Project Name: MotorsCertification
Tools Used: Microsoft SQL Server

Internship Program: Data Analysis and Machine Learning
Submitted By: Mythily Arunprasad
Submitted To: Tutor - Anu
Date: January 2025

MotorsCertification project was designed **for HerculesMotoCorp** to create a **robust database system**. **SQL** was used to **ensure smooth operations, accurate data, and automated processes**, making it easier for employees and stakeholders to access and **understand critical business information.**

Main Objective of the MotorsCertification Project:

The main objective is to **design and implement a comprehensive database system** for **HerculesMotoCorp**, a leading automotive retailer and garage handler. This **database will efficiently store, manage, and track customer and product data,** optimizing the company's operations, improving customer service, and **enabling the generation of insightful reports for internal stakeholders**.

Project Expectations:

1. **Create tables** with **primary key** and **foreign key** constraints.
2. **Insert records** into the **orderdetails, employees, payments, products, customers, offices, and orders** tables.
3. **Add comments** before each task to describe what is being performed.
4. **Delete unnecessary columns** in the productlines table.
5. **Verify data insertions and updates** using **SELECT statements.**
6. **Find the highest and lowest amounts** in the payments table.
7. **Get the unique count** of customerName from the customers table.
8. **Create a view** (cust_payment) combining customers and payments, then truncate and drop the view after use.
9. **Create a stored procedure** to display productLine for Classic Cars in the products table.
10. **Create a function** to return customers with a creditLimit less than 96800.
11. **Create a trigger** to store transaction records in the employee_transaction_log for new employee insertions.
12. **Create a trigger** to display customerNumber when the payment amount exceeds 10,000.

Project Outcome:

By following these steps, the project achieves a robust and organized database for **HerculesMotoCorp**. The key benefits include:

- **Data Integrity and Consistency**: With triggers, constraints, and views, the data remains consistent and valid across different operations.
- **Automation**: Tasks like logging transactions and triggering actions based on data inputs are automated, reducing the potential for human error.
- **Efficient Querying**: Optimized data retrieval using views, functions, and stored procedures improves performance and ease of access.
- **Scalability**: The design allows for easy modification and expansion as business requirements evolve.
- **Real-time Tracking**: Triggers and transaction logs enable real-time tracking of critical changes like employee additions and large payments.

This **structured approach** ensures the **system meets the business's operational, regulatory, and reporting needs effectively.**

Steps Followed TO Ensure Data Integrity:

- **Primary Key Constraints:**

Each table must have a primary key to uniquely identify each

- **Foreign Key Constraints:**

Foreign keys are used to maintain relationships between tables and ensure referential integrity.

- **Data Type Constraints:**

Defining appropriate data types and constraints for each column (e.g., NOT NULL, UNIQUE, CHECK) to prevent invalid data from being inserted.

- **Deleting Invalid Data:**

Removed orphaned records before adding foreign keys or performing updates to avoid data inconsistencies.

Example:

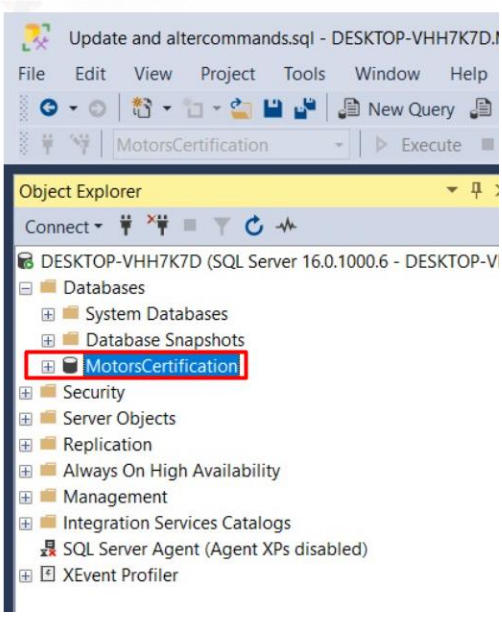
```
DELETE FROM payments

WHERE customerNumber NOT IN (SELECT customerNumber FROM customers);
```

Question 1:

Name the database as **MotorsCertification**. Design an ER model based on the following parameters:
Note: Build a ER diagram with proper entities, relationship etc.
Solution:

CREATE DATABASE MotorsCertification;

Purpose: This Query creates a **DB** Named **MotorsCertification.**
The database will serve as the foundation for **storing all data related to the HerculesMotoCorp business operations, including customer, employee, order, product, and payment information**.
By creating this database, you **are ensuring that all related data is stored in an organized and isolated environment, allowing for efficient management and querying**.



MotorCertification DB has been **successfully** instantiated within the database management system (DBMS). This output typically indicates the execution of a `CREATE DATABASE` statement, which

reserves space for the database schema and initializes the necessary system files for managing its data.

To facilitate the creation of all tables, the following methodical steps were carried out:

1. **Table Creation**: A new table was created with the appropriate structure to store the required data.
2. **Data Insertion**: Values were inserted into the newly created table to populate it with relevant information.
3. **Foreign Key Creation**: A foreign key constraint was established to maintain referential integrity between the related tables.
4. **Verification**: The foreign key constraint was successfully implemented, ensuring consistent and valid data relationships.
5. **Final Validation**: Queries were executed to verify the accuracy and integrity of the data, confirming the successful application of all changes.

Foreign Key Creation & Its Importance:

1. **Identify Tables**: Choose the parent table (with the primary key) and the child table (where the foreign key will be added).
2. **Add Foreign Key**: Use the ALTER TABLE statement to add the foreign key in the child table, linking it to the parent table.
3. **Enforce Integrity**: Ensure the foreign key column in the child table references a valid primary key in the parent table.

Question 1:

Design a table/database object named **orderdetails** with the following attributes/columns:

orderNumber int(), Primary Key

productCode varchar()

quantityOrdered int()

priceEach float

orderLineNumber smallint()

Foreign Key: orders (orderNumber → orderNumber) and products (productCode → productCode)

Index 1: PRIMARY, Type: BTREE, Unique Yes, Visible No, Columns orderNumber.
Index 2: productCode, Type: BTREE, Unique: No, Visible: No, Columns productCode.

Solution:

Orderdetails Table:

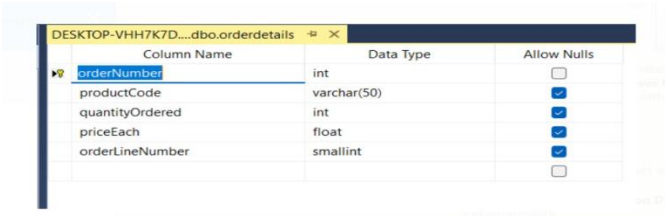
1.Table Creation: As part of the database implementation, the following SQL code was used to create the `orderdetails` table.

Code Used:

```
CREATE TABLE orderdetails(
    orderNumber INT PRIMARY KEY,
    productCode VARCHAR(50),
    quantityOrdered INT,
    priceEach FLOAT,
    orderLineNumber SMALLINT
       );

-- Indexes for orderdetails table
CREATE INDEX idx_orderNumber ON orderdetails(orderNumber);
```

```
CREATE INDEX idx_productCode ON orderdetails(productCode);
```

A table named orderdetails is created to store data related to order details. The table structure includes the following columns: **orderNumber** as the primary key**, productCode, quantityOrdered, priceEach, and orderLineNumber**, with appropriate data types and constraints for efficient data storage and integrity.



2.Data Insertion: The **INSERT INTO** command is used to **insert values into the orderdetails table**. This operation populates the table with specific rows of data, where each row represents an order detail, including values for columns like **orderNumber, productCode, quantityOrdered, priceEach, and orderLineNumber**. The **INSERT command** ensures that the table is populated with the necessary information for further **data processing and analysis**.

Code Used:

```
insert into orderdetails(orderNumber,productCode,quantityOrdered,priceEach,orderLineNumber)
values

	(10100,'S18_1749',30,'136.00',3),

	(10101,'S18_2248',50,'55.09',2),

	(10102,'S18_4409',22,'75.46',4),

	(10103,'S24_3969',49,'35.29',1),

	(10104,'S18_2325',25,'108.06',4),

	(10105,'S18_2795',26,'167.06',1),

	(10106,'S24_1937',45,'32.53',3),

	(10107,'S24_2022',46,'44.35',2),

	(10108,'S18_1342',39,'95.55',2),

	(10109,'S18_1367',41,'43.13',1),

	(10110,'S10_1949',26,'214.30',11),

	(10111,'S10_4962',42,'119.67',4),

	(10112,'S12_1666',27,'121.64',8),

	(10113,'S18_1097',35,'94.50',10),

	(10114,'S18_2432',22,'58.34',2),

	(10115,'S18_2949',27,'92.19',12),

	(10116,'S18_2957',35,'61.84',14),

	(10117,'S18_3136',25,'86.92',13),

	(10118,'S18_3320',46,'86.31',16),

	(10119,'S18_4600',36,'98.07',5);
```

DESKTOP-VHH7K7D....dbo.orderdetails | DESKTOP-VHH7K7D....dbo.orderdetails

orderNum...	productCo...	quantityOr...	priceEach	orderLineN...
10100	S18_1749	30	136	3
10101	S18_2248	50	55.09	2
10102	S18_4409	22	75.46	4
10103	S24_3969	49	35.29	1
10104	S18_2325	25	108.06	4
10105	S18_2795	26	167.06	1
10106	S24_1937	45	32.53	3
10107	S24_2022	46	44.35	2
10108	S18_1342	39	95.55	2
10109	S18_1367	41	43.13	1
10110	S10_1949	26	214.3	11
10111	S10_4962	42	119.67	4
10112	S12_1666	27	121.64	8
10113	S18_1097	35	94.5	10
10114	S18_2432	22	58.34	2
10115	S18_2949	27	92.19	12
10116	S18_2957	35	61.84	14
10117	S18_3136	25	86.92	13
10118	S18_3320	46	86.31	16
10119	S18_4600	36	98.07	5
NULL	NULL	NULL	NULL	NULL

3. Foreign Key Creation for orderdetails Table and Steps Involved:

Foreign Key Creation Using ALTER Statement:

1. The **ALTER TABLE** statement is used to add a foreign key constraint between the **orderdetails table and the orders table.** This ensures that each orderNumber in orderdetails corresponds to a valid orderNumber in the orders table, enforcing referential integrity.
2. Similarly, another ALTER TABLE statement is used to add a foreign key constraint between the **orderdetails table and the products table**. This ensures that each productCode in orderdetails corresponds to a valid productCode in the products table, maintaining consistency between the tables.

These operations are necessary to ensure that data in the orderdetails table is consistent and only references valid records from the orders and products tables.

```
ALTER TABLE orderdetails
ADD CONSTRAINT FK_orderdetails_products
FOREIGN KEY (productCode) REFERENCES products(productCode);

SELECT DISTINCT od.productCode
FROM orderdetails od
LEFT JOIN products p ON od.productCode = p.productCode
WHERE p.productCode IS NULL;

SELECT DISTINCT od.orderNumber
FROM orderdetails od
LEFT JOIN orders o ON od.orderNumber = o.orderNumber
WHERE o.orderNumber IS NULL;

INSERT INTO products (productCode, productName)
SELECT DISTINCT od.productCode, 'Unknown Product'
FROM orderdetails od
```

Messages

```
Msg 547, Level 16, State 0, Line 23
The ALTER TABLE statement conflicted with the FOREIGN KEY constraint "FK_orderdetails_products". The conflict occurred in database "MotorsCertification", table "dbo.products", column 'productCode'.

Completion time: 2025-01-23T14:12:03.6439205+05:30
```

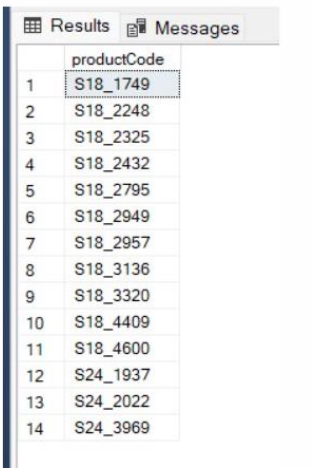
SELECT DISTINCT od.productCode

FROM orderdetails od

LEFT JOIN products p ON od.productCode = p.productCode

WHERE p.productCode IS NULL;

Purpose: The above query is used to identify productCode values from the orderdetails table that **do not exist** in the products table.



Results Messages

	productCode
1	S18_1749
2	S18_2248
3	S18_2325
4	S18_2432
5	S18_2795
6	S18_2949
7	S18_2957
8	S18_3136
9	S18_3320
10	S18_4409
11	S18_4600
12	S24_1937
13	S24_2022
14	S24_3969

Code Used:

```
SELECT DISTINCT od.orderNumber
FROM orderdetails od
LEFT JOIN orders o ON od.orderNumber = o.orderNumber
WHERE o.orderNumber IS NULL;
```

Purpose: This query is used to identify orderNumber values from the orderdetails table that **do not exist** in the orders table.

Results Messages

orderNumber

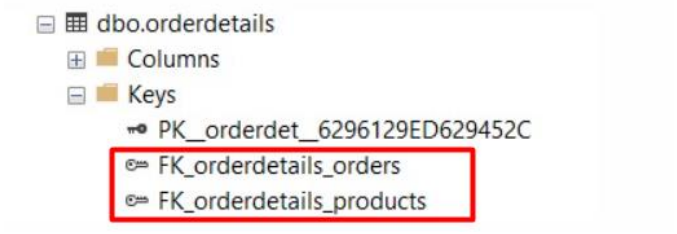
Code Used:

```
INSERT INTO products (productCode, productName)
SELECT DISTINCT od.productCode, 'Unknown Product'
FROM orderdetails od
LEFT JOIN products p ON od.productCode = p.productCode
WHERE p.productCode IS NULL;
```

Purpose: This query is used to **insert productCode** values from the orderdetails table that do not exist in the products table, and assigns them a **default productName of 'Unknown Product'**.

4.Verification: Re-execute the ALTER TABLE command to verify the changes applied and ensure the modifications are in effect.

```
ALTER TABLE orderdetails
ADD CONSTRAINT FK_orderdetails_products
FOREIGN KEY (productCode) REFERENCES products(productCode);
```



5. **Final Validation**: The **foreign key constraint** has been **successfully created**, ensuring **referential integrity between the related tables.**

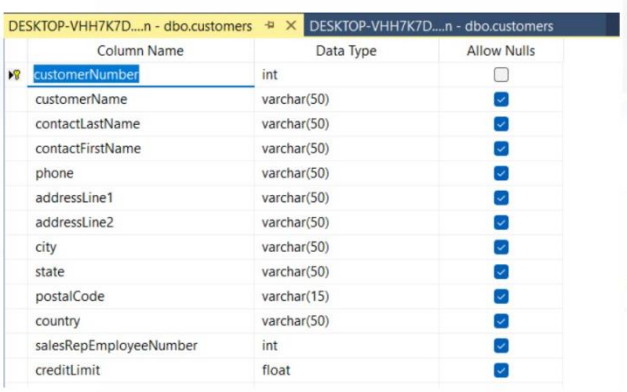
Customers table:

Table Creation: As part of the database implementation, the following SQL code was used to create the `customers` table.

Code Used:

```
    CREATE TABLE customers (
    customerNumber INT PRIMARY KEY,
    customerName VARCHAR(50),
    contactLastName VARCHAR(50),
    contactFirstName VARCHAR(50),
    phone VARCHAR(50),
    addressLine1 VARCHAR(50),
    addressLine2 VARCHAR(50),
    city VARCHAR(50),
    state VARCHAR(50),
    postalCode VARCHAR(15),
    country VARCHAR(50),
    salesRepEmployeeNumber INT,
    creditLimit FLOAT
       );

-- Index on customerNumber (primary key already done)
CREATE INDEX idx_customerNumber ON customers(customerNumber);
```

A table named customers is created to store data related to customer information. The table structure includes the following columns: **customerNumber** as the **primary key**, **customerName, contactLastName, contactFirstName, phone, addressLine1, addressLine2, city, state, postalCode, country, salesRepEmployeeNumber, and creditLimit**. These columns are defined with appropriate constraints to ensure efficient data storage and integrity.



2.**Data Insertion**: The **INSERT INTO** command is used to **insert values into the customers table**. This operation populates the table with specific rows of data, where each row represents a customer, including values for columns like **customerNumber, customerName, contactLastName, contactFirstName, phone, addressLine1, addressLine2, city, state, postalCode, country, salesRepEmployeeNumber, and creditLimit**. The **INSERT** command ensures that the table is populated with the necessary customer information for further **data processing and analysis**.

Code Used:

```
INSERT INTO customers VALUES

       (103,'Atelier graphique','Schmitt','Carine ','40.32.2555','54, rue
Royale',NULL,'Nantes',NULL,'44000','France',1370,'21000.00'),

       (112,'Signal Gift Stores','King','Jean','7025551838','8489 Strong St.',NULL,'Las
Vegas','NV','83030','USA',1166,'71800.00'),

       (114,'Australian Collectors, Co.','Ferguson','Peter','03 9520 4555','636 St Kilda
Road','Level 3','Melbourne','Victoria','3004','Australia',1611,'117300.00'),

       (119,'La Rochelle Gifts','Labrune','Janine ','40.67.8555','67, rue des Cinquante
Otages',NULL,'Nantes',NULL,'44000','France',1370,'118200.00'),

       (121,'Baane Mini Imports','Bergulfsen','Jonas ','07-98 9555','Erling Skakkes gate
78',NULL,'Stavern',NULL,'4110','Norway',1504,'81700.00'),
```

```
(124,'Mini Gifts Distributors Ltd.','Nelson','Susan','4155551450','5677 Strong
St.',NULL,'San Rafael','CA','97562','USA',1165,'210500.00'),

(125,'Havel & Zbyszek Co','Piestrzeniewicz','Zbyszek ','(26) 642-7555','ul. Filtrowa
68',NULL,'Warszawa',NULL,'01-012','Poland',NULL,'0.00'),

(128,'Blauer See Auto, Co.','Keitel','Roland','+49 69 66 90 2555','Lyonerstr.
34',NULL,'Frankfurt',NULL,'60528','Germany',1504,'59700.00'),

(129,'Mini Wheels Co.','Murphy','Julie','6505555787','5557 North Pendale
Street',NULL,'San Francisco','CA','94217','USA',1165,'64600.00'),

(131,'Land of Toys Inc.','Lee','Kwai','2125557818','897 Long Airport
Avenue',NULL,'NYC','NY','10022','USA',1323,'114900.00'),

(141,'Euro+ Shopping Channel','Freyre','Diego ','(91) 555 94 44','C/ Moralzarzal,
86',NULL,'Madrid',NULL,'28034','Spain',1370,'227600.00'),

(144,'Volvo Model Replicas, Co','Berglund','Christina ','0921-12 3555','Berguvsvägen
8',NULL,'Luleå',NULL,'S-958 22','Sweden',1504,'53100.00'),

(145,'Danish Wholesale Imports','Petersen','Jytte ','31 12 3555','Vinbæltet
34',NULL,'Kobenhavn',NULL,'1734','Denmark',1401,'83400.00'),

(146,'Saveley & Henriot, Co.','Saveley','Mary ','78.32.5555','2, rue du
Commerce',NULL,'Lyon',NULL,'69004','France',1337,'123900.00'),

(148,'Dragon Souveniers, Ltd.','Natividad','Eric','+65 221 7555','Bronz Sok.','Bronz Apt.
3/6 Tesvikiye','Singapore',NULL,'079903','Singapore',1621,'103800.00'),

(151,'Muscle Machine Inc','Young','Jeff','2125557413','4092 Furth Circle','Suite
400','NYC','NY','10022','USA',1286,'138500.00'),

(157,'Diecast Classics Inc.','Leong','Kelvin','2155551555','7586 Pompton
St.',NULL,'Allentown','PA','70267','USA',1216,'100600.00'),

(161,'Technics Stores Inc.','Hashimoto','Juri','6505556809','9408 Furth
Circle',NULL,'Burlingame','CA','94217','USA',1165,'84600.00'),

(166,'Handji Gifts& Co','Victorino','Wendy','+65 224 1555','106 Linden Road Sandown','2nd
Floor','Singapore',NULL,'069045','Singapore',1612,'97900.00'),

(167,'Herkku Gifts','Oeztan','Veysel','+47 2267 3215','Brehmen St. 121','PR 334
Sentrum','Bergen',NULL,'N 5804','Norway  ',1504,'96800.00');
```

DESKTOP-VHH7K7D....n - dbo.customers | DESKTOP-VHH7K7D....dbo.orderdetails | DESKTOP-VHH7K7D....dbo.orderdetails | MotorsCertificatio...K7D\aaruthraa (55))

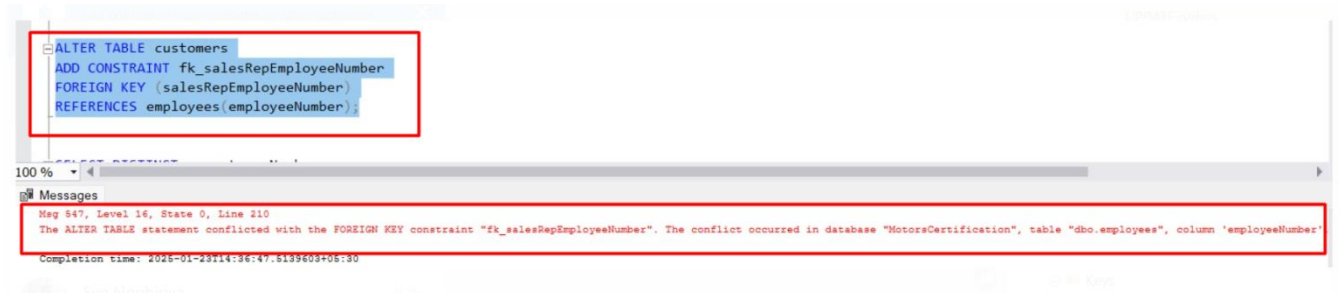
customerN...	customerN...	contactLast...	contactFirst...	phone	addressLin...	addressLin...	city	state	postalCode	country	salesRepE...	creditLimit
103	Atelier grap...	Schmitt	Carine	40.32.2555	54, rue Roya...	NULL	Nantes	NULL	44000	France	1370	21000
112	Signal Gift S...	King	Jean	7025551838	8489 Strong...	NULL	Las Vegas	NV	83030	USA	1166	71800
114	Australian C...	Ferguson	Peter	03 9520 4555	636 St Kilda...	Level 3	Melbourne	Victoria	3004	Australia	1611	117300
119	La Rochelle ...	Labrune	Janine	40.67.8555	67, rue des ...	NULL	Nantes	NULL	44000	France	1370	118200
121	Baane Mini ...	Bergulfsen	Jonas	07-98 9555	Erling Skakk...	NULL	Stavern	NULL	4110	Norway	1504	81700
124	Mini Gifts D...	Nelson	Susan	4155551450	5677 Strong...	NULL	San Rafael	CA	97562	USA	1165	210500
125	Havel & Zb...	Piestrzenie...	Zbyszek	(26) 642-75...	ul. Filtrowa ...	NULL	Warszawa	NULL	01-012	Poland	NULL	0
128	Blauer See ...	Keitel	Roland	+49 69 66 9...	Lyonerstr. 34	NULL	Frankfurt	NULL	60528	Germany	1504	59700
129	Mini Wheels...	Murphy	Julie	6505555787	5557 North ...	NULL	San Francisco	CA	94217	USA	1165	64600
131	Land of Toy...	Lee	Kwai	2125557818	897 Long Ai...	NULL	NYC	NY	10022	USA	1323	114900
141	Euro+ Shop...	Freyre	Diego	(91) 555 94 ...	C/ Moralzar...	NULL	Madrid	NULL	28034	Spain	1370	227600
144	Volvo Mode...	Berglund	Christina	0921-12 3555	Berguvsväg...	NULL	Luleå	NULL	S-958 22	Sweden	1504	53100
145	Danish Who...	Petersen	Jytte	31 12 3555	Vinbæltet 34	NULL	Kobenhavn	NULL	1734	Denmark	1401	83400
146	Saveley & H...	Saveley	Mary	78.32.5555	2, rue du Co...	NULL	Lyon	NULL	69004	France	1337	123900
148	Dragon Sou...	Natividad	Eric	+65 221 7555	Bronz Sok.	Bronz Apt. 3...	Singapore	NULL	079903	Singapore	1621	103800
151	Muscle Mac...	Young	Jeff	2125557413	4092 Furth ...	Suite 400	NYC	NY	10022	USA	1286	138500
157	Diecast Clas...	Leong	Kelvin	2155551555	7586 Pompt...	NULL	Allentown	PA	70267	USA	1216	100600
161	Technics Sto...	Hashimoto	Juri	6505556809	9408 Furth ...	NULL	Burlingame	CA	94217	USA	1165	84600
166	Handji Gifts...	Victorino	Wendy	+65 224 1555	106 Linden ...	2nd Floor	Singapore	NULL	069045	Singapore	1612	97900
167	Herkku Gifts	Oeztan	Veysel	+47 2267 3...	Brehmen St....	PR 334 Sent...	Bergen	NULL	N 5804	Norway	1504	96800
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

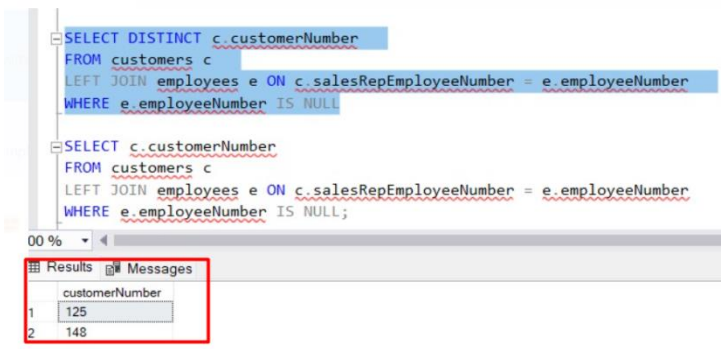
3.Foreign Key Creation for Customers Table and Steps Involved:

Foreign Key Creation Using ALTER Statement for Customers Table:

The **ALTER TABLE** statement is used to add a **foreign key constraint** between the **customers table and the employees table.** This ensures that each **salesRepEmployeeNumber in the customers table corresponds to a valid employeeNumber in the employees table**, enforcing referential integrity.

This operation is necessary to ensure that data in the customers table is consistent and only references valid records from the employees table. By linking these tables through a **foreign key, we maintain data accuracy and prevent invalid data entry.**



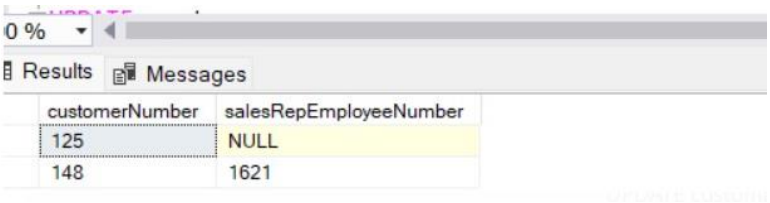


Purpose: This query is used to identify **customerNumber values from the customers table that do not have a valid salesRepEmployeeNumber** corresponding to any `employeeNumber` in the `employees` table.

Code Used.

```
SELECT customerNumber, salesRepEmployeeNumber

FROM customers
WHERE customerNumber IN (125, 148);
UPDATE customers
SET salesRepEmployeeNumber = 1002
WHERE customerNumber = 125;
UPDATE customers
SET salesRepEmployeeNumber = NULL
WHERE customerNumber = 125;
UPDATE customers
SET salesRepEmployeeNumber = NULL
WHERE salesRepEmployeeNumber NOT IN (SELECT employeeNumber FROM employees);
```

Purpose:This set of queries is used to **update the salesRepEmployeeNumber** in the customers table based on certain conditions and to **ensure that only valid employee numbers are assigned to customers**.



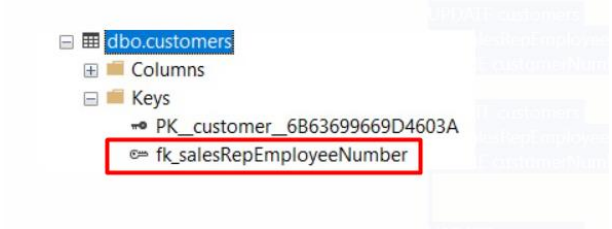
4. Verification: Re-execute the ALTER TABLE command to verify the changes applied and ensure the modifications are in effect.

ALTER TABLE customers

ADD CONSTRAINT fk_salesRepEmployeeNumber

FOREIGN KEY (salesRepEmployeeNumber)

REFERENCES employees(employeeNumber);



5. **Final Validation**: The **foreign key constraint** has been **successfully created**, ensuring **referential integrity between the related tables**

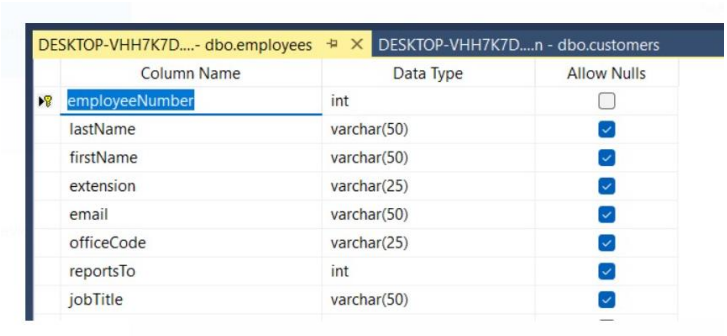
Employees table:

1.Table Creation: As part of the database implementation, the following SQL code was used to create the `employees` table.

Code Used:

```
CREATE TABLE employees (
    employeeNumber INT PRIMARY KEY,
    lastName VARCHAR(50),
    firstName VARCHAR(50),
    extension VARCHAR(25),
    email VARCHAR(50),
    officeCode VARCHAR(25),
    reportsTo INT,
    jobTitle VARCHAR(50)
       );

-- Indexes for employees table
CREATE INDEX idx_employeeNumber ON employees(employeeNumber);
CREATE INDEX idx_reportsTo ON employees(reportsTo);

CREATE INDEX idx_officeCode ON employees(officeCode);
```

A table named **`employees`** is created to store **data related to employee information**. The table structure includes the following columns: **`employeeNumber`** as the **primary key, `lastName`, `firstName`, `extension`, `email`, `officeCode`, `reportsTo`, and `jobTitle`.** These columns are defined with appropriate data types and constraints for efficient data storage and integrity. The **`employeeNumber` serves as a unique identifier for each employee,** and **the `reportsTo` column is used to establish a relationship with the `employeeNumber` of other employees**, **facilitating hierarchical reporting structures within the organization.**



2.Data Insertion: The **`INSERT INTO`** command is used to insert values into the **`employees`** table. This operation populates the table with specific rows of data, where each row represents an employee, including values for columns like **`employeeNumber`**, **`lastName`**, **`firstName`**, **`extension`**, **`email`**, **`officeCode`**, **`reportsTo`**, **and** **`jobTitle`**. The **`INSERT`** command ensures that the table is populated with the necessary employee information, facilitating the management and analysis of employee data within the system.

Code Used:

```
insert into employees(employeeNumber,lastName,firstName,extension
,email,officeCode,reportsTo,jobTitle)
values

	(1002,'Murphy','Diane','x5800','dmurphy@classicmodelcars.com','1',NULL,'President'),

	(1056,'Patterson','Mary','x4611','mpatterso@classicmodelcars.com','1',1002,'VP Sales'),

	(1076,'Firrelli','Jeff','x9273','jfirrelli@classicmodelcars.com','1',1002,'VP
Marketing'),

	(1088,'Patterson','William','x4871','wpatterson@classicmodelcars.com','6',1056,'Sales
Manager (APAC)'),

	(1102,'Bondur','Gerard','x5408','gbondur@classicmodelcars.com','4',1056,'Sale Manager
(EMEA)'),

	(1143,'Bow','Anthony','x5428','abow@classicmodelcars.com','1',1056,'Sales Manager (NA)'),

	(1165,'Jennings','Leslie','x3291','ljennings@classicmodelcars.com','1',1143,'Sales Rep'),

	(1166,'Thompson','Leslie','x4065','lthompson@classicmodelcars.com','1',1143,'Sales Rep'),

	(1188,'Firrelli','Julie','x2173','jfirrelli@classicmodelcars.com','2',1143,'Sales Rep'),

	(1216,'Patterson','Steve','x4334','spatterson@classicmodelcars.com','2',1143,'Sales
Rep'),

	(1286,'Tseng','Foon Yue','x2248','ftseng@classicmodelcars.com','3',1143,'Sales Rep'),

	(1323,'Vanauf','George','x4102','gvanauf@classicmodelcars.com','3',1143,'Sales Rep'),

	(1337,'Bondur','Loui','x6493','lbondur@classicmodelcars.com','4',1102,'Sales Rep'),

	(1370,'Hernandez','Gerard','x2028','ghernande@classicmodelcars.com','4',1102,'Sales
Rep'),

	(1401,'Castillo','Pamela','x2759','pcastillo@classicmodelcars.com','4',1102,'Sales Rep'),

	(1501,'Bott','Larry','x2311','lbott@classicmodelcars.com','7',1102,'Sales Rep'),

	(1504,'Jones','Barry','x102','bjones@classicmodelcars.com','7',1102,'Sales Rep'),

	(1611,'Fixter','Andy','x101','afixter@classicmodelcars.com','6',1088,'Sales Rep'),

	(1612,'Marsh','Peter','x102','pmarsh@classicmodelcars.com','6',1088,'Sales Rep'),

	(1619,'King','Tom','x103','tking@classicmodelcars.com','6',1088,'Sales Rep');
```

DESKTOP-VHH7K7D....- dbo.employees | MotorsCertificatio...K7D\aaruthraa (55)) | Update and alterco...7D\aaruthra

employee...	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1002	Murphy	Diane	x5800	dmurphy@c...	1	NULL	President
1056	Patterson	Mary	x4611	mpatterso...	1	1002	VP Sales
1076	Firrelli	Jeff	x9273	jfirrelli@cla...	1	1002	VP Marketing
1088	Patterson	William	x4871	wpatterson...	6	1056	Sales Mana...
1102	Bondur	Gerard	x5408	gbondur@c...	4	1056	Sale Manag...
1143	Bow	Anthony	x5428	abow@clas...	1	1056	Sales Mana...
1165	Jennings	Leslie	x3291	ljennings@c...	1	1143	Sales Rep
1166	Thompson	Leslie	x4065	lthompson...	1	1143	Sales Rep
1188	Firrelli	Julie	x2173	jfirrelli@cla...	2	1143	Sales Rep
1216	Patterson	Steve	x4334	spatterson...	2	1143	Sales Rep
1286	Tseng	Foon Yue	x2248	ftseng@clas...	3	1143	Sales Rep
1323	Vanauf	George	x4102	gvanauf@cl...	3	1143	Sales Rep
1337	Bondur	Loui	x6493	lbondur@cl...	4	1102	Sales Rep
1370	Hernandez	Gerard	x2028	ghernande...	4	1102	Sales Rep
1401	Castillo	Pamela	x2759	pcastillo@cl...	4	1102	Sales Rep
1501	Bott	Larry	x2311	lbott@classi...	7	1102	Sales Rep
1504	Jones	Barry	x102	bjones@cla...	7	1102	Sales Rep
1611	Fixter	Andy	x101	afixter@clas...	6	1088	Sales Rep
1612	Marsh	Peter	x102	pmarsh@cl...	6	1088	Sales Rep
1619	King	Tom	x103	tking@class...	6	1088	Sales Rep
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3.Foreign Key Creation for employees Table and Steps Involved:

Foreign Key for Self-Referencing Relationship: This foreign key ensures that the **reportsTo** field in the **employees** table refers to a valid **employeeNumber** within the **same table**. The **ALTER TABLE command is used to add this foreign key.**

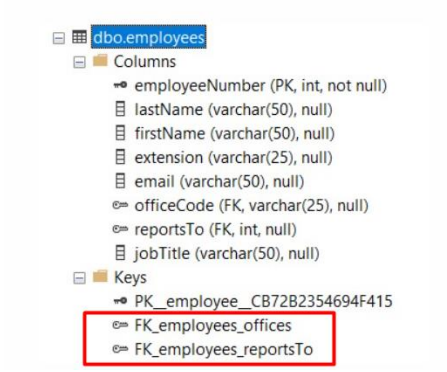
Code Used:

```
ALTER TABLE employees
ADD CONSTRAINT FK_employees_reportsTo
 FOREIGN KEY (reportsTo) REFERENCES offices(employeeNumber);
```

Foreign Key for Linking to the `offices` Table: This foreign key ensures that each `officeCode` in the `employees` table corresponds to a valid `officeCode` **in the** `offices` **table**, maintaining **consistency between these two tables.**

Code Used:

```
ALTER TABLE employees
ADD CONSTRAINT FK_employees_offices
FOREIGN KEY (officeCode) REFERENCES offices(officeCode);
```



5. Final Validation: The **foreign key constraint** has been **successfully created**, ensuring **referential integrity between the related tables**. **The referential integrity** between the **employees** table and itself (for **reportsTo**) as well as the relationship with the **offices** table (for **officeCode)** is now enforced. You can now **rely on these relationships for data consistency and integrity across your database.**

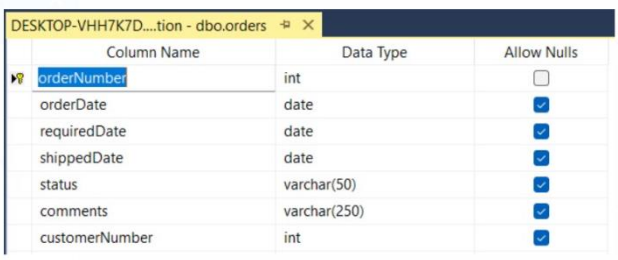
Orders table:

1.Table Creation: As part of the database implementation, the following SQL code was used to create the `orders` table.

Code Used:

```
CREATE TABLE orders (
orderNumber INT PRIMARY KEY,
orderDate DATE,
requiredDate DATE,
shippedDate DATE,
status VARCHAR(50),
comments VARCHAR(250),
customerNumber INT
);
```

A table named **orders** is created to **store data related to orders**. The table structure includes the following columns**: orderNumber as the primary key, requiredDate, shippedDate, status, comments, and customerNumber,** with appropriate data types and constraints for efficient data storage and integrity.

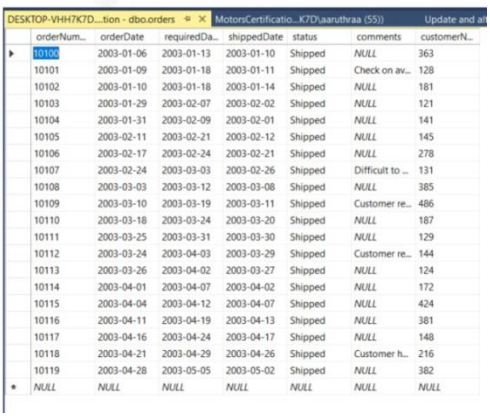


2.Data Insertion: The **INSERT INTO** command is used to **insert values into the orders table**. This operation populates the table with specific rows of data, where each row represents an order, including values for columns like **orderNumber, requiredDate, shippedDate, status, comments, and customerNumber**. The INSERT command ensures that the table is populated with the necessary information for further data processing and analysis.

Code Used:

```
insert into
orders(orderNumber,orderDate,requiredDate,shippedDate,status,comments,customerNumber)
values
(10100,'2003-01-06','2003-01-13','2003-01-10','Shipped',NULL,363),

(10101,'2003-01-09','2003-01-18','2003-01-11','Shipped','Check on availability.',128),

(10102,'2003-01-10','2003-01-18','2003-01-14','Shipped',NULL,181),

(10103,'2003-01-29','2003-02-07','2003-02-02','Shipped',NULL,121),

(10104,'2003-01-31','2003-02-09','2003-02-01','Shipped',NULL,141),

(10105,'2003-02-11','2003-02-21','2003-02-12','Shipped',NULL,145),

(10106,'2003-02-17','2003-02-24','2003-02-21','Shipped',NULL,278),

(10107,'2003-02-24','2003-03-03','2003-02-26','Shipped','Difficult to negotiate with customer.
We need more marketing materials',131),

(10108,'2003-03-03','2003-03-12','2003-03-08','Shipped',NULL,385),

(10109,'2003-03-10','2003-03-19','2003-03-11','Shipped','Customer requested that FedEx Ground is
used for this shipping',486),

(10110,'2003-03-18','2003-03-24','2003-03-20','Shipped',NULL,187),

(10111,'2003-03-25','2003-03-31','2003-03-30','Shipped',NULL,129),

(10112,'2003-03-24','2003-04-03','2003-03-29','Shipped','Customer requested that ad materials
(such as posters, pamphlets) be included in the shippment',144),

(10113,'2003-03-26','2003-04-02','2003-03-27','Shipped',NULL,124),

(10114,'2003-04-01','2003-04-07','2003-04-02','Shipped',NULL,172),

(10115,'2003-04-04','2003-04-12','2003-04-07','Shipped',NULL,424),
```

```
(10116,'2003-04-11','2003-04-19','2003-04-13','Shipped',NULL,381),

(10117,'2003-04-16','2003-04-24','2003-04-17','Shipped',NULL,148),

(10118,'2003-04-21','2003-04-29','2003-04-26','Shipped','Customer has worked with some of our
vendors in the past and is aware of their MSRP',216),

(10119,'2003-04-28','2003-05-05','2003-05-02','Shipped',NULL,382);
```

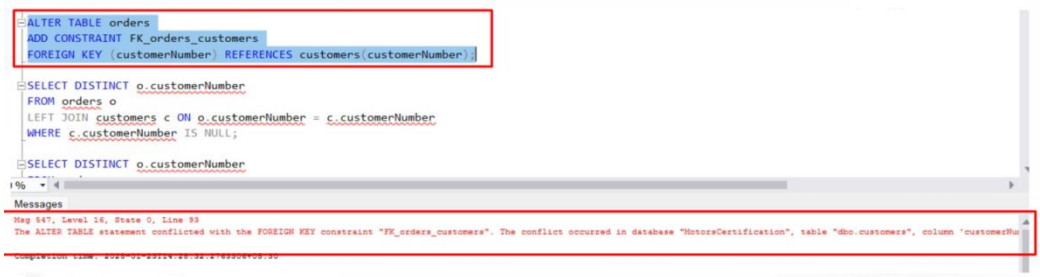


orderNum...	orderDate	requiredDa...	shippedDate	status	comments	customerN...
10100	2003-01-06	2003-01-13	2003-01-10	Shipped	NULL	363
10101	2003-01-09	2003-01-18	2003-01-11	Shipped	Check on av...	128
10102	2003-01-10	2003-01-18	2003-01-14	Shipped	NULL	181
10103	2003-01-29	2003-02-07	2003-02-02	Shipped	NULL	121
10104	2003-01-31	2003-02-09	2003-02-01	Shipped	NULL	141
10105	2003-02-11	2003-02-21	2003-02-12	Shipped	NULL	145
10106	2003-02-17	2003-02-24	2003-02-21	Shipped	NULL	278
10107	2003-02-24	2003-03-03	2003-02-26	Shipped	Difficult to ...	131
10108	2003-03-03	2003-03-12	2003-03-08	Shipped	NULL	385
10109	2003-03-10	2003-03-19	2003-03-11	Shipped	Customer re...	486
10110	2003-03-18	2003-03-24	2003-03-20	Shipped	NULL	187
10111	2003-03-25	2003-03-31	2003-03-30	Shipped	NULL	129
10112	2003-03-24	2003-04-03	2003-03-29	Shipped	Customer re...	144
10113	2003-03-26	2003-04-02	2003-03-27	Shipped	NULL	124
10114	2003-04-01	2003-04-07	2003-04-02	Shipped	NULL	172
10115	2003-04-04	2003-04-12	2003-04-07	Shipped	NULL	424
10116	2003-04-11	2003-04-19	2003-04-13	Shipped	NULL	381
10117	2003-04-16	2003-04-24	2003-04-17	Shipped	NULL	148
10118	2003-04-21	2003-04-29	2003-04-26	Shipped	Customer h...	216
10119	2003-04-28	2003-05-05	2003-05-02	Shipped	NULL	382
NULL	NULL	NULL	NULL	NULL	NULL	NULL

3.Foreign Key Creation for orders **Table and Steps Involved**:

Foreign Key Creation Using ALTER Statement:

The **ALTER TABLE statement** is used to add a **foreign key constraint** between the **orders table and the customers table.** This ensures that **each customerNumber in the orders table corresponds to a valid customerNumber in the customers table,** enforcing referential integrity**.**



```
ALTER TABLE orders
ADD CONSTRAINT FK_orders_customers
FOREIGN KEY (customerNumber) REFERENCES customers(customerNumber);

SELECT DISTINCT o.customerNumber
FROM orders o
LEFT JOIN customers c ON o.customerNumber = c.customerNumber
WHERE c.customerNumber IS NULL;

SELECT DISTINCT o.customerNumber
```

Messages

Msg 547, Level 16, State 0, Line 99
The ALTER TABLE statement conflicted with the FOREIGN KEY constraint "FK_orders_customers". The conflict occurred in database "MotorsCertification", table "dbo.customers", column 'customerNu

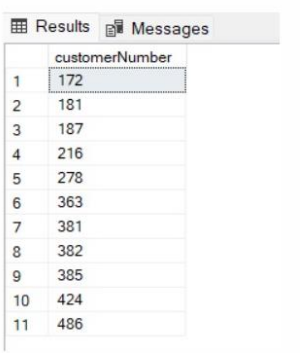
SELECT DISTINCT o.customerNumber

FROM orders o

LEFT JOIN customers c ON o.customerNumber = c.customerNumber

WHERE c.customerNumber IS NULL;

Purpose: This query **returns the customerNumber of customers** who have **placed an order but do not exist in the customers table.** This could happen if there are records in the orders table with a customerNumber that doesn't match any customerNumber in the customers table.

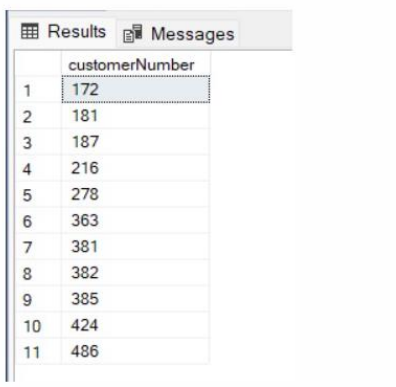


Results | Messages

	customerNumber
1	172
2	181
3	187
4	216
5	278
6	363
7	381
8	382
9	385
10	424
11	486

```
SELECT DISTINCT o.customerNumber
FROM orders o
LEFT JOIN customers c ON o.customerNumber = c.customerNumber
WHERE c.customerNumber IS NULL;
```

Purpose: This query helps to **find** customer numbers **in the orders table that do not have a corresponding entry** in the customers table.



Results | Messages

	customerNumber
1	172
2	181
3	187
4	216
5	278
6	363
7	381
8	382
9	385
10	424
11	486

4.Verification: Re-execute the ALTER TABLE command to verify the changes applied and ensure the modifications are in effect.

Code Used:

```
ALTER TABLE orders
ADD CONSTRAINT fk_orders_customer
FOREIGN KEY (customerNumber) REFERENCES customers(customerNumber);
```

5. Final Validation: The **foreign key constraint** has been **successfully created**, ensuring **referential integrity between the related tables.**

Offices:

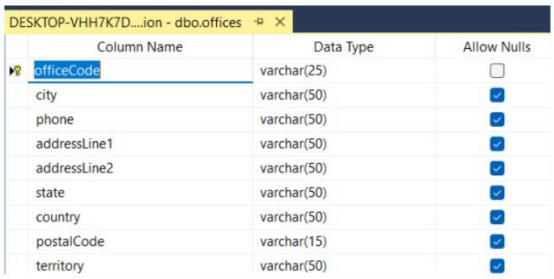
1.Table Creation: The `offices` table contains details about the organization's office locations, including city, state, and country. Below is the SQL code used to create this table:

Code Used:

```
CREATE TABLE offices (
    officeCode VARCHAR(25) PRIMARY KEY,
    city VARCHAR(50),
    phone VARCHAR(50),
    addressLine1 VARCHAR(50),
    addressLine2 VARCHAR(50),
    state VARCHAR(50),
    country VARCHAR(50),
    postalCode VARCHAR(15),
    territory VARCHAR(50)
);
```

A table named **offices** is created to store data **related to office locations**. The table structure includes the following columns: **officeCode as the primary key, city, phone, addressLine1, addressLine2, state,**

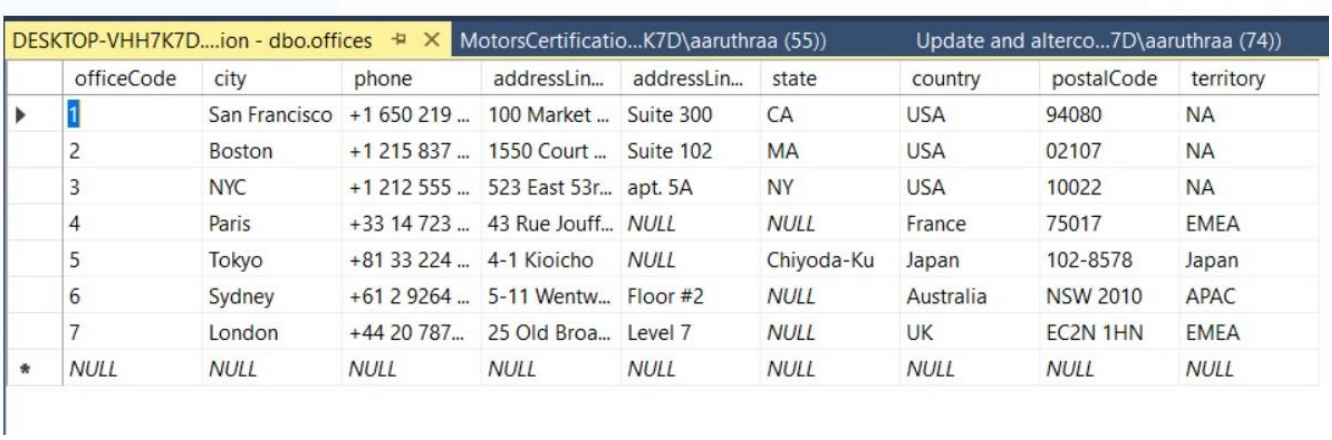
country, postalCode, and territory. These columns are defined with appropriate constraints **to ensure efficient data storage and integrity.**



2.Data Insertion: The **INSERT INTO** command is used to **insert values into the offices table.** This operation populates the table with specific rows of data, **where each row represents an office, including values for columns like officeCode, city, phone, addressLine1, addressLine2, state, country, postalCode, and territory.** The INSERT command ensures that the table is populated with the necessary information for further data processing and analysis.

Code Used:

```
insert into
offices(officeCode,city,phone,addressLine1,addressLine2,state,country,postalcode,territory)
values

('1','San Francisco','+1 650 219 4782','100 Market Street','Suite 300','CA','USA','94080','NA'),

('2','Boston','+1 215 837 0825','1550 Court Place','Suite 102','MA','USA','02107','NA'),

('3','NYC','+1 212 555 3000','523 East 53rd Street','apt. 5A','NY','USA','10022','NA'),

('4','Paris','+33 14 723 4404','43 Rue Jouffroy Dabbans',NULL,NULL,'France','75017','EMEA'),

('5','Tokyo','+81 33 224 5000','4-1 Kioicho',NULL,'Chiyoda-Ku','Japan','102-8578','Japan'),

('6','Sydney','+61 2 9264 2451','5-11 Wentworth Avenue','Floor #2',NULL,'Australia','NSW
2010','APAC'),

('7','London','+44 20 7877 2041','25 Old Broad Street','Level 7',NULL,'UK','EC2N 1HN','EMEA');
```



officeCode	city	phone	addressLin...	addressLin...	state	country	postalCode	territory
1	San Francisco	+1 650 219 ...	100 Market ...	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 ...	1550 Court ...	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 ...	523 East 53r...	apt. 5A	NY	USA	10022	NA
4	Paris	+33 14 723 ...	43 Rue Jouff...	*NULL*	*NULL*	France	75017	EMEA
5	Tokyo	+81 33 224 ...	4-1 Kioicho	*NULL*	Chiyoda-Ku	Japan	102-8578	Japan
6	Sydney	+61 2 9264 ...	5-11 Wentw...	Floor #2	*NULL*	Australia	NSW 2010	APAC
7	London	+44 20 787...	25 Old Broa...	Level 7	*NULL*	UK	EC2N 1HN	EMEA
NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Payments:

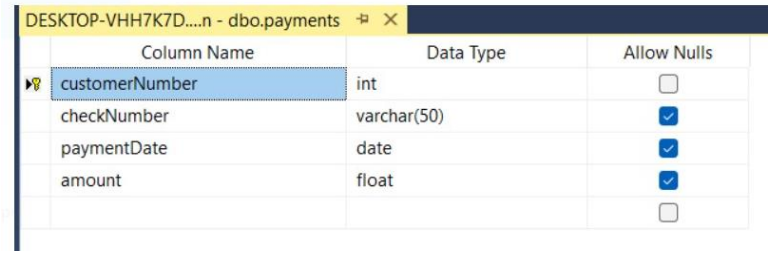
1.Table Creation: As part of the database implementation, the following SQL code was used to create the `of Payments table`.

Code Used:

```
CREATE TABLE payments (
    customerNumber INT PRIMARY KEY,
    checkNumber VARCHAR(50),
    paymentDate DATE,
    amount FLOAT
       );
```

```
CREATE INDEX idx_checkNumber ON payments(checkNumber);
```

A table named **payments** is created to **store data related to payment information.** The table structure includes the following columns: **customerNumber as the primary key**, **checkNumber, paymentDate, and amount.**, with appropriate data types and constraints to ensure efficient data storage and integrity.



2.Data Insertion: INSERT INTO command is used to **insert values into the payments table**. This operation populates the table with specific rows of data, where each row represents a payment, including values for columns **like customerNumber,checkNumber, paymentDate, and amount.** The INSERT command ensures that the table is populated with the necessary information for further data processing and analysis.

Code Used:

```
insert into payments(customerNumber,checkNumber,paymentDate,amount)
       values
(101,'HQ336336','2004-10-19','6066.78'),

(102,'JM555205','2003-06-05','14571.44'),

(103,'OM314933','2004-12-18','1676.14'),

(104,'BO864823','2004-12-17','14191.12'),

(105,'HQ55022','2003-06-06','32641.98'),

(106,'ND748579','2004-08-20','33347.88'),

(107,'GG31455','2003-05-20','45864.03'),

(108,'MA765515','2004-12-15','82261.22'),

(109,'NP603840','2003-05-31','7565.08'),

(110,'NR27552','2004-03-10','44894.74'),

(111,'DB933704','2004-11-14','19501.82'),

(112,'LN373447','2004-08-08','47924.19'),

(113,'NG94694','2005-02-22','49523.67'),

(114,'DB889831','2003-02-16','50218.95'),

(115,'FD317790','2003-10-28','1491.38'),

(116,'KI831359','2004-11-04','17876.32'),

(117,'MA302151','2004-11-28','34638.14'),

(118,'AE215433','2005-03-05','101244.59'),

(119,'BG255406','2004-08-28','85410.87'),

(120,'CQ287967','2003-04-11','11044.30');
```

DESKTOP-VHH7K7D....n - dbo.payments | MotorsCertificatio...K7D\aar

customerN...	checkNum...	paymentD...	amount
101	HQ336336	2004-10-19	6066.78
102	JM555205	2003-06-05	14571.44
103	OM314933	2004-12-18	1676.14
104	BO864823	2004-12-17	14191.12
105	HQ55022	2003-06-06	32641.98
106	ND748579	2004-08-20	33347.88
107	GG31455	2003-05-20	45864.03
108	MA765515	2004-12-15	82261.22
109	NP603840	2003-05-31	7565.08
110	NR27552	2004-03-10	44894.74
111	DB933704	2004-11-14	19501.82
112	LN373447	2004-08-08	47924.19
113	NG94694	2005-02-22	49523.67
114	DB889831	2003-02-16	50218.95
115	FD317790	2003-10-28	1491.38
116	KI831359	2004-11-04	17876.32
117	MA302151	2004-11-28	34638.14
118	AE215433	2005-03-05	101244.59
119	BG255406	2004-08-28	85410.87
120	CQ287967	2003-04-11	11044.3
NULL	NULL	NULL	NULL

3. Foreign Key Creation for Payments Table and Steps Involved:

Foreign Key Creation Using ALTER Statement:

1. The **ALTER TABLE** statement is used to add **a foreign key constraint** between **the `payments` table and the `customers`** table. This ensures that **each `customerNumber` in the `payments` table corresponds to a valid `customerNumber` in the `customers` table**, enforcing referential integrity.
2. These operations are necessary to **ensure that data in the `payments` table is consistent and only references valid records from the `customers` table**.

```
ALTER TABLE payments
ADD CONSTRAINT FK_payments_customers
FOREIGN KEY (customerNumber) REFERENCES customers(customerNumber);

SELECT DISTINCT customerNumber
FROM payments
WHERE customerNumber NOT IN (SELECT customerNumber FROM customers);


To ensure data integrity before creating a foreign key, Remove Invalid Rows
```

Messages

```
Msg 547, Level 16, State 0, Line 73
The ALTER TABLE statement conflicted with the FOREIGN KEY constraint "FK_payments_customers". The conflict occurred in database "MotorsCertification", table "dbo.customers", column 'customerl

Completion time: 2025-01-23T14:24:17.0813064+05:30
```

SELECT DISTINCT customerNumber

FROM payments

WHERE customerNumber NOT IN (SELECT customerNumber FROM customers);

Purpose: The query is used to find **customers in the payments table** who do not exist in the customers table. It helps identify invalid or orphaned payment records.

Results | Messages

	customerNumber
1	101
2	102
3	104
4	105
5	106
6	107
7	108
8	109
9	110
10	111
11	113
12	115
13	116
14	117
15	118
16	120

Code Used:

```
DELETE FROM payments
WHERE customerNumber NOT IN (SELECT customerNumber FROM customers);
```

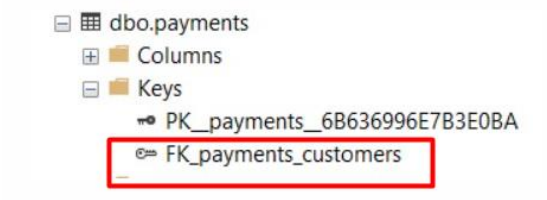
Purpose: This **DELETE** statement **removes invalid rows from the payments table** where **customerNumber does not exist in the customers table**. This ensures data consistency, allowing the foreign key to be added successfully.

4.Verification: Re-execute the ALTER TABLE command to verify the changes applied and ensure the modifications are in effect.

Code Used:

```
ALTER TABLE payments
ADD CONSTRAINT FK_payments_customers
FOREIGN KEY (customerNumber) REFERENCES customers(customerNumber);
```

5. Final Validation: The **foreign key constraint** has been **successfully created**, ensuring **referential integrity between the related tables.**



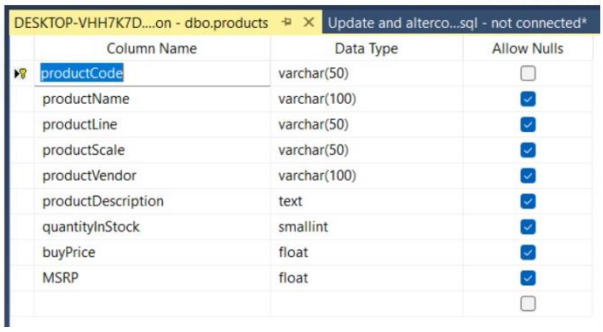
Products:

1.Table Creation: As part of the database implementation, the following SQL code was used to create the **`Products`** table.

Code Used:

```
CREATE TABLE products (
       productCode VARCHAR(50) PRIMARY KEY,
       productName VARCHAR(100),
       productLine VARCHAR(50),
       productScale VARCHAR(50),
       productVendor VARCHAR(100),
       productDescription TEXT,
       quantityInStock SMALLINT,
       buyPrice FLOAT,
       MSRP FLOAT
          );

CREATE INDEX idx_productLine ON products(productLine);
```

A table named `products` is created to store data related to product information. The table structure includes the following columns: **`productCode`** **as the primary key,** **`productName`**, **`productLine`**, **`productScale`**, **`productVendor`**, **`productDescription`**, **`quantityInStock`**, **`buyPrice`**, **and** **`MSRP`**. These columns are defined with appropriate constraints to **ensure efficient data storage and integrity**.

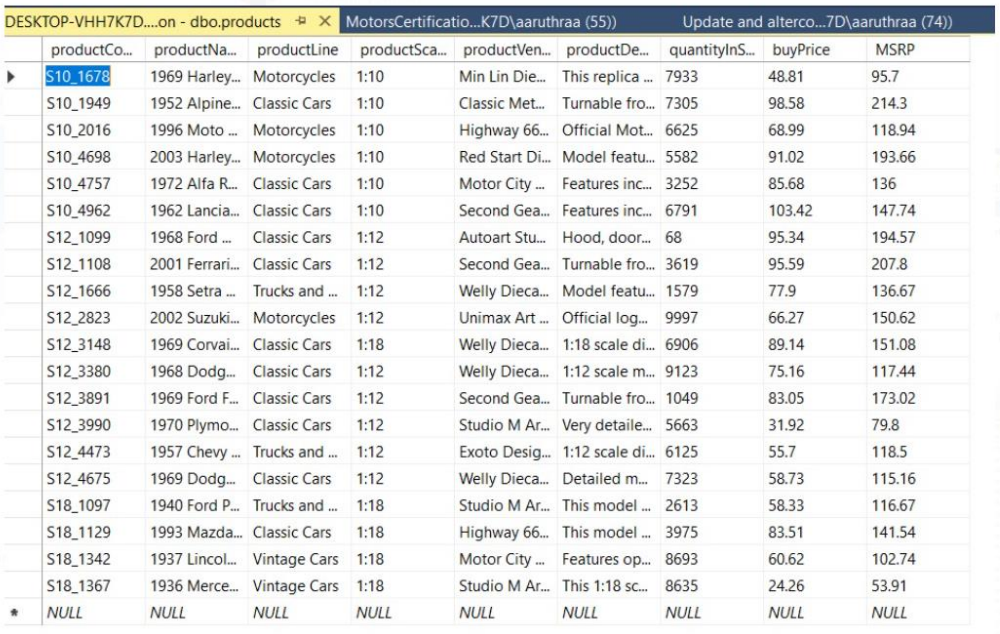


2.Data Insertion: The `INSERT INTO` command is used to **insert values into the `products` table**. This operation populates the table with specific rows of data, where each row represents a product, including values for columns like **`productCode`, `productName`, `productLine`, `productScale`, `productVendor`, `productDescription`, `quantityInStock`, `buyPrice`, and `MSRP`.** The `INSERT` command ensures that the table is populated with the necessary information for further data processing and analysis.

Code Used:

```
insert  into products(productCode,productName,productLine,productScale,
productVendor,productDescription,quantityInStock,buyPrice,MSRP)
values

('S10_1678','1969 Harley Davidson Ultimate Chopper','Motorcycles','1:10','Min Lin Diecast','This
replica features working kickstand, front suspension, gear-shift lever, footbrake lever, drive
chain, wheels and steering. All parts are particularly delicate due to their precise scale and
require special care and attention.',7933,'48.81','95.70'),

('S10_1949','1952 Alpine Renault 1300','Classic Cars','1:10','Classic Metal Creations','Turnable
front wheels; steering function; detailed interior; detailed engine; opening hood; opening
trunk; opening doors; and detailed chassis.',7305,'98.58','214.30'),

('S10_2016','1996 Moto Guzzi 1100i','Motorcycles','1:10','Highway 66 Mini Classics','Official
Moto Guzzi logos and insignias, saddle bags located on side of motorcycle, detailed engine,
working steering, working suspension, two leather seats, luggage rack, dual exhaust pipes, small
saddle bag located on handle bars, two-tone paint with chrome accents, superior die-cast detail
, rotating wheels , working kick stand, diecast metal with plastic parts and baked enamel
finish.',6625,'68.99','118.94'),

('S10_4698','2003 Harley-Davidson Eagle Drag Bike','Motorcycles','1:10','Red Start
Diecast','Model features, official Harley Davidson logos and insignias, detachable rear wheelie
bar, heavy diecast metal with resin parts, authentic multi-color tampo-printed graphics,
separate engine drive belts, free-turning front fork, rotating tires and rear racing slick,
certificate of authenticity, detailed engine, display stand\r\n, precision diecast replica,
baked enamel finish, 1:10 scale model, removable fender, seat and tank cover piece for
displaying the superior detail of the v-twin engine',5582,'91.02','193.66'),

('S10_4757','1972 Alfa Romeo GTA','Classic Cars','1:10','Motor City Art Classics','Features
include: Turnable front wheels; steering function; detailed interior; detailed engine; opening
hood; opening trunk; opening doors; and detailed chassis.',3252,'85.68','136.00'),

('S10_4962','1962 LanciaA Delta 16V','Classic Cars','1:10','Second Gear Diecast','Features
include: Turnable front wheels; steering function; detailed interior; detailed engine; opening
hood; opening trunk; opening doors; and detailed chassis.',6791,'103.42','147.74'),

('S12_1099','1968 Ford Mustang','Classic Cars','1:12','Autoart Studio Design','Hood, doors and
trunk all open to reveal highly detailed interior features. Steering wheel actually turns the
front wheels. Color dark green.',68,'95.34','194.57'),

('S12_1108','2001 Ferrari Enzo','Classic Cars','1:12','Second Gear Diecast','Turnable front
wheels; steering function; detailed interior; detailed engine; opening hood; opening trunk;
opening doors; and detailed chassis.',3619,'95.59','207.80'),

('S12_1666','1958 Setra Bus','Trucks and Buses','1:12','Welly Diecast Productions','Model
features 30 windows, skylights & glare resistant glass, working steering system, original
logos',1579,'77.90','136.67'),

('S12_2823','2002 Suzuki XREO','Motorcycles','1:12','Unimax Art Galleries','Official logos and
insignias, saddle bags located on side of motorcycle, detailed engine, working steering, working
suspension, two leather seats, luggage rack, dual exhaust pipes, small saddle bag located on
handle bars, two-tone paint with chrome accents, superior die-cast detail , rotating wheels ,
working kick stand, diecast metal with plastic parts and baked enamel
finish.',9997,'66.27','150.62'),

('S12_3148','1969 Corvair Monza','Classic Cars','1:18','Welly Diecast Productions','1:18 scale
die-cast about 10\" long doors open, hood opens, trunk opens and wheels
roll',6906,'89.14','151.08'),

('S12_3380','1968 Dodge Charger','Classic Cars','1:12','Welly Diecast Productions','1:12 scale
model of a 1968 Dodge Charger. Hood, doors and trunk all open to reveal highly detailed interior
features. Steering wheel actually turns the front wheels. Color black',9123,'75.16','117.44'),
```

```
('S12_3891','1969 Ford Falcon','Classic Cars','1:12','Second Gear Diecast','Turnable front wheels; steering function; detailed interior; detailed engine; opening hood; opening trunk; opening doors; and detailed chassis.',1049,'83.05','173.02'),

('S12_3990','1970 Plymouth Hemi Cuda','Classic Cars','1:12','Studio M Art Models','Very detailed 1970 Plymouth Cuda model in 1:12 scale. The Cuda is generally accepted as one of the fastest original muscle cars from the 1970s. This model is a reproduction of one of the orginal 652 cars built in 1970. Red color.',5663,'31.92','79.80'),

('S12_4473','1957 Chevy Pickup','Trucks and Buses','1:12','Exoto Designs','1:12 scale die-cast about 20\" long Hood opens, Rubber wheels',6125,'55.70','118.50'),

('S12_4675','1969 Dodge Charger','Classic Cars','1:12','Welly Diecast Productions','Detailed model of the 1969 Dodge Charger. This model includes finely detailed interior and exterior features. Painted in red and white.',7323,'58.73','115.16'),

('S18_1097','1940 Ford Pickup Truck','Trucks and Buses','1:18','Studio M Art Models','This model features soft rubber tires, working steering, rubber mud guards, authentic Ford logos, detailed undercarriage, opening doors and hood,  removable split rear gate, full size spare mounted in bed, detailed interior with opening glove box',2613,'58.33','116.67'),

('S18_1129','1993 Mazda RX-7','Classic Cars','1:18','Highway 66 Mini Classics','This model features, opening hood, opening doors, detailed engine, rear spoiler, opening trunk, working steering, tinted windows, baked enamel finish. Color red.',3975,'83.51','141.54'),

('S18_1342','1937 Lincoln Berline','Vintage Cars','1:18','Motor City Art Classics','Features opening engine cover, doors, trunk, and fuel filler cap. Color black',8693,'60.62','102.74'),

('S18_1367','1936 Mercedes-Benz 500K Special Roadster','Vintage Cars','1:18','Studio M Art Models','This 1:18 scale replica is constructed of heavy die-cast metal and has all the features of the original: working doors and rumble seat, independent spring suspension, detailed interior, working steering system, and a bifold hood that reveals an engine so accurate that it even includes the wiring. All this is topped off with a baked enamel finish. Color white.',8635,'24.26','53.91');
```



DESKTOP-VHH7K7D....on - dbo.products | MotorsCertificatio...K7D\aaruthraa (55)) | Update and alterco...7D\aaruthraa (74))

productCo...	productNa...	productLine	productSca...	productVen...	productDe...	quantityInS...	buyPrice	MSRP
S10_1678	1969 Harley...	Motorcycles	1:10	Min Lin Die...	This replica ...	7933	48.81	95.7
S10_1949	1952 Alpine...	Classic Cars	1:10	Classic Met...	Turnable fro...	7305	98.58	214.3
S10_2016	1996 Moto ...	Motorcycles	1:10	Highway 66...	Official Mot...	6625	68.99	118.94
S10_4698	2003 Harley...	Motorcycles	1:10	Red Start Di...	Model featu...	5582	91.02	193.66
S10_4757	1972 Alfa R...	Classic Cars	1:10	Motor City ...	Features inc...	3252	85.68	136
S10_4962	1962 Lancia...	Classic Cars	1:10	Second Gea...	Features inc...	6791	103.42	147.74
S12_1099	1968 Ford ...	Classic Cars	1:12	Autoart Stu...	Hood, door...	68	95.34	194.57
S12_1108	2001 Ferrari...	Classic Cars	1:12	Second Gea...	Turnable fro...	3619	95.59	207.8
S12_1666	1958 Setra ...	Trucks and ...	1:12	Welly Dieca...	Model featu...	1579	77.9	136.67
S12_2823	2002 Suzuki...	Motorcycles	1:12	Unimax Art ...	Official log...	9997	66.27	150.62
S12_3148	1969 Corvai...	Classic Cars	1:18	Welly Dieca...	1:18 scale di...	6906	89.14	151.08
S12_3380	1968 Dodg...	Classic Cars	1:12	Welly Dieca...	1:12 scale m...	9123	75.16	117.44
S12_3891	1969 Ford F...	Classic Cars	1:12	Second Gea...	Turnable fro...	1049	83.05	173.02
S12_3990	1970 Plymo...	Classic Cars	1:12	Studio M Ar...	Very detaile...	5663	31.92	79.8
S12_4473	1957 Chevy ...	Trucks and ...	1:12	Exoto Desig...	1:12 scale di...	6125	55.7	118.5
S12_4675	1969 Dodg...	Classic Cars	1:12	Welly Dieca...	Detailed m...	7323	58.73	115.16
S18_1097	1940 Ford P...	Trucks and ...	1:18	Studio M Ar...	This model ...	2613	58.33	116.67
S18_1129	1993 Mazda...	Classic Cars	1:18	Highway 66...	This model ...	3975	83.51	141.54
S18_1342	1937 Lincol...	Vintage Cars	1:18	Motor City ...	Features op...	8693	60.62	102.74
S18_1367	1936 Merce...	Vintage Cars	1:18	Studio M Ar...	This 1:18 sc...	8635	24.26	53.91
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3.Foreign Key Creation for Products **Table and Steps Involved**:

Foreign Key Creation Using ALTER Statement:

The **`ALTER TABLE`** statement is used to **add a foreign key constraint** between the **`products` table and the `productlines`** table. This ensures that each `productLine` in the `products` table corresponds to a valid `productLine` in the `productlines` table, enforcing referential integrity.

4.Verification: Re-execute the ALTER TABLE command to verify the changes applied and ensure the modifications are in effect.

Code Used:

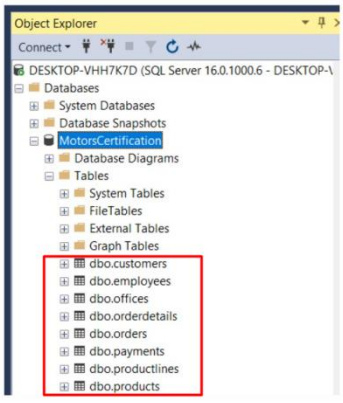
```
ALTER TABLE products
ADD CONSTRAINT FK_products_productlines
FOREIGN KEY (productLine) REFERENCES productlines(productLine);
```

5. Final Validation: The **foreign key constraint** has been **successfully created**, ensuring **referential integrity between the related tables.**

SQL Implementation: Entire Table Creation

As part of the database implementation in SSMS, 8 tables were created to organize and manage data effectively, **ensuring the system meets all project requirements**.

The below Screenchot showcases the **creation** of all **tables essential for the project's successful implementation**. With the foundational structures in place, the focus now shifts to the **ER diagram, which visually represents the relationships and ensures a comprehensive understanding of the database design**.



Question 3:

Provide comments before every task that is performed describing the operation that is being performed and **attach a screenshot of ER diagram from SSMS.**

Solution:

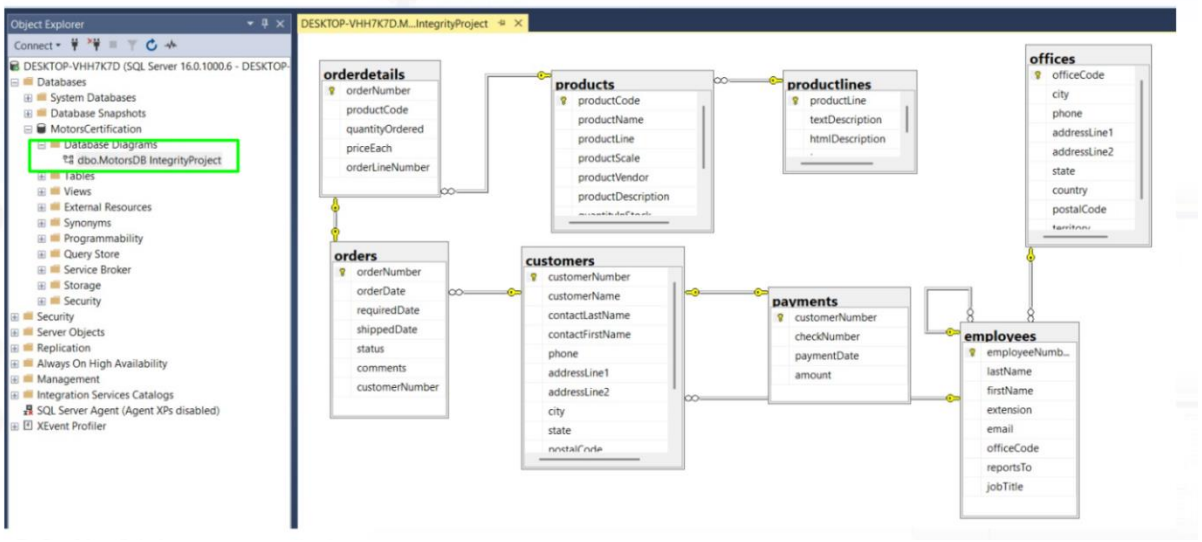
ER Diagram: Which visually represents the relationships and ensures a comprehensive understanding of the database design .

The **MotorsDBIntegrityProject** (ER Diagram) explains the following:

1. **Database Structure:** It defines how the data is organized in tables (**e.g., `customers`, `orders`, `products`, `payments`, etc.)** to store all the relevant information about **motor products, customers, orders, and payments.**
2. **Relationships Between Data:** It shows how different tables are related (e.g., linking orders to customers and products), ensuring that **data is properly connected and consistent**.
3. **Data Integrity:** It enforces rules to keep the **data accurate and consistent**, such as:
    - Using **primary keys** to uniquely identify each row in a table.
    - Using **foreign keys** to **maintain relationships between tables** (e.g., each order should be linked to a valid customer).

4. **Security and Access Control:** It helps define how **sensitive information is protected and ensures that only authorized users can access or modify specific data**.
5. **Business Processes:** It explains how the system handles **key operations like processing customer orders, tracking payments, and managing inventory of motor products**, ensuring that **everything** is **well-organized and error-free**.

In summary, the **motorsdbintegrityproject** is designed to help **HerculesMotoCorp** manage its **motor products, customers, and orders efficiently, ensuring data accuracy, integrity, and compliance with industry standards**.



Question 4:

Delete the columns in **productlines** which are useless that do not infer anything.

Solution:

LAPTOPARUN.sample - dbo.productlines | SQLQuery2.sql - L...TOPARUN\Arun (64)) | LAPTOPARUN.sample - dbo.productlines

productLine	textDescription	htmlDescription	image
Classic Cars	Attention car enthusiasts: Make yo...	*NULL*	*NULL*
Motorcycles	Our motorcycles are state of the ar...	*NULL*	*NULL*
Planes	Unique, diecast airplane and helico...	*NULL*	*NULL*
Ships	The perfect holiday or anniversary ...	*NULL*	*NULL*
Trains	Model trains are a rewarding hobb...	*NULL*	*NULL*
Trucks and Buses	The Truck and Bus models are reali...	*NULL*	*NULL*
Vintage Cars	Our Vintage Car models realisticall...	*NULL*	*NULL*
NULL	*NULL*	*NULL*	*NULL*

Purpose:

- **Simplify the Table**: This makes the table more efficient by deleting unnecessary data.
- **Remove Unwanted Data**: The operation removes columns (`htmlDescription` and `image`) that are no longer needed.

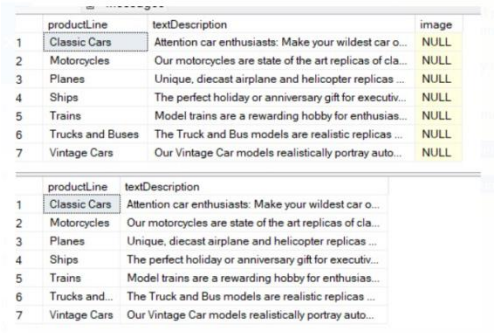
Code Used:

```
ALTER TABLE productLines
Drop Column htmlDescription;
SELECT * FROM productlines;
```

ALTER TABLE: Modifies the structure of the productLines table.
Drop Column htmlDescription: **Deletes the htmlDescription column** from the table, removing any data stored in that column.

```
ALTER TABLE productLines
Drop Column image;
SELECT * FROM productlines;
```

Similar to the first block, this **removes the image column** from the productLines table and then shows the updated table without that column.



	productLine	textDescription	image
1	Classic Cars	Attention car enthusiasts: Make your wildest car o...	NULL
2	Motorcycles	Our motorcycles are state of the art replicas of cla...	NULL
3	Planes	Unique, diecast airplane and helicopter replicas ...	NULL
4	Ships	The perfect holiday or anniversary gift for executiv...	NULL
5	Trains	Model trains are a rewarding hobby for enthusias...	NULL
6	Trucks and Buses	The Truck and Bus models are realistic replicas ...	NULL
7	Vintage Cars	Our Vintage Car models realistically portray auto...	NULL

	productLine	textDescription
1	Classic Cars	Attention car enthusiasts: Make your wildest car o...
2	Motorcycles	Our motorcycles are state of the art replicas of cla...
3	Planes	Unique, diecast airplane and helicopter replicas ...
4	Ships	The perfect holiday or anniversary gift for executiv...
5	Trains	Model trains are a rewarding hobby for enthusias...
6	Trucks and...	The Truck and Bus models are realistic replicas ...
7	Vintage Cars	Our Vintage Car models realistically portray auto...

Question 5:

Use a select statement to verify all insertions as well as updates.

Solution:

Here are the **SELECT statements** for verifying the data insertion for all the tables mentioned in your project:

1. orderdetails Table:

To verify data insertion into the orderdetails table:

```
SELECT * FROM orderdetails;
```

2. employees Table:

To verify data insertion into the employees table:

```
SELECT * FROM employees;
```

3. payments Table:

To verify data insertion into the payments table:

```
SELECT * FROM payments;
```

4. products Table:

To verify data insertion into the products table:

```
SELECT * FROM products;
```

5. productlines Table:

To verify data insertion into the productlines table:

```
SELECT * FROM productlines;
```

6. customers Table:

To verify data insertion into the customers table:

```
SELECT * FROM customers;
```

7. offices Table:

To verify data insertion into the offices table:

```
SELECT * FROM offices;
```

8. orders Table:

To verify data insertion into the orders table:

```
SELECT * FROM orders;
```

These **SELECT statements will help you verify that data has been inserted correctly** into the respective tables. After performing the insertions, running these **SELECT queries will display all the rows in each table to confirm the success of the data insertion operation**.

2. Verify Data Updates:

After performing an update on a table, you can use **SELECT** to check the modified data.

Example:

```
-- After updating the price in products table
SELECT productCode, priceEach FROM products WHERE productCode = 'S10_1678';

-- After updating customer details
SELECT * FROM customers WHERE customerNumber = 103;
```

3. Verify Data Deletions:

If rows have been deleted, use **SELECT** to verify that the rows are no longer present in the table.

Example:

```
-- After deleting from the productLines table
SELECT * FROM productLines WHERE htmlDescription IS NULL;

-- After deleting from the customers table
SELECT * FROM customers WHERE customerNumber = 103;
```

6. Verify Distinct Values for Uniqueness:

To ensure there **are no duplicate entries or invalid data,** use **SELECT DISTINCT**.

Example:

Code Used:

```
SELECT DISTINCT customerNumber FROM payments;
```

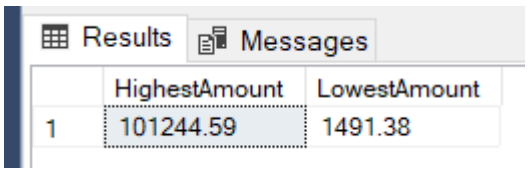
```
SELECT DISTINCT orderNumber FROM orderdetails;
```

Question 6:

Find out the highest and the lowest amount from Payments Table.
Solution:
Code Used:

```
SELECT
    MAX(amount) AS highest_amount,
    MIN(amount) AS lowest_amount
FROM payments;
```

The purpose of the following SELECT query is to **retrieve the maximum and minimum payment amounts** from the payments table.



Question 7:

Give the unique count of customerName from **customers.**
Solution:

Code Used:

```
SELECT COUNT(DISTINCT customerName) AS unique_customer_count
FROM customers;
```

The purpose: This **SELECT query** is to **count the number of unique customer names** in the customers table.

- **COUNT(DISTINCT customerName)**: Counts the number of unique (non-duplicate) customer names in the customerName column.
- **AS unique_customer_count**: The alias **unique_customer_count** renames the result column, making **the output more understandable**.

Purpose:

- **Data Analysis**: This query helps determine **how many unique customers (by their names) exist in the database, excluding any duplicates.**
- **Customer Insights**: It can be used to understand **the number of distinct customers who have made purchases or interacted with the company**, **which is valuable for marketing, customer segmentation, and reporting purposes.**

Question 8:

Create a view from **customers** and **payments** named cust_payment and select customerName, amount, contactLastName, contactFirstName who have paid.Truncate and Drop the view after operation.

Solutions: Step 1: Create the View

Code Used:

```
CREATE VIEW cust_payment AS

SELECT c.customerName,

    p.amount,

    c.contactLastName,

    c.contactFirstName

FROM

    customers c

JOIN

    payments p

ON

    c.customerNumber = p.customerNumber;
```

Purpose: This step creates a **view** named **cust_payment**. A **view is a virtual table** that **combines data from multiple tables (in this case, customers and payments) based on a JOIN condition**.
Operation: It selects the customerName, amount, contactLastName, and contactFirstName fields from the customers and payments tables, where the customerNumber matches between the two tables.
Why Use Views: **Views allow you to simplify complex queries by encapsulating the query logic in a reusable object.** You can query the view as if it were a table.

Code Used:

```
SELECT * FROM cust_payment ;
```

When you run the SELECT * FROM cust_payment;it will dynamically execute the query inside the view and display the results based on the current data in the customers and payments tables.

Results Messages

	customerName	amount	contactLastName	contactFirstName
1	Atelier graphique	1676.14	Schmitt	Carine
2	Signal Gift Stores	47924.19	King	Jean
3	Australian Collectors, Co.	50218.95	Ferguson	Peter
4	La Rochelle Gifts	85410.87	Labrune	Janine

Step 2: Select Data from the View

Code Used:

```
SELECT

    customerName,

    amount,

    contactLastName,
```

```
    contactFirstName

FROM cust_payment;
```

Why Use This: After creating the view, you can use it like a table to **query data easily without repeating the complex JOIN logic.** It's useful for frequently used, **complex queries that you want to simplify.**

Step3:

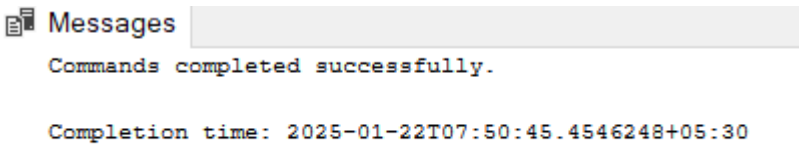
TRUNCATE the view:

Why Can't TRUNCATE Be Used: TRUNCATE is used to remove all records from a table and is applicable only to physical tables, not views. A view doesn't store data physically; it's a representation of a query result.

Step 4: Drop the View

DROP VIEW cust_payment;

Why Drop the View: Once the view is no longer needed, it can be dropped to clean up the database. Dropping a view doesn't affect the underlying tables; it only removes the virtual table representation.



Question 9:

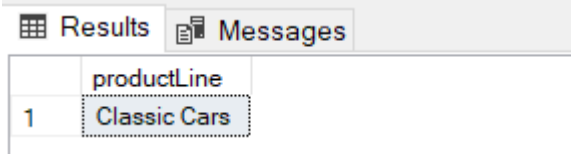
Create a stored procedure on **products** which displays productLine for Classic Cars.
Solution:
Code Used:

```
CREATE PROCEDURE GetClassicCarsProductLine
AS
BEGIN
    SELECT DISTINCT productLine
    FROM products
    WHERE productLine = 'Classic Cars';
END;

EXEC GetClassicCarsProductLine;
```

Purpose : This code is to **create and execute a stored procedure** that retrieves the distinct product lines from the products table where the productLine is 'Classic Cars'.

Executing the Stored Procedure (EXEC GetClassicCarsProductLine):

This statement **executes the stored procedure that was just created**, returning the result of the query inside it, which is a **list of distinct product lines where the product line is 'Classic Cars'**.



Question 10:

Create a function to get the creditLimit of **customers** less than 96800.

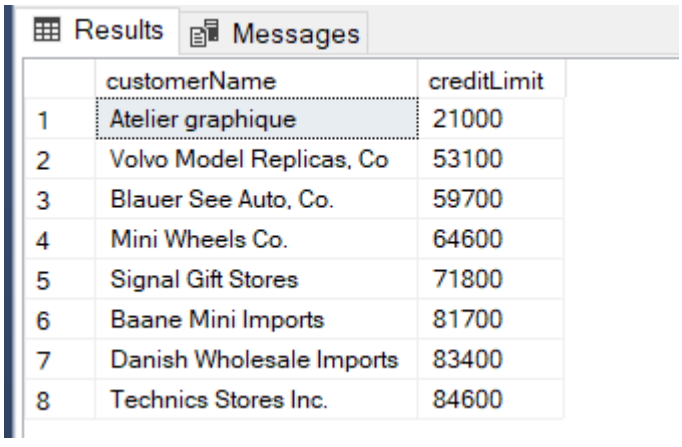
Solution:

Code Used:

```
CREATE FUNCTION GetCustomersByCreditLimit()
RETURNS TABLE
AS
RETURN
(
    SELECT customerNumber, customerName, creditLimit
    FROM customers
    WHERE creditLimit < 96800
        AND creditLimit > 0 -- Exclude invalid or zero credit limits
        AND creditLimit IS NOT NULL -- Exclude NULL values
);
SELECT customerName, creditLimit
FROM dbo.GetCustomersByCreditLimit()
ORDER BY creditLimit;
```

Purpose: This SQL code is used to **create and execute a table-valued function that retrieves customers with specific credit limits from the customers table.**

CREATE FUNCTION GetCustomersByCreditLimit(): This defines **a table-valued function** named **GetCustomersByCreditLimit.** A table-valued **function returns a table as a result**, which can be used in SELECT queries, just like querying a regular table


Results Messages

	customerName	creditLimit
1	Atelier graphique	21000
2	Volvo Model Replicas, Co	53100
3	Blauer See Auto, Co.	59700
4	Mini Wheels Co.	64600
5	Signal Gift Stores	71800
6	Baane Mini Imports	81700
7	Danish Wholesale Imports	83400
8	Technics Stores Inc.	84600

Purpose:

- The code creates a **table-valued function** that filters customers based on their credit limits and returns a table with the relevant customer information.
- **Encapsulation of Logic**: By using a function, you encapsulate the filtering logic, making it reusable in different queries.

- **Data Integrity and Validation**: The function ensures that only valid credit limit values are included (non-zero, positive, and non-NULL values), maintaining data integrity.
- **Data Analysis**: The function is useful for reporting or further data analysis where you want to retrieve customers with certain credit limit conditions.

Question 11:

Create Trigger to store transaction record for **employee** table which displays employeeNumber, lastName, FirstName and office code upon insertion

Solution:

Code Used:

```
CREATE TABLE employee_transaction_log (
    transactionID INT IDENTITY(1,1) PRIMARY KEY,
    employeeNumber INT,
    lastName VARCHAR(50),
    firstName VARCHAR(50),
    officeCode VARCHAR(10),
    transactionDate DATETIME
);
CREATE TRIGGER trg_after_employee_insert
ON employees
AFTER INSERT
AS
BEGIN
    INSERT INTO employee_transaction_log (employeeNumber, lastName, firstName, officeCode, transactionDate)
    SELECT employeeNumber, lastName, firstName, officeCode, GETDATE()
    FROM inserted;
    PRINT 'Transaction recorded for new employee';
END;
INSERT INTO employees (employeeNumber, lastName, firstName, officeCode)
VALUES (999, 'Smith', 'John', '1');
SELECT * FROM employee_transaction_log;
```

Purpose: Above SQL code is to create an **employee transaction log system** that logs each new employee insertion into a dedicated transaction log table. It also demonstrates the use of a **trigger** to automatically record changes when a new employee is inserted into the `employees` table.

	transactionID	employeeNumber	lastName	firstName	officeCode	transactionDate
1	1	999	Smith	John	1	2025-01-22 10:31:17.190

Purpose of Trigger Function:

1. **Data Integrity and Auditing**: Automatically logs new employee additions to keep a record of each change for auditing or tracking purposes.
2. **Automation with Trigger**: The trigger automatically logs employee data, reducing manual errors and ensuring consistency.
3. **Real-Time Tracking**: Records the exact time an employee is added, allowing for real-time tracking of changes.

Question 12:

Create a Trigger to display customer number if the amount is greater than 10,000
Solution:

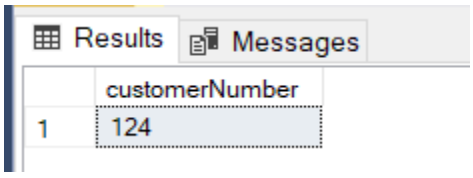
Code Used:

```
CREATE TRIGGER trg_after_payment_insert

ON payments

AFTER INSERT

AS

BEGIN

   IF EXISTS (SELECT 1 FROM inserted WHERE amount > 10000)

   BEGIN

      SELECT customerNumber

      FROM inserted

      WHERE amount > 10000;

   END

END;

INSERT INTO payments (customerNumber, checkNumber, paymentDate, amount)

VALUES (124, 'pN373445', '2004-08-08', 15000);
```

Purpose: This SQL code is to create a **trigger** that automatically checks for **payments greater than 10,000** in the `payments` table and displays the **`customerNumber` of the customers making those payments**.

Purpose of Trigger:

- **Automatic Check for Large Payments**: The trigger is set to automatically check when a payment greater than 10,000 is made.
- **Customer Identification**: If a payment exceeds 10,000, the trigger displays the `customerNumber` of the customer who made the payment, helping to track significant transactions. Here's a summarized documentation of the steps followed for creating the trigger:



Dear HerculesMotoCorp Team,

I am excited to **present the database system** designed specifically to **meet the needs of HerculesMotoCorp.** As your Database Administrator, My goal was to **create a comprehensive solution** that not only simplifies the daily tasks of your employees but also **provides valuable insights for other parties, such as shareholders**. This database system is crafted to deliver:

- Accurate and consistent data for smooth operations.
- Automation to reduce manual effort and errors.
- Real-time insights into critical data, such as transactions and employee records.
- Easy adaptability to support future growth and evolving requirements.

I would also like to **express my gratitude** to **SQL** for **being the foundation of this system, enabling efficient data management, querying, and seamless integration.** I believe this system will optimize your operations, enhance decision-making, and ensure compliance with regulatory standards.

Warm regards,
[Mythily Arunprasad].

Acknowledgments

I would like to **express my sincere gratitude to my trainer for her immense support and continuous guidance** throughout the project. **Her dedication and assistance** in clearing doubts helped me **navigate through challenges, ensuring the project's success.** I truly appreciate Her commitment to my learning and the valuable insights provided during the course of the project.

Thank you for your unwavering support!