# Module 3: Querying Data using Built-in Functions and T-SQL

## Demo 1 – Querying using Built-in Functions

edureka!

## Querying using Built-in Functions

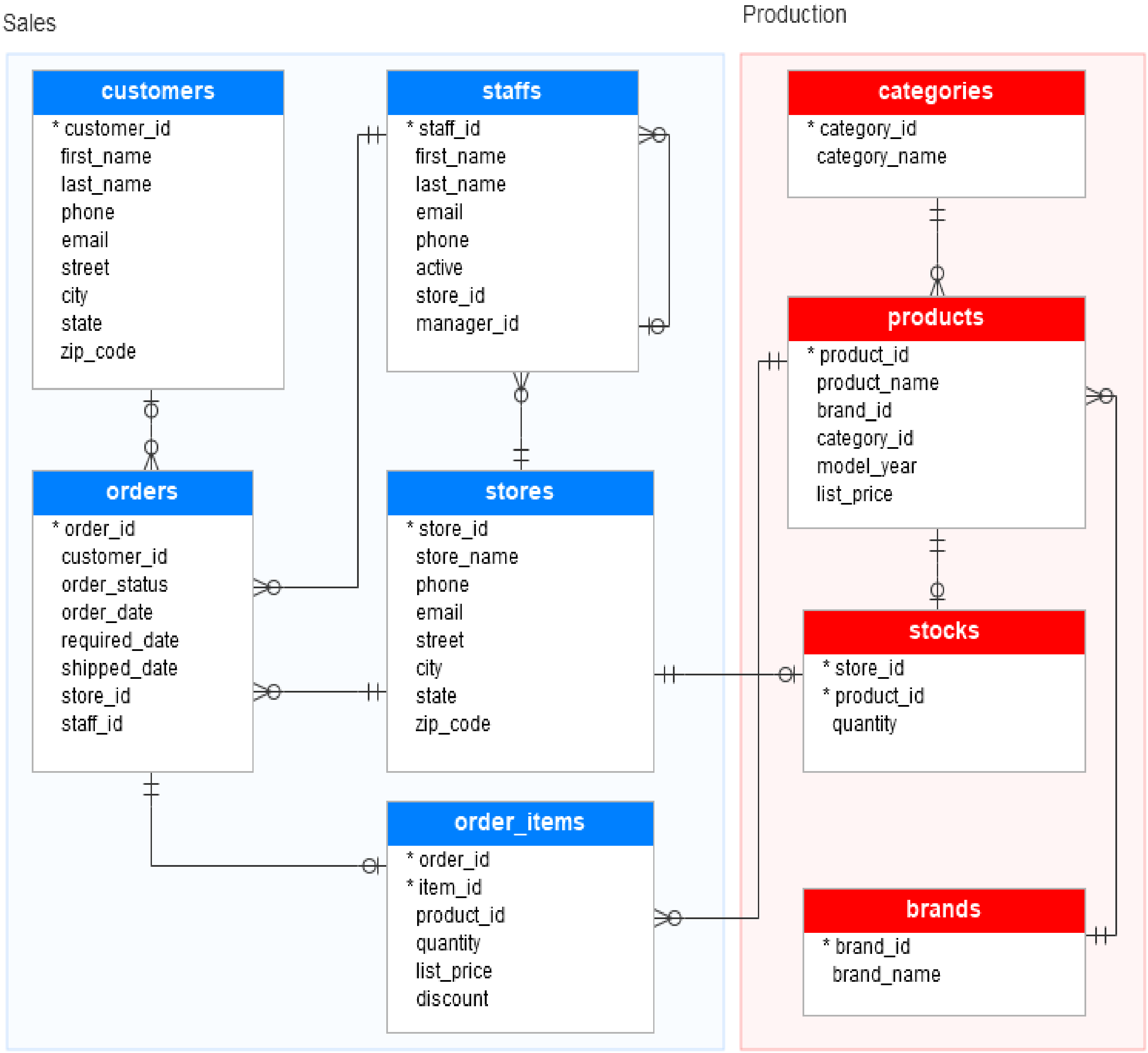
**Problem Statement:** A Bike store is a global motor media company, focusing on Auto exhibition magazine, covers annual brand value reports, offers clients various professional business services, including accounting, auditing, human resources consulting, and strategy management.

Create a database name BikeStores, as shown in database relational schema diagram create the specific tables database has two schemas' sales and production, and these schemas have nine tables.

Write a SQL server Query to find out the following:

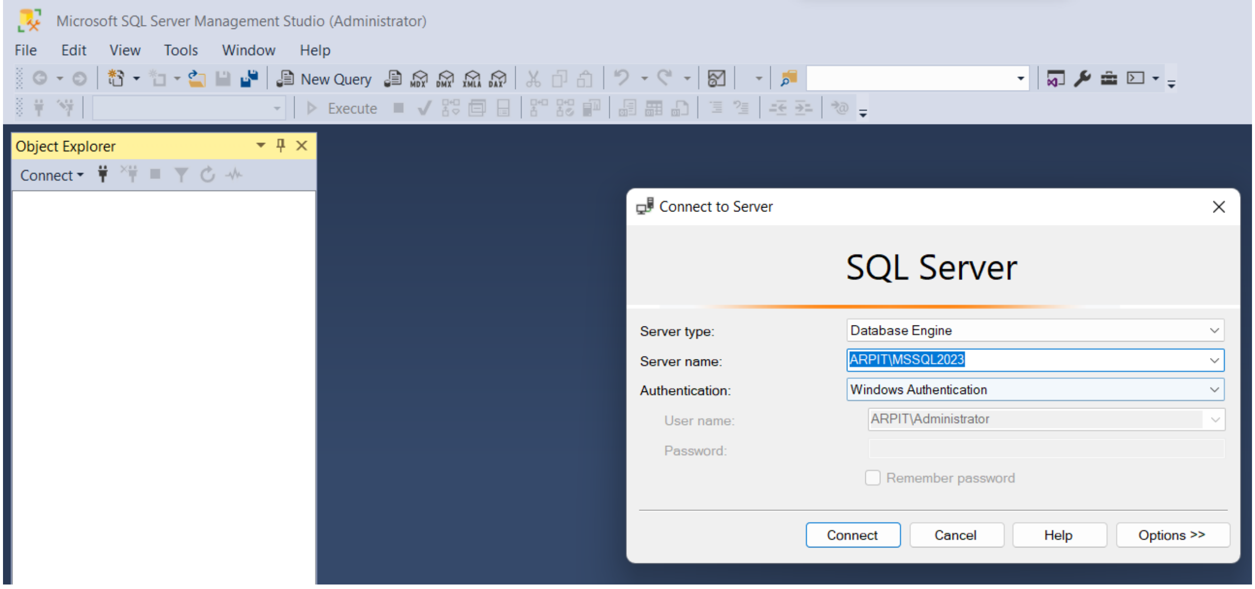
- Create all the Database tables shown in Database Diagram.
- Create database tables of both schema production and sales.
- Load the database to insert the data into the tables.
- Combines names of staff and customers into a single list and sort the first names and last names of customers and staff.
- Find all cities of the customers and the cities of the stores also sort the result set.
- Find the products that had no sales and sorts the products by their id in ascending order.
- Draw the database relational schema diagram.

## Database Diagram:



## Working on the Demo

**Step 1:** Connect to the SQL Server using the Windows Authentication credential and selecting the server's name and server type.



Connect to Server

SQL Server

Server type: Database Engine

Server name: ARPIT\MSSQL2023

Authentication: Windows Authentication

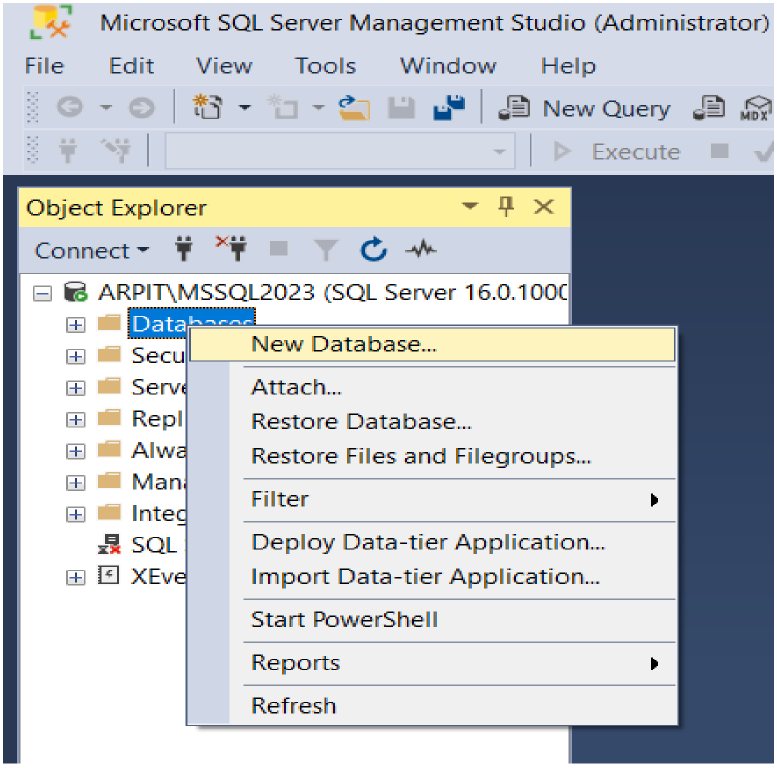
User name: ARPIT\Administrator

Password:

□ Remember password

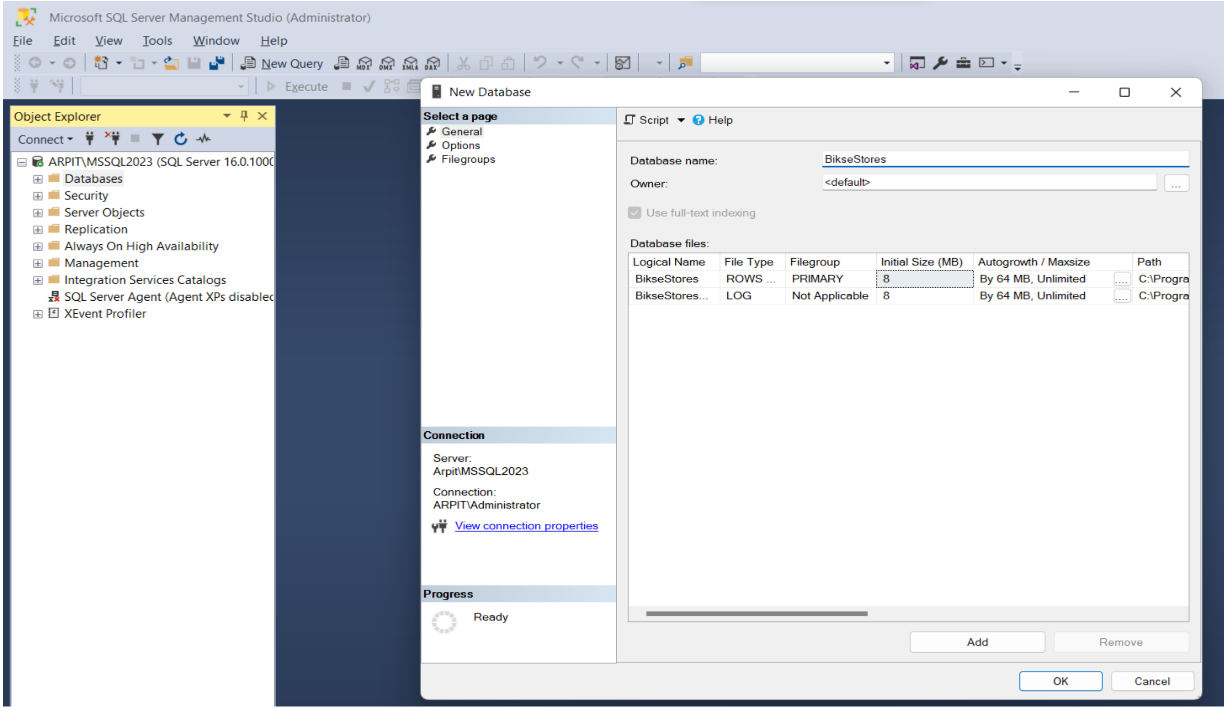
Microsoft SQL Server Management Studio (Administrator)
File Edit View Tools Window Help
New Query
Execute
Object Explorer
Connect
Connect
Cancel
Help
Options >>

**Step 2:** Create a database named **BikeStores** by right-clicking on databases.



Microsoft SQL Server Management Studio (Administrator)
File Edit View Tools Window Help
New Query
Execute
Object Explorer
Connect
ARPIT\MSSQL2023 (SQL Server 16.0.100(
Databases
Secu
Serv
Repl
Alwa
Man
Integ
SQL
XEve
New Database...
Attach...
Restore Database...
Restore Files and Filegroups...
Filter
Deploy Data-tier Application...
Import Data-tier Application...
Start PowerShell
Reports
Refresh

**Step 3:** Give a database a **BikeStores** and check the connection and server name than click on the OK.


Microsoft SQL Server Management Studio (Administrator)
File Edit View Tools Window Help
New Query
Execute
Object Explorer
Connect
ARPIT\MSSQL2023 (SQL Server 16.0.1000
Databases
Security
Server Objects
Replication
Always On High Availability
Management
Integration Services Catalogs
SQL Server Agent (Agent XPs disabled
XEvent Profiler
New Database
Select a page
General
Options
Filegroups
Script
Help
Add
Remove
OK
Cancel

Database name: BikseStores

Owner: <default>

☑ Use full-text indexing

Database files:

| Logical Name | File Type | Filegroup | Initial Size (MB) | Autogrowth / Maxsize | Path |
|---|---|---|---|---|---|
| BikseStores | ROWS ... | PRIMARY | 8 | By 64 MB, Unlimited | C:\Progra |
| BikseStores... | LOG | Not Applicable | 8 | By 64 MB, Unlimited | C:\Progra |

Connection

Server:
Arpit\MSSQL2023

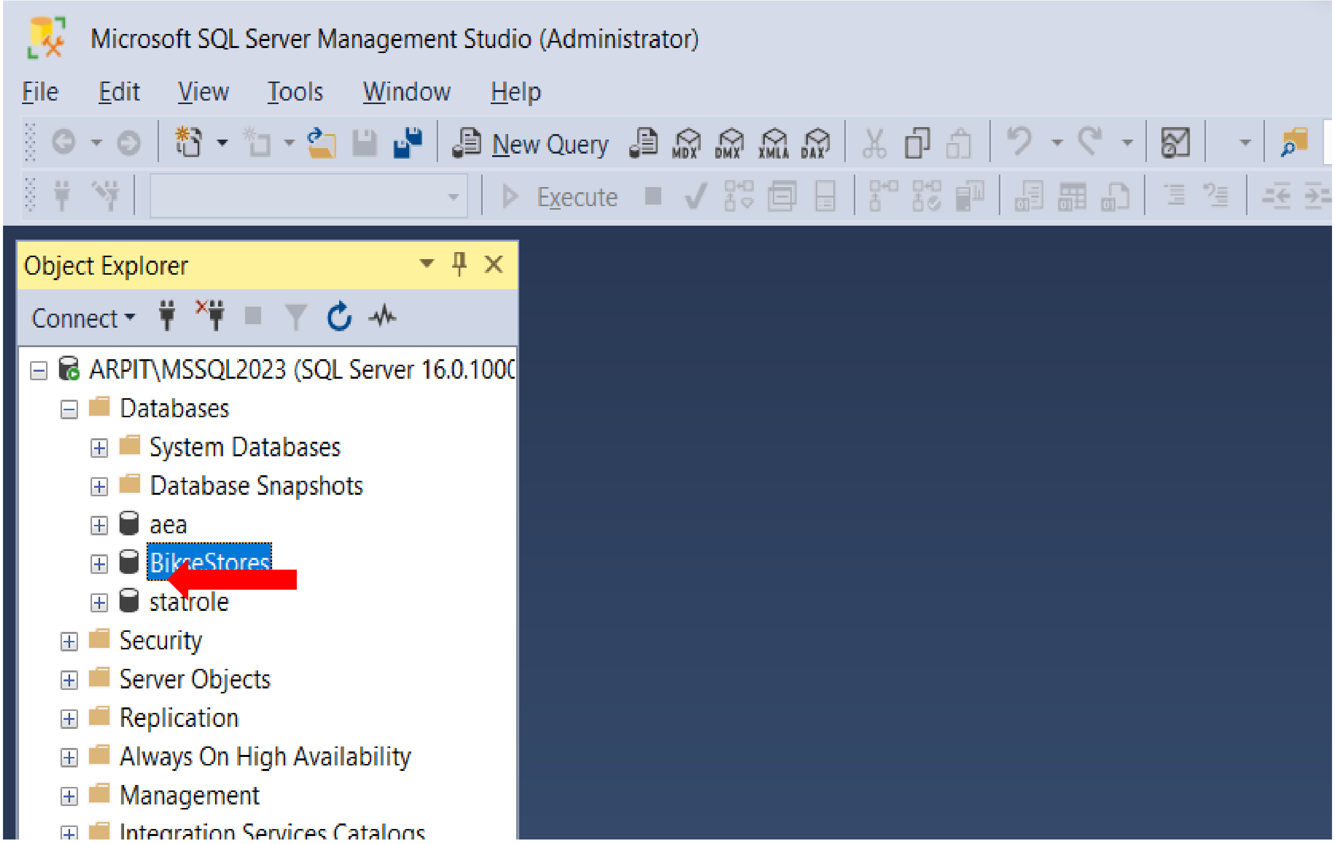
Connection:
ARPIT\Administrator

View connection properties

Progress

Ready

**Step 4:** Select the created database in left hand navigation panel under the Databases.


Microsoft SQL Server Management Studio (Administrator)
File Edit View Tools Window Help
New Query
Execute
Object Explorer
Connect
ARPIT\MSSQL2023 (SQL Server 16.0.1000
Databases
System Databases
Database Snapshots
aea
BikseStores
statrole
Security
Server Objects
Replication
Always On High Availability
Management
Integration Services Catalogs

**Step 5:** Now, we will create the required all Database tables as shown database relational scheme.

### Database Tables:

### Table sales.stores

The sales.stores table includes the store's information. Each store has a store name, contact information such as phone and email, and an address including street, city, state, and zip code.

```
CREATE TABLE sales.stores (
      store_id INT IDENTITY (1, 1) PRIMARY KEY,
      store_name VARCHAR (255) NOT NULL,
      phone VARCHAR (25),
      email VARCHAR (255),
      street VARCHAR (255),
      city VARCHAR (255),
      state VARCHAR (10),
      zip_code VARCHAR (5)
);
```

### Table sales.staffs

The sales.staffs table stores the essential information of staffs including first name, last name. It also contains the communication information such as email and phone.

A staff works at a store specified by the value in the store_id column. A store can have one or more staffs.

A staff reports to a store manager specified by the value in the manager_id column. If the value in the manager_id is null, then the staff is the top manager.

If a staff no longer works for any stores, the value in the active column is set to zero.

```
CREATE TABLE sales.staffs (
      staff_id INT IDENTITY (1, 1) PRIMARY KEY,
      first_name VARCHAR (50) NOT NULL,
      last_name VARCHAR (50) NOT NULL,
      email VARCHAR (255) NOT NULL UNIQUE,
      phone VARCHAR (25),
      active tinyint NOT NULL,
      store_id INT NOT NULL,
      manager_id INT,
      FOREIGN KEY (store_id)
        REFERENCES sales.stores (store_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (manager_id)
        REFERENCES sales.staffs (staff_id)
        ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

### Table production.categories

The production.categories table stores the bike's categories such as children bicycles, comfort bicycles, and electric bikes.

```
CREATE TABLE production.categories (
      category_id INT IDENTITY (1, 1) PRIMARY KEY,
      category_name VARCHAR (255) NOT NULL
);
```

### Table production.brands

The production.brands table stores the brand's information of bikes, for example, Electra, Haro, and Heller.

```
CREATE TABLE production.brands (
      brand_id INT IDENTITY (1, 1) PRIMARY KEY,
      brand_name VARCHAR (255) NOT NULL
);
```

### Table production.products

The production.products table stores the product's information such as name, brand, category, model year, and list price.

Each product belongs to a brand specified by the brand_id column. Hence, a brand may have zero or many products.

Each product also belongs a category specified by the category_id column. Also, each category may have zero or many products.

```
CREATE TABLE production.products (
      product_id INT IDENTITY (1, 1) PRIMARY KEY,
      product_name VARCHAR (255) NOT NULL,
      brand_id INT NOT NULL,
      category_id INT NOT NULL,
      model_year SMALLINT NOT NULL,
      list_price DECIMAL (10, 2) NOT NULL,
      FOREIGN KEY (category_id)
        REFERENCES production.categories (category_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (brand_id)
        REFERENCES production.brands (brand_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

### Table sales.customers

The sales.customers table stores customer's information including first name, last name, phone, email, street, city, state and zip code.

```
CREATE TABLE sales.customers (
      customer_id INT IDENTITY (1, 1) PRIMARY KEY,
      first_name VARCHAR (255) NOT NULL,
      last_name VARCHAR (255) NOT NULL,
      phone VARCHAR (25),
      email VARCHAR (255) NOT NULL,
      street VARCHAR (255),
      city VARCHAR (50),
      state VARCHAR (25),
      zip_code VARCHAR (5)
);
```

### Table sales.orders

The sales.orders table stores the sales order's header information including customer, order status, order date, required date, shipped date.

It also stores the information on where the sales transaction was created (store) and who created it (staff).

Each sales order has a row in the sales_orders table. A sales order has one or many line items stored in the sales.order_items table.

```
CREATE TABLE sales.orders (
      order_id INT IDENTITY (1, 1) PRIMARY KEY,
      customer_id INT,
      order_status tinyint NOT NULL,
      -- Order status: 1 = Pending; 2 = Processing; 3 =
Rejected; 4 = Completed
      order_date DATE NOT NULL,
      required_date DATE NOT NULL,
      shipped_date DATE,
      store_id INT NOT NULL,
      staff_id INT NOT NULL,
      FOREIGN KEY (customer_id)
        REFERENCES sales.customers (customer_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (store_id)
        REFERENCES sales.stores (store_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (staff_id)
        REFERENCES sales.staffs (staff_id)
        ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

**Table sales.order_items**
The sales.order_items table stores the line items of a sales order. Each line item belongs to a sales order specified by the order_id column.

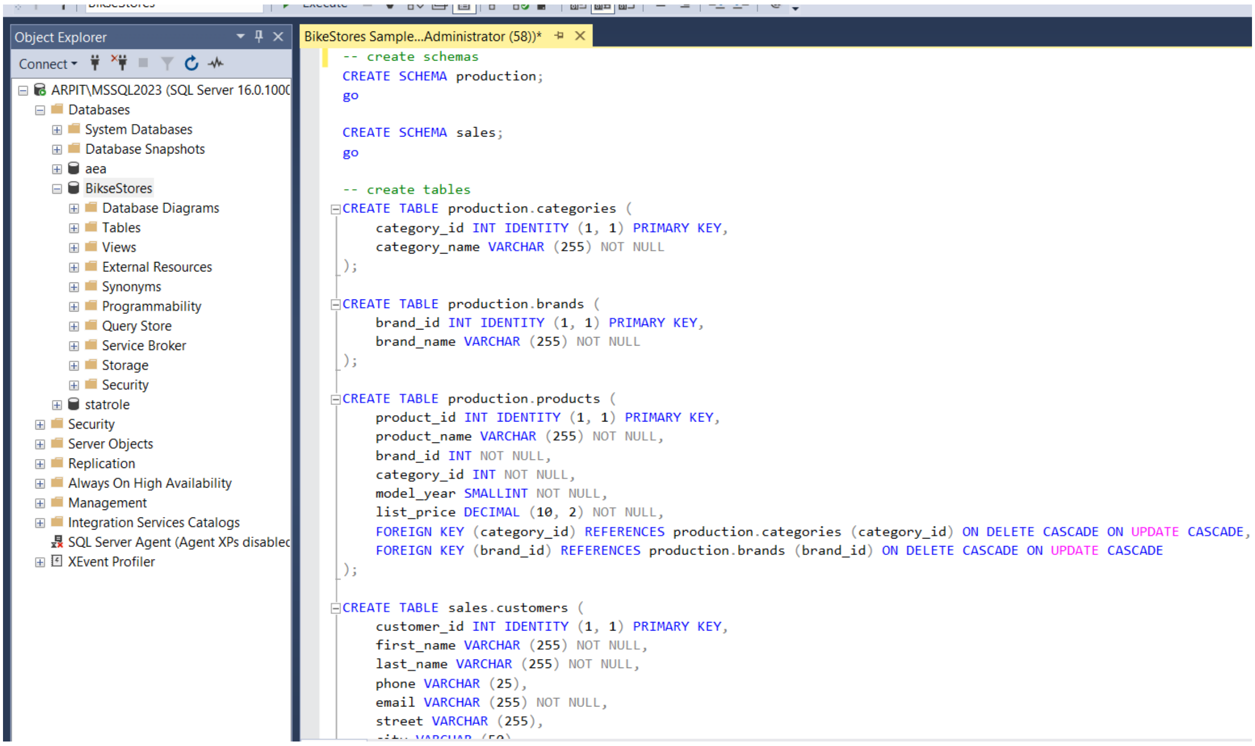
A sales order line item includes product, order quantity, list price, and discount.

```
CREATE TABLE sales.order_items(
      order_id INT,
      item_id INT,
      product_id INT NOT NULL,
      quantity INT NOT NULL,
      list_price DECIMAL (10, 2) NOT NULL,
      discount DECIMAL (4, 2) NOT NULL DEFAULT 0,
      PRIMARY KEY (order_id, item_id),
      FOREIGN KEY (order_id)
        REFERENCES sales.orders (order_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (product_id)
        REFERENCES production.products (product_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

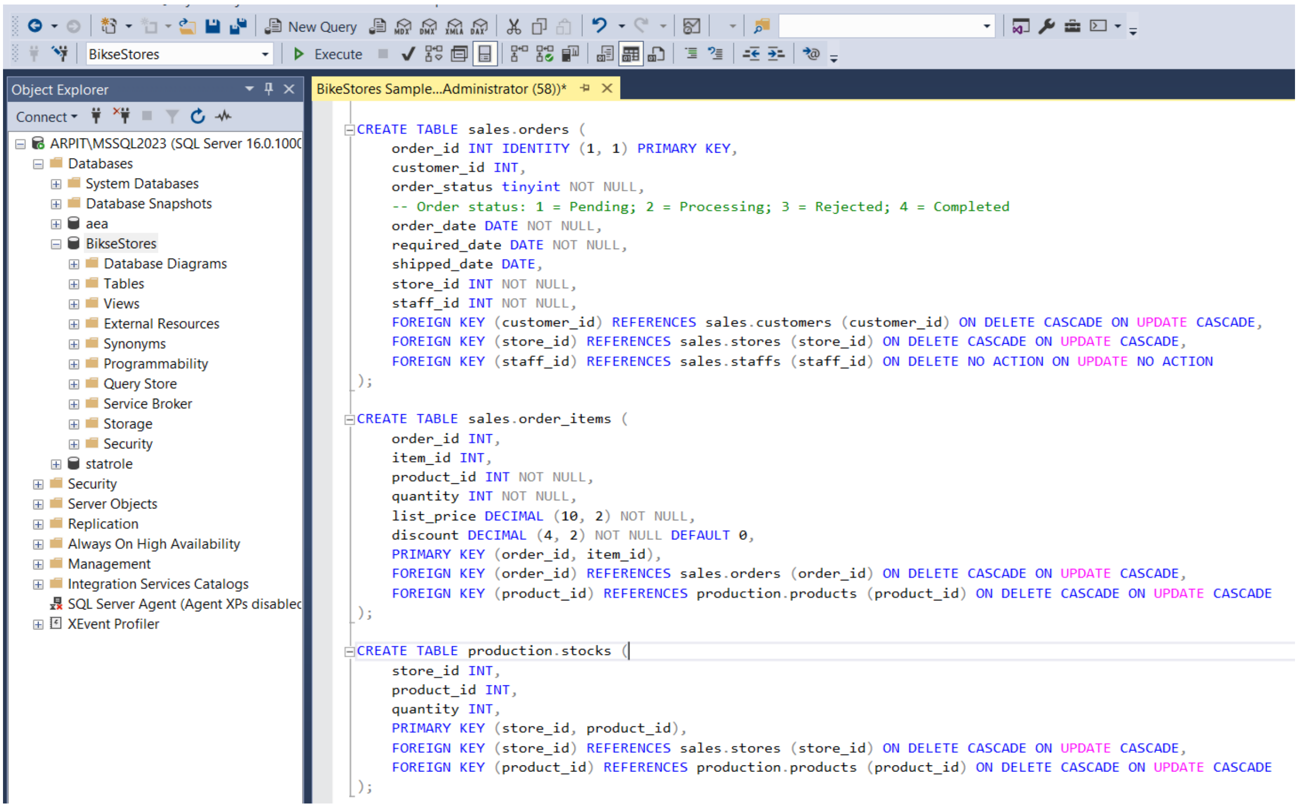
**Table production.stocks**
The production.stocks table stores the inventory information i.e. the quantity of a particular product in a specific store.

```
CREATE TABLE production.stocks (
      store_id INT,
      product_id INT,
      quantity INT,
      PRIMARY KEY (store_id, product_id),
      FOREIGN KEY (store_id)
   REFERENCES sales.stores (store_id)
   ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (product_id)
   REFERENCES production.products (product_id)
   ON DELETE CASCADE ON UPDATE CASCADE
);
```

**Step 6:** Run all the queries to create database tables of both schema production and sales.



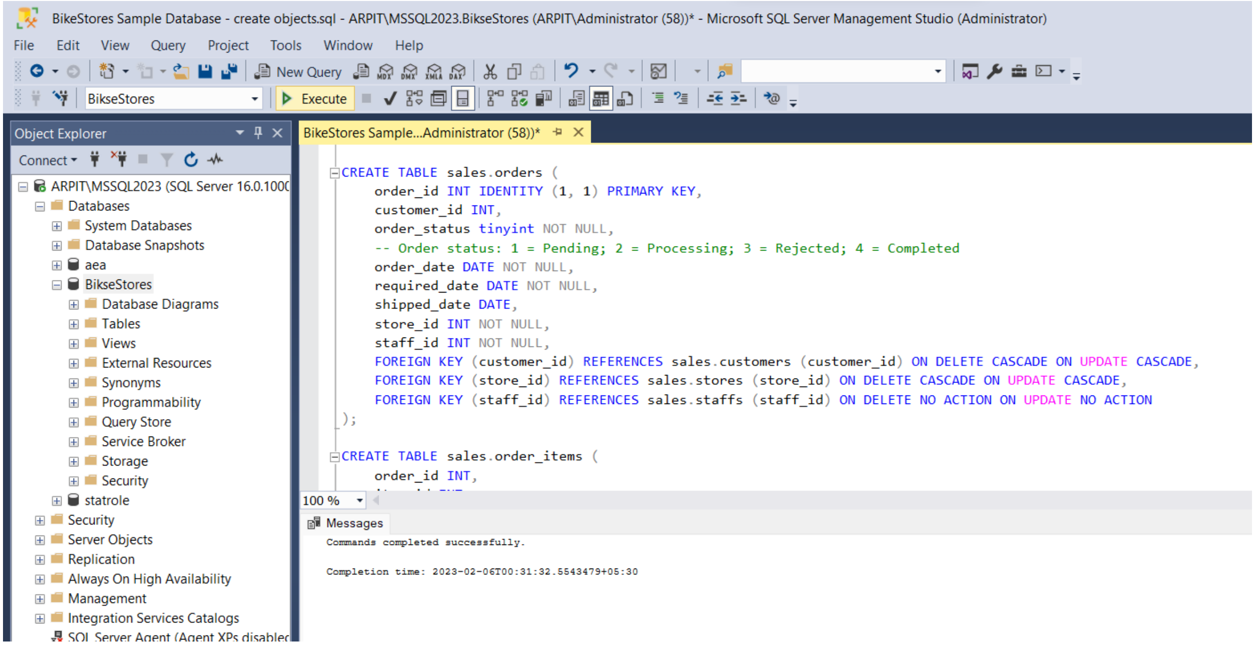
```
-- create schemas
CREATE SCHEMA production;
go

CREATE SCHEMA sales;
go

-- create tables
CREATE TABLE production.categories (
    category_id INT IDENTITY (1, 1) PRIMARY KEY,
    category_name VARCHAR (255) NOT NULL
);

CREATE TABLE production.brands (
    brand_id INT IDENTITY (1, 1) PRIMARY KEY,
    brand_name VARCHAR (255) NOT NULL
);

CREATE TABLE production.products (
    product_id INT IDENTITY (1, 1) PRIMARY KEY,
    product_name VARCHAR (255) NOT NULL,
    brand_id INT NOT NULL,
    category_id INT NOT NULL,
    model_year SMALLINT NOT NULL,
    list_price DECIMAL (10, 2) NOT NULL,
    FOREIGN KEY (category_id) REFERENCES production.categories (category_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (brand_id) REFERENCES production.brands (brand_id) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE sales.customers (
    customer_id INT IDENTITY (1, 1) PRIMARY KEY,
    first_name VARCHAR (255) NOT NULL,
    last_name VARCHAR (255) NOT NULL,
    phone VARCHAR (25),
    email VARCHAR (255) NOT NULL,
    street VARCHAR (255),
```



```
CREATE TABLE sales.orders (
    order_id INT IDENTITY (1, 1) PRIMARY KEY,
    customer_id INT,
    order_status tinyint NOT NULL,
    -- Order status: 1 = Pending; 2 = Processing; 3 = Rejected; 4 = Completed
    order_date DATE NOT NULL,
    required_date DATE NOT NULL,
    shipped_date DATE,
    store_id INT NOT NULL,
    staff_id INT NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES sales.customers (customer_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (store_id) REFERENCES sales.stores (store_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (staff_id) REFERENCES sales.staffs (staff_id) ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE TABLE sales.order_items (
    order_id INT,
    item_id INT,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    list_price DECIMAL (10, 2) NOT NULL,
    discount DECIMAL (4, 2) NOT NULL DEFAULT 0,
    PRIMARY KEY (order_id, item_id),
    FOREIGN KEY (order_id) REFERENCES sales.orders (order_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (product_id) REFERENCES production.products (product_id) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE production.stocks (
    store_id INT,
    product_id INT,
    quantity INT,
    PRIMARY KEY (store_id, product_id),
    FOREIGN KEY (store_id) REFERENCES sales.stores (store_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (product_id) REFERENCES production.products (product_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

**Step 7:** Execute all the queries to create database tables of both schema production and sales.

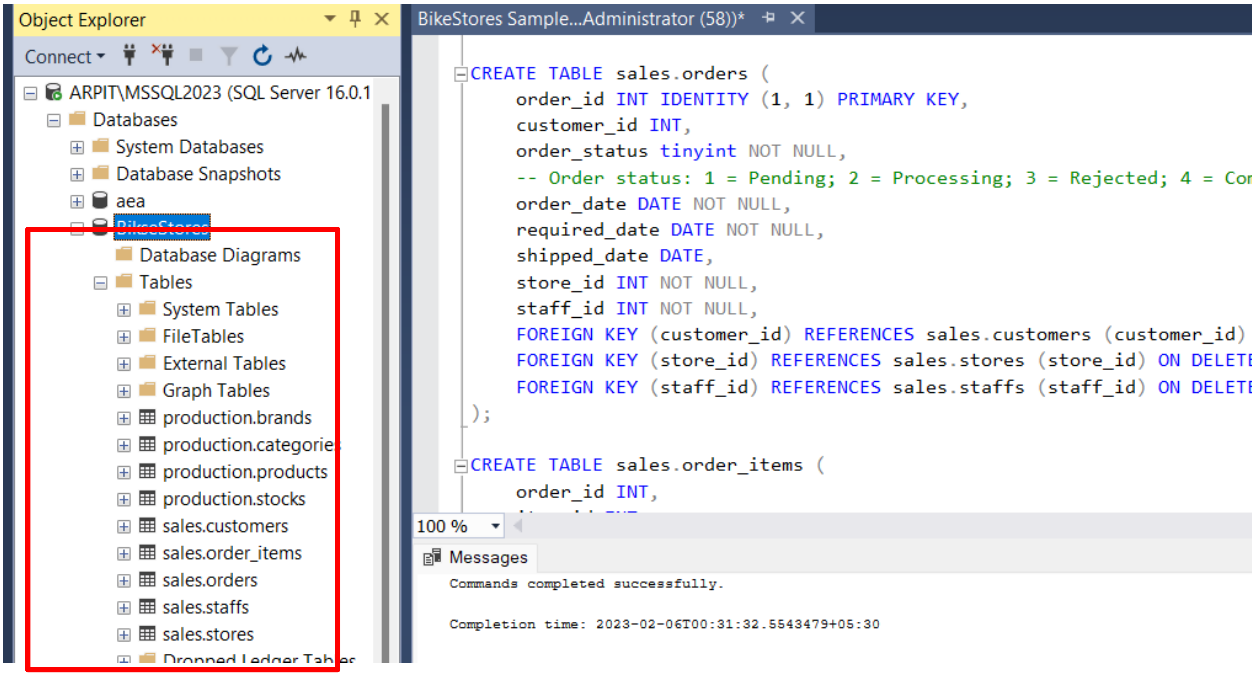
Once it gets executed you can see the message, commands completed successfully.


BikeStores Sample Database - create objects.sql - ARPIT\MSSQL2023.BikseStores (ARPIT\Administrator (58))* - Microsoft SQL Server Management Studio (Administrator)
File
Edit
View
Query
Project
Tools
Window
Help
New Query
BikseStores
Execute
Object Explorer
Connect
BikeStores Sample...Administrator (58))*
ARPIT\MSSQL2023 (SQL Server 16.0.1000
Databases
System Databases
Database Snapshots
aea
BikseStores
Database Diagrams
Tables
Views
External Resources
Synonyms
Programmability
Query Store
Service Broker
Storage
Security
statrole
Security
Server Objects
Replication
Always On High Availability
Management
Integration Services Catalogs
SQL Server Agent (Agent XPs disabled
100 %
Messages

```
CREATE TABLE sales.orders (
	order_id INT IDENTITY (1, 1) PRIMARY KEY,
	customer_id INT,
	order_status tinyint NOT NULL,
	-- Order status: 1 = Pending; 2 = Processing; 3 = Rejected; 4 = Completed
	order_date DATE NOT NULL,
	required_date DATE NOT NULL,
	shipped_date DATE,
	store_id INT NOT NULL,
	staff_id INT NOT NULL,
	FOREIGN KEY (customer_id) REFERENCES sales.customers (customer_id) ON DELETE CASCADE ON UPDATE CASCADE,
	FOREIGN KEY (store_id) REFERENCES sales.stores (store_id) ON DELETE CASCADE ON UPDATE CASCADE,
	FOREIGN KEY (staff_id) REFERENCES sales.staffs (staff_id) ON DELETE NO ACTION ON UPDATE NO ACTION
);

CREATE TABLE sales.order_items (
	order_id INT,
```

```
Commands completed successfully.

Completion time: 2023-02-06T00:31:32.5543479+05:30
```

**Step 8:** Click on the BikeStores database in the left-hand navigation panel. Under the database click on the Tables you can see the created tables are listed down.

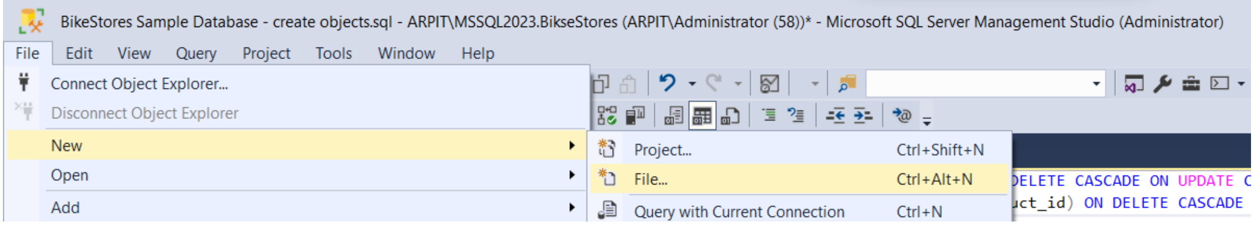


```
CREATE TABLE sales.orders (
	order_id INT IDENTITY (1, 1) PRIMARY KEY,
	customer_id INT,
	order_status tinyint NOT NULL,
	-- Order status: 1 = Pending; 2 = Processing; 3 = Rejected; 4 = Cor
	order_date DATE NOT NULL,
	required_date DATE NOT NULL,
	shipped_date DATE,
	store_id INT NOT NULL,
	staff_id INT NOT NULL,
	FOREIGN KEY (customer_id) REFERENCES sales.customers (customer_id)
	FOREIGN KEY (store_id) REFERENCES sales.stores (store_id) ON DELETE
	FOREIGN KEY (staff_id) REFERENCES sales.staffs (staff_id) ON DELETE
);

CREATE TABLE sales.order_items (
	order_id INT,
```

```
Commands completed successfully.

Completion time: 2023-02-06T00:31:32.5543479+05:30
```

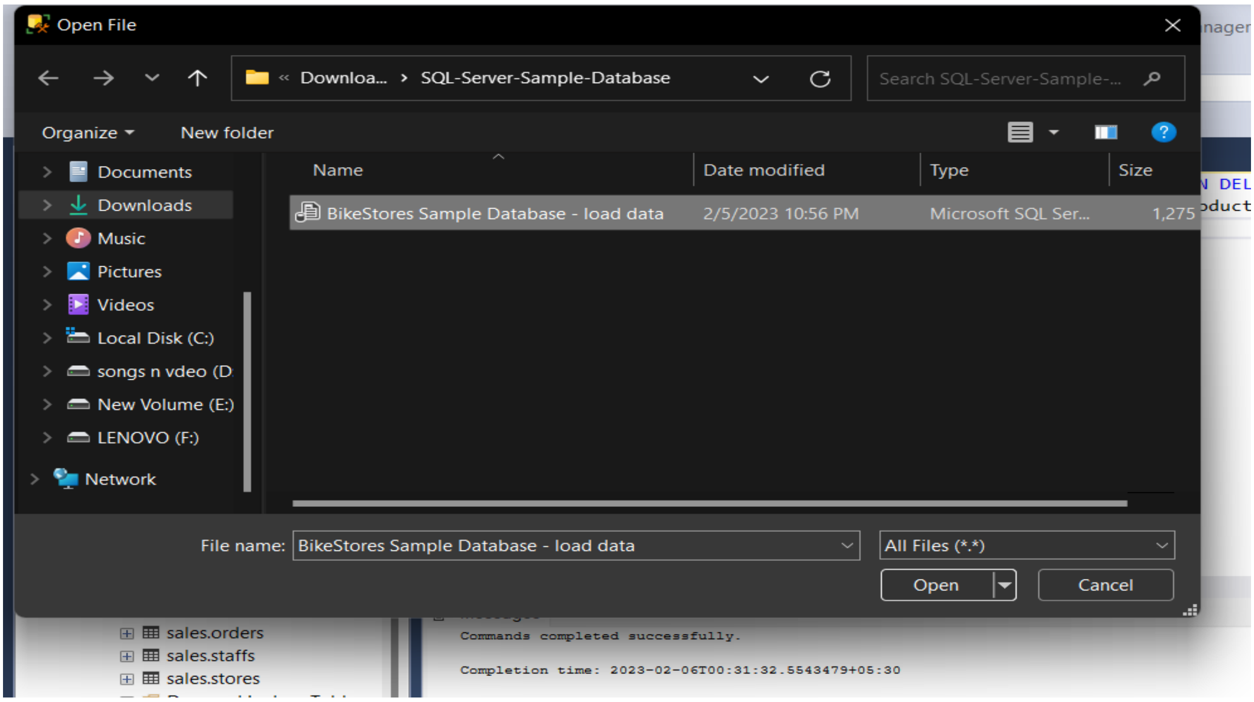
**Step 9:** Once the tables are created and we will insert the data into the tables.

Click on the File > New> File


BikeStores Sample Database - create objects.sql - ARPIT\MSSQL2023.BikseStores (ARPIT\Administrator (58))* - Microsoft SQL Server Management Studio (Administrator)
File
Edit
View
Query
Project
Tools
Window
Help
Connect Object Explorer...
Disconnect Object Explorer
New
Open
Add
Project... Ctrl+Shift+N
File... Ctrl+Alt+N
Query with Current Connection Ctrl+N

```
DELETE CASCADE ON UPDATE C
uct_id) ON DELETE CASCADE
```

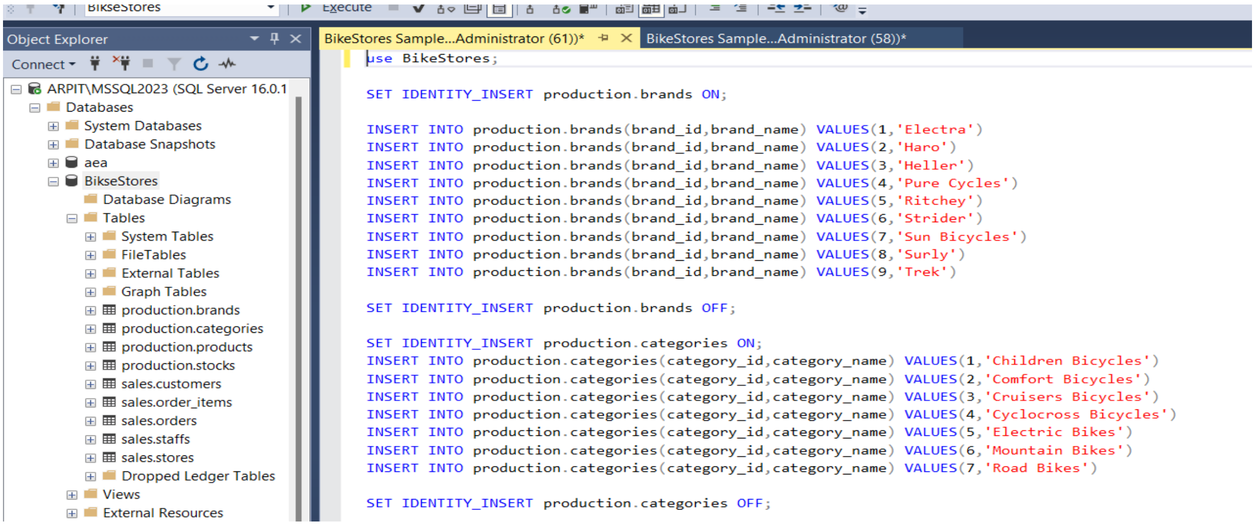
And select the file to load data into the tables of  BikeStores database.



File name: BikeStores Sample Database - load data

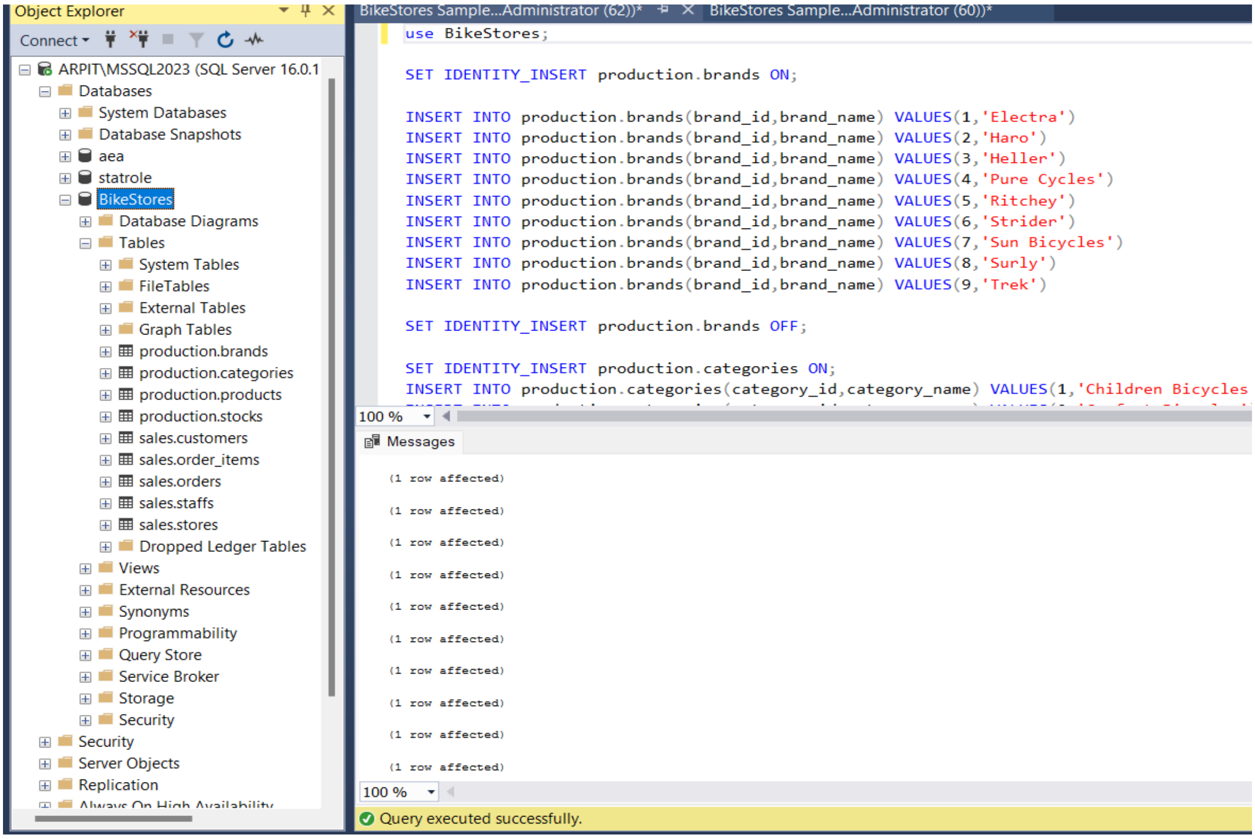
```
Commands completed successfully.

Completion time: 2023-02-06T00:31:32.5543479+05:30
```

Once the data are loaded in the database, check the database name used correctly in order to get connected.



```
use BikeStores;

SET IDENTITY_INSERT production.brands ON;

INSERT INTO production.brands(brand_id,brand_name) VALUES(1,'Electra')
INSERT INTO production.brands(brand_id,brand_name) VALUES(2,'Haro')
INSERT INTO production.brands(brand_id,brand_name) VALUES(3,'Heller')
INSERT INTO production.brands(brand_id,brand_name) VALUES(4,'Pure Cycles')
INSERT INTO production.brands(brand_id,brand_name) VALUES(5,'Ritchey')
INSERT INTO production.brands(brand_id,brand_name) VALUES(6,'Strider')
INSERT INTO production.brands(brand_id,brand_name) VALUES(7,'Sun Bicycles')
INSERT INTO production.brands(brand_id,brand_name) VALUES(8,'Surly')
INSERT INTO production.brands(brand_id,brand_name) VALUES(9,'Trek')

SET IDENTITY_INSERT production.brands OFF;

SET IDENTITY_INSERT production.categories ON;
INSERT INTO production.categories(category_id,category_name) VALUES(1,'Children Bicycles')
INSERT INTO production.categories(category_id,category_name) VALUES(2,'Comfort Bicycles')
INSERT INTO production.categories(category_id,category_name) VALUES(3,'Cruisers Bicycles')
INSERT INTO production.categories(category_id,category_name) VALUES(4,'Cyclocross Bicycles')
INSERT INTO production.categories(category_id,category_name) VALUES(5,'Electric Bikes')
INSERT INTO production.categories(category_id,category_name) VALUES(6,'Mountain Bikes')
INSERT INTO production.categories(category_id,category_name) VALUES(7,'Road Bikes')

SET IDENTITY_INSERT production.categories OFF;
```

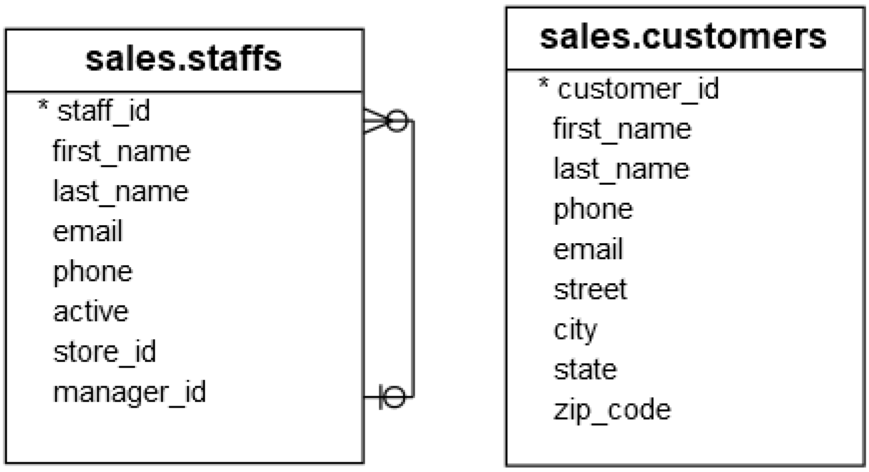
**Step 10:** Execute the query to load the data, wait for while once the query gets executed successfully.



```
use BikeStores;

SET IDENTITY_INSERT production.brands ON;

INSERT INTO production.brands(brand_id,brand_name) VALUES(1,'Electra')
INSERT INTO production.brands(brand_id,brand_name) VALUES(2,'Haro')
INSERT INTO production.brands(brand_id,brand_name) VALUES(3,'Heller')
INSERT INTO production.brands(brand_id,brand_name) VALUES(4,'Pure Cycles')
INSERT INTO production.brands(brand_id,brand_name) VALUES(5,'Ritchey')
INSERT INTO production.brands(brand_id,brand_name) VALUES(6,'Strider')
INSERT INTO production.brands(brand_id,brand_name) VALUES(7,'Sun Bicycles')
INSERT INTO production.brands(brand_id,brand_name) VALUES(8,'Surly')
INSERT INTO production.brands(brand_id,brand_name) VALUES(9,'Trek')

SET IDENTITY_INSERT production.brands OFF;

SET IDENTITY_INSERT production.categories ON;
INSERT INTO production.categories(category_id,category_name) VALUES(1,'Children Bicycles
```

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

Query executed successfully.



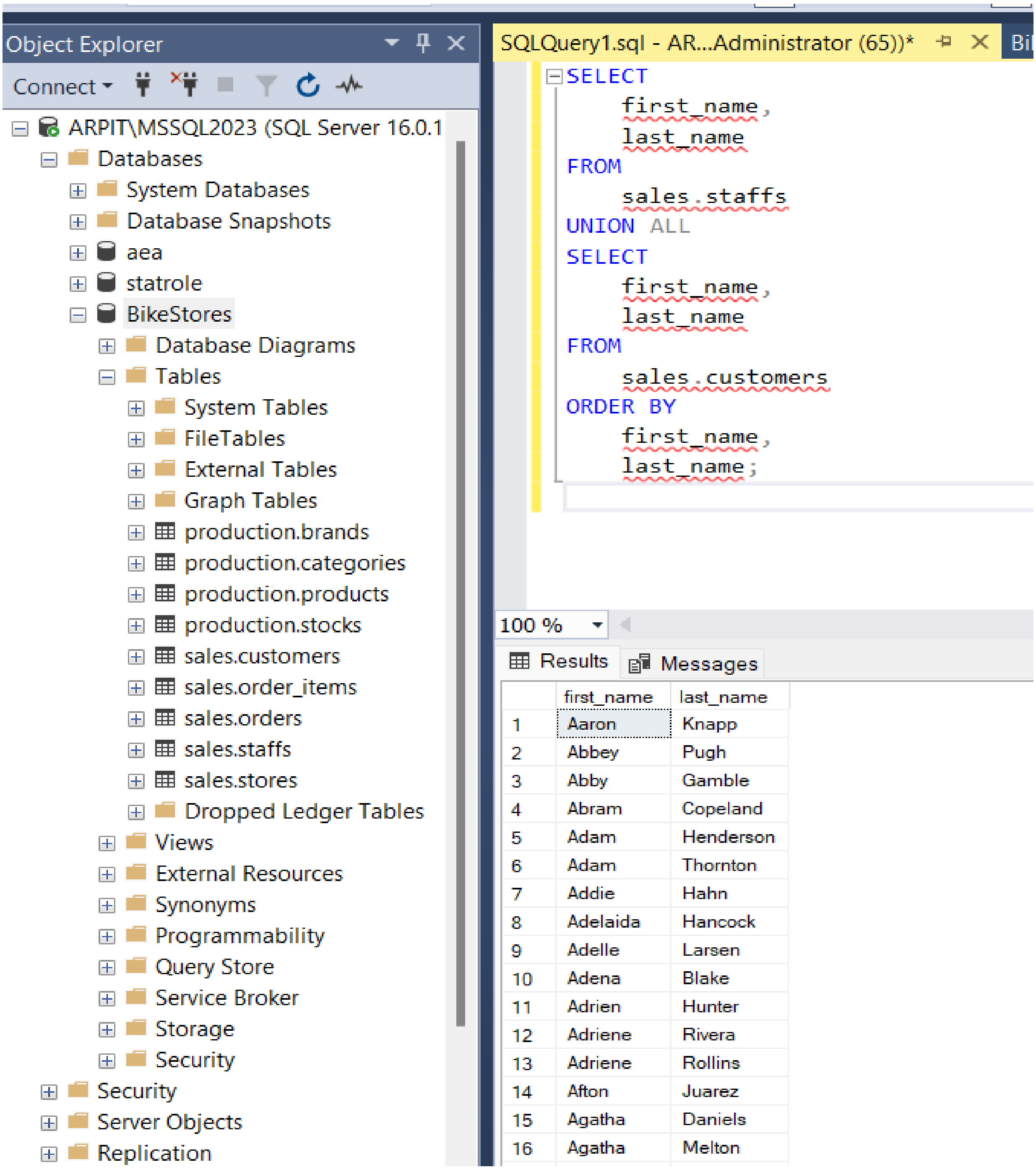
sales.staffs

* staff_id
first_name
last_name
email
phone
active
store_id
manager_id

sales.customers

* customer_id
first_name
last_name
phone
email
street
city
state
zip_code

**Step 11**: Combines names of staff and customers into a single list and sort the first names and last names of customers and staff.

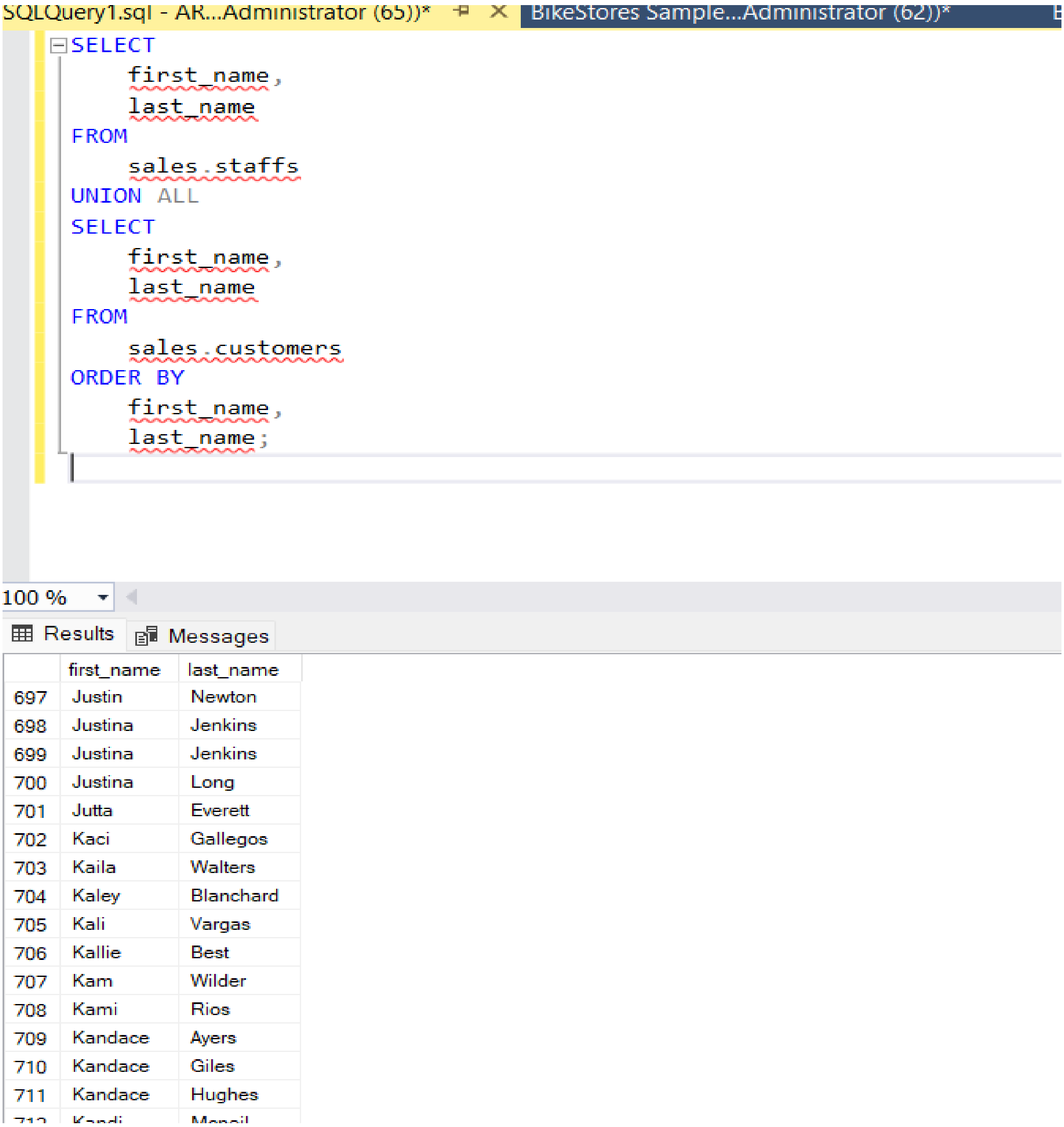


```
SELECT
    first_name,
    last_name
FROM
    sales.staffs
UNION ALL
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    first_name,
    last_name;
```

| | first_name | last_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 | Rivera |
| 13 | Adriene | Rollins |
| 14 | Afton | Juarez |
| 15 | Agatha | Daniels |
| 16 | Agatha | Melton |

After the query gets executed, and the query returns 1,455 rows as expected.

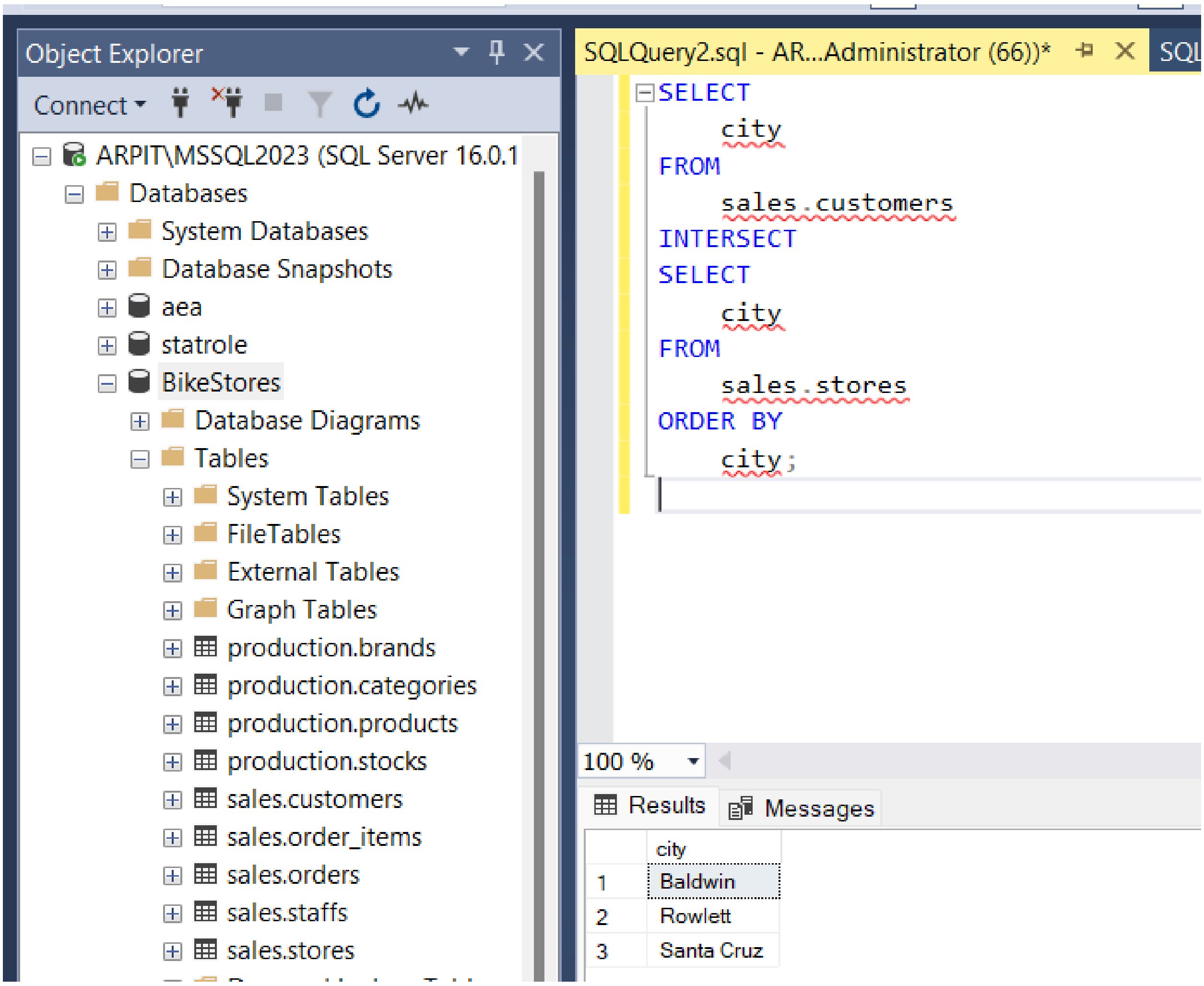
In results the expected table is listed.


SQLQuery1.sql - AR...Administrator (65))*
BikeStores Sample...Administrator (62))*
100 %
Results
Messages

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

|  | first_name | last_name |
|---|---|---|
| 697 | Justin | Newton |
| 698 | Justina | Jenkins |
| 699 | Justina | Jenkins |
| 700 | Justina | Long |
| 701 | Jutta | Everett |
| 702 | Kaci | Gallegos |
| 703 | Kaila | Walters |
| 704 | Kaley | Blanchard |
| 705 | Kali | Vargas |
| 706 | Kallie | Best |
| 707 | Kam | Wilder |
| 708 | Kami | Rios |
| 709 | Kandace | Ayers |
| 710 | Kandace | Giles |
| 711 | Kandace | Hughes |
| 712 | Kandi | Mcneil |

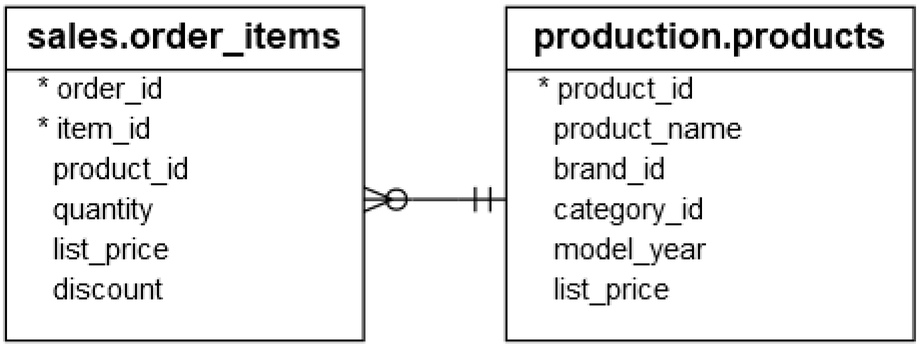
**Step 12**: Now we will find all cities of the customers and the second query finds the cities of the stores. The whole query, which uses INTERSECT, returns the common cities of customers and stores, which are the cities output by both input queries. we added the ORDER BY clause to the last query to sort the result set.



```
SELECT
    city
FROM
    sales.customers
INTERSECT
SELECT
    city
FROM
    sales.stores
ORDER BY
    city;
```

| | city |
|---|---|
| 1 | Baldwin |
| 2 | Rowlett |
| 3 | Santa Cruz |

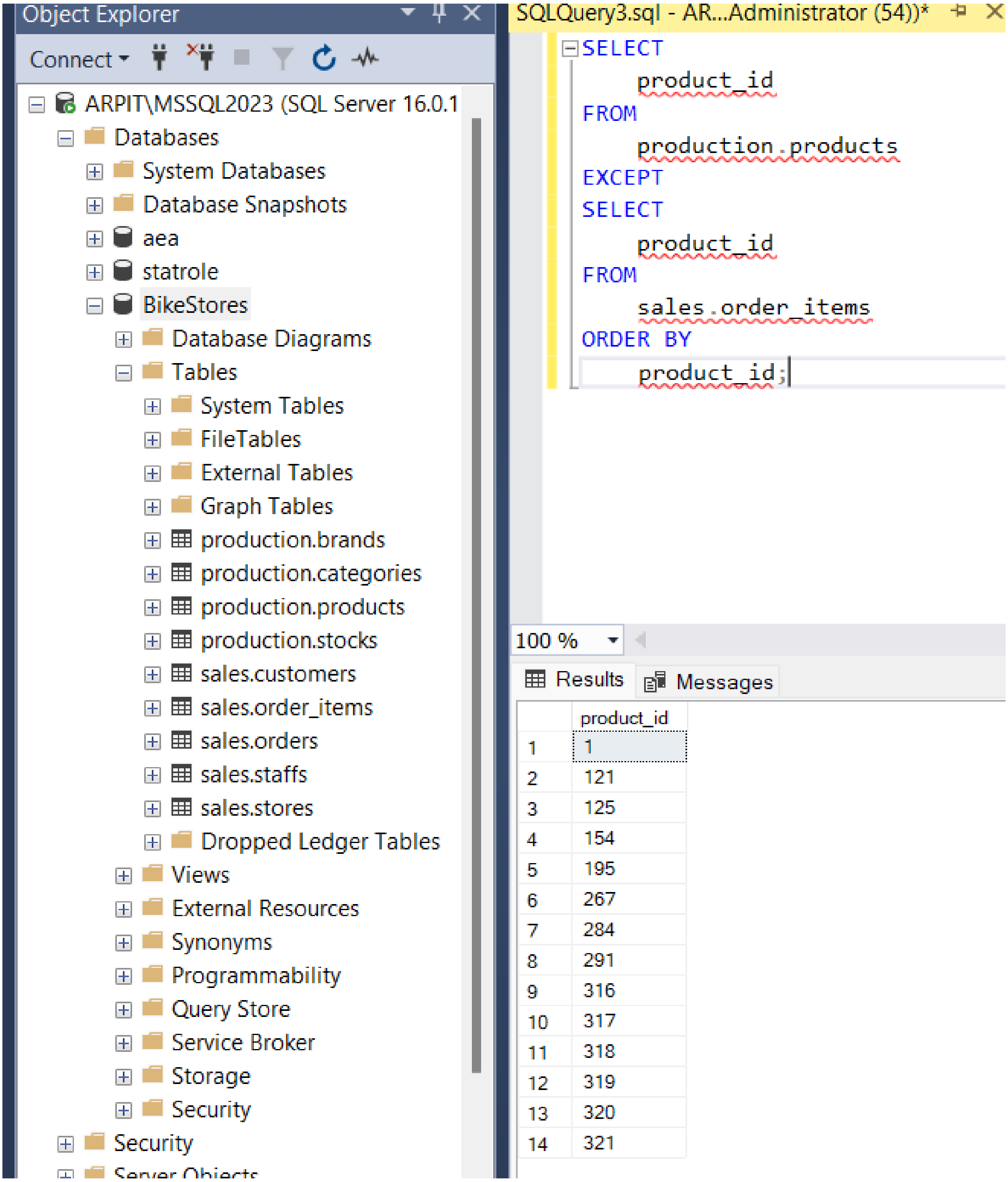
**Step 13**: Now we will find the products that had no sales and sorts the products by their id in ascending order:



| sales.order_items |
|---|
| * order_id |
| * item_id |
| product_id |
| quantity |
| list_price |
| discount |

| production.products |
|---|
| * product_id |
| product_name |
| brand_id |
| category_id |
| model_year |
| list_price |

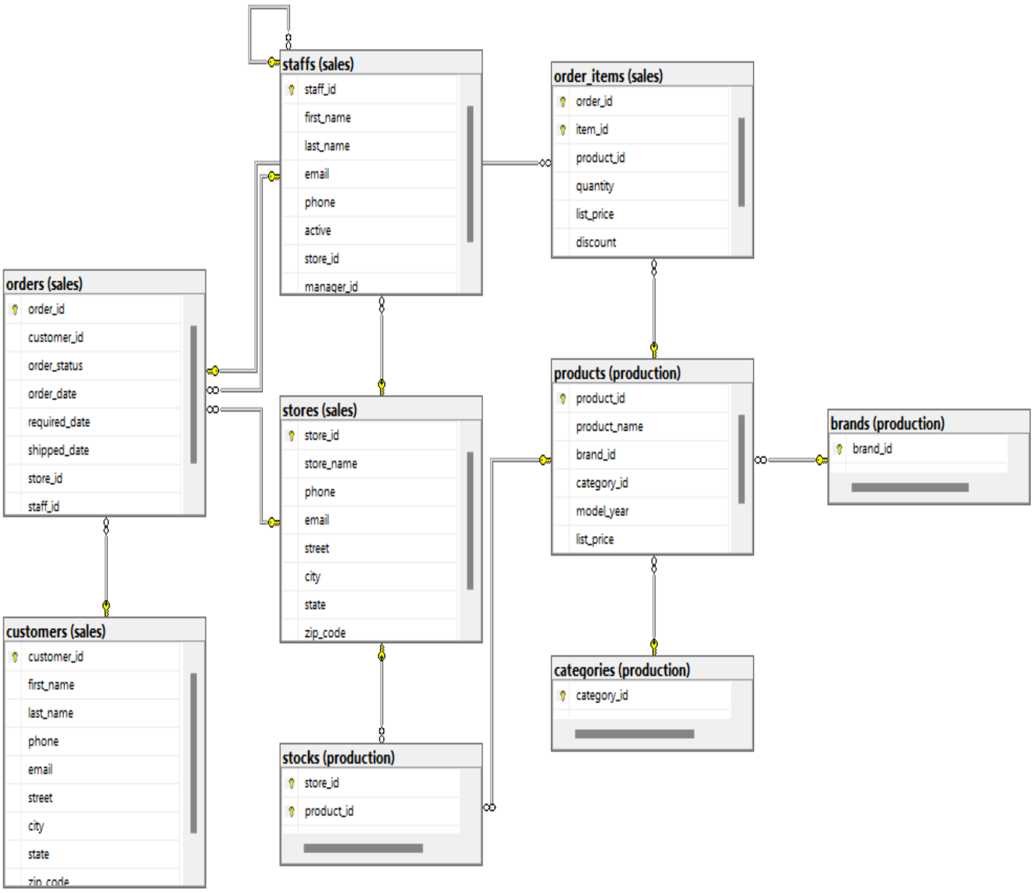
The first query returns all the products. The second query returns the products that have sales. Therefore, the result set includes only the products that have no sales.



```sql
SELECT
    product_id
FROM
    production.products
EXCEPT
SELECT
    product_id
FROM
    sales.order_items
ORDER BY
    product_id;
```

|  | product_id |
|---|---|
| 1 | 1 |
| 2 | 121 |
| 3 | 125 |
| 4 | 154 |
| 5 | 195 |
| 6 | 267 |
| 7 | 284 |
| 8 | 291 |
| 9 | 316 |
| 10 | 317 |
| 11 | 318 |
| 12 | 319 |
| 13 | 320 |
| 14 | 321 |

**Step 14:** Now we will draw the BikeStores database diagram.


staffs (sales)
staff_id
first_name
last_name
email
phone
active
store_id
manager_id
order_items (sales)
order_id
item_id
product_id
quantity
list_price
discount
orders (sales)
order_id
customer_id
order_status
order_date
required_date
shipped_date
store_id
staff_id
stores (sales)
store_id
store_name
phone
email
street
city
state
zip_code
products (production)
product_id
product_name
brand_id
category_id
model_year
list_price
brands (production)
brand_id
customers (sales)
customer_id
first_name
last_name
phone
email
street
city
state
zip_code
stocks (production)
store_id
product_id
categories (production)
category_id