******SQL Server Research Project Repository******

This is my project/portfolio for SQL Server class
at Saint Martin's University. Purpose of this research
project is to demostrate my understanding of SQL Server 2016
through actual queries and its output. My goal is to input queries
that have combination of statements that would be beyond the basic
fundamentals that I have learned either through class or through the
textbook.

This specific repository should have 50 queries along with it's output
and my feedback on understanding of the query. The sample database that I
am
using is from the website SQLServerTutorials.net [Insert Hyperlink]

Please feel free to reach out to me if you have any feedback regarding
the queries or if it can be further improved! Also, I have another
repository
that have comparison between SQL Server and SQLite on their capabilities
and
structure, that you can view it here!
[Insert Hyperlink]

Alright, let's get started!

On the below would be a sample of the structure that I will be
consistently
using throughout the 50 queries.

Sample Structure:
[SELECT statement]

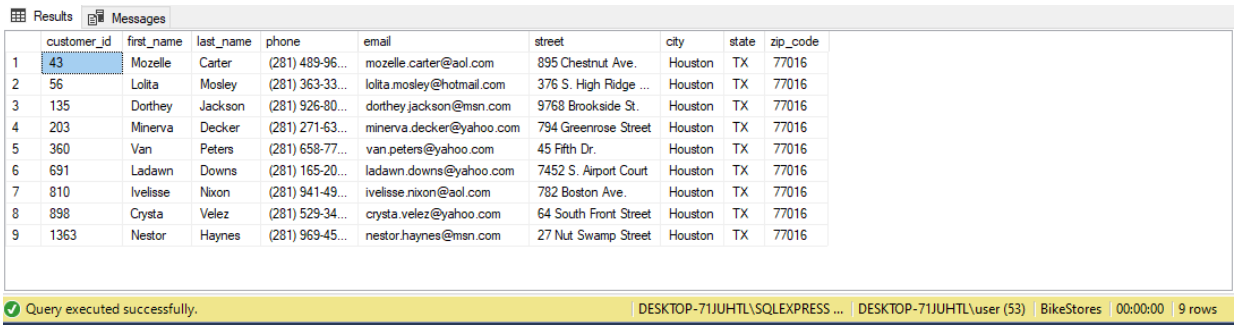
```
SELECT
    first_name,
    last_name
FROM
    sales.customers;
```

Source: Self-Programmed

```
/* The query would select the first and last name
   of the customers from sales table */
```


#1
[SELECT | FROM | WHERE]

```
select * from sales.customers
where city = 'houston'
```



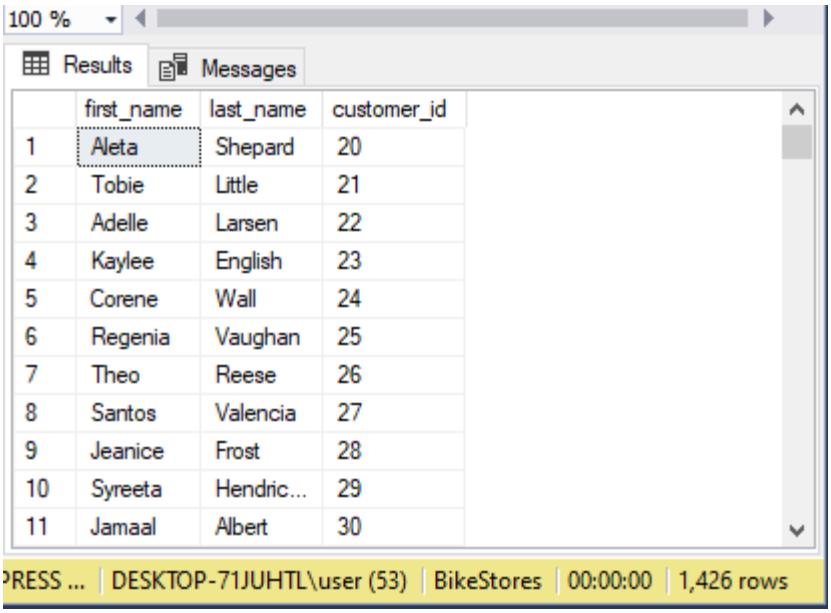
Results | Messages

	customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	43	Mozelle	Carter	(281) 489-96...	mozelle.carter@aol.com	895 Chestnut Ave.	Houston	TX	77016
2	56	Lolita	Mosley	(281) 363-33...	lolita.mosley@hotmail.com	376 S. High Ridge ...	Houston	TX	77016
3	135	Dorthey	Jackson	(281) 926-80...	dorthey.jackson@msn.com	9768 Brookside St.	Houston	TX	77016
4	203	Minerva	Decker	(281) 271-63...	minerva.decker@yahoo.com	794 Greenrose Street	Houston	TX	77016
5	360	Van	Peters	(281) 658-77...	van.peters@yahoo.com	45 Fifth Dr.	Houston	TX	77016
6	691	Ladawn	Downs	(281) 165-20...	ladawn.downs@yahoo.com	7452 S. Airport Court	Houston	TX	77016
7	810	Ivelisse	Nixon	(281) 941-49...	ivelisse.nixon@aol.com	782 Boston Ave.	Houston	TX	77016
8	898	Crysta	Velez	(281) 529-34...	crysta.velez@yahoo.com	64 South Front Street	Houston	TX	77016
9	1363	Nestor	Haynes	(281) 969-45...	nestor.haynes@msn.com	27 Nut Swamp Street	Houston	TX	77016

Query executed successfully. | DESKTOP-71JUHTL\SQLEXPRESS ... | DESKTOP-71JUHTL\user (53) | BikeStores | 00:00:00 | 9 rows

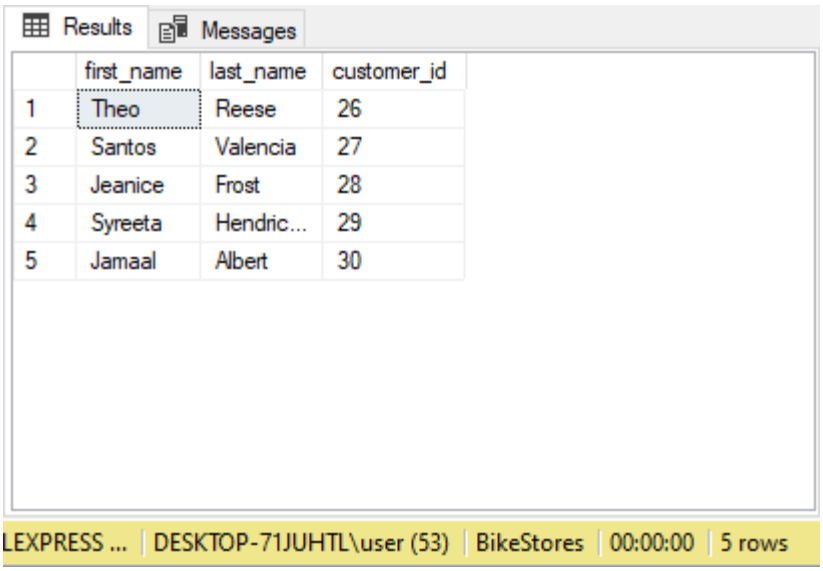
Source: Self-Programmed

```
/* The classic SELECT statement selects FROM a schema ('sales'),
   while 'customers' is the table that is related to 'sales.'
   Therefore using FROM statement should have format like this
   [schema_name.table_name]

   WHERE statement can further specify the condition of the query
   that it will further narrowing down the search.
   WHERE statement along with HAVING, GROUP BY, ORDER BY,
   and others are SQL Clauses.
*/

--------------------------------------------------------------------------------

#2
[WHERE | GROUP BY | ORDER BY]

SELECT
    first_name,
    last_name,
    customer_id
FROM
    sales.customers
WHERE
    customer_id >= 20
GROUP BY
    first_name,
    last_name,
    customer_id
ORDER BY
    customer_id
```



Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-where/

```
/* With ORDER BY, I can sort the table specified in an ascending
   order. In this example, I have used 'customer_id' to be sorted.
   Noticed that 'customer_id' started with 20, which that is because
   the WHERE have a condition that 'customer_id' is at or more than 20.
*/

--------------------------------------------------------------------------------

#3
[HAVING]

SELECT
    first_name,
    last_name,
    customer_id
FROM
    sales.customers
WHERE
    customer_id BETWEEN 20 AND 30
GROUP BY
    first_name,
    last_name,
    customer_id
HAVING
    customer_id >25
ORDER BY
    customer_id
```




```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
having/

/* HAVING clause works with a condition and the condition in the
   example is 'customer_id' is more than 25. Even though I can specify
   than in the WHERE clause, however, this to shows that WHERE and HAVING
   can be in the same query.
*/

--------------------------------------------------------------------------
--------

#4
[HAVING AND WHERE Comparison]

SELECT
    first_name,
    last_name,
    customer_id,
    state
FROM
    sales.customers
WHERE
    zip_code >= 30000
GROUP BY
    first_name,
    last_name,
    customer_id,
    state
HAVING
    customer_id >25
ORDER BY
    state
```

Results | Messages

	first_name	last_name	customer_id	state
1	Jamaal	Albert	30	CA
2	Williemae	Holloway	31	CA
3	Araceli	Golden	32	CA
4	Deloris	Burke	33	CA
5	Ronna	Butler	40	CA
6	Monika	Berg	46	CA
7	Bridgette	Guerra	47	CA
8	Saturnina	Garner	53	CA
9	Neil	Mccall	60	CA
10	Tommie	Melton	67	CA
11	Katelin	Kennedy	69	CA

KPRESS ... | DESKTOP-71JUHTL\user (53) | BikeStores | 00:00:00 | 418 rows

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-having/

```
/* WHERE clause can be based off on any column in the base table and
   HAVING clause only can refer to the result of the SELECT statement
   or the GROUP BY clause.
   To illustrate this concept, the WHERE clause is set to have condition
   of a 'zipcode' table that is or higher than 30000. Notice that zipcode
   are not part of the SELECT statement. Whereas HAVING is bound by the
   SELECT and GROUP BY
*/

--------------------------------------------------------------------------------

#5
[SELECT DISTINCT]

SELECT DISTINCT state

from sales.customers;
```

Results | Messages

	state
1	TX
2	CA
3	NY

.EXPRESS ... | DESKTOP-71JUHTL\user (52) | BikeStores | 00:00:00 | 3 rows

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-select-distinct/

```
/* SELECT DISTINCT as simple as it may seems, it does the same thing as
   query #1 does, which it is to narrow down the search in a particular
   search but with leaving out WHERE clause. I believe the reasoning
behind
   it for using distinct instead of WHERE clause, is when you know
exactly
   what you're searching and that the search will return with a true
result
   instead of WHERE clause that is a condition, and that condition may or
may
   not return a true result.
*/

--------------------------------------------------------------------------
--------

#6
[LIKE operator]

SELECT first_name, email, city, state, zip_code
FROM sales.customers
WHERE email LIKE '%gmail%';
```

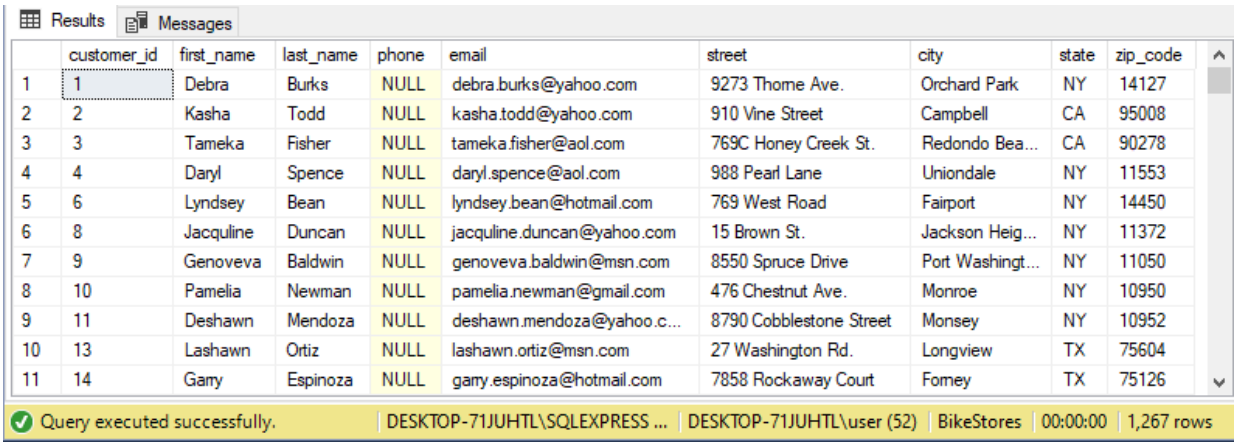
Results | Messages

	first_name	email	city	state	zip_code
1	Pamelia	pamelia.newman@gmail.c...	Monroe	NY	10950
2	Linnie	linnie.branch@gmail.com	Plattsburgh	NY	12901
3	Tobie	tobie.little@gmail.com	Victoria	TX	77904
4	Adelle	adelle.larsen@gmail.com	East Northp...	NY	11731
5	Regenia	regenia.vaughan@gmail.c...	Mahopac	NY	10541
6	Theo	theo.reese@gmail.com	Long Beach	NY	11561
7	Jamaal	jamaal.albert@gmail.com	Torrance	CA	90505
8	Melia	melia.brady@gmail.com	Maspeth	NY	11378
9	Ronna	ronna.butler@gmail.com	Encino	CA	91316
10	Monika	monika.berg@gmail.com	Encino	CA	91316
11	Cesar	cesar.jackson@gmail.com	Liverpool	NY	13090

JHTL\SQLEXPRESS ... | DESKTOP-71JUHTL\user (52) | BikeStores | 00:00:00 | 305 rows

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
like/

/* The query I was looking for in the bike shop is to find the first
name, email,
   city, state and zipcodes of users that have email account that is
GMAIL.
   Therefore in this query I have used the WHERE clause to define the
condition
   and LIKE operator to filter the results from the sales.customer table
with the
   specific result list using SELECT statement
*/
```

```
-------------------------------------------------------------------------
---------

#7
[NULL logic]

SELECT *
FROM sales.customers
WHERE phone is NULL
```



Results | Messages

	customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Bea...	CA	90278
4	4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
6	8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heig...	NY	11372
7	9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washingt...	NY	11050
8	10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950
9	11	Deshawn	Mendoza	NULL	deshawn.mendoza@yahoo.c...	8790 Cobblestone Street	Monsey	NY	10952
10	13	Lashawn	Ortiz	NULL	lashawn.ortiz@msn.com	27 Washington Rd.	Longview	TX	75604
11	14	Garry	Espinoza	NULL	garry.espinoza@hotmail.com	7858 Rockaway Court	Forney	TX	75126

Query executed successfully. | DESKTOP-71JUHTL\SQLEXPRESS ... | DESKTOP-71JUHTL\user (52) | BikeStores | 00:00:00 | 1,267 rows

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
null/

/* The NULL logic is used to find empty value by assigning it to a search
condition
   or clause. Which in this case, I will be using WHERE clause to find
customer's
   contact information that does not have a phone number. From a business
perspective,
   I'll be able to email or send them a personal letter to connect and
build relationships
   with the customers, as well as updating personal information such as
phone number.
   I specifically SELECT all columns, which is the customer contact
information, but
   with the result with only customers that do not have phone number on
file in the database.
*/

-------------------------------------------------------------------------
---------

#8
[AND operator]

SELECT *
FROM sales.customers
WHERE phone is NULL
AND state = 'CA'
```

```
AND zip_code < 93000
ORDER BY zip_code ASC;
```

Results | Messages

	customer_id	first_name	last_name	phone	email	street	city	state	zip_co
1	224	Moses	Pope	NULL	moses.pope@yahoo.com	654 Theatre Street	Lawndale	CA	90260
2	254	Cristobal	Hutchins...	NULL	cristobal.hutchinson@gmail.c...	58 Washington Aven...	Lawndale	CA	90260
3	774	Nenita	Mooney	NULL	nenita.mooney@hotmail.com	10 W. Bishop Street	Lawndale	CA	90260
4	1163	Carmela	Hays	NULL	carmela.hays@aol.com	149 Pennington Ave.	Lawndale	CA	90260
5	1288	Allie	Conley	NULL	allie.conley@msn.com	96 High Point Road	Lawndale	CA	90260
6	1301	Raven	Curtis	NULL	raven.curtis@aol.com	18 Summit Lane	Lawndale	CA	90260
7	1328	Cindie	Franklin	NULL	cindie.franklin@yahoo.com	7249 Franklin St.	Lawndale	CA	90260
8	947	Angele	Castro	NULL	angele.castro@yahoo.com	15 Acacia Drive	Palos Verdes Peninsula	CA	90274
9	245	Delila	Hamilton	NULL	delila.hamilton@yahoo.com	7451 East James Ave.	Palos Verdes Peninsula	CA	90274
10	247	Muriel	Juarez	NULL	muriel.juarez@gmail.com	8073 Cemetery Drive	Palos Verdes Peninsula	CA	90274

Query executed successfully. | DESKTOP-71JUHTL\SQLEXPRESS ... | DESKTOP-71JUHTL\user (52) | BikeStores | 00:00:00 | 140 rows

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-and/

```
/* The AND operator is a boolean expression where it would return a value that is
   either TRUE, FALSE, or UNKNOWN. I'm using the previous query result to showcase
   AND operator to find a specific state in which in the example would be CA and
   I further define the condition to be zipcode number would be anything below '93000.'
*/

-------------------------------------------------------------------------------
---------

#9
[OR operator]

SELECT *
FROM
    production.products
WHERE
category_id = 6
or
brand_id > 7
ORDER BY list_price ASC;
```

Results | Messages

	product_id	product_name	brand_id	category_id	model_year	list_price
1	83	Trek Boy's Kickster - 2015/20...	9	1	2017	149.99
2	86	Trek Girl's Kickster - 2017	9	1	2017	149.99
3	268	Trek Kickster - 2018	9	1	2018	159.99
4	87	Trek Precaliber 12 Boys - 2017	9	1	2017	189.99
5	88	Trek Precaliber 12 Girls - 2017	9	1	2017	189.99
6	269	Trek Precaliber 12 Boy's - 2018	9	1	2018	199.99
7	267	Trek Precaliber 12 Girl's - 2018	9	1	2018	199.99
8	270	Trek Precaliber 16 Boy's - 2018	9	1	2018	209.99
9	271	Trek Precaliber 16 Girl's - 2018	9	1	2018	209.99
10	89	Trek Precaliber 16 Boys - 2017	9	1	2017	209.99

DESKTOP-71JUHTL\SQLEXPRESS ... | DESKTOP-71JUHTL\user (57) | BikeStores | 00:00:00 | 171 rows

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-or/

/* The OR operator would take the condition define and combine them as a
result if
   the condition result is TRUE. In the example, I placed 2 conditions to
be met
   where category by ID have to be 6 and brand ID can be anything above
7, and then
   order them by ascending in price.
*/

-------------------------------------------------------------------------
---------

#10
[BETWEEN operator]

SELECT *
FROM
    production.products
WHERE
category_id = 6 or category_id = 7
ORDER BY list_price ASC;
-------------------------------------
SELECT *
FROM
    production.products
WHERE
category_id between 6 and 7
ORDER BY list_price ASC;
```

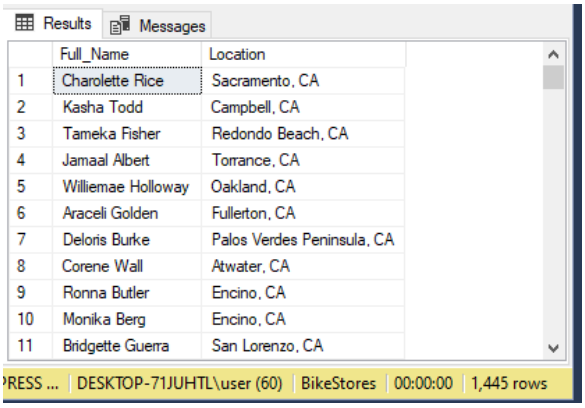
Results | Messages

	product_id	product_name	brand_id	category_id	model_year	list_price
1	1	Trek 820 - 2016	9	6	2016	379.99
2	37	Haro Flightline One ST - 2017	2	6	2017	379.99
3	112	Trek 820 - 2018	9	6	2018	379.99
4	119	Trek Kids' Neko - 2018	9	6	2018	469.99
5	125	Trek Kids' Dual Sport - 2018	9	6	2018	469.99
6	126	Surly Big Fat Dummy Frameset - 2018	8	6	2018	469.99
7	127	Surly Pack Rat Frameset - 2018	8	6	2018	469.99
8	6	Surly Ice Cream Truck Frameset - 2016	8	6	2016	469.99
9	32	Trek Farley Alloy Frameset - 2017	9	6	2017	469.99
10	33	Surly Wednesday Frameset - 2017	8	6	2017	469.99

Que... | DESKTOP-71JUHTL\SQLEXPRESS ... | DESKTOP-71JUHTL\user (52) | BikeStores | 00:00:00 | 120 rows

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
between/
```

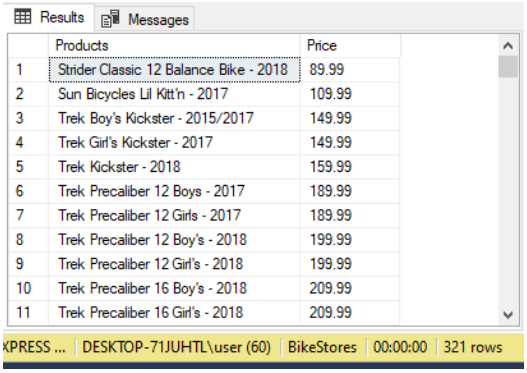
```
/* The BETWEEN operator works like the < and > comparison and both can
replace BETWEEN
   and would work the same. Another operator that would give the same
result would be
   the OR operator.
*/

-------------------------------------------------------------------------
---------

#11
[Alias]

SELECT first_name + ' ' + last_name AS Full_Name,
       city + ', ' + state AS Location
FROM sales.customers
ORDER BY state;
```



	Full_Name	Location
1	Charolette Rice	Sacramento, CA
2	Kasha Todd	Campbell, CA
3	Tameka Fisher	Redondo Beach, CA
4	Jamaal Albert	Torrance, CA
5	Williemae Holloway	Oakland, CA
6	Araceli Golden	Fullerton, CA
7	Deloris Burke	Palos Verdes Peninsula, CA
8	Corene Wall	Atwater, CA
9	Ronna Butler	Encino, CA
10	Monika Berg	Encino, CA
11	Bridgette Guerra	San Lorenzo, CA


```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
alias/

/* Alias work by putting 'AS' to combine two information into one and
named it
   entirely different. In the example, I have took the last name and
first name
   combined it to make the full name. Same as for the location. This
makes the data
   information more complete for user to read it easily.
*/

-------------------------------------------------------------------------
---------

#12
[Alias]

SELECT
    product_name 'Products', list_price 'Price'
FROM
    production.products
```

```
ORDER BY
   'Price';
```

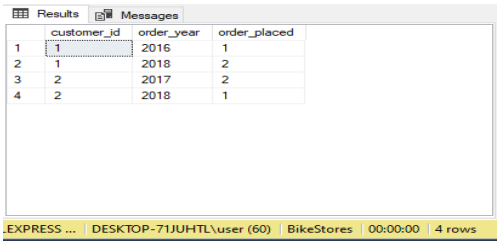


	Products	Price
1	Strider Classic 12 Balance Bike - 2018	89.99
2	Sun Bicycles Lil Kitt'n - 2017	109.99
3	Trek Boy's Kickster - 2015/2017	149.99
4	Trek Girl's Kickster - 2017	149.99
5	Trek Kickster - 2018	159.99
6	Trek Precaliber 12 Boys - 2017	189.99
7	Trek Precaliber 12 Girls - 2017	189.99
8	Trek Precaliber 12 Boy's - 2018	199.99
9	Trek Precaliber 12 Girl's - 2018	199.99
10	Trek Precaliber 16 Boy's - 2018	209.99
11	Trek Precaliber 16 Girl's - 2018	209.99

Results Messages

XPRESS ... | DESKTOP-71JUHTL\user (60) | BikeStores | 00:00:00 | 321 rows

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
alias/

/* While the example above combines information of 2 columns, this
example shows
   that alias can be assigned to a column without the combination and
without
   'AS'
*/

-------------------------------------------------------------------------
---------

#13
[GROUP BY]

SELECT
    customer_id,
    YEAR (order_date) order_year,
    COUNT (order_id) order_placed
FROM
    sales.orders
WHERE
    customer_id IN (1, 2)
GROUP BY
    customer_id,
    YEAR (order_date)
ORDER BY
    customer_id;
```

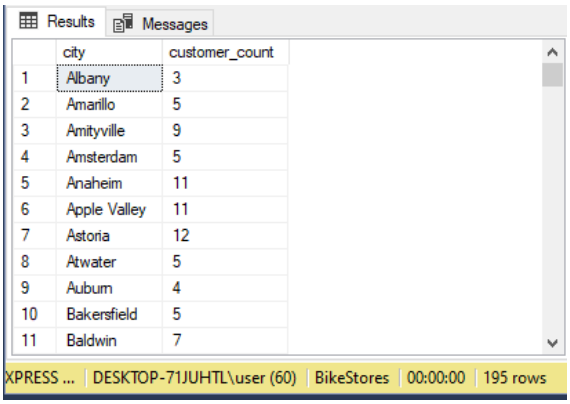


	customer_id	order_year	order_placed
1	1	2016	1
2	1	2018	2
3	2	2017	2
4	2	2018	1

Results Messages

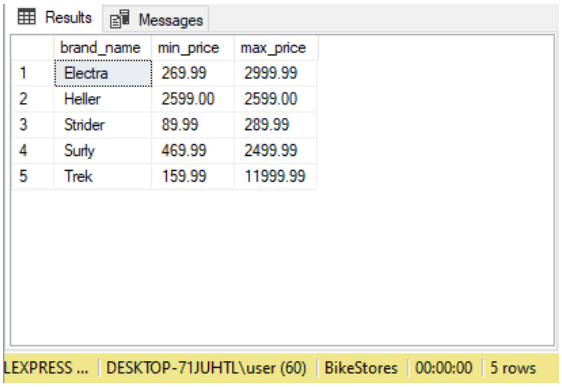
EXPRESS ... | DESKTOP-71JUHTL\user (60) | BikeStores | 00:00:00 | 4 rows

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
group-by/

/* This query with GROUP BY clause uses aggregate function like 'COUNT'
to perform
   calculation. For this matter, the count will increase by the number of
times
   the customer would order each year based on customer_id and order_id.
*/

-------------------------------------------------------------------------
---------

#14
[GROUP BY]

SELECT
    city,
    COUNT (customer_id) customer_count
FROM
    sales.customers
GROUP BY
    city
ORDER BY
    city;
```




```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
group-by/

/* Just like the example on #13, this type of queries can improve the
quality of
   business strategy in a company. If a specific location that has low
customer
   count but in a large city (ratio of customer to number of people in
the city)
   then maybe this bike store strategies marketing effort in the specific
location.
*/

-------------------------------------------------------------------------
---------
```

```
#15
[Aggregate Functions]

SELECT
    brand_name,
    MIN (list_price) min_price,
    MAX (list_price) max_price
FROM
    production.products p
INNER JOIN production.brands b ON b.brand_id = p.brand_id
WHERE
    model_year = 2018
GROUP BY
    brand_name
ORDER BY
    brand_name;
```




```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
inner-join/

/* Using MIN and MAX, this query is able to pull information from
list_price and
   run calculation of the minimum and maximum price sorted by brand name
and model
   year. As goes to analysis of possible bike category and also the
spending from
   each city or state. Just a simple query provides valuable information
in helping
   to analyse a business strategy.
*/

---------------------------------------------------------------------------
---------

#16
[Subqueries]

SELECT
    product_name,
    list_price
```

```
FROM
    production.products
WHERE
    list_price > (
        SELECT
            AVG (list_price)
        FROM
            production.products
        WHERE
            brand_id IN (
                SELECT
                    brand_id
                FROM
                    production.brands
                WHERE
                    brand_name = 'Strider'
                OR brand_name = 'Trek'
            )
    )
ORDER BY
    list_price;
```

Results | Messages

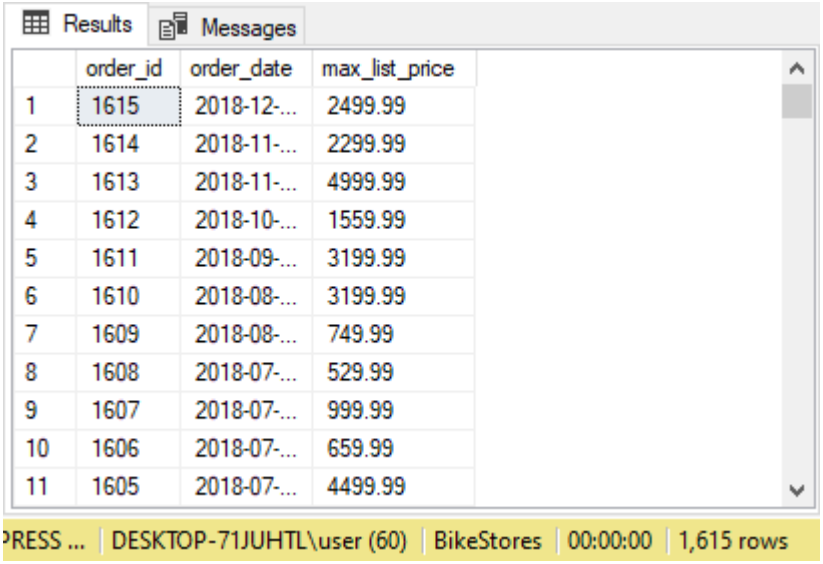
	product_name	list_price
1	Surly Karate Monkey 27.5+ Frameset - 2017	2499.99
2	Trek Fuel EX 7 29 - 2018	2499.99
3	Surly Krampus Frameset - 2018	2499.99
4	Surly Troll Frameset - 2018	2499.99
5	Trek Domane SL 5 Disc Women's - 2018	2499.99
6	Trek 1120 - 2018	2499.99
7	Trek Domane SL 5 Disc - 2018	2499.99
8	Heller Bloodhound Trail - 2018	2599.00
9	Heller Shagamaw GX1 - 2018	2599.00
10	Trek Domane S 5 Disc - 2017	2599.99
11	Electra Townie Go! 8i Ladies' - 2018	2599.99

EXPRESS ... | DESKTOP-71JUHTL\user (60) | BikeStores | 00:00:00 | 80 rows

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-subquery/

```
/* This query is example of a subquery nesting where if information and
filteration
   of data is complex, then subqueries can be used. In SQL Server,
subqueries can be
   used up to 32 levels of nesting.
*/

-------------------------------------------------------------------------
---------
```

```
#17
[Subqueries and Expression]

SELECT
    order_id,
    order_date,
    (
        SELECT
            MAX (list_price)
        FROM
            sales.order_items i
        WHERE
            i.order_id = o.order_id
    ) AS max_list_price
FROM
    sales.orders o
order by order_date desc;
```



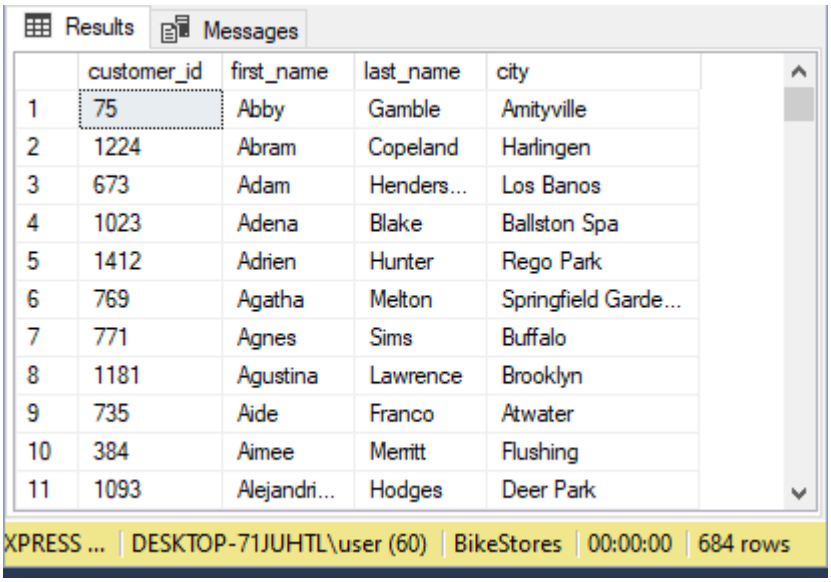
Results | Messages

	order_id	order_date	max_list_price
1	1615	2018-12-...	2499.99
2	1614	2018-11-...	2299.99
3	1613	2018-11-...	4999.99
4	1612	2018-10-...	1559.99
5	1611	2018-09-...	3199.99
6	1610	2018-08-...	3199.99
7	1609	2018-08-...	749.99
8	1608	2018-07-...	529.99
9	1607	2018-07-...	999.99
10	1606	2018-07-...	659.99
11	1605	2018-07-...	4499.99

PRESS ... | DESKTOP-71JUHTL\user (60) | BikeStores | 00:00:00 | 1,615 rows

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
subquery/

/* Within the subquery and nesting, expression can further narrow down
the data
   information. In this example, the expression compute the maximum price
from
   list_price and list it as max_list_price within the subquery nesting.
*/

---------------------------------------------------------------------------
---------

#18
[EXIST operator]

SELECT
```

```
    customer_id,
    first_name,
    last_name,
    city
FROM
    sales.customers c
WHERE
    EXISTS (
        SELECT
            customer_id
        FROM
            sales.orders o
        WHERE
            o.customer_id = c.customer_id
        AND YEAR (order_date) = 2017
    )
ORDER BY
    first_name,
    last_name;
```



Results | Messages

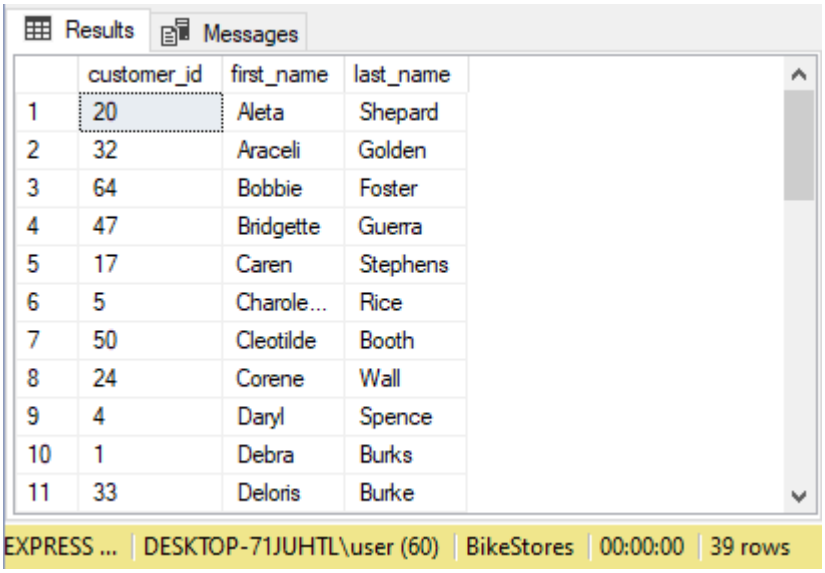
	customer_id	first_name	last_name	city
1	75	Abby	Gamble	Amityville
2	1224	Abram	Copeland	Harlingen
3	673	Adam	Henders...	Los Banos
4	1023	Adena	Blake	Ballston Spa
5	1412	Adrien	Hunter	Rego Park
6	769	Agatha	Melton	Springfield Garde...
7	771	Agnes	Sims	Buffalo
8	1181	Agustina	Lawrence	Brooklyn
9	735	Aide	Franco	Atwater
10	384	Aimee	Merritt	Flushing
11	1093	Alejandri...	Hodges	Deer Park

XPRESS ... | DESKTOP-71JUHTL\user (60) | BikeStores | 00:00:00 | 684 rows

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-exists/

```
/* Using the subquery example, this query also illustrate the EXIST
operator where
   if the subquery return result, then the EXIST operator will return
TRUE. This
   example would find customers that bought products from the bike store
in the year
   of 2017.
*/

---------------------------------------------------------------------------
---------
```

```
#19
[Correlation subquery]

SELECT
    customer_id,
    first_name,
    last_name
FROM
    sales.customers c
WHERE
    EXISTS (
        SELECT
            COUNT (*)
        FROM
            sales.orders o
        WHERE
            customer_id = c.customer_id
        GROUP BY
            customer_id
        HAVING
            COUNT (*) > 2
    )
ORDER BY
    first_name,
    last_name;
```



Results | Messages

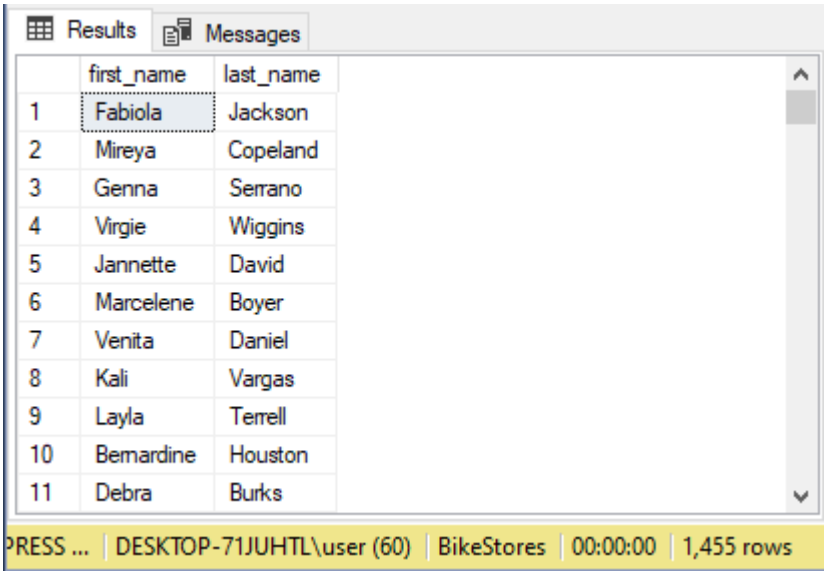
	customer_id	first_name	last_name
1	20	Aleta	Shepard
2	32	Araceli	Golden
3	64	Bobbie	Foster
4	47	Bridgette	Guerra
5	17	Caren	Stephens
6	5	Charole...	Rice
7	50	Cleotilde	Booth
8	24	Corene	Wall
9	4	Daryl	Spence
10	1	Debra	Burks
11	33	Deloris	Burke

EXPRESS ... | DESKTOP-71JUHTL\user (60) | BikeStores | 00:00:00 | 39 rows

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-subquery/

```
/* Within this query, it uses EXIST within a subquery and also HAVING
clause that
   includes count more than 2. It means that this query would find
customers that
   have ordered from the bike store for more than twice. This query can
also further
```

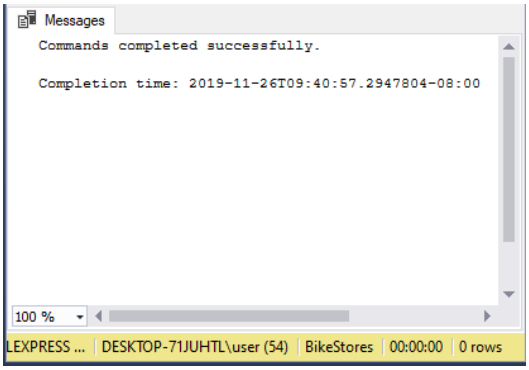
```
   break down into by year, city or state.
*/

-------------------------------------------------------------------------
---------

#20
[UNION]

SELECT
    first_name,
    last_name
FROM
    sales.staffs
UNION ALL
SELECT
    first_name,
    last_name
FROM
    sales.customers;
```




```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
union/

/* Using union, it combines rows from queries into a single result set
where in
   this example, it combines the first name and last name from staffs and
customers
   into a single result set.
*/

-------------------------------------------------------------------------
---------

#21
```

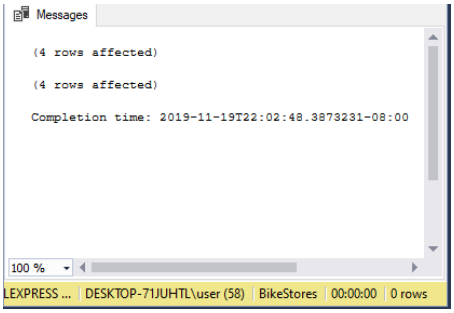
[SCHEMA]

```
CREATE SCHEMA hr;
```



Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-joins/

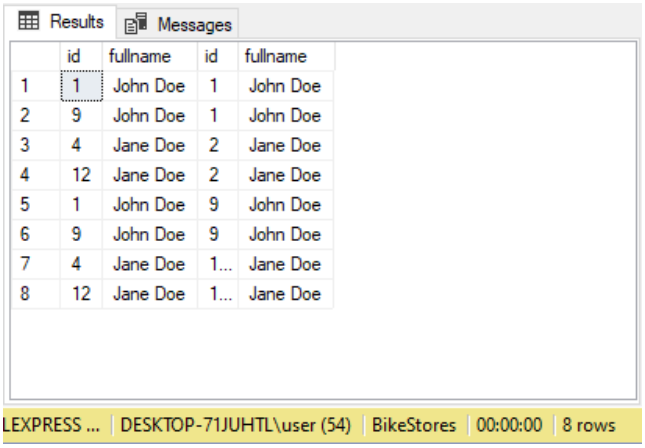
```
/* Using create, I can create schema or even tables into the schema
*/

-------------------------------------------------------------------------
---------

#22
[CREATE]

CREATE TABLE hr.candidates(
    id INT PRIMARY KEY IDENTITY,
    fullname VARCHAR(100) NOT NULL
);

CREATE TABLE hr.employees(
    id INT PRIMARY KEY IDENTITY,
    fullname VARCHAR(100) NOT NULL
);
```



Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-create-table/

```
/* Using create, I can create schema or even tables into the schema. The
'id'
```

```
   will be the PRIMARY KEY and would contain full name.
*/

-------------------------------------------------------------------------
---------

#23
[I]

INSERT INTO
    hr.candidates(fullname)
VALUES
    ('Kenneth Monroe'),
    ('Lewis Wang'),
    ('Peter Chap'),
    ('Jane Lane');


INSERT INTO
    hr.employees(fullname)
VALUES
    ('Kenny Lew'),
    ('Joe Langley'),
    ('Michael Scotty'),
    ('Jack Sparrow');
```

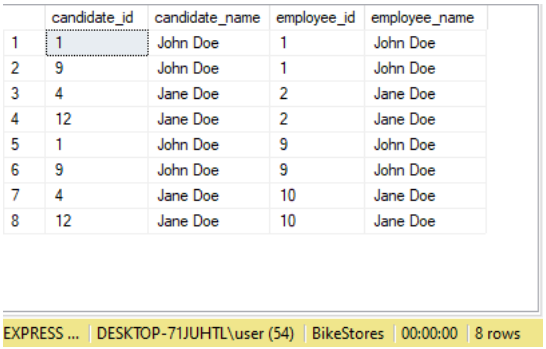


Results | Messages

	id	fullname	id	fullname
1	1	John Doe	1	John Doe
2	9	John Doe	1	John Doe
3	4	Jane Doe	2	Jane Doe
4	12	Jane Doe	2	Jane Doe
5	1	John Doe	9	John Doe
6	9	John Doe	9	John Doe
7	4	Jane Doe	1...	Jane Doe
8	12	Jane Doe	1...	Jane Doe


```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
joins/

[210_IMG23]

/* Insert into is self explanatory where I can enter information
regarding
   the table I've just created. All those values will be inserted into
candidates
   and employees tables under 'hr' schema respectively.
*/
```

```
-------------------------------------------------------------------------
---------

#24
[INNER JOIN]

SELECT
    c.id ,
    c.fullname ,
    e.id ,
    e.fullname
FROM
    hr.candidates c
    INNER JOIN hr.employees e
        ON e.fullname = c.fullname;


Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
joins/


/* Basically by selecting 'id' and 'fullname' from 'c' which is the
customer, and
   the same for employee, to joint them when the condition (fullname that
is in both
   employee and candidate) is met. This is to ensure that HR department
able to clean
   up candidate records if and when the candidate is hired as an employee
but would
   still have records as a candidate.
*/

-------------------------------------------------------------------------
---------

#25
[Header]

SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    INNER JOIN hr.employees e
        ON e.fullname = c.fullname;
```



	candidate_id	candidate_name	employee_id	employee_name
1	1	John Doe	1	John Doe
2	9	John Doe	1	John Doe
3	4	Jane Doe	2	Jane Doe
4	12	Jane Doe	2	Jane Doe
5	1	John Doe	9	John Doe
6	9	John Doe	9	John Doe
7	4	Jane Doe	10	Jane Doe
8	12	Jane Doe	10	Jane Doe

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-joins/

[210_IMG25]

```
/* The same as example #24, I can rename/replace the header with any name that
   needed to be changed. From #24, I have c.id and e.id where now, I can replace
   it with Candidate_id and Employee_id or other any name that make sense to the
   data. This little change is important as organization to do data manipulation
   where information constantly gets updated.
*/

-------------------------------------------------------------------------------

#26
[LEFT JOIN]

SELECT
 c.id candidate_id,
 c.fullname candidate_name,
 e.id employee_id,
 e.fullname employee_name
FROM
 hr.candidates c
 LEFT JOIN hr.employees e
 ON e.fullname = c.fullname;
```

Results | Messages

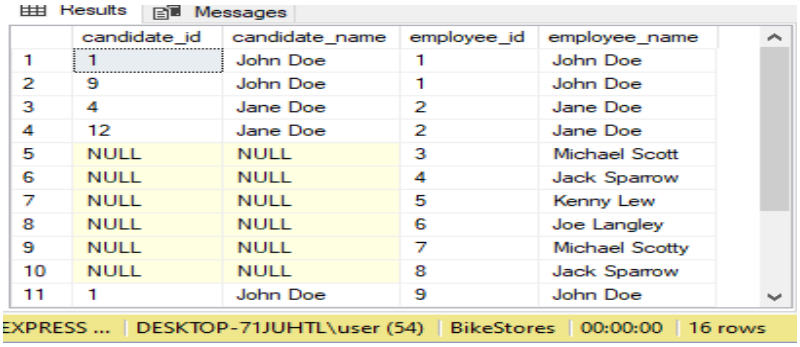
	candidate_id	candidate_name	employee_id	employee_name
1	1	John Doe	1	John Doe
2	1	John Doe	9	John Doe
3	2	Lily Bush	NULL	NULL
4	3	Peter Drucker	NULL	NULL
5	4	Jane Doe	2	Jane Doe
6	4	Jane Doe	10	Jane Doe
7	5	Kenneth Monr...	NULL	NULL
8	6	Lewis Wang	NULL	NULL
9	7	Peter Chap	NULL	NULL
10	8	Jane Lane	NULL	NULL
11	9	John Doe	1	John Doe

XPRESS ... | DESKTOP-71JUHTL\user (54) | BikeStores | 00:00:00 | 16 rows

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-joins/

[210_IMG26]

```
/* While example #25 uses inner join, where the data from tables with the
condition
   create a combined data as the information. However for #26, LEFT JOIN
is used
   to put more emphasis on 'hr.candidates' and the condition for this
LEFT JOIN is
   to find all candidate names that also might be in the 'hr.employees'
table. If not
   'null' is present in the employee section.
*/

-------------------------------------------------------------------------
---------

#27
[RIGHT JOIN]

SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    RIGHT JOIN hr.employees e
        ON e.fullname = c.fullname;
```



	candidate_id	candidate_name	employee_id	employee_name
1	1	John Doe	1	John Doe
2	9	John Doe	1	John Doe
3	4	Jane Doe	2	Jane Doe
4	12	Jane Doe	2	Jane Doe
5	NULL	NULL	3	Michael Scott
6	NULL	NULL	4	Jack Sparrow
7	NULL	NULL	5	Kenny Lew
8	NULL	NULL	6	Joe Langley
9	NULL	NULL	7	Michael Scotty
10	NULL	NULL	8	Jack Sparrow
11	1	John Doe	9	John Doe

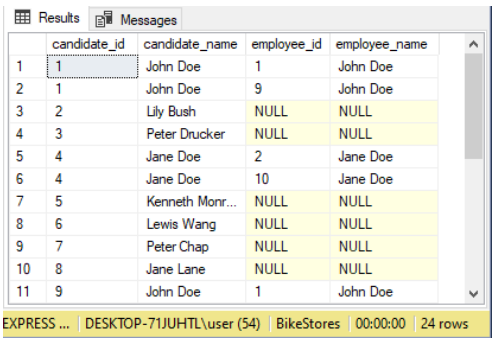
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-joins/

[210_IMG27]

```
/* Same as the example above where LEFT JOIN put more emphasis on
'hr.candidates',
   RIGHT JOIN emphasis more on 'hr.employees'. This is where all
information of the
   table from employees will appear and if employee is not a candidate,
then in
   'hr.candidate' will appear as NULL
*/
```

#28
[FULL JOIN]

```
SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    FULL JOIN hr.employees e
        ON e.fullname = c.fullname;
```



	candidate_id	candidate_name	employee_id	employee_name
1	1	John Doe	1	John Doe
2	1	John Doe	9	John Doe
3	2	Lily Bush	NULL	NULL
4	3	Peter Drucker	NULL	NULL
5	4	Jane Doe	2	Jane Doe
6	4	Jane Doe	10	Jane Doe
7	5	Kenneth Monr...	NULL	NULL
8	6	Lewis Wang	NULL	NULL
9	7	Peter Chap	NULL	NULL
10	8	Jane Lane	NULL	NULL
11	9	John Doe	1	John Doe

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-joins/

[210_IMG28]

```
/* FULL JOIN will combine both RIGHT and LEFT JOIN as well can see where
both
   employees and candidates have NULL information in their respective
column.
   This gives more visiblility to the overall data, however because it is
broad,
   depending on what information is requested, RIGHT and LEFT join will
narrow down
   the specific query.
*/

-------------------------------------------------------------------------
---------

#29
[LEFT JOIN | ORDER BY]

SELECT
    product_name,
    order_id
FROM
    production.products p
LEFT JOIN sales.order_items o ON o.product_id = p.product_id
ORDER BY
    order_id;
```

Results | Messages

	product_name	order_id
1	Electra Savannah 1 (20-inch) - Girl's - 2018	NULL
2	Electra Townie Go! 8i Ladies' - 2018	NULL
3	Trek Checkpoint ALR 5 Women's - 2019	NULL
4	Trek Checkpoint ALR Frameset - 2019	NULL
5	Trek Precaliber 12 Girl's - 2018	NULL
6	Surly Krampus Frameset - 2018	NULL
7	Trek Checkpoint SL 5 Women's - 2019	NULL
8	Trek 820 - 2016	NULL
9	Trek Checkpoint ALR 4 Women's - 2019	NULL
10	Trek Kids' Dual Sport - 2018	NULL
11	Trek Checkpoint ALR 5 - 2019	NULL

PRESS ... | DESKTOP-71JUHTL\user (54) | BikeStores | 00:00:00 | 4,736 rows

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-left-join/

[210_IMG29]

```
/* Using LEFT JOIN, provided that the two tables have link between the
table column,
   this shows that by querying the product and the order, adn order it by
'order_id',
   would help business to have visiblity which product was not being sold
or have any
   orders.
*/

-------------------------------------------------------------------------
---------

#30
[RIGHT JOIN | WHERE | ORDER BY]

SELECT
    product_name,
    order_id
FROM
    sales.order_items o
    RIGHT JOIN production.products p
        ON o.product_id = p.product_id
WHERE
    order_id IS NULL
ORDER BY
    product_name;
```

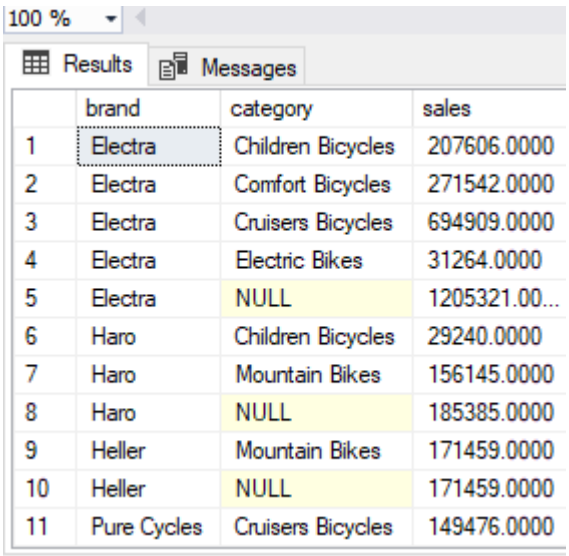
Results | Messages

	product_name	order_id
1	Electra Savannah 1 (20-inch) - Girl's - 2018	NULL
2	Electra Sweet Ride 1 (20-inch) - Girl's - 20...	NULL
3	Electra Townie Go! 8i Ladies' - 2018	NULL
4	Surly Krampus Frameset - 2018	NULL
5	Trek 820 - 2016	NULL
6	Trek Checkpoint ALR 4 Women's - 2019	NULL
7	Trek Checkpoint ALR 5 - 2019	NULL
8	Trek Checkpoint ALR 5 Women's - 2019	NULL
9	Trek Checkpoint ALR Frameset - 2019	NULL
10	Trek Checkpoint SL 5 Women's - 2019	NULL
11	Trek Checkpoint SL 6 - 2019	NULL

EXPRESS ... | DESKTOP-71JUHTL\user (54) | BikeStores | 00:00:00 | 14 rows

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
left-join/

[210_IMG30]

/* Alternatively, using RIGHT JOIN will give almost the same result as
#29.
   The focus now shift to the product in this RIGHT JOIN. Include 'WHERE'
will
```

```
   narrow down the search even more concise. That the focus is on the
product,
   the query result will only be order_id that is NULL.
*/

-------------------------------------------------------------------------
---------

#31
[INNER JOIN | ROLLUP setup]

SELECT
    b.brand_name AS brand,
    c.category_name AS category,
    p.model_year,
    round(
        SUM (
            quantity * i.list_price * (1 - discount)
        ),
        0
    ) sales INTO sales.sales_summary
FROM
    sales.order_items i
INNER JOIN production.products p ON p.product_id = i.product_id
INNER JOIN production.brands b ON b.brand_id = p.brand_id
INNER JOIN production.categories c ON c.category_id = p.category_id
GROUP BY
    b.brand_name,
    c.category_name,
    p.model_year
ORDER BY
    b.brand_name,
    c.category_name,
    p.model_year;
```



100 %

Results | Messages

	brand	category	sales
1	Electra	Children Bicycles	207606.0000
2	Electra	Comfort Bicycles	271542.0000
3	Electra	Cruisers Bicycles	694909.0000
4	Electra	Electric Bikes	31264.0000
5	Electra	NULL	1205321.00...
6	Haro	Children Bicycles	29240.0000
7	Haro	Mountain Bikes	156145.0000
8	Haro	NULL	185385.0000
9	Heller	Mountain Bikes	171459.0000
10	Heller	NULL	171459.0000
11	Pure Cycles	Cruisers Bicycles	149476.0000

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-rollup/

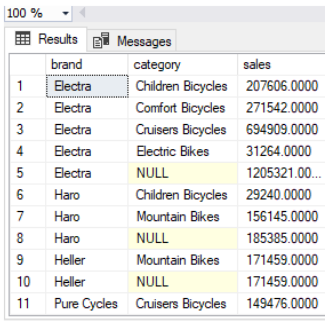
Output:
(41 rows affected)

Completion time: 2019-12-09T08:59:19.7200263-08:00

```
/* Using inner join, this query combined information from brand,
category, and
   model into respective names and then retrieve sales amount data by
those
   information into the sales.sales_summary table
*/
```

#32
[Grouping Set]

```
SELECT
    brand,
    category,
    SUM (sales) sales
FROM
    sales.sales_summary
GROUP BY
    brand,
    category
ORDER BY
    brand,
    category;
```



100 %

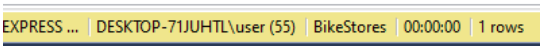
Results | Messages

	brand	category	sales
1	Electra	Children Bicycles	207606.0000
2	Electra	Comfort Bicycles	271542.0000
3	Electra	Cruisers Bicycles	694909.0000
4	Electra	Electric Bikes	31264.0000
5	Electra	NULL	1205321.00...
6	Haro	Children Bicycles	29240.0000
7	Haro	Mountain Bikes	156145.0000
8	Haro	NULL	185385.0000
9	Heller	Mountain Bikes	171459.0000
10	Heller	NULL	171459.0000
11	Pure Cycles	Cruisers Bicycles	149476.0000

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-grouping-sets/

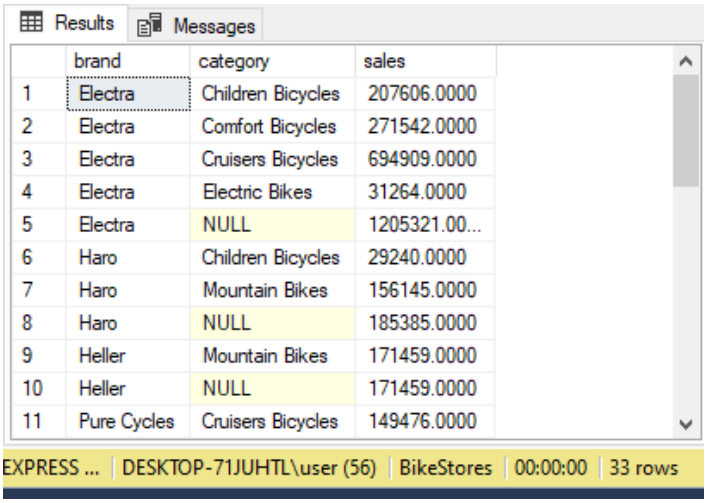
```
/* This query shows that we can calculate the sum of the sales by sorting
into
```

```
   brand and category to create a total amount. Respectively, we can also
sort
   by brand or just the category to get the total amount of sales
according
   to the data requested.
*/

-------------------------------------------------------------------------
---------

#33
[Grouping Set]

SELECT
    SUM (sales) sales
FROM
    sales.sales_summary;
```




```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
grouping-sets/

/* Another useful query would be that, the total sum of the sales can be
calculated
   by using the same concept but without sorting by brand nor category,
then it will
   show an output of the total sales from the sales summary. In the
database that I
   have used, which would be the bike store, have a total sales of
$7,689,113.00
*/

-------------------------------------------------------------------------
---------

#34
[ROLLUP]

SELECT
    brand,
    category,
```

```
    SUM (sales) sales
FROM
    sales.sales_summary
GROUP BY
    ROLLUP(brand, category);
```



Results | Messages

	brand	category	sales
1	Electra	Children Bicycles	207606.0000
2	Electra	Comfort Bicycles	271542.0000
3	Electra	Cruisers Bicycles	694909.0000
4	Electra	Electric Bikes	31264.0000
5	Electra	NULL	1205321.00...
6	Haro	Children Bicycles	29240.0000
7	Haro	Mountain Bikes	156145.0000
8	Haro	NULL	185385.0000
9	Heller	Mountain Bikes	171459.0000
10	Heller	NULL	171459.0000
11	Pure Cycles	Cruisers Bicycles	149476.0000

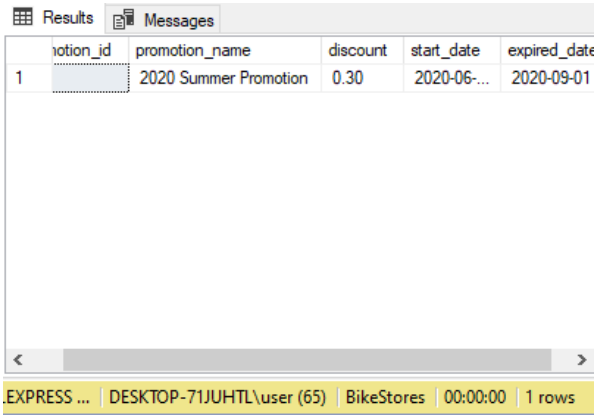
EXPRESS ... | DESKTOP-71JUHTL\user (56) | BikeStores | 00:00:00 | 33 rows

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-rollup/

```
/* ROLLUP usage is to calculate aggregation of value based upon
hierarchical order.
   In this query, it goes from brand > category, and where SUM (sales)
sales, will
   pull in information/data from brand and category and display the total
sales of
   those columns. The output is similar to the one we looked at in query
#32, as
   ROLLUP is using group set but can generate multiple group set of a
hierarchical
   input column.
*/

-------------------------------------------------------------------------
---------

#35
[PRE - INSERT setup]

CREATE TABLE sales.promotions (
    promotion_id INT PRIMARY KEY IDENTITY (1, 1),
    promotion_name VARCHAR (255) NOT NULL,
    discount NUMERIC (3, 2) DEFAULT 0,
    start_date DATE NOT NULL,
    expired_date DATE NOT NULL
);
```

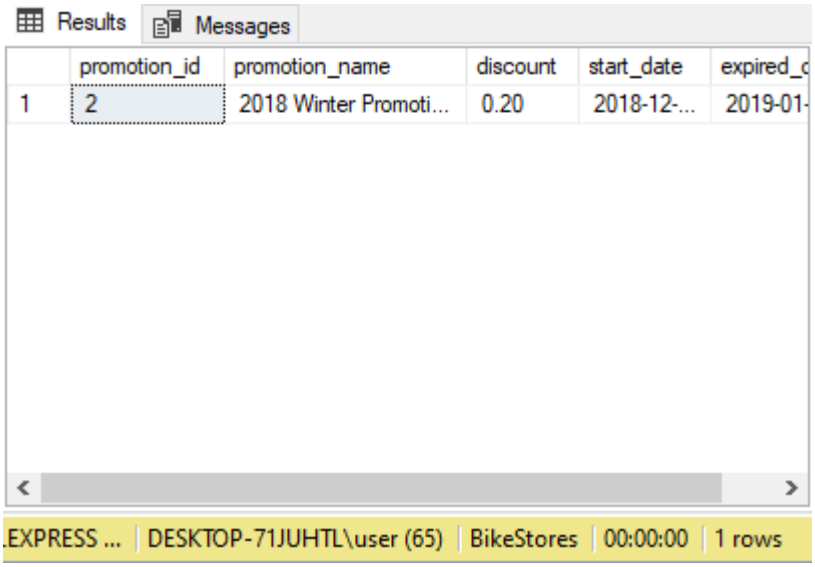
```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
insert/

Commands completed successfully.

Completion time: 2019-12-09T09:49:32.4190259-08:00


/* Before we can insert information or data into the table, we first need
to create
   a table with defining columns and what contains in that column.
*/

------------------------------------------------------------------------
---------

#36
[INSERT]

CREATE TABLE sales.promotions (
    promotion_id INT PRIMARY KEY IDENTITY (1, 1),
    promotion_name VARCHAR (255) NOT NULL,
    discount NUMERIC (3, 2) DEFAULT 0,
    start_date DATE NOT NULL,
    expired_date DATE NOT NULL
);


Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
insert/

Commands completed successfully.

Completion time: 2019-12-09T09:49:32.4190259-08:00


/* Before we can insert information or data into the table, we first need
to create
   a table with defining columns and what contains in that column.
*/

------------------------------------------------------------------------
---------


#37
[INSERT -- 1]

INSERT INTO sales.promotions (
    promotion_name,
    discount,
    start_date,
    expired_date
)
```

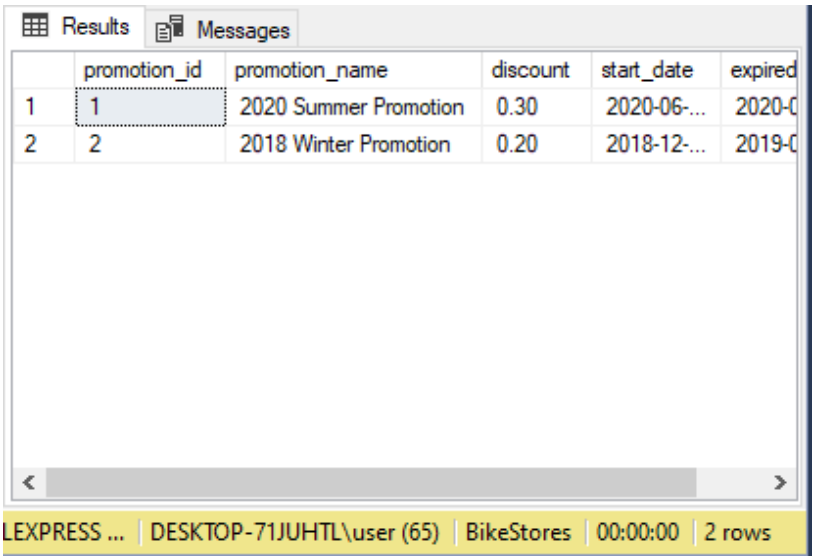
```
VALUES
    (
        '2020 Summer Promotion',
        0.30,
        '20200601',
        '20200901'
    );

-----------------------------------

SELECT
    *
FROM
    sales.promotions;

-----------------------------------
```




```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
insert/

(1 row affected)

Completion time: 2019-12-09T13:40:18.1565897-08:00

[210_IMG37]

/* After creation of the table, we can input data into the column that we
have
   created
*/

-------------------------------------------------------------------------
---------

#38
[INSERT / OUTPUT -- 2]

INSERT INTO sales.promotions (
    promotion_name,
    discount,
```

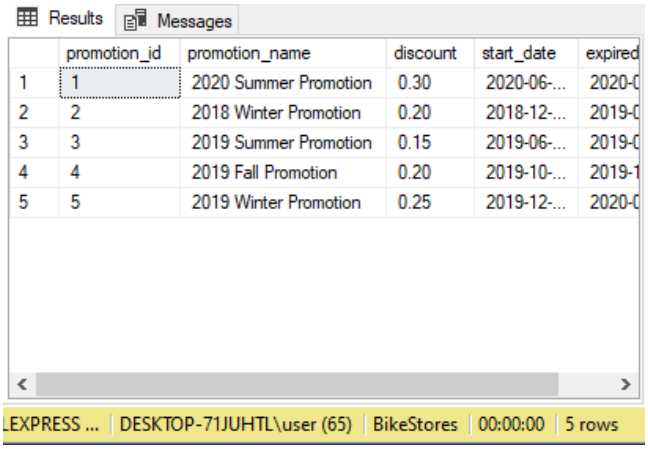
```
    start_date,
    expired_date
) OUTPUT inserted.promotion_id,
 inserted.promotion_name,
 inserted.discount,
 inserted.start_date,
 inserted.expired_date
VALUES
    (
        '2018 Winter Promotion',
        0.2,
        '20181201',
        '20190101'
    );
```





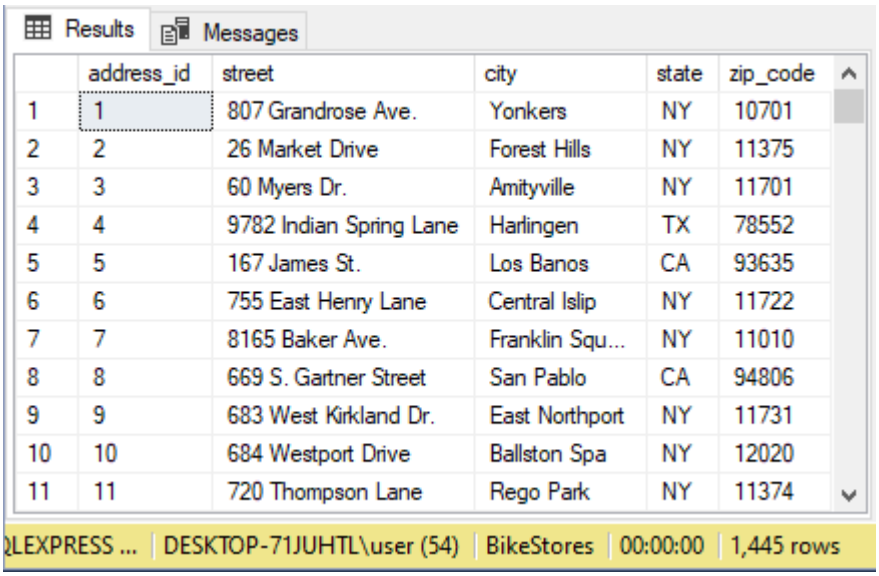
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-insert/

```
/* Using output clause, I can insert the specific values into the columns
in tables
   and show the values that was inserted into specific column using
OUTPUT clause.
*/

-------------------------------------------------------------------------
---------

#39
[INSERT -- 3]

INSERT INTO sales.promotions (
    promotion_name,
    discount,
    start_date,
    expired_date
)
VALUES
    (
        '2019 Summer Promotion',
        0.15,
        '20190601',
        '20190901'
    ),
    (
        '2019 Fall Promotion',
        0.20,
        '20191001',
        '20191101'
    ),
    (
        '2019 Winter Promotion',
        0.25,
        '20191201',
        '20200101'
    );
-----------------------------------

SELECT
    *
FROM
    sales.promotions;

-----------------------------------
```




```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
insert/

(3 rows affected)

Completion time: 2019-12-09T14:12:48.2077251-08:00

/* If multiple values are known, then I would be able to insert multiple
entries in
   one query. Then it gives more efficiency, if the values are known
ahead of time,
   without multiple data entry/queries.
*/

------------------------------------------------------------------------
---------

#40
[INSERT INTO SELECT]

CREATE TABLE sales.addresses (
    address_id INT IDENTITY PRIMARY KEY,
    street VARCHAR (255) NOT NULL,
    city VARCHAR (50),
    state VARCHAR (25),
    zip_code VARCHAR (5)
);
--------------------------------------------
INSERT INTO sales.addresses (street, city, state, zip_code)
SELECT
    street,
    city,
    state,
    zip_code
FROM
    sales.customers
ORDER BY
    first_name,
    last_name;
```

```
-----------------------------------------------
SELECT
    *
FROM
    sales.addresses;
-----------------------------------------------
```



Results | Messages

	address_id	street	city	state	zip_code
1	1	807 Grandrose Ave.	Yonkers	NY	10701
2	2	26 Market Drive	Forest Hills	NY	11375
3	3	60 Myers Dr.	Amityville	NY	11701
4	4	9782 Indian Spring Lane	Harlingen	TX	78552
5	5	167 James St.	Los Banos	CA	93635
6	6	755 East Henry Lane	Central Islip	NY	11722
7	7	8165 Baker Ave.	Franklin Squ...	NY	11010
8	8	669 S. Gartner Street	San Pablo	CA	94806
9	9	683 West Kirkland Dr.	East Northport	NY	11731
10	10	684 Westport Drive	Ballston Spa	NY	12020
11	11	720 Thompson Lane	Rego Park	NY	11374

QLEXPRESS ... | DESKTOP-71JUHTL\user (54) | BikeStores | 00:00:00 | 1,445 rows

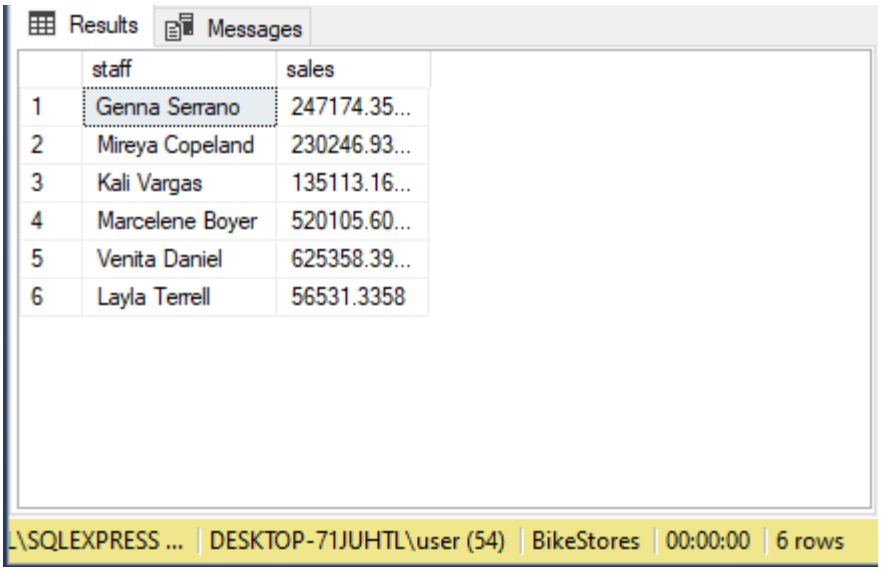
```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
insert-into-select/

/* To demostrate the INSERT INTO clause, I need to create a table named
sales.addresses.
   then, would "copy" the addresess from customer table into the newly
created
   addresses table by using 'INSERT INTO' clause. As a verification,
SELECT statement
   is used to display the output.
*/

-------------------------------------------------------------------------
---------

#41
[INSERT INTO | WHERE CLAUSE]

INSERT INTO
    sales.addresses (street, city, state, zip_code)
SELECT
    street,
    city,
    state,
    zip_code
FROM
    sales.stores
WHERE
```

```
    city IN ('Santa Cruz', 'Baldwin')

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
insert-into-select/

(2 rows affected)

Completion time: 2019-12-09T20:58:14.5918588-08:00


/* Similar to the query #40, using the INSERT INTO, #41 is essentially
the same
   query as #40 except that it also has the WHERE clause that will sort
out the
   condition from sales.stores before 'INSERT INTO' clause is used to
insert the
   information into the sales.addresses
*/

------------------------------------------------------------------------
---------

#42
[TRUNCATE]


TRUNCATE TABLE sales.addresses;


Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
insert-into-select/

/* Using this query, basically it would remove all rows from the
addresses table.
*/

------------------------------------------------------------------------
---------

#43
[INSERT TOP]

INSERT TOP (50) PERCENT
INTO sales.addresses (street, city, state, zip_code)
SELECT
    street,
    city,
    state,
    zip_code
FROM
    sales.customers
ORDER BY
    first_name,
    last_name;
```

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
left-join/

(723 rows affected)

Completion time: 2019-12-09T21:32:35.2282981-08:00

/* Beside having INSERT INTO, additionally, I may just want to enter the
first 50
   percent or whatever that number might be into the new table. Along
with that,
   the data will be sorted out with ORDER BY condition. In this example
query, it
   has the 50 percent of the total data, which the total row is 1445 and
half of that
   would the 723 rows. This provides more visibility and narrowing down
the search
   according to the business requirement.
*/

--------------------------------------------------------------------------
---------

#44
[UPDATE]

CREATE TABLE sales.taxes (
 tax_id INT PRIMARY KEY IDENTITY (1, 1),
 state VARCHAR (50) NOT NULL UNIQUE,
 state_tax_rate DEC (3, 2),
 avg_local_tax_rate DEC (3, 2),
 combined_rate AS state_tax_rate + avg_local_tax_rate,
 max_local_tax_rate DEC (3, 2),
 updated_at datetime
);

-----------------------------------------

INSERT INTO
sales.taxes(state,state_tax_rate,avg_local_tax_rate,max_local_tax_rate)
VALUES('Alabama',0.04,0.05,0.07);
INSERT INTO
sales.taxes(state,state_tax_rate,avg_local_tax_rate,max_local_tax_rate)
VALUES('Alaska',0,0.01,0.07);
INSERT INTO
sales.taxes(state,state_tax_rate,avg_local_tax_rate,max_local_tax_rate)
VALUES('Arizona',0.05,0.02,0.05);
INSERT INTO
sales.taxes(state,state_tax_rate,avg_local_tax_rate,max_local_tax_rate)
VALUES('Arkansas',0.06,0.02,0.05);
```

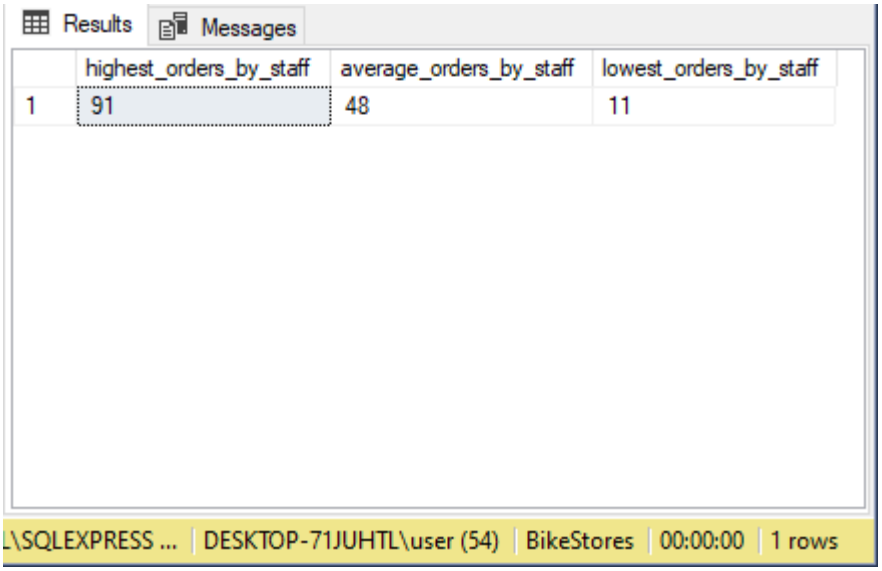
```
INSERT INTO
sales.taxes(state,state_tax_rate,avg_local_tax_rate,max_local_tax_rate)
VALUES('California',0.07,0.01,0.02);

-----------------------------------------

UPDATE sales.taxes
SET updated_at = GETDATE();

-----------------------------------------

UPDATE sales.taxes
SET max_local_tax_rate += 0.02,
    avg_local_tax_rate += 0.01
WHERE
    max_local_tax_rate = 0.01;

-----------------------------------------

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
insert/

(5 row affected)

Completion time: 2019-12-09T13:40:18.1565897-08:00


/* First of all, a table is created with the values. Then using INSERT
INTO,
   more information is then inserted into the taxes table. If updates
needs to be
   done, then UPDATE clause is utilized along with SET. This query shows
that,
   updating same information across rows in table can be done all
simultaneously
*/

-------------------------------------------------------------------------
---------

#45
[MERGE]

MERGE sales.category t
    USING sales.category_staging s
ON (s.category_id = t.category_id)
WHEN MATCHED
    THEN UPDATE SET
        t.category_name = s.category_name,
        t.amount = s.amount
WHEN NOT MATCHED BY TARGET
    THEN INSERT (category_id, category_name, amount)
         VALUES (s.category_id, s.category_name, s.amount)
WHEN NOT MATCHED BY SOURCE
```

```
    THEN DELETE;


Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
merge/

(6 rows affected)

Completion time: 2019-12-09T22:05:21.9155920-08:00

/* This query is merging sales.category (target table) with
sales.category_staging
   (source table) Those conditions are powerful in terms of the
condition, where
   it acts like a programming type (if..else) where if the category_id in
't' and 's'
   matches, then update the set and it not matched in target table then
INSERT while,
   it is not matched with the source, then data row will be deleted. Just
like GITHUB,
   this query works like a source control, ensuring all data is updated
with just
   a simple query.
*/

------------------------------------------------------------------------
---------

#46
[Common Table Expression]

WITH cte_sales_amounts (staff, sales, year) AS (
    SELECT
        first_name + ' ' + last_name,
        SUM(quantity * list_price * (1 - discount)),
        YEAR(order_date)
    FROM
        sales.orders o
    INNER JOIN sales.order_items i ON i.order_id = o.order_id
    INNER JOIN sales.staffs s ON s.staff_id = o.staff_id
    GROUP BY
        first_name + ' ' + last_name,
        year(order_date)
)

SELECT
    staff,
    sales
FROM
    cte_sales_amounts
WHERE
    year = 2018;
```



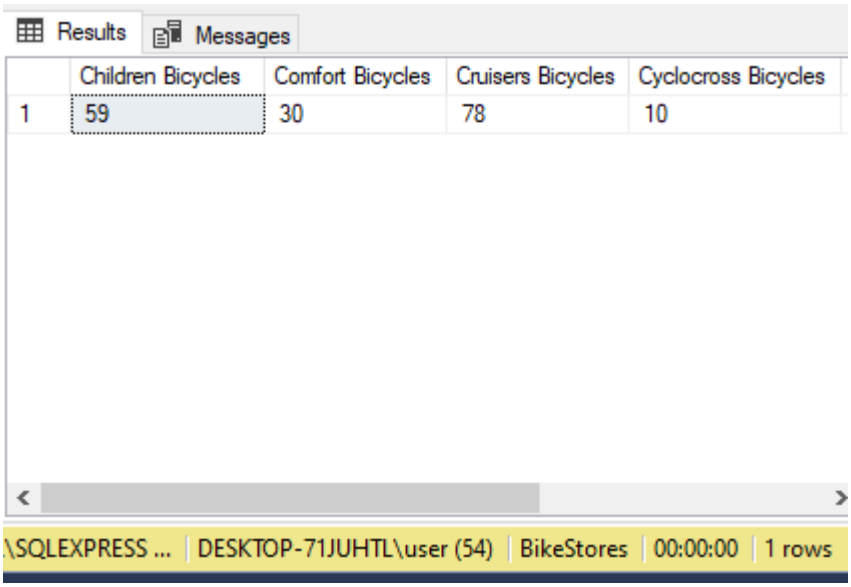

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
cte/

/* CTE means common table expression. Basically it's a temporary named
result set.
   With this query, the CTE is the cte_sales_amount where the information
is derived
   from first name, last name, sum of sales, and year.

*/

---------------------------------------------------------------------------
---------

#47
[Common Table Expression]

WITH cte_sales AS (
    SELECT
        staff_id,
        COUNT(*) order_count
    FROM
        sales.orders
    WHERE
        YEAR(order_date) = 2018
    GROUP BY
        staff_id

)
SELECT
        max(order_count) highest_orders_by_staff,
      avg(order_count) average_orders_by_staff,
      min(order_count) lowest_orders_by_staff
FROM
```

```
    cte_sales;
```




```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
cte/

/* This query allows me to find out which staff/employee have the highest
orders,
   on average orders, and the most minimum orders.

*/

---------------------------------------------------------------------------
---------

#48
[PIVOT]

SELECT * FROM
(
    SELECT
        category_name,
        product_id
    FROM
        production.products p
        INNER JOIN production.categories c
            ON c.category_id = p.category_id
) t
PIVOT(
    COUNT(product_id)
    FOR category_name IN (
        [Children Bicycles],
        [Comfort Bicycles],
        [Cruisers Bicycles],
        [Cyclocross Bicycles],
        [Electric Bikes],
```

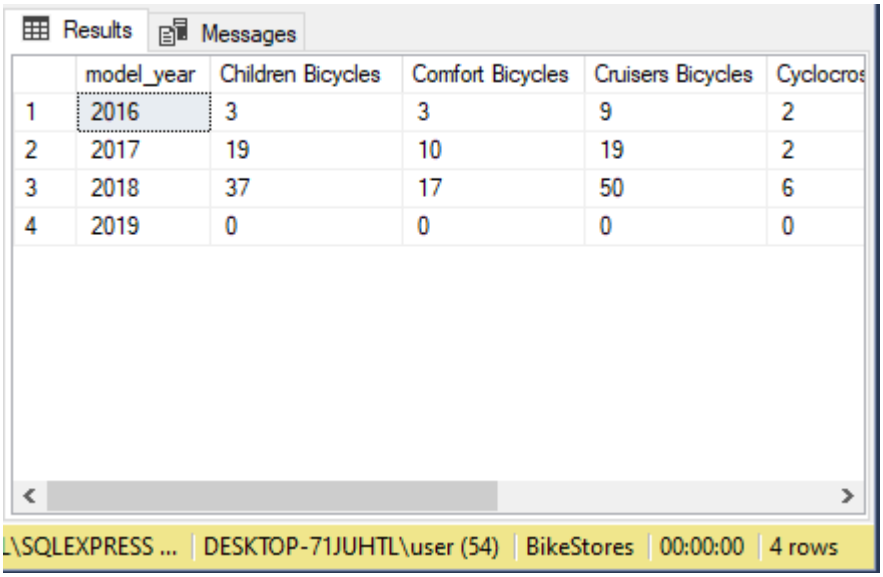
```
        [Mountain Bikes],
        [Road Bikes])
) AS pivot_table;
```



Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-pivot/

```
/* This PIVOT statement enables the query to turn values in one column
into multiple
   column depending on the condition defined.
*/

-------------------------------------------------------------------------
---------

#49
[PIVOT]

SELECT * FROM
(
    SELECT
        category_name,
        product_id,
        model_year
    FROM
        production.products p
        INNER JOIN production.categories c
            ON c.category_id = p.category_id
) t
PIVOT(
    COUNT(product_id)
    FOR category_name IN (
        [Children Bicycles],
        [Comfort Bicycles],
        [Cruisers Bicycles],
```

```
        [Cyclocross Bicycles],
        [Electric Bikes],
        [Mountain Bikes],
        [Road Bikes])
) AS pivot_table;
```



Results | Messages

	model_year	Children Bicycles	Comfort Bicycles	Cruisers Bicycles	Cyclocros
1	2016	3	3	9	2
2	2017	19	10	19	2
3	2018	37	17	50	6
4	2019	0	0	0	0

\SQLEXPRESS ... | DESKTOP-71JUHTL\user (54) | BikeStores | 00:00:00 | 4 rows

```
Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
pivot/

/* Same as before, PIVOT is used however this time, the query would
return the output
   based upon the model_year.
*/

-------------------------------------------------------------------------
---------

#50
[ALTER TABLE ADD]

CREATE TABLE sales.quotations (
    quotation_no INT IDENTITY PRIMARY KEY,
    valid_from DATE NOT NULL,
    valid_to DATE NOT NULL
);
-----------------------------------------

ALTER TABLE sales.quotations
ADD description VARCHAR (255) NOT NULL;

-----------------------------------------

ALTER TABLE sales.quotations
    ADD
        amount DECIMAL (10, 2) NOT NULL,
```

```
        customer_name VARCHAR (50) NOT NULL;

Source: http://www.sqlservertutorial.net/sql-server-basics/sql-server-
alter-table-add-column/

/* In this query, when new information arises and just needed to be added
into
   the existing table, 'ALTER TABLE' can be use to create additional rows
to the
   table without changing or deleting the table at all.
*/

-------------------------------------------------------------------------
---------
```