether or not they go with a pizza, and the names of

the pizzas the toppings go with. Order the list by topping name in reverse alphabetical order.
Just as with step #5, there are two kinds of joins that can be used to satisfy this request. Write two queries using each type of join to satisfy this request.

```
111 SELECT topping_name, name
112 FROM Pizza
113 RIGHT JOIN Topping ON Pizza.pizza_id = Topping.pizza_id
114 ORDER BY topping_name DESC;
```

121 %

Results | Messages

	topping_name	name
1	Onions	Baby Spinach and Onions
2	Mushroom	Downtown Masterpiece
3	Mushroom	Bell Pepper and Mushroom
4	Bell Pepper	Bell Pepper and Mushroom
5	Baby Spinach	Baby Spinach and Onions
6	Baby Spinach	Downtown Masterpiece
7	Artichoke	NULL

```
117 SELECT topping_name, name
118 FROM Pizza
119 FULL JOIN Topping ON Pizza.pizza_id = Topping.pizza_id
120 WHERE topping_name IS NOT NULL
121 ORDER BY topping_name DESC;
```

121 %

Results | Messages

	topping_name	name
1	Onions	Baby Spinach and Onions
2	Mushroom	Downtown Masterpiece
3	Mushroom	Bell Pepper and Mushroom
4	Bell Pepper	Bell Pepper and Mushroom
5	Baby Spinach	Baby Spinach and Onions
6	Baby Spinach	Downtown Masterpiece
7	Artichoke	NULL

7. *Listing All Pizzas and All Toppings* – Fulfill the following request with a single SQL query:

   List the names of all pizzas and all toppings, as well as which pizzas go with which toppings. Order the list alphabetically by pizza name then by topping name.

```
124 SELECT name, topping_name
125 FROM Pizza
126 FULL JOIN Topping ON Pizza.pizza_id = Topping.pizza_id
127 ORDER BY name, topping_name;
128
```

121 %

Results | Messages

	name	topping_name
1	NULL	Artichoke
2	Baby Spinach and Onions	Baby Spinach
3	Baby Spinach and Onions	Onions
4	Bell Pepper and Mushroom	Bell Pepper
5	Bell Pepper and Mushroom	Mushroom
6	Downtown Masterpiece	Baby Spinach
7	Downtown Masterpiece	Mushroom
8	Plain	NULL

Section Two – Expressing Data

Section Background

While it is certainly useful to directly extract values as they are stored in a database, it is more useful in some contexts to manipulate these values to derive a different result. In this section we practice using value manipulation techniques to transform data values in useful ways. For example, what if we want to tell a customer exactly how much money they need to give for a purchase? We could extract a price and sales tax from the database, but it would be more useful to compute a price with tax as a single value by multiplying the two together and rounding appropriately, and formatting it as a currency, as illustrated in the figure below.

Less Useful to Customer	
price	**tax_percent**
7.99	8.5

More Useful to Customer
price_with_tax
$8.67

We do not need to store the price with tax, because we can derive it when we need it.

As another example, what if we need to send an email communication to a customer by name? We could extract the prefix, first name, and last name of the customer, but it would be more useful to properly format the name by combining them in proper order, as illustrated below.

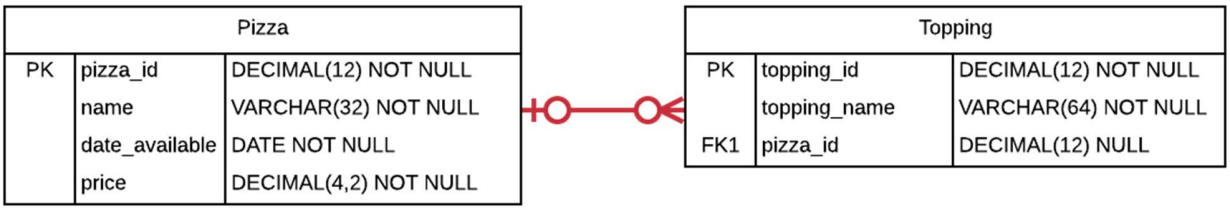
Less Useful to Customer		
prefix	**first_name**	**last_name**
Mr.	Seth	Nemes

More Useful to Customer
name
Mr. Seth Nemes

Again, we do not need to store the formatted name, because we can derive it when we need it from its constituent parts. Manipulating raw data values stored in database tables can yield a variety of useful results we need without adding the burden of storing every such result.

In this section, you use expressions to manipulate and format data values. The first several steps in this section teach you several important concepts needed to correctly use expressions, including attributes of SQL clients, operator precedence, datatype precedence, and formatting functions. The later steps have you use this knowledge to manipulate and format data values.

You work with the same Pizza and Toppings schema from Section One. The schema is illustrated below again for your review.



Section Steps

8. *Formatting as Money* – Fulfill the following request with a single query:

   Management of the pizza shop wants to review its pizza pricing. List the names and prices of all pizzas, making sure to format the price monetarily in U.S. dollars (for example, "$11.99").

```
130 SELECT name, FORMAT(price, '$.00') AS price_formatted
131 FROM Pizza;
```

121 %

Results | Messages

	name	price_formatted
1	Plain	$9.89
2	Downtown Masterpiece	$10.79
3	Baby Spinach and Onions	$12.89
4	Bell Pepper and Mushroom	$8.79

9. *Using Expressions* – Fulfill the following request with a single query:

   The pizza shop is running a special where every pizza is discounted by a $1.75. List the names and discounted prices of all pizzas, making sure to format the price monetarily in U.S. dollars.

```
134 SELECT name, FORMAT(price - 1.75, '$.00') AS discounted_price_formatted
135 FROM Pizza;
```

121 %

Results | Messages

	name	discounted_price_formatted
1	Plain	$8.14
2	Downtown Masterpiece	$9.04
3	Baby Spinach and Onions	$11.14
4	Bell Pepper and Mushroom	$7.04

10. *Advanced Formatting* – Fulfill the following request with a single query:

    The pizza shop wants to mail out mailers that promotes the toppings it offers, tied into the pizzas it sells. The shop wants each line in the mailer formatted like

"ToppingName (PizzaName - Price)", and wants the lines ordered alphabetically by topping name. For example, if a "Meat Lover's" pizza costs $10.00 and has two toppings – Sausage and Pepperoni – the results would have two lines for this pizza:
Pepperoni (Meat Lover's - $10.00)
Sausage (Meat Lover's - $10.00)

```
138 SELECT topping_name + ' (' + name + ' - ' + FORMAT(price, '$.00') + ')' AS promotion_entries
139 FROM Pizza
140 RIGHT JOIN Topping ON Pizza.pizza_id = Topping.pizza_id
141 WHERE name is NOT NULL
142 ORDER BY topping_name;
```

121 %

Results | Messages

	promotion_entries
1	Baby Spinach (Downtown Masterpiece - $10.79)
2	Baby Spinach (Baby Spinach and Onions - $12.89)
3	Bell Pepper (Bell Pepper and Mushroom - $8.79)
4	Mushroom (Bell Pepper and Mushroom - $8.79)
5	Mushroom (Downtown Masterpiece - $10.79)
6	Onions (Baby Spinach and Onions - $12.89)

Note, the question says '(…) mailers that promotes the toppings it offers, tied into the pizzas it sells'; since the add-on topping 'Artichoke' is not included on any pizza, I did not list it.

Section Three – Advanced Data Expression

Section Background

Boolean expressions can become complex, yet are essential to filtering results in SQL. Boolean expressions can determine if a set of columns meet a possibly complex set of conditions. In this section, you learn to work with more advanced Boolean expressions.

Modern relational databases have the ability to calculate a column automatically. Such a column is identified by many terms – *generated, computed, calculated, derived, and virtual*. If one column can be calculated from the values in other columns, it is best practice to avoid storing the extra value, because it can become out of sync with the other columns. That is, if one of the columns change value, but the derived column is not updated, the data becomes inconsistent. In fact, storing derived columns is a form of data redundancy.

As described in Section Two, one option to avoid storage is to dynamically calculate derived values in a SQL query using an expression. Another option, the topic of this section, is to create a generated column, and have the database calculate it automatically.

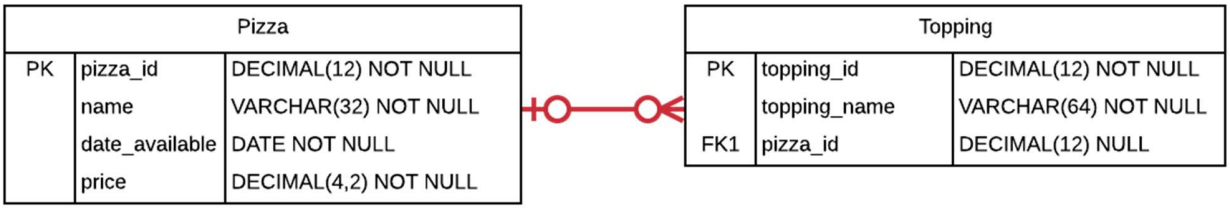
Using an example from Section Two, if one column contains a price, and another column contains a tax percentage, a third column could contain the price after tax.

price	**tax_percent**
7.99	8.5

price_with_tax
$8.67

This third value can be calculated by multiplying the price by the tax percentage, and performing proper rounding to two decimal points. We would not want to store the price with tax, because we can derive it. In this section, you learn how to create generated columns to contain values that can be derived from other columns.

You work with the same Pizza and Toppings schema from prior sections. The schema is illustrated below again for your review.



Section Steps

11. *Evaluating Boolean Expressions* – Indicate the final values for each of the Boolean expressions below. You must show your work for full credit, by showing the value of each operation step-by-step.
    a. (true AND false) OR (true AND true)
    Final value is: **true**
    (true AND false) evaluates to: false
    (true AND true) evaluates to: true
    Thus, (true AND false) OR (true AND true) is the same as: false OR true;
    which evaluates to: true.

    b. (true OR false) AND NOT(false OR false) AND (false AND true)
    Final value is: **false**
    (true OR false) evaluates to: true
    (false OR false) evaluates to: false
    (false AND true) evaluates to: false
    Thus, main Boolean expression is same as: true AND NOT(false) AND false;
    which evaluates to: true AND true AND false;
    which evaluates to: true AND false;
    which evaluates to: false.

    c. NOT((false OR true) AND NOT(true AND true) AND (false OR true))
    Final value is: **true**
    (false OR true) evaluates to: true
    (true AND true) evaluates to: true
    (false OR true) evaluates to: true
    Thus, main Boolean expression is same as: NOT(true AND NOT(true) AND true);
    which evaluates to: NOT(true AND false AND true);
    which evaluates to: NOT(false AND true);
    which evaluates to: NOT(false);
    which evaluates to: true.

12. *Using Boolean Expressions in Queries* – Address the following scenarios.
    a. Any pizza matching the following condition is considered a *signature pizza* for the pizza shop: Any pizza, except for the "Plain" pizza, that is available on or after 5/1/2020, with a price of $9.55 or higher, is a signature pizza. Write a query that shows the name and price of all signature pizzas.

```
152 SELECT name, FORMAT(price, '$.00') AS price
153 FROM Pizza
154 WHERE name <> 'Plain' AND date_available >= '5/1/2020' AND price >= 9.55;
```

121 %

Results | Messages

	name	price
1	Downtown Masterpiece	$10.79
2	Baby Spinach and Onions	$12.89

b. The pizza shop also has one *flagship pizza* that sets the shop apart from other pizza shops. First, define your own conditions for this flagship pizza, making sure the conditions include the name, date, and price. Then write a query that shows the name and price of the flagship pizza. It's fine if you'd like to insert another row of pizzas to become your flagship pizza.

```
154 SELECT name, FORMAT(price, '$.00') AS price
155 FROM Pizza
156 WHERE name LIKE '%Baby Spinach%' AND date_available >= '3/13/2018' AND price >= 12.15;
```

121 %

Results | Messages

	name	price
1	Baby Spinach and Onions	$12.89

13. *Using Generated Columns* – Address the following.

a. Define a new generated column named *special_price*, which gives a lower price for the pizza for when the pizza shop offers specials (such as on holidays or during weekly specials). You determine the percentage or fixed value discount for the special price. Then write a query that lists out the name of all pizzas, along with their regular and special prices.

```
160 ALTER TABLE Pizza
161 ADD special_price AS (price - 2);
```

121 %

Messages

```
Commands completed successfully.
```

```
163 SELECT name, FORMAT(price, '$.00') AS regular_price, special_price
164 FROM Pizza;
```

121 %

Results | Messages

	name	regular_price	special_price
1	Plain	$9.89	7.89
2	Downtown Masterpiece	$10.79	8.79
3	Baby Spinach and Onions	$12.89	10.89
4	Bell Pepper and Mushroom	$8.79	6.79

The special price was deliberately not formatted with the $ symbol in front to show that the generated column named special_price was actually added.

b. Address #12a again in a different way. First, define a generated column named *is_signature* on the Pizza table, which indicates whether it's a signature pizza or not. Then write a query that lists only the signature pizzas. Include relevant columns in the result.

```
167 ALTER TABLE Pizza
168 ADD is_signature AS
169 (CASE
170    WHEN name <> 'Plain' AND date_available >= '5/1/2020' AND price >= 9.55 THEN 1
171    ELSE 0
172 END);
```

121 %

Messages

```
Commands completed successfully.
```

```
174 SELECT name, FORMAT(price, '$.00') AS price
175 FROM Pizza
176 WHERE is_signature = 1;
```

121 %

Results | Messages

	name	price
1	Downtown Masterpiece	$10.79
2	Baby Spinach and Onions	$12.89