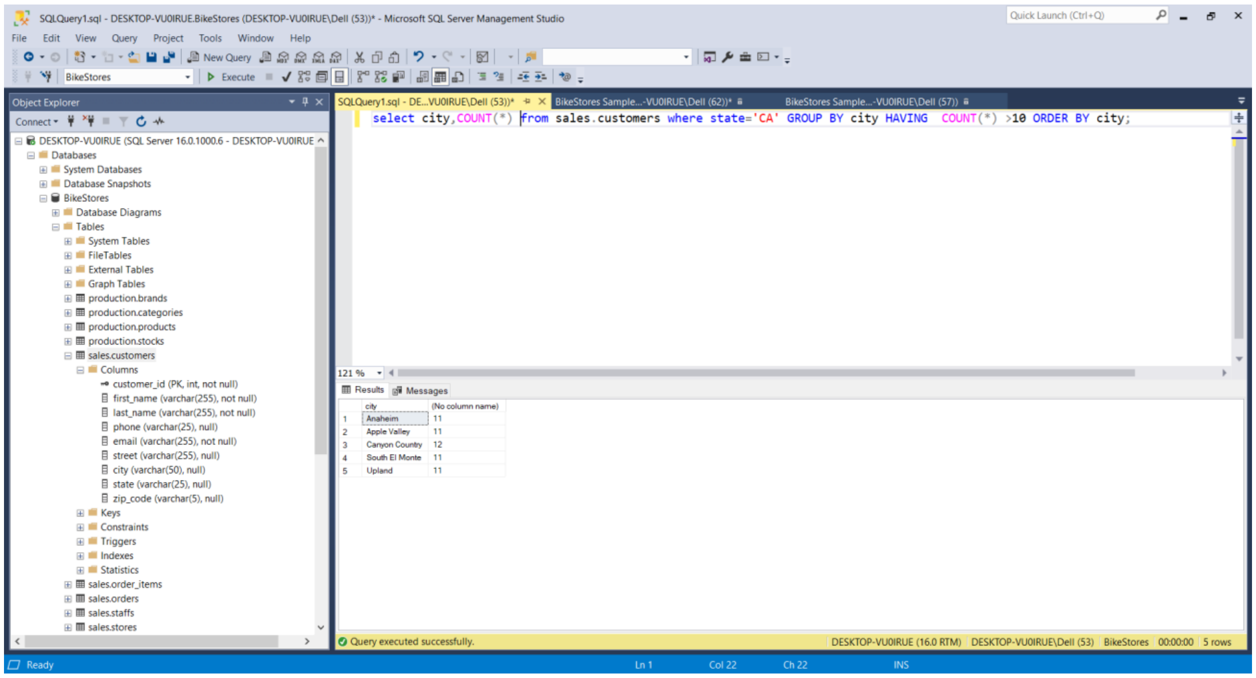
TASK ABOUT DATABASE ON DATA MANIUPILATION QUERIES

FIRST STEP is loading sample database, then practice the data manipulation queries

SELECT statement with **WHARE** clause to filter rows by one condition, to sort result based on one or more column use **OREDER BY**, to group rows into groups use GROUP BY, and to filter Groups based one or more conditions use HAVING clause

The **WHERE** clause filters rows while the **HAVING** clause filter groups.

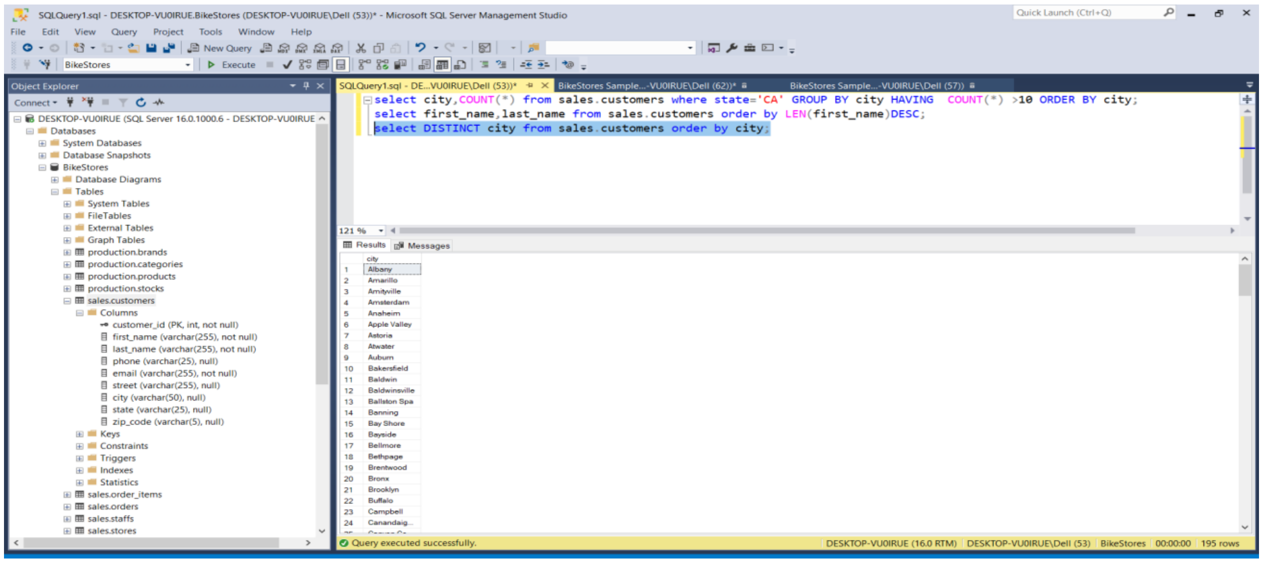


```
select city,COUNT(*) from sales.customers where state='CA' GROUP BY city HAVING  COUNT(*) >10 ORDER BY city;
```

	city	(No column name)
1	Anaheim	11
2	Apple Valley	11
3	Canyon Country	12
4	South El Monte	11
5	Upland	11

Query executed successfully.

Using **DISTINCT** keyword ,it removed all duplicate cities from the result set.

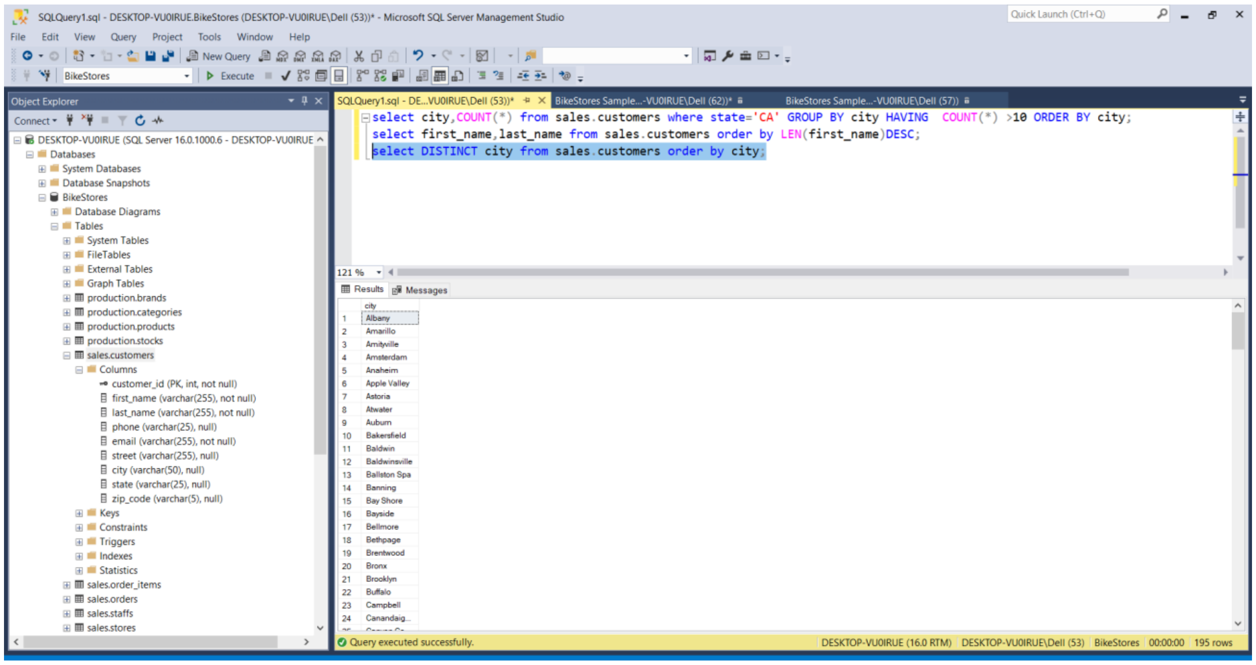


```
select city,COUNT(*) from sales.customers where state='CA' GROUP BY city HAVING  COUNT(*) >10 ORDER BY city;
select first_name,last_name from sales.customers order by LEN(first_name)DESC;
select DISTINCT city from sales.customers order by city;
```

	city
1	Albany
2	Amarillo
3	Amityville
4	Amsterdam
5	Anaheim
6	Apple Valley
7	Astoria
8	Atwater
9	Auburn
10	Bakersfield
11	Baldwin
12	Baldwinsville
13	Ballston Spa
14	Banning
15	Bay Shore
16	Bayside
17	Bellmore
18	Bethpage
19	Brentwood
20	Bronx
21	Brooklyn
22	Buffalo
23	Campbell
24	Canandaig...

Query executed successfully. 195 rows

uses the **GROUP BY** clause to return **distinct** cities together with state and zip code from the sales. Customers



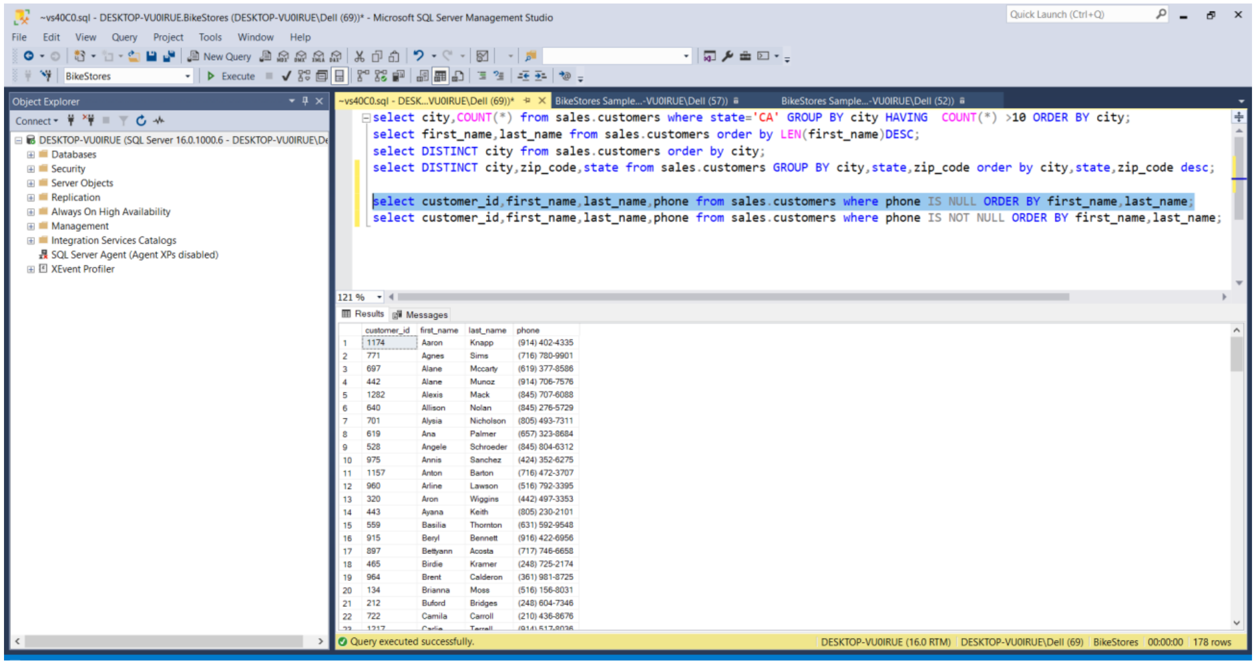
```sql
select city,COUNT(*) from sales.customers where state='CA' GROUP BY city HAVING  COUNT(*) >10 ORDER BY city;
select first_name,last_name from sales.customers order by LEN(first_name)DESC;
select DISTINCT city from sales.customers order by city;
```

	city
1	Albany
2	Amarillo
3	Amityville
4	Amsterdam
5	Anaheim
6	Apple Valley
7	Astoria
8	Atwater
9	Auburn
10	Bakersfield
11	Baldwin
12	Baldwinsville
13	Ballston Spa
14	Banning
15	Bay Shore
16	Bayside
17	Bellmore
18	Bethpage
19	Brentwood
20	Bronx
21	Brooklyn
22	Buffalo
23	Campbell
24	Canandaig...

```
Query executed successfully.    DESKTOP-VU0IRUE (16.0 RTM) | DESKTOP-VU0IRUE\Dell (53) | BikeStores | 00:00:00 | 195 rows
```

NULL

To test whether a value is NULL or not, you always use the IS NULL operator.IS NOT NULL to test the value is not null



```sql
select city,COUNT(*) from sales.customers where state='CA' GROUP BY city HAVING  COUNT(*) >10 ORDER BY city;
select first_name,last_name from sales.customers order by LEN(first_name)DESC;
select DISTINCT city from sales.customers order by city;
select DISTINCT city,zip_code,state from sales.customers GROUP BY city,state,zip_code order by city,state,zip_code desc;

select customer_id,first_name,last_name,phone from sales.customers where phone IS NULL ORDER BY first_name,last_name;
select customer_id,first_name,last_name,phone from sales.customers where phone IS NOT NULL ORDER BY first_name,last_name;
```

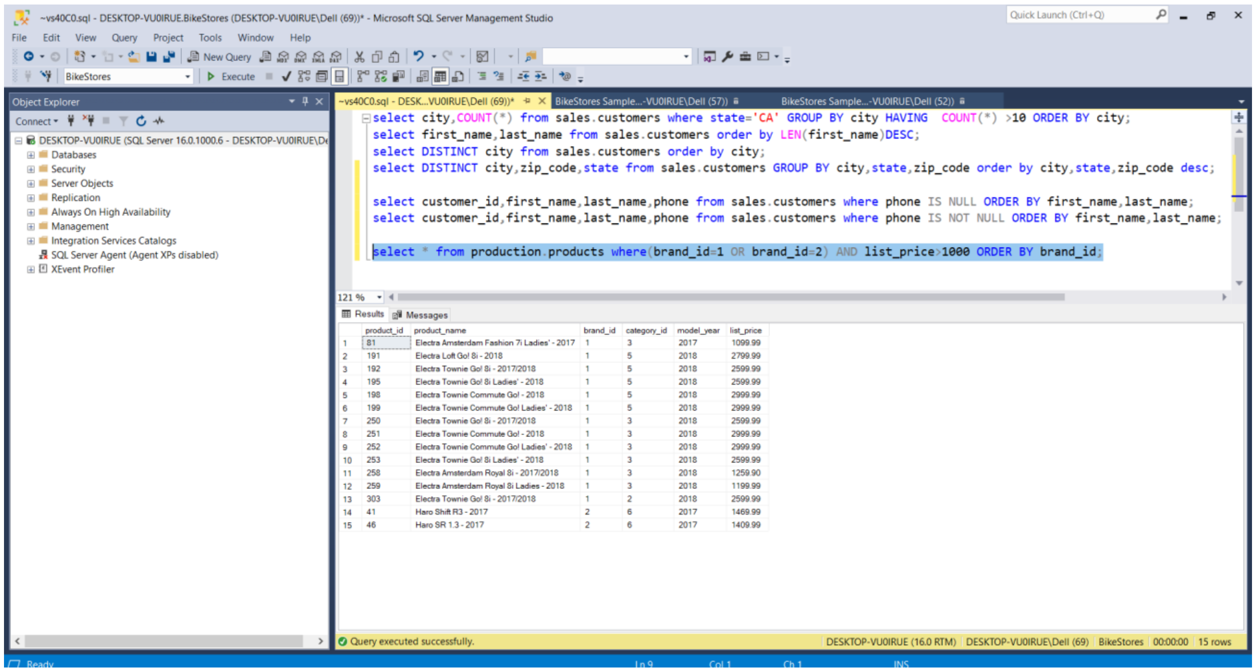
	customer_id	first_name	last_name	phone
1	1174	Aaron	Knapp	(914) 402-4335
2	771	Agnes	Sims	(716) 780-9901
3	697	Alane	Mccarty	(619) 377-8586
4	442	Alane	Munoz	(914) 706-7576
5	1282	Alexis	Mack	(845) 707-6088
6	640	Allison	Nolan	(845) 276-5729
7	701	Alysia	Nicholson	(805) 493-7311
8	619	Ana	Palmer	(657) 323-8684
9	528	Angele	Schroeder	(845) 804-6312
10	975	Annis	Sanchez	(424) 352-6275
11	1157	Anton	Barton	(716) 472-3707
12	960	Arline	Lawson	(516) 792-3395
13	320	Aron	Wiggins	(442) 497-3353
14	443	Ayana	Keith	(805) 230-2101
15	559	Basilia	Thornton	(631) 592-9548
16	915	Beryl	Bennett	(916) 422-6956
17	897	Bettyann	Acosta	(717) 746-6658
18	465	Birdie	Kramer	(248) 725-2174
19	964	Brent	Calderon	(361) 981-8725
20	134	Brianna	Moss	(516) 156-8031
21	212	Buford	Bridges	(248) 604-7346
22	722	Camila	Carroll	(210) 436-8676
23	1217	Carlie	Terrell	(914) 517-8036

```
Query executed successfully.    DESKTOP-VU0IRUE (16.0 RTM) | DESKTOP-VU0IRUE\Dell (69) | BikeStores | 00:00:00 | 178 rows
```

WHERE:

we used both **OR** and **AND** operators in the condition

To get the product whose brand id is one **or** two **and** list price is larger than 1,000

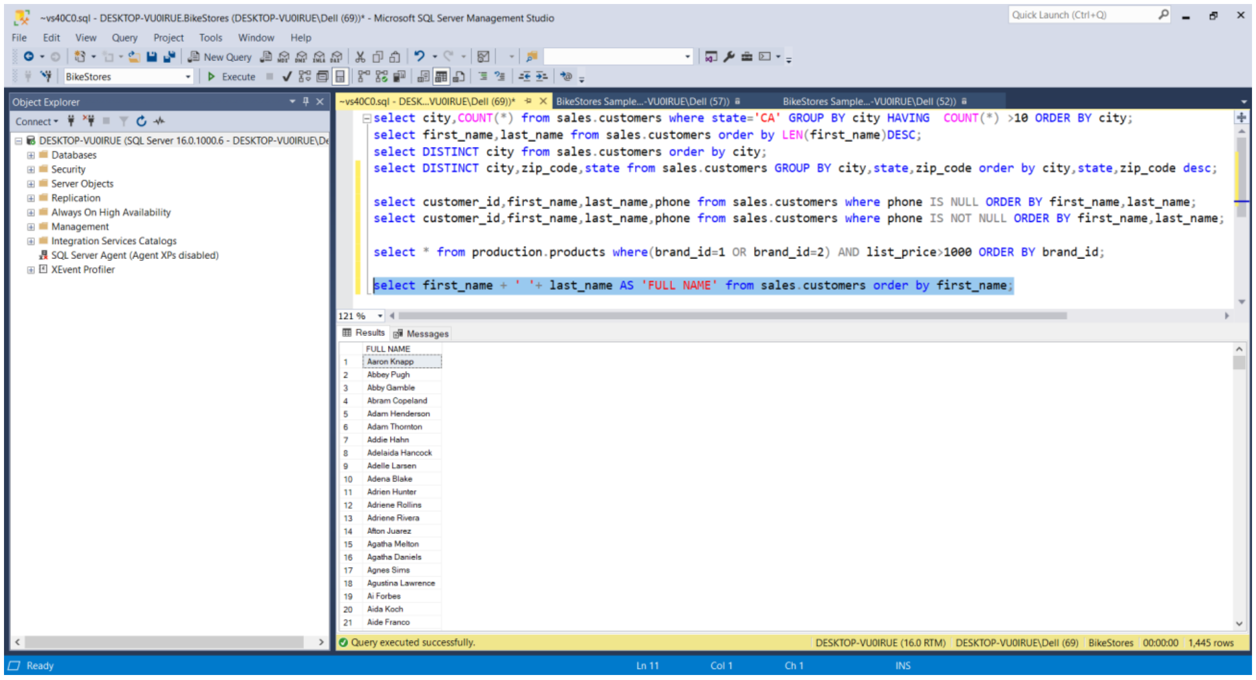


```sql
select city,COUNT(*) from sales.customers where state='CA' GROUP BY city HAVING  COUNT(*) >10 ORDER BY city;
select first_name,last_name from sales.customers order by LEN(first_name)DESC;
select DISTINCT city from sales.customers order by city;
select DISTINCT city,zip_code,state from sales.customers GROUP BY city,state,zip_code order by city,state,zip_code desc;

select customer_id,first_name,last_name,phone from sales.customers where phone IS NULL ORDER BY first_name,last_name;
select customer_id,first_name,last_name,phone from sales.customers where phone IS NOT NULL ORDER BY first_name,last_name;

select * from production.products where(brand_id=1 OR brand_id=2) AND list_price>1000 ORDER BY brand_id;
```

	product_id	product_name	brand_id	category_id	model_year	list_price
1	81	Electra Amsterdam Fashion 7i Ladies' - 2017	1	3	2017	1099.99
2	191	Electra Loft Go! 8i - 2018	1	5	2018	2799.99
3	192	Electra Townie Go! 8i - 2017/2018	1	5	2018	2599.99
4	195	Electra Townie Go! 8i Ladies' - 2018	1	5	2018	2599.99
5	198	Electra Townie Commute Go! - 2018	1	5	2018	2999.99
6	199	Electra Townie Commute Go! Ladies' - 2018	1	5	2018	2999.99
7	250	Electra Townie Go! 8i - 2017/2018	1	3	2018	2599.99
8	251	Electra Townie Commute Go! - 2018	1	3	2018	2999.99
9	252	Electra Townie Commute Go! Ladies' - 2018	1	3	2018	2999.99
10	253	Electra Townie Go! 8i Ladies' - 2018	1	3	2018	2599.99
11	258	Electra Amsterdam Royal 8i - 2017/2018	1	3	2018	1259.90
12	259	Electra Amsterdam Royal 8i Ladies - 2018	1	3	2018	1199.99
13	303	Electra Townie Go! 8i - 2017/2018	1	2	2018	2599.99
14	41	Haro Shift R3 - 2017	2	6	2017	1469.99
15	46	Haro SR 1.3 - 2017	2	6	2017	1409.99

```
Query executed successfully.    DESKTOP-VU0IRUE (16.0 RTM)  DESKTOP-VU0IRUE\Dell (69)  BikeStores  00:00:00  15 rows
```

column alias:



```sql
select city,COUNT(*) from sales.customers where state='CA' GROUP BY city HAVING  COUNT(*) >10 ORDER BY city;
select first_name,last_name from sales.customers order by LEN(first_name)DESC;
select DISTINCT city from sales.customers order by city;
select DISTINCT city,zip_code,state from sales.customers GROUP BY city,state,zip_code order by city,state,zip_code desc;

select customer_id,first_name,last_name,phone from sales.customers where phone IS NULL ORDER BY first_name,last_name;
select customer_id,first_name,last_name,phone from sales.customers where phone IS NOT NULL ORDER BY first_name,last_name;

select * from production.products where(brand_id=1 OR brand_id=2) AND list_price>1000 ORDER BY brand_id;

select first_name + ' '+ last_name AS 'FULL NAME' from sales.customers order by first_name;
```

	FULL NAME
1	Aaron Knapp
2	Abbey Pugh
3	Abby Gamble
4	Abram Copeland
5	Adam Henderson
6	Adam Thornton
7	Addie Hahn
8	Adelaida Hancock
9	Adelle Larsen
10	Adena Blake
11	Adrien Hunter
12	Adriene Rollins
13	Adriene Rivera
14	Afton Juarez
15	Agatha Melton
16	Agatha Daniels
17	Agnes Sims
18	Agustina Lawrence
19	Ai Forbes
20	Aida Koch
21	Aide Franco

```
Query executed successfully.    DESKTOP-VU0IRUE (16.0 RTM)  DESKTOP-VU0IRUE\Dell (69)  BikeStores  00:00:00  1,445 rows
```

Joins

Inner join produces a data set that includes rows from the left table, matching rows from the right table**.**

Left join selects data starting from the left table and matching rows in the right table. The left join returns all rows from the left table and the matching rows from the right table. If a row in the left table does not have a matching row in the right table, the columns of the right table will have **nulls**.

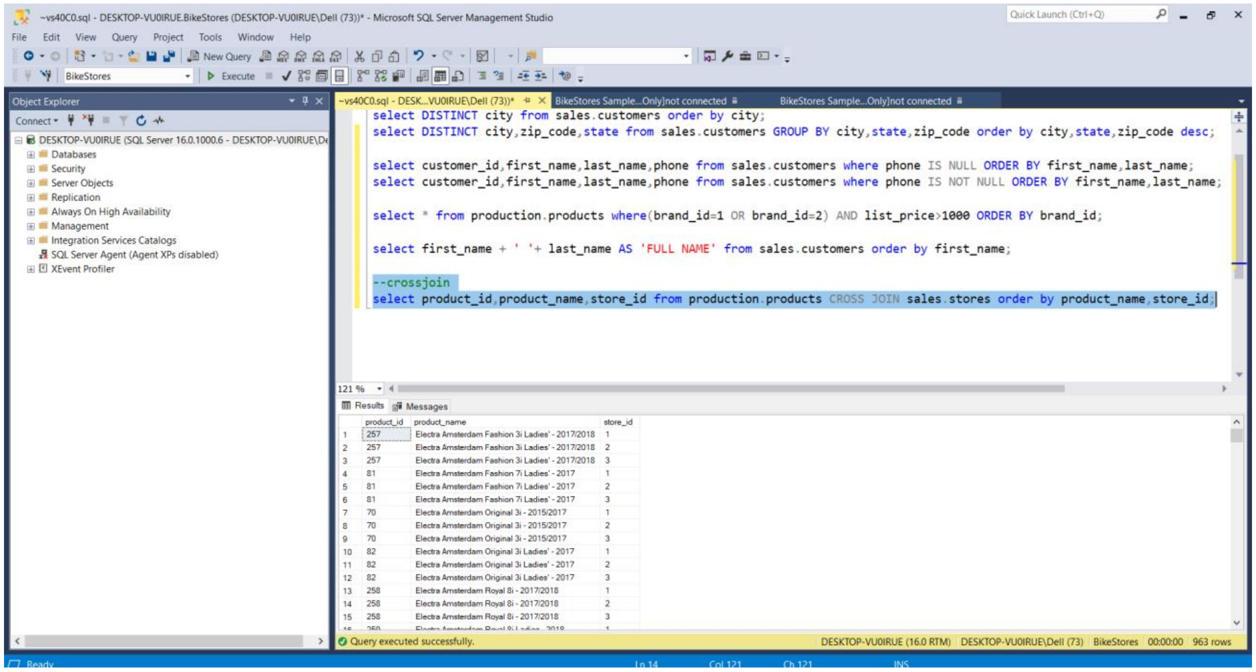
The right join returns a result set that contains all rows from the right table and the matching rows in the left table. If a row in the right table does not have a matching row in the left table, all columns in the left table will contain **nulls**.

The **full outer join or full join** returns a result set that contains all rows from both left and right tables, with the matching rows from both sides where available. In case there is no match, the missing side will have **NULL** values.

A self join used to join a table to itself. It helps query hierarchical data or compare rows within the same table.

A self join uses the inner join or left join clause. Because the query that uses **the self join** references the same table, the table alias is used to assign different names to the same table within the query.

CROSS JOIN to join two or more unrelated tables.

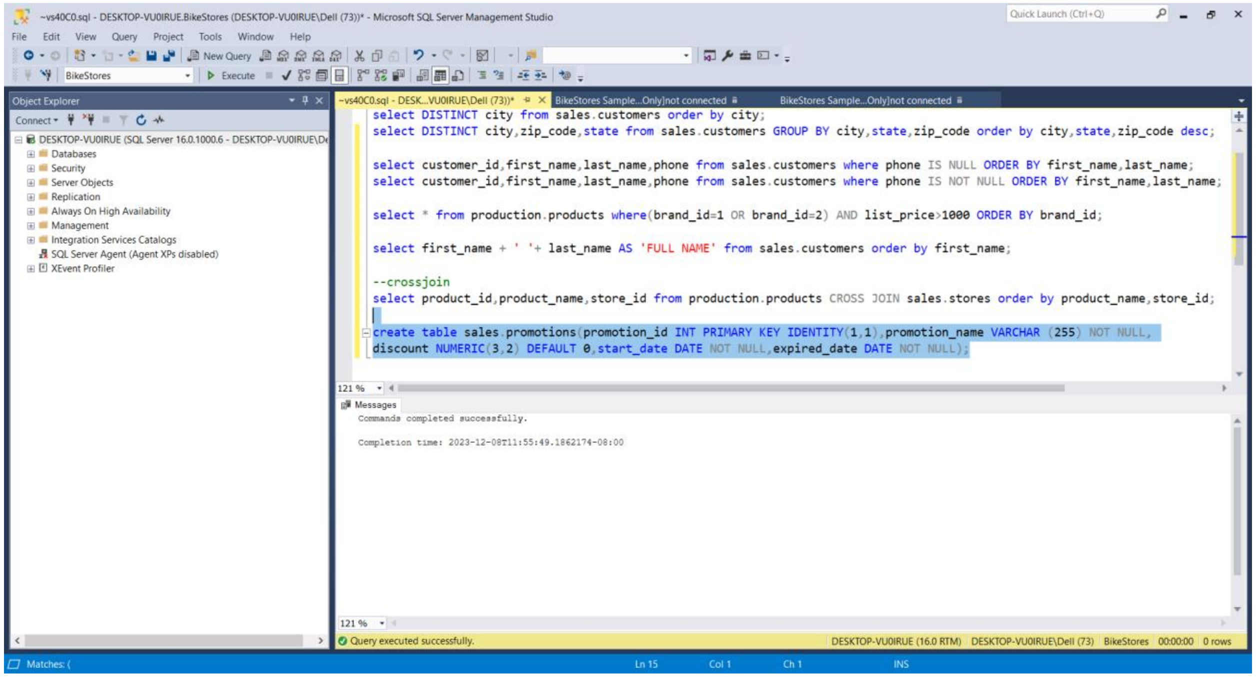


```sql
select DISTINCT city from sales.customers order by city;
select DISTINCT city,zip_code,state from sales.customers GROUP BY city,state,zip_code order by city,state,zip_code desc;

select customer_id,first_name,last_name,phone from sales.customers where phone IS NULL ORDER BY first_name,last_name;
select customer_id,first_name,last_name,phone from sales.customers where phone IS NOT NULL ORDER BY first_name,last_name;

select * from production.products where(brand_id=1 OR brand_id=2) AND list_price>1000 ORDER BY brand_id;

select first_name + ' '+ last_name AS 'FULL NAME' from sales.customers order by first_name;

--crossjoin
select product_id,product_name,store_id from production.products CROSS JOIN sales.stores order by product_name,store_id;
```

```
    product_id  product_name                                      store_id
1   257         Electra Amsterdam Fashion 3i Ladies' - 2017/2018  1
2   257         Electra Amsterdam Fashion 3i Ladies' - 2017/2018  2
3   257         Electra Amsterdam Fashion 3i Ladies' - 2017/2018  3
4   81          Electra Amsterdam Fashion 7i Ladies' - 2017       1
5   81          Electra Amsterdam Fashion 7i Ladies' - 2017       2
6   81          Electra Amsterdam Fashion 7i Ladies' - 2017       3
7   70          Electra Amsterdam Original 3i - 2015/2017         1
8   70          Electra Amsterdam Original 3i - 2015/2017         2
9   70          Electra Amsterdam Original 3i - 2015/2017         3
10  82          Electra Amsterdam Original 3i Ladies' - 2017      1
11  82          Electra Amsterdam Original 3i Ladies' - 2017      2
12  82          Electra Amsterdam Original 3i Ladies' - 2017      3
13  258         Electra Amsterdam Royal 8i - 2017/2018            1
14  258         Electra Amsterdam Royal 8i - 2017/2018            2
15  258         Electra Amsterdam Royal 8i - 2017/2018            3
```

Query executed successfully. DESKTOP-VU0IRUE (16.0 RTM) DESKTOP-VU0IRUE\Dell (73) BikeStores 00:00:00 963 rows

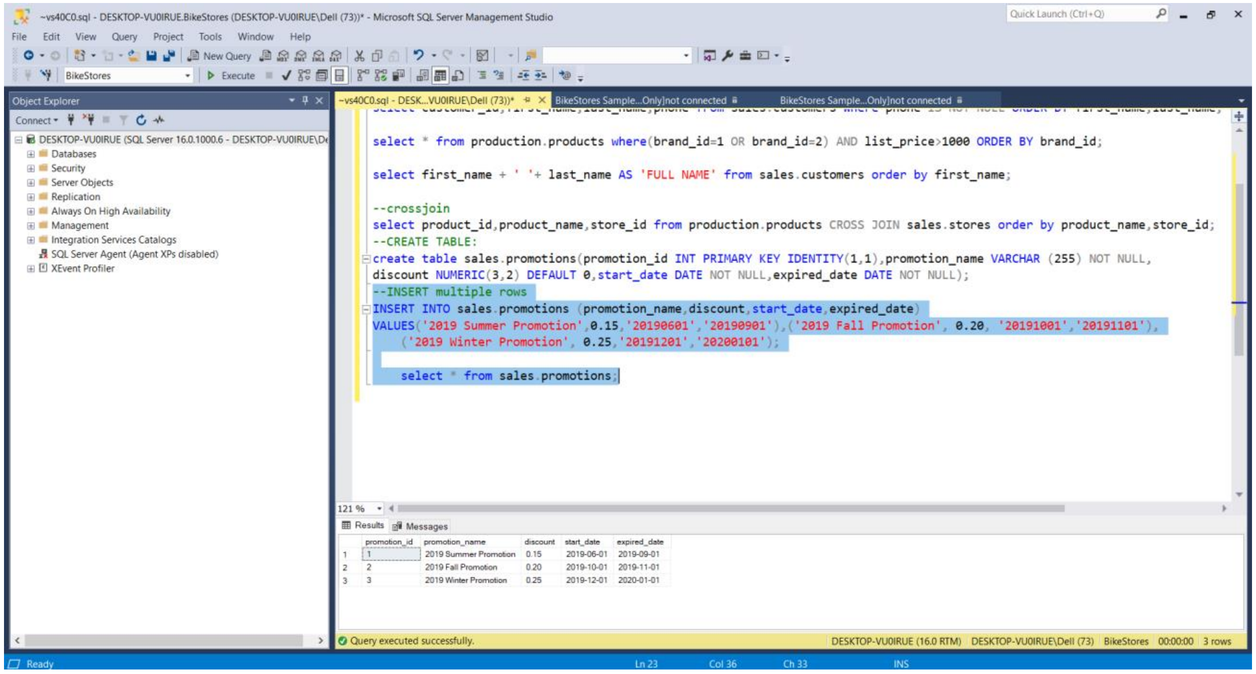
CREAT TABLE



```sql
select DISTINCT city from sales.customers order by city;
select DISTINCT city,zip_code,state from sales.customers GROUP BY city,state,zip_code order by city,state,zip_code desc;

select customer_id,first_name,last_name,phone from sales.customers where phone IS NULL ORDER BY first_name,last_name;
select customer_id,first_name,last_name,phone from sales.customers where phone IS NOT NULL ORDER BY first_name,last_name;

select * from production.products where(brand_id=1 OR brand_id=2) AND list_price>1000 ORDER BY brand_id;

select first_name + ' '+ last_name AS 'FULL NAME' from sales.customers order by first_name;

--crossjoin
select product_id,product_name,store_id from production.products CROSS JOIN sales.stores order by product_name,store_id;

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);
```

```
Commands completed successfully.

Completion time: 2023-12-08T11:55:49.1862174-08:00
```

INSERT MULTIPLE ROWS:

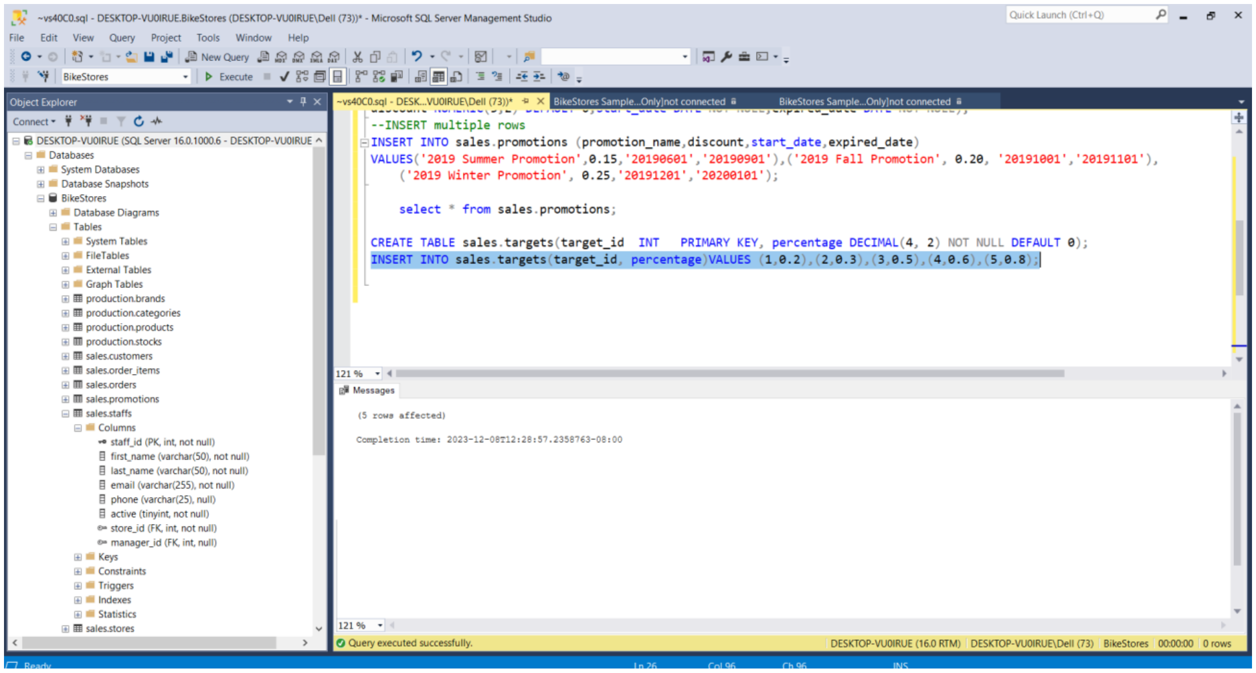


```sql
select * from production.products where(brand_id=1 OR brand_id=2) AND list_price>1000 ORDER BY brand_id;

select first_name + ' '+ last_name AS 'FULL NAME' from sales.customers order by first_name;

--crossjoin
select product_id,product_name,store_id from production.products CROSS JOIN sales.stores order by product_name,store_id;
--CREATE TABLE:
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);
--INSERT multiple rows
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;
```

	promotion_id	promotion_name	discount	start_date	expired_date
1	1	2019 Summer Promotion	0.15	2019-06-01	2019-09-01
2	2	2019 Fall Promotion	0.20	2019-10-01	2019-11-01
3	3	2019 Winter Promotion	0.25	2019-12-01	2020-01-01

Query executed successfully.

First table creation

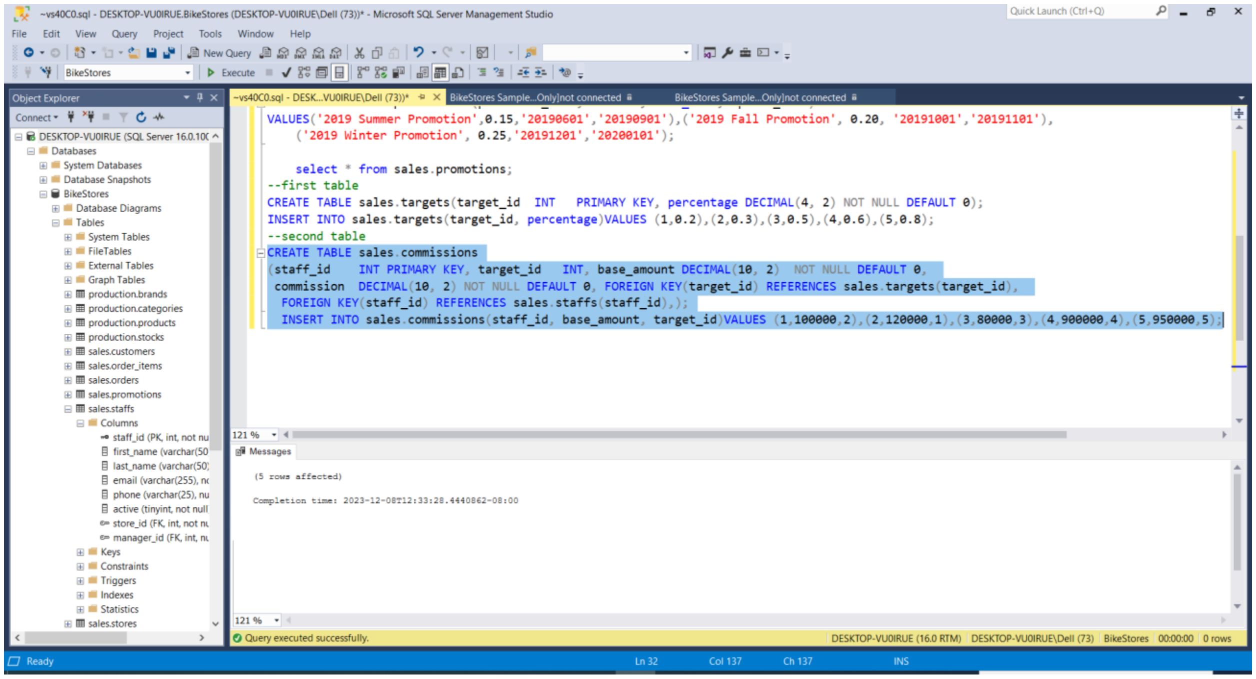


```sql
--INSERT multiple rows
INSERT INTO sales.promotions (promotion_name,discount,start_date,expired_date)
VALUES('2019 Summer Promotion',0.15,'20190601','20190901'),('2019 Fall Promotion', 0.20, '20191001','20191101'),
    ('2019 Winter Promotion', 0.25,'20191201','20200101');

    select * from sales.promotions;

CREATE TABLE sales.targets(target_id  INT   PRIMARY KEY, percentage DECIMAL(4, 2) NOT NULL DEFAULT 0);
INSERT INTO sales.targets(target_id, percentage)VALUES (1,0.2),(2,0.3),(3,0.5),(4,0.6),(5,0.8);
```

```
(5 rows affected)

Completion time: 2023-12-08T12:28:57.2358763-08:00
```

Query executed successfully.

Second table creation

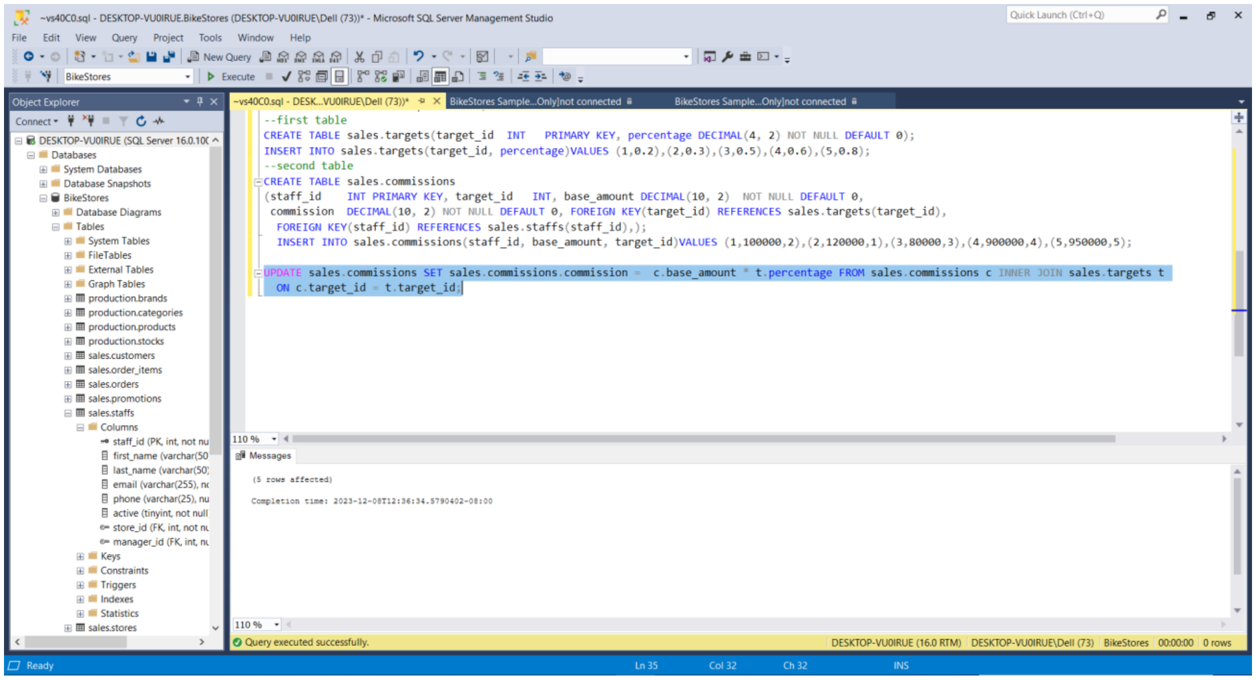


```sql
VALUES('2019 Summer Promotion',0.15,'20190601','20190901'),('2019 Fall Promotion', 0.20, '20191001','20191101'),
    ('2019 Winter Promotion', 0.25,'20191201','20200101');

    select * from sales.promotions;
--first table
CREATE TABLE sales.targets(target_id  INT   PRIMARY KEY, percentage DECIMAL(4, 2) NOT NULL DEFAULT 0);
INSERT INTO sales.targets(target_id, percentage)VALUES (1,0.2),(2,0.3),(3,0.5),(4,0.6),(5,0.8);
--second table
CREATE TABLE sales.commissions
(staff_id    INT PRIMARY KEY, target_id   INT, base_amount DECIMAL(10, 2)  NOT NULL DEFAULT 0,
 commission  DECIMAL(10, 2) NOT NULL DEFAULT 0, FOREIGN KEY(target_id) REFERENCES sales.targets(target_id),
  FOREIGN KEY(staff_id) REFERENCES sales.staffs(staff_id),);
  INSERT INTO sales.commissions(staff_id, base_amount, target_id)VALUES (1,100000,2),(2,120000,1),(3,80000,3),(4,900000,4),(5,950000,5);
```

```
(5 rows affected)

Completion time: 2023-12-08T12:33:28.4440862-08:00
```

Query executed successfully.

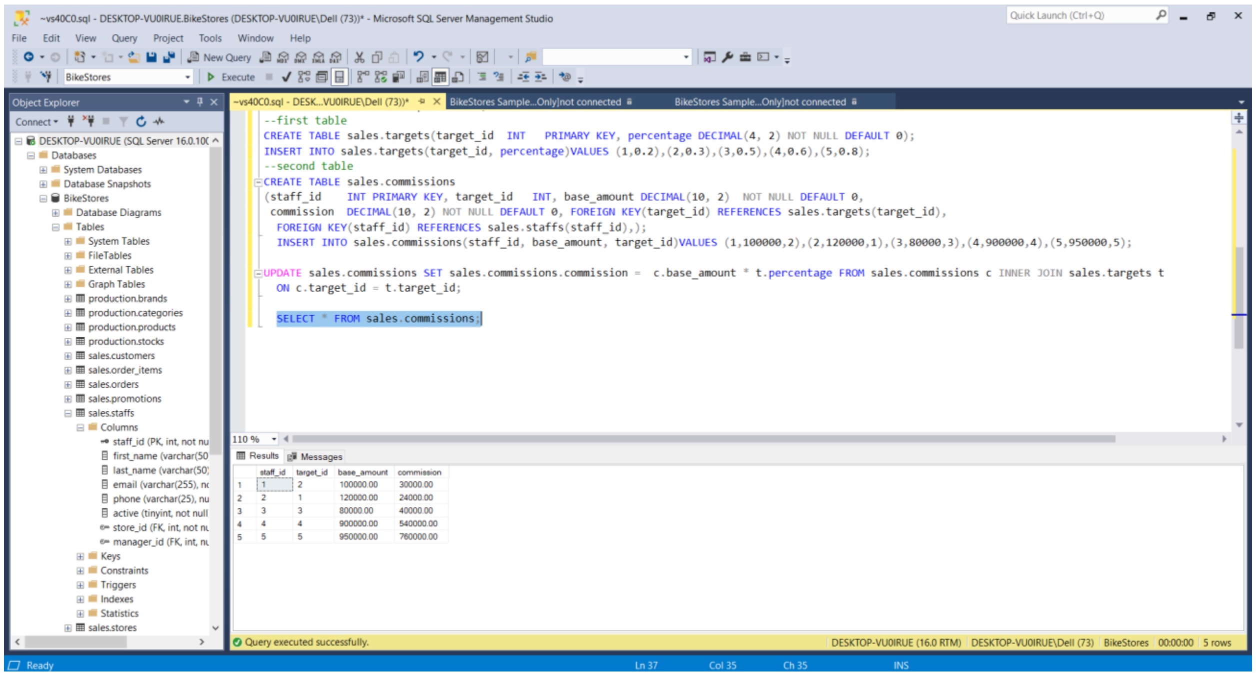
UPDATE INNER JOIN to calculate the sales commission for all sales staffs.



```sql
--first table
CREATE TABLE sales.targets(target_id  INT   PRIMARY KEY, percentage DECIMAL(4, 2) NOT NULL DEFAULT 0);
INSERT INTO sales.targets(target_id, percentage)VALUES (1,0.2),(2,0.3),(3,0.5),(4,0.6),(5,0.8);
--second table
CREATE TABLE sales.commissions
(staff_id    INT PRIMARY KEY, target_id   INT, base_amount DECIMAL(10, 2)  NOT NULL DEFAULT 0,
 commission  DECIMAL(10, 2) NOT NULL DEFAULT 0, FOREIGN KEY(target_id) REFERENCES sales.targets(target_id),
  FOREIGN KEY(staff_id) REFERENCES sales.staffs(staff_id),);
  INSERT INTO sales.commissions(staff_id, base_amount, target_id)VALUES (1,100000,2),(2,120000,1),(3,80000,3),(4,900000,4),(5,950000,5);

UPDATE sales.commissions SET sales.commissions.commission =  c.base_amount * t.percentage FROM sales.commissions c INNER JOIN sales.targets t
  ON c.target_id = t.target_id;
```

```
(5 rows affected)

Completion time: 2023-12-08T12:36:34.5790402-08:00
```

Query executed successfully.

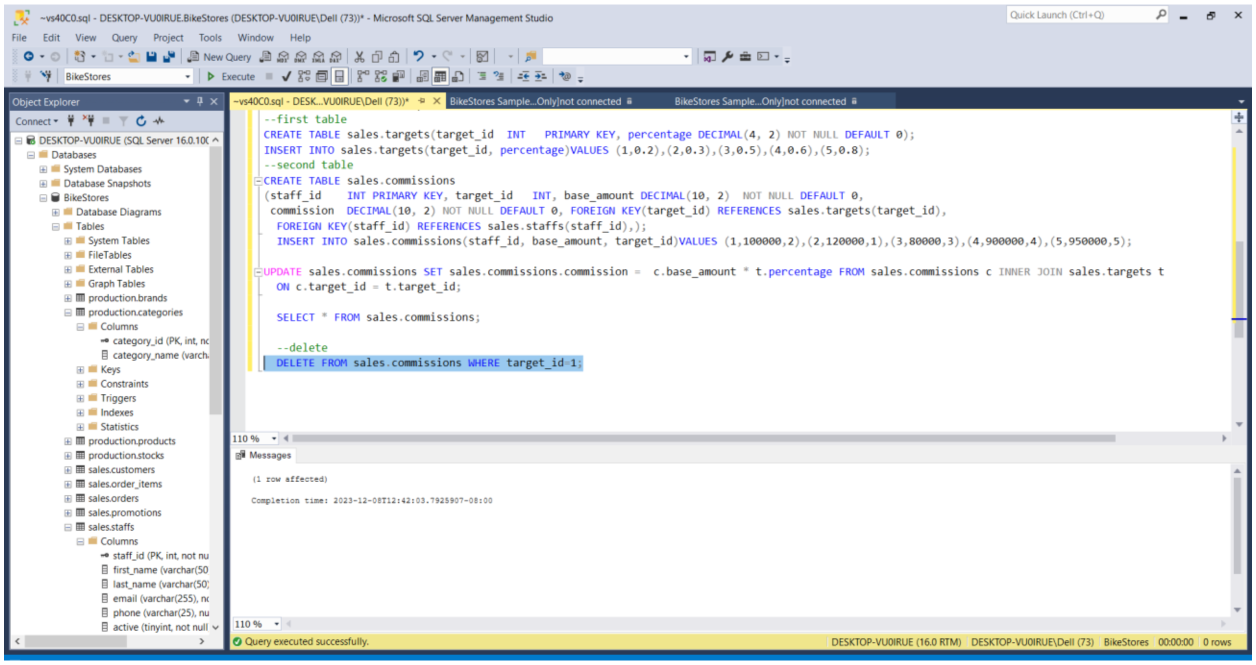


```sql
--first table
CREATE TABLE sales.targets(target_id  INT   PRIMARY KEY, percentage DECIMAL(4, 2) NOT NULL DEFAULT 0);
INSERT INTO sales.targets(target_id, percentage)VALUES (1,0.2),(2,0.3),(3,0.5),(4,0.6),(5,0.8);
--second table
CREATE TABLE sales.commissions
(staff_id    INT PRIMARY KEY, target_id   INT, base_amount DECIMAL(10, 2)  NOT NULL DEFAULT 0,
 commission  DECIMAL(10, 2) NOT NULL DEFAULT 0, FOREIGN KEY(target_id) REFERENCES sales.targets(target_id),
  FOREIGN KEY(staff_id) REFERENCES sales.staffs(staff_id),);
  INSERT INTO sales.commissions(staff_id, base_amount, target_id)VALUES (1,100000,2),(2,120000,1),(3,80000,3),(4,900000,4),(5,950000,5);

UPDATE sales.commissions SET sales.commissions.commission =  c.base_amount * t.percentage FROM sales.commissions c INNER JOIN sales.targets t
  ON c.target_id = t.target_id;

  SELECT * FROM sales.commissions;
```

	staff_id	target_id	base_amount	commission
1	1	2	100000.00	30000.00
2	2	1	120000.00	24000.00
3	3	3	80000.00	40000.00
4	4	4	900000.00	540000.00
5	5	5	950000.00	760000.00

Query executed successfully.

Delete statement with where condition



```
--first table
CREATE TABLE sales.targets(target_id  INT   PRIMARY KEY, percentage DECIMAL(4, 2) NOT NULL DEFAULT 0);
INSERT INTO sales.targets(target_id, percentage)VALUES (1,0.2),(2,0.3),(3,0.5),(4,0.6),(5,0.8);
--second table
CREATE TABLE sales.commissions
(staff_id    INT PRIMARY KEY, target_id    INT, base_amount DECIMAL(10, 2)  NOT NULL DEFAULT 0,
 commission  DECIMAL(10, 2) NOT NULL DEFAULT 0, FOREIGN KEY(target_id) REFERENCES sales.targets(target_id),
  FOREIGN KEY(staff_id) REFERENCES sales.staffs(staff_id),);
  INSERT INTO sales.commissions(staff_id, base_amount, target_id)VALUES (1,100000,2),(2,120000,1),(3,80000,3),(4,900000,4),(5,950000,5);

UPDATE sales.commissions SET sales.commissions.commission =  c.base_amount * t.percentage FROM sales.commissions c INNER JOIN sales.targets t
  ON c.target_id = t.target_id;

  SELECT * FROM sales.commissions;

  --delete
  DELETE FROM sales.commissions WHERE target_id=1;
```

```
(1 row affected)

Completion time: 2023-12-08T12:42:03.7925907-08:00
```

Query executed successfully.

Sheikha khalifa ALQuyudhi