

# **FUTURE DESIGN CAR DEALERSHIP**

## **Phase 2: Implementation**

**BUAN 6320.004 – Database Foundations  
for Business Analytics**

Car Dealership Management System

### **Group 6:**

Krish Sonani

Nihal Sharma

Vijay Venkatesan

Ishan Waghmare

Yao-Hui Tseng

Anamika Das

Ninad Patil

**Business Description:**

- Many business procedures, such as customer relationship management, service operations, sales monitoring, vehicle inventory management, and financial transactions, are involved in running a car dealership. Streamlining these processes using a centralized database system that offers a thorough picture of dealership activity is the goal of the Car Dealership Management System.
- The goal of this project is to create a database system that will store and arrange financial alternatives, service histories, customer information, sales records, vehicle inventories, and other relevant information. Dealership managers and employees can simply access real-time data, create comprehensive reports, and enhance company strategy based on precise insights by combining all this information into a single system.

The Car Dealership Management System's main goals are to:

1. Make inventory management easier by giving users real-time visibility into the availability and status of cars.
2. Improve customer relationship management by keeping accurate records of sales, services rendered, and client preferences.
3. Manage financing options, promotional offers, and payment histories to increase financial transparency.
4. Track and schedule auto maintenance and repairs to maximize service management.
5. Assist in decision-making by offering insights into supplier management, technician performance, and sales patterns.

**Objective:** This database system will enable smooth daily operations and make strategic planning easier for long-term company success.

## Data Requirements and Scope:

The scope of the project is to develop a complete database solution for a car dealership that includes data storage and management for the following entities:

1. **Car Models:** All car model we have for sale, including engine type for the model.
2. **Cars:** All cars available for sale, including specifications, pricing, and stock status.
3. **Customers:** Customer profiles, contact information, purchase history, and preferences.
4. **Salespersons:** Sales staff details, performance metrics, and sales history.
5. **Sales:** Transactions and associated details, including the car, customer, salesperson, and payment information.
6. **Suppliers:** Information on suppliers, including contact details and relationships with the dealership.
7. **Inventory:** Incoming and outgoing stock management, tracking purchase dates, supplier data, and current availability.
8. **Services:** Maintenance and repair services provided, including service type, costs, and technicians involved.
9. **Technicians:** Employee details of technicians performing services, including specialties and certifications.
10. **Finance Options:** Financing options available to customers, including terms, interest rates, and eligibility.
11. **Orders:** Orders placed for new cars from suppliers, tracking status, delivery dates, and quantities.
12. **Car Features:** Additional features available for cars, such as navigation systems, sunroofs, and customization options.
13. **Payments:** Payment details for completed sales, including amounts, payment methods, and dates.

**Functionality:** The system will support the management and tracking of these entities and their interrelationships, ensuring the dealership can maintain accurate records and generate meaningful insights.

## Team Roles and Responsibilities:

### Nihal Sharma - **Project Manager**

- Oversees project planning, coordination, and ensures all team members stay aligned with project objectives.

### Krish Sonani - **Business Analyst**

- Collects requirements, defines project scope, and ensures the final database meets business goals.

### Anamika Das - **Database Designer**

- Creates conceptual and logical database (ER) designs and ensures data normalization and relationships are accurate.

### Yao-Hui Tseng - **Data Modeler**

- Defines data structures, attributes, and relationships to translate business needs into the data model.

### Vijay Venkatesan - **Backend Developer**

- Converts the database design into a physical schema using SQL and ensures efficient data handling.

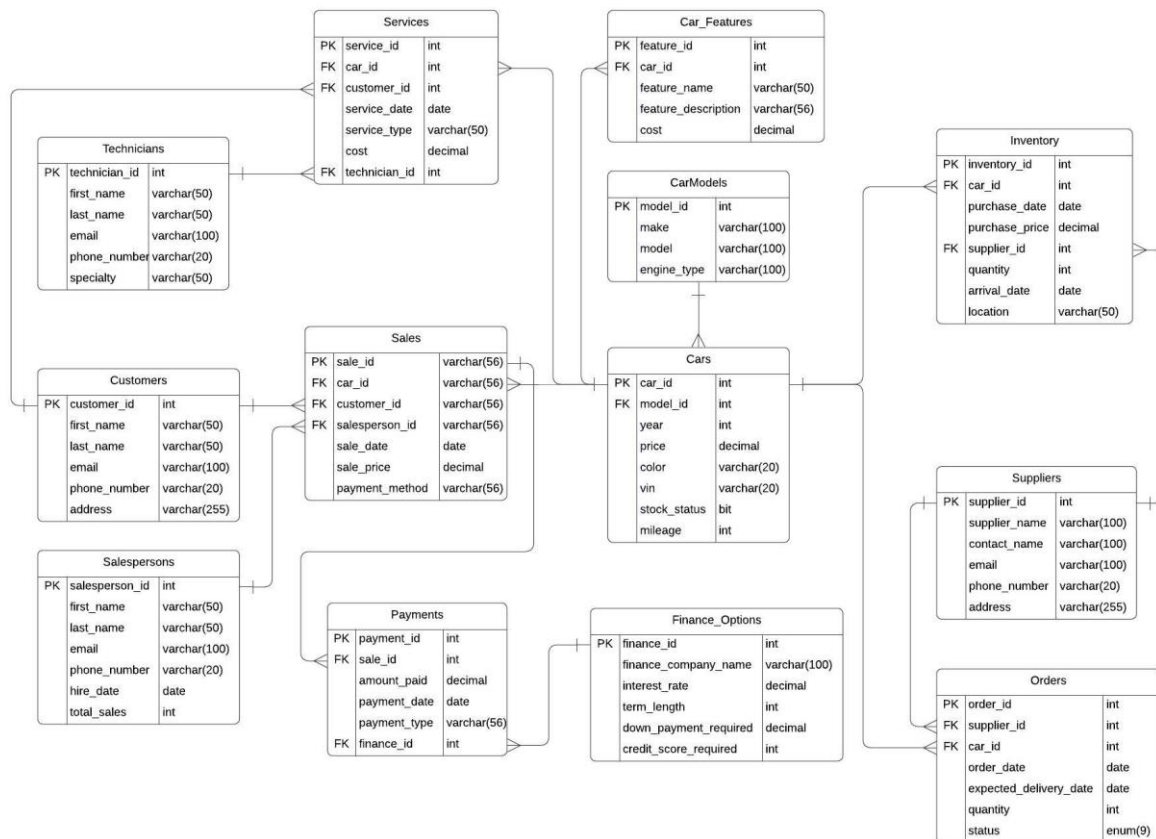
### Ishan Waghmare - **Documentation Specialist**

- Compiles detailed documentation for the project.

### Ninad Patil - **Tester**

- Validates the database design through testing for the project.

## ER Diagram (revised)



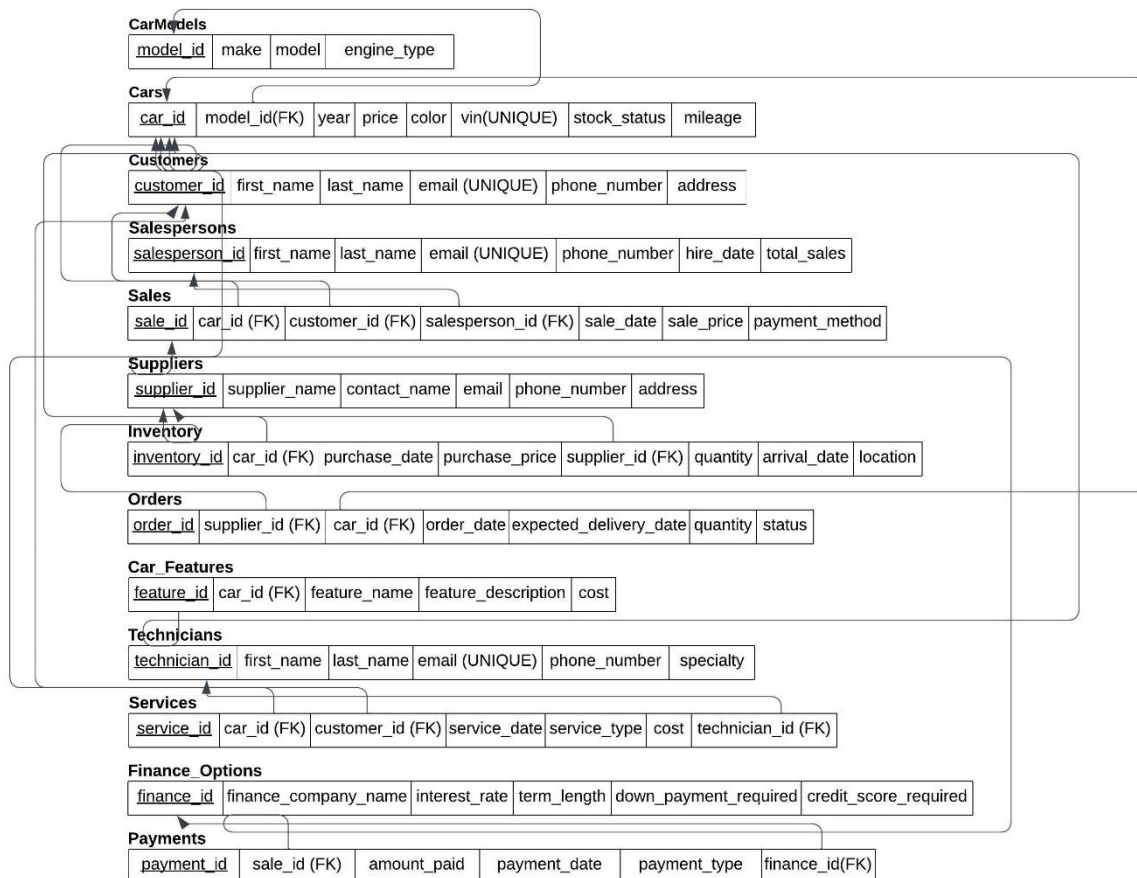
Crow's foot notes provide a clear and structured way to visualize the relationships and attributes of entities in a database schema. This approach helps in understanding the overall design and facilitates communication among stakeholders involved in database design and implementation in a very intuitive way.

- **Entities:** Each box is an entity, which represents an actual table in the database.  
Example: Cars, Customers, Salespersons.
- **Attributes:** Under the table name in each box.  
Example Attributes:
  - For Cars: car\_id, model\_id, year, price, etc.
  - For Customers: customer\_id, first\_name, last\_name, etc.
- **Relationships:** Connect between entities to show which entities are related.  
Example Relationships:
  - Sales connects Cars, Customers, and Salespersons.
- **Cardinality:** Indicated near the lines connecting entities, showing the nature of relationships (e.g., one-to-one, one-to-many, many-to-many).
- **Primary Keys:** Underlined attributes to indicate a unique identifier.

Example: car\_id in the Cars table.

Example: A Customer can have many Sales, but each Sale is associated with one Customer.

## Relational Schema (revised)



This summary captures the structure of our database schema, outlining the relationships between different entities through foreign keys and indicating how each table is organized.

Summary:

- **CarModels**  
model\_id, make, model, engine\_type  
PK: model\_id
- **Cars:**  
car\_id, model\_id, year, price, color, vin, stock\_status, mileage  
PK: car\_id  
FK: model\_id
- **Customers:**  
customer\_id, first\_name, last\_name, email, phone\_number, address  
PK: customer\_id
- **Sales:**

sale\_id, car\_id, customer\_id, salesperson\_id, sale\_date, sale\_price,  
payment\_method

**PK:** sale\_id

**FKs:** car\_id, customer\_id, salesperson\_id

Etc...

(**PK:** Primary Key; **FK:** Foreign Key)

This concise summary captures some of the main attributes and relationships in our database schema.

## Functional Dependencies & Normalization

In a relational database, functional dependency is a relationship between the attributes in a relation (table), where one attribute's value uniquely determines another attribute's value. And for this database, our goal is to make all tables comply with Third Normal Form(3NF). The following shows our steps to normalize our tables into 3NF.

Steps for Normalizing to 3NF

### First Normal Form (1NF):

1. Ensure the table has a primary key.
2. Each column must have atomic values (no multiple values or lists).
3. Eliminate duplicate rows.

As we check through all the tables, we do not identify any tables that violate the requirements for 1NF, so all the tables comply with 1NF. Following is a screenshot for Sales table as an example:

	sale_id	car_id	customer_id	salesperson_id	sale_date	sale_price	payment_method
1	1	2	1	1	2021-08-21	35000.00	Cash
2	2	3	2	2	2022-01-15	80000.00	Financing
3	3	1	4	1	2023-03-18	20000.00	Cash
4	4	5	6	2	2023-04-10	40000.00	Credit Card
5	5	4	7	1	2023-05-25	18000.00	Cash
6	6	7	8	3	2022-06-15	42000.00	Financing
7	7	9	9	4	2023-01-10	70000.00	Cash
8	8	8	10	5	2021-09-21	54000.00	Credit Card
9	9	6	11	6	2022-12-13	62000.00	Cash
10	10	14	12	7	2023-02-14	18000.00	Financing
11	11	11	13	8	2020-08-23	26000.00	Credit Card
12	12	12	14	9	2022-11-15	51000.00	Cash
13	13	13	15	10	2023-04-12	29000.00	Cash
14	14	15	16	1	2021-03-08	25000.00	Cash
15	15	10	17	2	2021-05-17	34000.00	Credit Card
16	16	6	18	3	2023-07-21	59000.00	Cash
17	17	7	19	4	2023-09-15	52000.00	Credit Card
18	18	9	20	5	2023-06-24	73000.00	Cash
19	19	8	21	6	2022-05-10	54000.00	Financing
20	20	15	22	7	2021-11-11	24000.00	Cash
21	21	5	10	2	2024-03-15	20000.00	Credit Card
22	22	12	11	3	2024-03-16	22500.00	Cash
23	23	21	12	1	2024-03-17	18000.00	Financing
24	24	8	13	4	2024-03-18	23500.00	Bank Transfer
25	25	3	14	5	2024-03-19	27000.00	Cash

**Second Normal Form (2NF):**

1. Ensure the table is in 1NF.
2. Remove partial dependencies to make sure that all non-key attributes are fully functional depending on primary key.

In order to check if all tables comply with 2NF, we look into all tables to determine all functional dependencies to see if there are any partial dependencies we must remove, and following the functional dependencies we identified:

Cars Table:

car\_id → make, model, year, price, color, engine\_type, vin, stock\_status, mileage

vin → car\_id

Customers Table:

customer\_id → first\_name, last\_name, email, phone\_number, address

Sales Table

sale\_id → car\_id, customer\_id, salesperson\_id, sale\_date, sale\_price, payment\_method

Suppliers Table

supplier\_id → supplier\_name, contact\_name, email, phone\_number, address

Technicians Table

technician\_id → first\_name, last\_name, email, phone\_number, specialty

Inventory Table

inventory\_id → car\_id, purchase\_date, purchase\_price, supplier\_id, quantity, arrival\_date, location

Services Table

service\_id → car\_id, customer\_id, service\_date, service\_type, cost, technician\_id

FinanceOptions Table

finance\_id → finance\_company\_name, interest\_rate, term\_length, down\_payment\_required, credit\_score\_required

Orders Table

order\_id → supplier\_id, car\_id, order\_date, expected\_delivery\_date, quantity, status

CarFeatures Table

feature\_id → car\_id, feature\_name, feature\_description, cost

Payments Table

payment\_id → sale\_id, amount\_paid, payment\_date, payment\_type

The only functional dependency that's does not depend on primary key is in the Cars table as the Vin number can also be the primary key and determine all of the attributes in Cars table, but since it would be inconvenience if we use Vin number to internally manage the



cars we have and it would be redundancy on table-wise if we only separate the Vin number out from Cars table, so we keep the Vin number as an unique attribute in the Cars table.

As a result, we identified all tables comply with 2NF.

### **Third Normal Form (3NF):**

1. Ensure the table is in 2NF.
2. Remove transitive dependencies (non-key attributes dependent on other non-key attributes).

Same as how we check on 2NF, we did the same process to all the tables we have to identify if there is any transitive dependency in our tables, and we only identified the following transitive dependency in Cars table and all other tables comply with 3NF:

#### **Cars Table**

model → make: Each model belongs to a specific make (e.g., Corolla → Toyota)

We must separate the model and make into another table. As since we considered the engine type is also an attribute for model, so we also take the engine\_type attribute out from Cars table into the new table called CarModels. And since we want to easily manage and reference on the models and cars, we create a model\_id attribute as primary key, so here is the normalized Cars table and CarsModels table:

#### **Cars Table**

car\_id → model\_id, year, price, color, vin, stock\_status, mileage

#### CarModels Table

model\_id → make, model, engine\_type

By performing this process, we now ensure that all our tables comply with 3NF.

## Database Design and Implementation

The database comprises 13 tables, each normalized to reduce redundancy and maintain data integrity. The tables are linked using primary and foreign keys to enforce relationships and ensure referential integrity.

```
CREATE DATABASE CarDealershipDataBase;
USE CarDealershipDataBase;
```

7 22:58:23 USE CarDealershipDataBase

0 row(s) affected

Table 1: **Car Models**: Stores information about car models, makes, and engine types.

```
-- 1. Cars Table
CREATE TABLE CarModels (
    model_id INT PRIMARY KEY,
    make VARCHAR(50),
    model VARCHAR(50),
    engine_type VARCHAR(50)
);
SELECT * FROM CarModels;

INSERT INTO CarModels VALUES
(1, 'Toyota', 'Corolla', 'Gasoline'),
(2, 'Ford', 'Mustang', 'Gasoline'),
(3, 'Tesla', 'Model S', 'Electric'),
(4, 'Honda', 'Civic', 'Gasoline'),
(5, 'Chevrolet', 'Camaro', 'Gasoline'),
```

output

	model_id	make	model	engine_type
▶	1	Toyota	Corolla	Gasoline
	2	Ford	Mustang	Gasoline
	3	Tesla	Model S	Electric
	4	Honda	Civic	Gasoline
	5	Chevrolet	Camaro	Gasoline
	6	BMW	X5	Gasoline
	7	Audi	A4	Gasoline
	8	Mercedes	C-Class	Gasoline
	9	Tesla	Model 3	Electric
	10	Toyota	Highlander	Gasoline

Table 2: **Cars**: Contains specific details about cars in inventory, including price, color, year, and stock status.

```
-- 2. Cars Table
CREATE TABLE Cars (
    car_id INT PRIMARY KEY,
    model_id INT,
    year INT,
    price DECIMAL(10, 2),
    color VARCHAR(50),
    vin VARCHAR(50),
    stock_status VARCHAR(20),
    mileage INT
);

INSERT INTO Cars VALUES
(1,1,2020,20000, 'White', 'JTMZK33V876009879', 'In Stock', 15000),
(2,2,2021,35000, 'Red', '1ZVBP8CF6B5123456', 'Sold', 5000),
(3,3,2022,80000, 'Black', '5YJSA1E25FF100001', 'In Stock', 1200),
(4,4,2019,18000, 'Blue', '19XFC2F59JE216546', 'In Stock', 12000),
(5,5,2021,40000, 'Yellow', '1G1F51R63J0108156', 'Sold', 7000),
```

Output

	car_id	model_id	year	price	color	vin	stock_status	mileage
▶	1	1	2020	20000.00	White	JTMZK33V876009879	In Stock	15000
	2	2	2021	35000.00	Red	1ZVBP8CF6B5123456	Sold	5000
	3	3	2022	80000.00	Black	5YJSA1E25FF100001	In Stock	1200
	4	4	2019	18000.00	Blue	19XFC2F59JE216546	In Stock	12000
	5	5	2021	40000.00	Yellow	1G1F51R63J0108156	Sold	7000
	6	6	2021	60000.00	Silver	WBAXH5C59F0K12345	In Stock	8000
	7	7	2020	45000.00	Black	WAUZZZF48HA123456	In Stock	9000
	8	8	2019	55000.00	White	WDBUF65J13B123456	Sold	12000
	9	9	2022	70000.00	Red	5YJ3E1EA1JF123456	In Stock	1500
	10	10	2018	32000.00	Blue	JTEDS41A582345678	Sold	18000

Table 3: **Customers**: Maintains customer profiles for tracking sales and service history.

```
-- 3. Customers Table
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone_number VARCHAR(20),
    address VARCHAR(200)
);

INSERT INTO Customers VALUES
(1, 'John', 'Doe', 'john.doe@example.com', '555-1234', '123 Elm St, Springfield'),
(2, 'Jane', 'Smith', 'jane.smith@example.com', '555-5678', '456 Oak St, Springfield'),
(3, 'Jim', 'Beam', 'jim.beam@example.com', '555-9876', '789 Pine St, Springfield'),
(4, 'Michael', 'Jordan', 'michael.jordan@example.com', '555-2345', '100 Basketball Ave, Chicago'),
(5, 'Emily', 'Clark', 'emily.clark@example.com', '555-3456', '500 Lakeview Rd, Springfield'),
```

Output

	customer_id	first_name	last_name	email	phone_number	address
▶	1	John	Doe	john.doe@example.com	555-1234	123 Elm St, Springfield
	2	Jane	Smith	jane.smith@example.com	555-5678	456 Oak St, Springfield
	3	Jim	Beam	jim.beam@example.com	555-9876	789 Pine St, Springfield
	4	Michael	Jordan	michael.jordan@example.com	555-2345	100 Basketball Ave, Chicago
	5	Emily	Clark	emily.clark@example.com	555-3456	500 Lakeview Rd, Springfield
	6	Robert	Downey	robert.downey@example.com	555-4567	987 Sunset Blvd, Los Angeles
	7	Jessica	Alba	jessica.alba@example.com	555-5679	231 Hollywood Blvd, Los Angeles

Table 4: **Salespersons**: Tracks the sales team members and their performance metrics.

```
-- 4. Salespersons Table
> CREATE TABLE Salespersons (
    salesperson_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone_number VARCHAR(20),
    hire_date DATE,
    total_sales INT
);

INSERT INTO Salespersons VALUES
(1, 'Alice', 'Brown', 'alice.brown@example.com', '555-1111', '2019-05-01', 50),
(2, 'Bob', 'Johnson', 'bob.johnson@example.com', '555-2222', '2020-06-15', 30),
(3, 'David', 'Clark', 'david.clark@example.com', '555-2345', '2021-04-12', 25),
(4, 'Sophia', 'Martinez', 'sophia.martinez@example.com', '555-6789', '2018-08-24', 45),
(5, 'James', 'Moore', 'james.moore@example.com', '555-5678', '2019-02-19', 60),
(6, 'William', 'Taylor', 'william.taylor@example.com', '555-7890', '2020-03-10', 30),
(7, 'Ava', 'Lee', 'ava.lee@example.com', '555-2347', '2022-05-14', 15),
```

## Output

	salesperson_id	first_name	last_name	email	phone_number	hire_date	total_sales
▶	1	Alice	Brown	alice.brown@example.com	555-1111	2019-05-01	50
	2	Bob	Johnson	bob.johnson@example.com	555-2222	2020-06-15	30
	3	David	Clark	david.clark@example.com	555-2345	2021-04-12	25
	4	Sophia	Martinez	sophia.martinez@example.com	555-6789	2018-08-24	45
	5	James	Moore	james.moore@example.com	555-5678	2019-02-19	60
	6	William	Taylor	william.taylor@example.com	555-7890	2020-03-10	30
	7	Ava	Lee	ava.lee@example.com	555-2347	2022-05-14	15

Table 5: **Sales**: Records details of completed car sales transactions.

```
-- 5. Sales Table
> CREATE TABLE Sales (
    sale_id INT PRIMARY KEY,
    car_id INT,
    customer_id INT,
    salesperson_id INT,
    sale_date DATE,
    sale_price DECIMAL(10, 2),
    payment_method VARCHAR(50),
    FOREIGN KEY (car_id) REFERENCES Cars(car_id),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
    FOREIGN KEY (salesperson_id) REFERENCES Salespersons(salesperson_id)
);

INSERT INTO Sales VALUES
(1, 2, 1, 1, '2021-08-21', 35000.00, 'Cash'),
(2, 3, 2, 2, '2022-01-15', 80000.00, 'Financing'),
(3, 1, 4, 1, '2023-03-18', 20000.00, 'Cash'),
(4, 5, 6, 2, '2023-04-10', 40000.00, 'Credit Card'),
(5, 4, 7, 1, '2023-05-25', 18000.00, 'Cash'),
(6, 7, 8, 3, '2022-06-15', 42000.00, 'Financing'),
(7, 9, 9, 4, '2023-01-10', 70000.00, 'Cash'),
(8, 8, 10, 5, '2021-09-21', 54000.00, 'Credit Card'),
(9, 6, 11, 6, '2022-12-13', 62000.00, 'Cash'),
(10, 14, 12, 7, '2023-02-14', 18000.00, 'Financing'),
```

## Output

	sale_id	car_id	customer_id	salesperson_id	sale_date	sale_price	payment_method
▶	1	2	1	1	2021-08-21	35000.00	Cash
	2	3	2	2	2022-01-15	80000.00	Financing
	3	1	4	1	2023-03-18	20000.00	Cash
	4	5	6	2	2023-04-10	40000.00	Credit Card
	5	4	7	1	2023-05-25	18000.00	Cash
	6	7	8	3	2022-06-15	42000.00	Financing
	7	9	9	4	2023-01-10	70000.00	Cash
	8	8	10	5	2021-09-21	54000.00	Credit Card
	9	6	11	6	2022-12-13	62000.00	Cash
	10	14	12	7	2023-02-14	18000.00	Financing

Table 6: **Suppliers**: Manages supplier information for parts and cars.

```
-- 6. Suppliers Table
CREATE TABLE Suppliers (
    supplier_id INT PRIMARY KEY,
    supplier_name VARCHAR(100),
    contact_name VARCHAR(50),
    email VARCHAR(100),
    phone_number VARCHAR(20),
    address VARCHAR(200)
);

INSERT INTO Suppliers VALUES
(1, 'AutoSupply Inc.', 'Tom White', 'tom.white@autosupply.com', '555-3333', '500 Industrial Rd, Cityville'),
(2, 'Parts4U', 'Sue Green', 'sue.green@parts4u.com', '555-4444', '123 Commerce Ave, Cityville'),
(3, 'Global Auto Parts', 'John Doe', 'john.doe@globalauto.com', '555-9876', '789 Industrial Ave, Detroit'),
(4, 'Prime Auto Supplies', 'Alice Johnson', 'alice.johnson@primeauto.com', '555-3456', '456 Commerce Blvd, Houston'),
(5, 'Vehicle Source Inc.', 'Mike Brown', 'mike.brown@vehiclesource.com', '555-8765', '123 Business St, Los Angeles'),
(6, 'Reliable Car Parts', 'Nancy White', 'nancy.white@reliableparts.com', '555-4321', '999 Supply Rd, Miami'),
```

## Output

	supplier_id	supplier_name	contact_name	email	phone_number	address
▶	1	AutoSupply Inc.	Tom White	tom.white@autosupply.com	555-3333	500 Industrial Rd, Cityville
	2	Parts4U	Sue Green	sue.green@parts4u.com	555-4444	123 Commerce Ave, Cityville
	3	Global Auto Parts	John Doe	john.doe@globalauto.com	555-9876	789 Industrial Ave, Detroit
	4	Prime Auto Supplies	Alice Johnson	alice.johnson@primeauto.com	555-3456	456 Commerce Blvd, Houston
	5	Vehicle Source Inc.	Mike Brown	mike.brown@vehiclesource.com	555-8765	123 Business St, Los Angeles
	6	Reliable Car Parts	Nancy White	nancy.white@reliableparts.com	555-4321	999 Supply Rd, Miami
	7	AutoNation Suppliers	Tom Green	tom.green@autonation.com	555-5670	78 Manufacturing Dr, Chicago

Table 7: **Inventory**: Tracks car purchases, arrival dates, suppliers, and locations.

```
-- 7. Inventory Table
CREATE TABLE Inventory (
    inventory_id INT PRIMARY KEY,
    car_id INT,
    purchase_date DATE,
    purchase_price DECIMAL(10, 2),
    supplier_id INT,
    quantity INT,
    arrival_date DATE,
    location VARCHAR(100),
    FOREIGN KEY (car_id) REFERENCES Cars(car_id),
    FOREIGN KEY (supplier_id) REFERENCES Suppliers(supplier_id)
);
```



**INSERT INTO Inventory VALUES**

```
(1, 1, '2020-01-20', 18000.00, 1, 10, '2020-01-25', 'Lot 1'),
(2, 3, '2022-02-10', 75000.00, 2, 5, '2022-02-15', 'Lot 2'),
(3, 2, '2021-07-18', 32000.00, 1, 8, '2021-07-20', 'Lot 3'),
(4, 4, '2023-01-12', 17000.00, 1, 15, '2023-01-18', 'Lot 4'),
(5, 5, '2023-03-05', 39000.00, 2, 7, '2023-03-10', 'Lot 5'),
(6, 3, '2023-05-30', 78000.00, 2, 3, '2023-06-02', 'Lot 6'),
(7, 7, '2021-07-14', 43000.00, 3, 12, '2021-07-20', 'Lot 7'),
(8, 9, '2022-09-18', 68000.00, 4, 5, '2022-09-23', 'Lot 8'),
(9, 8, '2021-03-10', 52000.00, 5, 10, '2021-03-15', 'Lot 9'),
(10, 6, '2020-12-24', 59000.00, 6, 15, '2020-12-30', 'Lot 10'),
```

**Output**

	inventory_id	car_id	purchase_date	purchase_price	supplier_id	quantity	arrival_date	location
▶	1	1	2020-01-20	18000.00	1	10	2020-01-25	Lot 1
	2	3	2022-02-10	75000.00	2	5	2022-02-15	Lot 2
	3	2	2021-07-18	32000.00	1	8	2021-07-20	Lot 3
	4	4	2023-01-12	17000.00	1	15	2023-01-18	Lot 4
	5	5	2023-03-05	39000.00	2	7	2023-03-10	Lot 5
	6	3	2023-05-30	78000.00	2	3	2023-06-02	Lot 6
	7	7	2021-07-14	43000.00	3	12	2021-07-20	Lot 7
	8	9	2022-09-18	68000.00	4	5	2022-09-23	Lot 8
	9	8	2021-03-10	52000.00	5	10	2021-03-15	Lot 9
	10	6	2020-12-24	59000.00	6	15	2020-12-30	Lot 10
	11	14	2019-11-10	17000.00	7	20	2019-11-15	Lot 11
	12	11	2022-01-15	24000.00	3	6	2022-01-20	Lot 12

**Table 8: Technicians:** Lists service technicians with their specialties.

```
-- 8. Technicians Table
CREATE TABLE Technicians (
    technician_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone_number VARCHAR(20),
    specialty VARCHAR(50)
);

INSERT INTO Technicians VALUES
(1, 'Mike', 'Wrench', 'mike.wrench@carservice.com', '555-5555', 'Engine Repair'),
(2, 'Sam', 'Fixer', 'sam.fixer@carservice.com', '555-6666', 'Electrical'),
(3, 'Emma', 'Stone', 'emma.stone@carservice.com', '555-7890', 'Brake Service'),
(4, 'Michael', 'Scott', 'michael.scott@carservice.com', '555-6789', 'Air Filter Change'),
(5, 'Rachel', 'Green', 'rachel.green@carservice.com', '555-8901', 'Transmission Repair'),
(6, 'Ross', 'Geller', 'ross.geller@carservice.com', '555-1236', 'Suspension Repair'),
(7, 'Monica', 'Geller', 'monica.geller@carservice.com', '555-2347', 'Tire Services');
```

## Output

	technician_id	first_name	last_name	email	phone_number	specialty
►	1	Mike	Wrench	mike.wrench@carservice.com	555-5555	Engine Repair
	2	Sam	Fixer	sam.fixer@carservice.com	555-6666	Electrical
	3	Emma	Stone	emma.stone@carservice.com	555-7890	Brake Service
	4	Michael	Scott	michael.scott@carservice.com	555-6789	Air Filter Change
	5	Rachel	Green	rachel.green@carservice.com	555-8901	Transmission Repair
	6	Ross	Geller	ross.geller@carservice.com	555-1236	Suspension Repair
	7	Monica	Geller	monica.geller@carservice.com	555-2347	Tire Services

Table 9: **Services:** Logs service history for cars, including costs and technicians involved.

```
-- 9. Services Table
```

```
CREATE TABLE Services (
  service_id INT PRIMARY KEY,
  car_id INT,
  customer_id INT,
  service_date DATE,
  service_type VARCHAR(50),
  cost DECIMAL(10, 2),
  technician_id INT,
  FOREIGN KEY (car_id) REFERENCES Cars(car_id),
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
  FOREIGN KEY (technician_id) REFERENCES Technicians(technician_id)
);
```

```
INSERT INTO Services VALUES
```

```
(1, 1, 1, '2021-03-01', 'Oil Change', 50.00, 1),
(2, 2, 2, '2022-04-12', 'Tire Rotation', 30.00, 2),
(3, 3, 3, '2022-08-10', 'Brake Inspection', 100.00, 1),
(4, 4, 4, '2023-02-20', 'Battery Replacement', 150.00, 2),
(5, 5, 5, '2023-07-01', 'Transmission Repair', 400.00, 1),
(6, 1, 6, '2023-08-15', 'Oil Change', 50.00, 1),
(7, 2, 7, '2023-09-05', 'Engine Repair', 500.00, 2),
(8, 7, 8, '2022-07-20', 'Tire Replacement', 150.00, 2),
(9, 9, 9, '2023-03-11', 'Battery Replacement', 180.00, 1),
(10, 8, 10, '2021-08-05', 'Brake Service', 120.00, 3),
```

## Output

	service_id	car_id	customer_id	service_date	service_type	cost	technician_id
▶	1	1	1	2021-03-01	Oil Change	50.00	1
	2	2	2	2022-04-12	Tire Rotation	30.00	2
	3	3	3	2022-08-10	Brake Inspection	100.00	1
	4	4	4	2023-02-20	Battery Replacement	150.00	2
	5	5	5	2023-07-01	Transmission	Transmission Repair	
	6	1	6	2023-08-15	Oil Change	50.00	1
	7	2	7	2023-09-05	Engine Repair	500.00	2
	8	7	8	2022-07-20	Tire Replacement	150.00	2
	9	9	9	2023-03-11	Battery Replacement	180.00	1
	10	8	10	2021-08-05	Brake Service	120.00	3

Table 10: **Finance Options:** Details financing plans available to customers.

```
-- 10. Finance Options Table
CREATE TABLE FinanceOptions (
    finance_id INT PRIMARY KEY,
    finance_company_name VARCHAR(100),
    interest_rate DECIMAL(5, 2),
    term_length INT,
    down_payment_required DECIMAL(10, 2),
    credit_score_required INT
);

INSERT INTO FinanceOptions VALUES
(1, 'Bank of Auto', 2.5, 60, 5000.00, 700),
(2, 'Finance4U', 3.0, 48, 4000.00, 680),
(3, 'AutoCredit Union', 3.2, 72, 6000.00, 720),
(4, 'VehicleLoans Corp.', 4.0, 60, 4000.00, 680),
(5, 'National Car Finance', 2.8, 48, 7000.00, 750),
(6, 'PrimeAuto Loans', 3.5, 36, 5000.00, 700),
(7, 'EasyAuto Finance', 4.2, 60, 4500.00, 660);
```

## Output

	finance_id	finance_company_name	interest_rate	term_length	down_payment_required	credit_score_required
▶	1	Bank of Auto	2.50	60	5000.00	700
	2	Finance4U	3.00	48	4000.00	680
	3	AutoCredit Union	3.20	72	6000.00	720
	4	VehicleLoans Corp.	4.00	60	4000.00	680
	5	National Car Finance	2.80	48	7000.00	750
	6	PrimeAuto Loans	3.50	36	5000.00	700
	7	EasyAuto Finance	4.20	60	4500.00	660

Table 11: **Orders:** Records orders placed with suppliers for inventory replenishment.

```
-- 11. Orders Table
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    supplier_id INT,
    car_id INT,
    order_date DATE,
    expected_delivery_date DATE,
    quantity INT,
    status VARCHAR(50),
    FOREIGN KEY (supplier_id) REFERENCES Suppliers(supplier_id),
    FOREIGN KEY (car_id) REFERENCES Cars(car_id)
);
```



```
INSERT INTO Orders VALUES
(1, 1, 1, '2023-02-01', '2023-02-20', 20, 'Delivered'),
(2, 2, 3, '2023-05-10', '2023-05-25', 10, 'Pending'),
(3, 1, 2, '2022-01-15', '2022-02-05', 15, 'Delivered'),
(4, 2, 4, '2023-03-25', '2023-04-10', 12, 'Delivered'),
(5, 1, 5, '2023-07-01', '2023-07-18', 25, 'Pending'),
(6, 2, 3, '2023-08-05', '2023-08-22', 8, 'Delivered'),
(7, 1, 1, '2023-09-10', '2023-09-30', 18, 'Pending'),
(8, 3, 7, '2023-07-10', '2023-07-20', 15, 'Delivered'),
(9, 4, 9, '2022-08-15', '2022-08-28', 10, 'Pending'),
(10, 5, 8, '2021-06-25', '2021-07-05', 12, 'Delivered'),
```

### Output

	order_id	supplier_id	car_id	order_date	expected_delivery_date	quantity	status
▶	1	1	1	2023-02-01	2023-02-20	20	Delivered
	2	2	3	2023-05-10	2023-05-25	10	Pending
	3	1	2	2022-01-15	2022-02-05	15	Delivered
	4	2	4	2023-03-25	2023-04-10	12	Delivered
	5	1	5	2023-07-01	2023-07-18	25	Pending
	6	2	3	2023-08-05	2023-08-22	8	Delivered
	7	1	1	2023-09-10	2023-09-30	18	Pending
	8	3	7	2023-07-10	2023-07-20	15	Delivered
	9	4	9	2022-08-15	2022-08-28	10	Pending
	10	5	8	2021-06-25	2021-07-05	12	Delivered

Table 12: **Car Features:** Captures additional features and upgrades available for cars.

```
-- 12. Car Features Table
CREATE TABLE CarFeatures (
    feature_id INT PRIMARY KEY,
    car_id INT,
    feature_name VARCHAR(100),
    feature_description VARCHAR(255),
    cost DECIMAL(10, 2),
    FOREIGN KEY (car_id) REFERENCES Cars(car_id)
);
```

```
INSERT INTO CarFeatures VALUES
(1, 1, 'Sunroof', 'Electric sunroof with tilt and slide feature', 1200.00),
(2, 3, 'Navigation System', 'Built-in GPS with real-time traffic updates', 2000.00),
(3, 7, 'Leather Seats', 'Premium leather seats with heating options', 1500.00),
(4, 9, 'Sports Package', 'Enhanced suspension and sporty aesthetics', 3000.00),
(5, 8, 'Navigation System', 'Integrated GPS with live traffic updates', 2000.00),
(6, 6, 'Sunroof', 'Panoramic sunroof with tilt and slide feature', 1800.00),
(7, 14, 'Automatic Parking Assist', 'Assists in parallel and perpendicular parking with minimal driver input', 2000.00),
(8, 11, 'Remote Start', 'Start the vehicle remotely using a smartphone app', 800.00),
(9, 12, 'Blind Spot Monitor', 'Alerts you to vehicles in your blind spot using radar sensors', 1000.00),
(10, 13, 'Ventilated Seats', 'Seats with ventilation for added comfort in hot weather', 1500.00),
```

### Output

	feature_id	car_id	feature_name	feature_description	cost
▶	1	1	Sunroof	Electric sunroof with tilt and slide feature	1200.00
	2	3	Navigation System	Built-in GPS with real-time traffic updates	2000.00
	3	7	Leather Seats	Premium leather seats with heating options	1500.00
	4	9	Sports Package	Enhanced suspension and sporty aesthetics	3000.00
	5	8	Navigation System	Integrated GPS with live traffic updates	2000.00
	6	6	Sunroof	Panoramic sunroof with tilt and slide feature	1800.00
	7	14	Automatic Parking Assist	Assists in parallel and perpendicular parking wit...	2000.00
	8	11	Remote Start	Start the vehicle remotely using a smartphone app	800.00
	9	12	Blind Spot Monitor	Alerts you to vehides in your blind spot using ra...	1000.00
	10	13	Ventilated Seats	Seats with ventilation for added comfort in hot ...	1500.00

Table 13: **Payments:** Tracks payment details for car sales, including payment methods.

```
-- 13. Payments Table
CREATE TABLE Payments (
    payment_id INT PRIMARY KEY,
    sale_id INT,
    amount_paid DECIMAL(10, 2),
    payment_date DATE,
    payment_type VARCHAR(50),
    FOREIGN KEY (sale_id) REFERENCES Sales(sale_id)
);

INSERT INTO Payments VALUES
(1, 1, 35000.00, '2021-08-21', 'Cash'),
(2, 2, 20000.00, '2022-01-15', 'Bank Transfer'),
(3, 3, 20000.00, '2023-03-18', 'Credit Card'),
(4, 4, 10000.00, '2023-04-11', 'Credit Card'),
(5, 5, 18000.00, '2023-05-25', 'Cash'),
(6, 6, 42000.00, '2022-06-15', 'Credit Card'),
(7, 7, 70000.00, '2023-01-10', 'Bank Transfer'),
(8, 8, 54000.00, '2021-09-21', 'Cash'),
(9, 9, 62000.00, '2022-12-13', 'Financing'),
(10, 10, 18000.00, '2023-02-14', 'Debit Card'),
```

## Output

	payment_id	sale_id	amount_paid	payment_date	payment_type
►	1	1	35000.00	2021-08-21	Cash
	2	2	20000.00	2022-01-15	Bank Transfer
	3	3	20000.00	2023-03-18	Credit Card
	4	4	10000.00	2023-04-11	Credit Card
	5	5	18000.00	2023-05-25	Cash
	6	6	42000.00	2022-06-15	Credit Card
	7	7	70000.00	2023-01-10	Bank Transfer
	8	8	54000.00	2021-09-21	Cash
	9	9	62000.00	2022-12-13	Financing
	10	10	18000.00	2023-02-14	Debit Card

**SQL/NOSQL Scripts:**

## SQL Queries (28):

Name: Vijay Venkatesan

1. Count the Total Number of Cars in Inventory that are still 'In Stock'. This helps the dealership track how many cars are ready for sale, aiding inventory management and sales forecasting.

```
533 #Count the Total Number of Cars in Inventory That Are Still 'In Stock'
534 • SELECT COUNT(*) AS TotalInStockCars
535 FROM Cars
536 WHERE stock_status = 'In Stock';
537
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	TotalInStockCars			
▶	62			

2. Find the Maximum Price of Cars in a Specific Engine Type (e.g., Electric). This provides insights into high-value vehicles in specific categories, supporting targeted marketing and pricing strategies.

```
538 #Find the Maximum Price of Cars in a Specific Engine Type
539 • SELECT MAX(price) AS MaxPrice
540 FROM Cars c
541 JOIN CarModels cm ON c.model_id = cm.model_id
542 WHERE cm.engine_type = 'Electric';
543
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	MaxPrice			
▶	80000.00			

3. Calculate the Average Sale Price of Cars Sold in 2023. This offers a benchmark for recent sales performance, helping the dealership evaluate pricing effectiveness and revenue trends

```
544 #Calculate the Average Sale Price of Cars Sold in 2023.
545 • SELECT AVG(sale_price) AS AvgSalePrice2023
546 FROM Sales
547 WHERE YEAR(sale_date) = 2023;
548
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	AvgSalePrice2023			
▶	42111.111111			

- Find the Total Revenue Generated by Salespersons Who Sold More Than 5 Cars. This identifies top-performing salespersons, enabling the dealership to reward success and strategize sales training.

```

549 #Find the Total Revenue Generated by Salespersons Who Sold More Than 5 Cars.
550 • SELECT s.salesperson_id, SUM(s.sale_price) AS TotalRevenue
551 FROM Sales s
552 GROUP BY s.salesperson_id
553 HAVING COUNT(s.sale_id) > 5;
554

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	salesperson_id	TotalRevenue		
▶	1	139000.00		

**Name:** Ishan Waghmare

- List All Cars Priced Above \$50,000, Ordered by Price. This highlights premium vehicles, allowing the dealership to focus on high-value sales opportunities and customer targeting.

```

555 #List All Cars Priced Above $50,000, Ordered by Price
556 • SELECT car_id, model_id, price, color
557 FROM Cars
558 WHERE price > 50000
559 ORDER BY price DESC;
560

```

Result Grid		Filter Rows:	Edit:	Export/Im
	car_id	model_id	price	color
▶	25	25	250000.00	Blue
	19	19	120000.00	Red
	24	24	120000.00	Black
	16	16	110000.00	Black
	23	23	110000.00	White
	18	18	105000.00	Silver
	17	17	100000.00	White
	22	22	95000.00	Gray
	20	20	90000.00	Blue
	21	21	85000.00	Green
	3	3	80000.00	Black

2. Find the Number of Cars Sold by Each Salesperson, Ordered by Total Cars Sold. This helps assess individual salesperson performance, informing incentives and resource allocation.

```

561  #Find the Number of Cars Sold by Each Salesperson, Ordered by Total Cars Sold
562  •  SELECT salesperson_id, COUNT(sale_id) AS CarsSold
563      FROM Sales
564      GROUP BY salesperson_id
565      ORDER BY CarsSold DESC;
566

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	salesperson_id	CarsSold		
▶	1	6		
	2	5		
	3	4		
	4	4		
	5	4		
	7	2		
	6	2		
	8	1		
	9	1		
	10	1		



3. List All Cars Sold by a Specific Salesperson (e.g., ID = 3). This tracks detailed sales history for individual employees, useful for performance reviews and customer follow-ups.


```


566
567  #List All Cars Sold by a Specific Salesperson
568  •  SELECT s.sale_id, c.car_id, c.vin, s.sale_price, s.sale_date
569      FROM Sales s
570      JOIN Cars c ON s.car_id = c.car_id
571      WHERE s.salesperson_id = 3
572      ORDER BY s.sale_date DESC;

```

Result Grid

  Filter Rows:

Export: 

Wrap Cell Content: 

	sale_id	car_id	vin	sale_price	sale_date
▶	27	9	5YJ3E1EA1JF123456	21000.00	2024-03-21
	22	12	1GNFK13027J123456	22500.00	2024-03-16
	16	6	WBAXH5C59F0K12345	59000.00	2023-07-21
	6	7	WAUZZZF48HA123456	42000.00	2022-06-15

- Find the Total Number of Cars Sold Per Year, Grouped by Year. This provides long-term sales trends, supporting strategic planning and market analysis.

573

574 #Find the Total Number of Cars Sold Per Year, Grouped by Year

575 • SELECT YEAR(s.sale\_date) AS SaleYear, COUNT(s.sale\_id) AS TotalCarsSold

576 FROM Sales s

577 GROUP BY SaleYear

578 ORDER BY SaleYear;

579

Result Grid		
	Filter Rows:	Export: Wrap Cell Content:
	SaleYear	TotalCarsSold
▶	2020	1
	2021	5
	2022	5
	2023	9
	2024	10

## Name: Yao-Hui Tseng

- List All Cars in Inventory that have yet to accumulate 8000 miles, Ordered by Their Mileage in Ascending Order. This helps identify low-mileage cars that may be more attractive to buyers, optimizing inventory promotion.

580 #List All Cars in Inventory that have mileage less than 8000, Ordered by Their Mileage in Ascending Order

581 • SELECT car\_id, vin, mileage

582 FROM Cars

583 WHERE mileage &lt; 8000

584 ORDER BY mileage ASC;

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	car_id	vin	mileage
	64	9SUBARUIMPREZAXXJPN123529	1300
	53	3JEEPRENEGADEXXJPN123518	1300
	54	4HYUNDAITUCSONXXJPN123519	1500
	35	0SUBARUCROSSTREKXXJPN123500	1500
	9	5YJ3E1EA1JF123456	1500
	40	5NISSANROGUEXXJPN123505	1500
	57	2HONDACIVICXXJPN123522	1500
	26	1FIATXXJPN123491	1500
	42	2HONDAFITXXJPN123507	1500



- Find the Total Cost of Features for a Specific Car (e.g., Car ID = 1). This provides cost breakdowns for customization, helping in pricing bundled features competitively.

```

586      #Find the Total Cost of Features for a Specific Car
587 •    SELECT car_id, SUM(cost) AS TotalFeatureCost
588      FROM CarFeatures
589      WHERE car_id = 1
590      GROUP BY car_id;

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content
	car_id	TotalFeatureCost		
▶	1	3500.00		

- Find the Minimum Sale Price for Cars Sold Using 'Cash' Payment Method. This evaluates cash sale patterns, supporting strategies to encourage cash or other preferred payment methods

```

592      #Find the Minimum Sale Price for Cars Sold Using 'Cash' Payment Method
593 •    SELECT MIN(s.sale_price) AS MinCashSalePrice
594      FROM Sales s
595      JOIN Payments p ON s.sale_id = p.sale_id
596      WHERE p.payment_type = 'Cash';
597

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	MinCashSalePrice			
▶	18000.00			

- Count the Number of Cars for Each Engine Type, Grouped by Engine Type. This helps understand inventory composition, supporting decisions on procurement and marketing for different car types.

```

598      #Count the Number of Cars for Each Engine Type, Grouped by Engine Type
599 •    SELECT engine_type, COUNT(model_id) AS CarCount
600      FROM CarModels
601      GROUP BY engine_type
602      ORDER BY CarCount DESC;
603

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	engine_type	CarCount		
▶	Gasoline	60		
	Electric	2		
	Hybrid	1		

**\* Nihal Sharma \***

-- query 1 -- Finding the top 3 customers based on total spendings at the dealership and listing details of each car they purchased.

[subquery total spendings of each customer, ordering by total spendings per customer/cars]

-- objective: salesperson can create customer relations with the best customers in the future.

```

477 • SELECT
478     CustomerSpending.Customer_Name,
479     c.make AS Manufacturer,
480     c.model AS Model,
481     c.year AS Year,
482     s.sale_price AS Sale_Price,
483     s.sale_date AS Sale_Date
484 FROM
485     (SELECT
486         cu.customer_id,
487         cu.first_name AS Customer_Name,
488         SUM(s.sale_price) AS Total_Spending
489     FROM
490         CarDealershipDataBase.Customers cu
491     JOIN
492         CarDealershipDataBase.Sales s ON cu.customer_id = s.customer_id
493     GROUP BY
494         cu.customer_id
495     ORDER BY
496         Total_Spending DESC
497     LIMIT 3) AS CustomerSpending
498 JOIN
499     CarDealershipDataBase.Sales s ON CustomerSpending.customer_id = s.customer_id
500 JOIN
501     CarDealershipDataBase.Cars c ON s.car_id = c.car_id
502 ORDER BY
503     CustomerSpending.Total_Spending DESC, s.sale_price DESC;

```

Output:

	Customer_Name	Manufacturer	Model	Year	Sale_Price	Sale_Date
▶	Amelia	BMW	X5	2021	62000.00	2022-12-13
	Amelia	Chevrolet	Tahoe	2021	22500.00	2024-03-16
	Evelyn	BMW	X5	2021	59000.00	2023-07-21
	Evelyn	Maserati	Quattroporte	2022	23000.00	2024-03-23
	Jane	Tesla	Model S	2022	80000.00	2022-01-15



-- query 2 -- Identifying the most frequent car color for each manufacturer/count descending  
 [having max color count grouped by make and color]  
 -- objective: salesperson realizes popular car color(s) based on demand for future ordering needs.

```

524 • SELECT
525     make AS Manufacturer,
526     color AS Most_Popular_Color,
527     COUNT(*) AS Count
528 FROM
529     CarDealershipDataBase.Cars
530 GROUP BY
531     make, color
532 HAVING
533     COUNT(*) = (
534         SELECT
535             MAX(color_count)
536         FROM (
537             SELECT
538                 make, color, COUNT(*) AS color_count
539             FROM
540                 CarDealershipDataBase.Cars
541             GROUP BY
542                 make, color
543         ) AS color
544         WHERE
545             color.make = Cars.make
546     )
547 ORDER BY
548     Count desc;
  
```

Output:

	Manufacturer	Most_Popular_Color	Count
▶	Ford	Red	2
	Chevrolet	Red	2
	Subaru	Orange	2
	Dodge	Black	2
	Chevrolet	Dark Green	2
	Hyundai	Blue	2
	Mercedes	White	1
	Tesla	Red	1
	Toyota	Blue	1
	Honda	Silver	1
	Nissan	Black	1
	Kia	Green	1
	Mercedes-Benz	Black	1
	BMW	White	1
	Audi	Silver	1
	Porsche	Red	1
	Lexus	Blue	1
	Jaguar	Green	1
	Tesla	Gray	1
	Land Rover	White	1
	Maserati	Black	1
	Bentley	Blue	1
	Fiat	Light Blue	1
	Mini	British Racing Green	1
	Honda	Dark Gray	1
	Nissan	Sunset Orange	1

-- query 3 -- Calculating the percentage of total revenue contributed by each manufacturer to get a distributed percentage of sales  
 [rounding sum of sales price as total revenue]  
 -- objective: dealership achieves an understanding of their sales percentages/ brand.

```

SELECT
  c.make AS Manufacturer,
  ROUND(SUM(s.sale_price), 2) AS Total_Revenue,
  ROUND((SUM(s.sale_price) / (SELECT SUM(sale_price) FROM CarDealershipDataBase.Sales)) * 100, 2) AS Revenue_Percentage
FROM
  CarDealershipDataBase.Cars c
JOIN
  CarDealershipDataBase.Sales s ON c.car_id = s.car_id
GROUP BY
  c.make
ORDER BY
  Total_Revenue DESC;

```

Output:

	Manufacturer	Total_Revenue	Revenue_Percentage
►	Tesla	271000.00	24.87
	BMW	164500.00	15.10
	Chevrolet	133500.00	12.25
	Mercedes	131500.00	12.07
	Audi	94000.00	8.63
	Kia	74000.00	6.79
	Ford	61000.00	5.60
	Toyota	54000.00	4.96
	Honda	47000.00	4.31
	Maserati	23000.00	2.11
	Nissan	18000.00	1.65
	Jaguar	18000.00	1.65

-- procedure (query 4) -- filtering car make, min/max sales price, year of sale to retrieve car sales data.

[4 input parameters, added dynamic filtering ability, defaulting ordering by sale date]

-- objective: providing an ability to the dealership/sales team to filter and retrieve car sales data based on inputs such as: the cars make, minimum sales price, maximum sales price, and the year of sale.

```

567 ● ○ CREATE PROCEDURE GetFilteredSalesData(
568     IN ManufacturerName VARCHAR(50),      -- Filter by car manufacturer (NULL for all manufacturers)
569     IN MinSalePrice DECIMAL(10,2),         -- Minimum sale price filter (NULL for no minimum)
570     IN MaxSalePrice DECIMAL(10,2),         -- Maximum sale price filter (NULL for no maximum)
571     IN SaleYear INT                        -- Year filter for sale date (NULL for all years)
572 )
573 ○ BEGIN
574     -- setting/getting data
575     SET @query = 'SELECT
576         c.make AS Manufacturer,
577         c.model AS Model,
578         c.year AS Year,
579         s.sale_price AS Sale_Price,
580         s.sale_date AS Sale_Date,
581         cu.first_name AS Customer_Name,
582         cu.email AS Customer_Email,
583         (SELECT ROUND(SUM(s2.sale_price), 2)
584          FROM CarDealershipDataBase.Sales s2
585          JOIN CarDealershipDataBase.Cars c2 ON s2.car_id = c2.car_id
586          WHERE c2.make = c.make) AS Total_Revenue,
587         (SELECT COUNT(*)
588          FROM CarDealershipDataBase.Sales s3
589          JOIN CarDealershipDataBase.Cars c3 ON s3.car_id = c3.car_id
590          WHERE c3.make = c.make) AS Total_Cars_Sold
591     FROM CarDealershipDataBase.Cars c
592     JOIN CarDealershipDataBase.Sales s ON c.car_id = s.car_id
593     JOIN CarDealershipDataBase.Customers cu ON s.customer_id = cu.customer_id
594     WHERE 1 = 1';
596     -- adding filters if input params are not null
597     IF ManufacturerName IS NOT NULL THEN
598         SET @query = CONCAT(@query, ' AND c.make = ', REPLACE(ManufacturerName, ' ', ' '), ');
599     END IF;
600
601     IF MinSalePrice IS NOT NULL THEN
602         SET @query = CONCAT(@query, ' AND s.sale_price >= ', MinSalePrice);
603     END IF;
604
605     IF MaxSalePrice IS NOT NULL THEN
606         SET @query = CONCAT(@query, ' AND s.sale_price <= ', MaxSalePrice);
607     END IF;
608
609     IF SaleYear IS NOT NULL THEN
610         SET @query = CONCAT(@query, ' AND YEAR(s.sale_date) = ', SaleYear);
611     END IF;
612
613     -- default ordering by sale date desc
614     SET @query = CONCAT(@query, ' ORDER BY s.sale_date DESC');
615
616     -- execution
617     PREPARE stmt FROM @query;
618     EXECUTE stmt;
619     DEALLOCATE PREPARE stmt;
620 END$$

```

Output (can call the proc in various ways depending on the requirement):

Example 1:

**call GetFilteredSalesData('Tesla', 25000, 40000, null);**

	Manufacturer	Model	Year	Sale_Price	Sale_Date	Customer_Name	Customer_Email	Total_Revenue	Total_Cars_Sold
►	Tesla	Model S	2022	27000.00	2024-03-19	Mason	mason.hernandez@example.com	271000.00	5

Example 2:

**call GetFilteredSalesData(null, null, null, null);**

	Manufacturer	Model	Year	Sale_Price	Sale_Date	Customer_Name	Customer_Email	Total_Revenue	Total_Cars_Sold
►	BMW	X5	2021	24000.00	2024-03-24	Liam	liam.perez@example.com	164500.00	4
	Maserati	Quattroporte	2022	23000.00	2024-03-23	Evelyn	evelyn.gonzalez@example.com	23000.00	1
	BMW	7 Series	2021	19500.00	2024-03-22	Isabella	isabella.lopez@example.com	164500.00	4
	Tesla	Model 3	2022	21000.00	2024-03-21	Mia	mia.martinez@example.com	271000.00	5
	Kia	Sorento	2018	25000.00	2024-03-20	Sophia	sophia.rodriguez@example.com	74000.00	3
	Tesla	Model S	2022	27000.00	2024-03-19	Mason	mason.hernandez@example.com	271000.00	5
	Mercedes	C-Class	2019	23500.00	2024-03-18	Henry	henry.martinez@example.com	131500.00	3
	Jaguar	XJ	2019	18000.00	2024-03-17	Lucas	lucas.garcia@example.com	18000.00	1
	Chevrolet	Tahoe	2021	22500.00	2024-03-16	Amelia	amelia.davis@example.com	133500.00	4
	Chevrolet	Camaro	2021	20000.00	2024-03-15	Charlotte	charlotte.miller@example.com	133500.00	4
	Audi	A4	2020	52000.00	2023-09-15	Liam	liam.perez@example.com	94000.00	2
	BMW	X5	2021	59000.00	2023-07-21	Evelyn	evelyn.gonzalez@example.com	164500.00	4
	Tesla	Model 3	2022	73000.00	2023-06-24	Aiden	aiden.johnson@example.com	271000.00	5
	Honda	Civic	2019	18000.00	2023-05-25	Jessica	jessica.alba@example.com	47000.00	2
	Honda	Accord	2019	29000.00	2023-04-12	Sophia	sophia.rodriguez@example.com	47000.00	2
	Chevrolet	Camaro	2021	40000.00	2023-04-10	Robert	robert.downey@example.com	133500.00	4
	Toyota	Corolla	2020	20000.00	2023-03-18	Michael	michael.jordan@example.com	54000.00	2
	Nissan	Altima	2017	18000.00	2023-02-14	Lucas	lucas.garcia@example.com	18000.00	1
	Tesla	Model 3	2022	70000.00	2023-01-10	James	james.williams@example.com	271000.00	5
	BMW	X5	2021	62000.00	2022-12-13	Amelia	amelia.davis@example.com	164500.00	4
	Chevrolet	Tahoe	2021	51000.00	2022-11-15	Mason	mason.hernandez@example.com	133500.00	4
	Audi	A4	2020	42000.00	2022-06-15	Olivia	olivia.brown@example.com	94000.00	2
	Mercedes	C-Class	2019	54000.00	2022-05-10	Ella	ella.thomas@example.com	131500.00	3
	Tesla	Model S	2022	80000.00	2022-01-15	Jane	jane.smith@example.com	271000.00	5
	Kia	Sorento	2018	24000.00	2021-11-11	Noah	noah.anderson@example.com	74000.00	3
	Mercedes	C-Class	2019	54000.00	2021-09-21	Charlotte	charlotte.miller@example.com	131500.00	3

## Name: Ninad Patil

**Query 1:** Analyze car stock status: List all models with cars that remain unsold for more than 2 years after manufacturing.

**Objective:** To check which car models are in the stock are old and send them back to the manufacturer to modify

**Query:**

**SELECT**

```

    c.vin,
    cm.make,
    cm.model,
    c.year AS manufacture_year,
    c.stock_status,
    DATEDIFF(CURDATE(), DATE(CONCAT(c.year, '-01-01')))/ 365 AS years_since_manufacture
FROM Cars c
JOIN CarModels cm ON c.model_id = cm.model_id
WHERE c.stock_status = 'In Stock'
AND DATEDIFF(CURDATE(), DATE(CONCAT(c.year, '-01-01')))/ 365 > 730;

```

**Output:**

Result Grid		Filter Rows:		Export:	Wrap Cell Content:	
	vin	make	model	manufacture_year	stock_status	years_since_manufacture
▶	JTMZK33V876009879	Toyota	Corolla	2020	In Stock	4.9315
	5YJSA1E25FF100001	Tesla	Model S	2022	In Stock	2.9288
	19XFC2F59JE216546	Honda	Civic	2019	In Stock	5.9315
	WBAXH5C59F0K12345	BMW	X5	2021	In Stock	3.9288
	WAUZZZF48HA123456	Audi	A4	2020	In Stock	4.9315
	5YJ3E1EA1JF123456	Tesla	Model 3	2022	In Stock	2.9288
	1FMCU9GX5HUA12345	Ford	Escape	2020	In Stock	4.9315
	1GNFK13027J123456	Chevrolet	Tahoe	2021	In Stock	3.9288
	1N4AL3AP8DC123456	Nissan	Altima	2017	In Stock	7.9315
	1MBEJXXJXPN123456	Mercedes-Benz	S-Class	2022	In Stock	2.9288
	2BMWJXXJXPN123457	BMW	7 Series	2021	In Stock	3.9288
	3AUDIXXJXPN123458	Audi	A8	2022	In Stock	2.9288
	4PORSCHEXXJXPN123...	Porsche	Panam...	2021	In Stock	3.9288
	5LEXUSXXJXPN123460	Lexus	LS	2020	In Stock	4.9315
	7TESLAXXJXPN123462	Land Rover	Range ...	2022	In Stock	2.9288
	8LANDROVERXXJXPN...	Maserati	Quattr...	2021	In Stock	3.9288
	9MASERATIXXJXPN12...	Bentley	Contin...	2022	In Stock	2.9288
	1FIATXXJXPN123491	Mini	Cooper	2022	In Stock	2.9288
	2MINIXXJXPN123492	Honda	HR-V	2021	In Stock	3.9288
	3HONDAHRVXXJXPN1...	Nissan	Kicks	2022	In Stock	2.9288
	4NISSANKICKSXXJXP...	Toyota	C-HR	2021	In Stock	3.9288
	5TOYOTACHRXXJXPN...	Hyundai	Venue	2022	In Stock	2.9288
	6HYUNDAIVENUEXXJX...	Kia	Soul	2022	In Stock	2.9288
	7KIASOULXXJXPN123...	Chevrolet	Spark	2021	In Stock	3.9288
	8CHEVYSXXJXPN123498	Mazda	CX-30	2022	In Stock	2.9288
	9MAZDACX30XXJXPN...	Subaru	Crosstrek	2021	In Stock	3.9288
	0SUBARUCROSSTREK...	Volkswagen	Tiguan	2022	In Stock	2.9288
	1VOLKSTIGUANXXJXP...	Chrysler	Pacifica	2021	In Stock	3.9288
	2CHRYSLERPACTICA	Dodge	Journey	2022	In Stock	2.9288

**Query 2:** 10 Cars with the highest profit margins



**Objective:** To get an idea which are the top 10 cars that generate the most amount of profit

**Query:**


```
SELECT
    cm.make,
    cm.model,
    c.year,
    s.sale_price,
    i.purchase_price,
    (s.sale_price - i.purchase_price) AS profit_margin
FROM Cars c
JOIN CarModels cm ON c.model_id = cm.model_id
JOIN Sales s ON c.car_id = s.car_id
JOIN Inventory i ON c.car_id = i.car_id
ORDER BY profit_margin DESC
LIMIT 10;
```


**Output:**

Result Grid

  Filter Rows:

Export:

 Wrap Cell Content:

 Fetch rows:

	make	model	year	sale_price	purchase_price	profit_margin
▶	Tesla	Model S	2022	80000.00	26000.00	54000.00
	Tesla	Model 3	2022	73000.00	20500.00	52500.00
	Tesla	Model 3	2022	70000.00	20500.00	49500.00
	BMW	X5	2021	62000.00	24500.00	37500.00
	BMW	X5	2021	59000.00	24500.00	34500.00
	Mercedes	C-Class	2019	54000.00	22500.00	31500.00
	Mercedes	C-Class	2019	54000.00	22500.00	31500.00
	Chevrolet	Tahoe	2021	51000.00	21500.00	29500.00
	Chevrolet	Camaro	2021	40000.00	18500.00	21500.00
	Audi	A4	2020	52000.00	43000.00	9000.00

**Query 3:** Find customers who have made multiple purchases

**Objective:** To find the customers who have bought more than 1 car to come up with some additional offers for them in the future

**Query:**

```
SELECT c.customer_id, c.first_name, c.last_name, COUNT(*) as purchase_count
FROM Customers c
JOIN Sales s ON c.customer_id = s.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name
HAVING COUNT(*) > 1;
```

**Output:**

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
customer_id	first_name	last_name	purchase_count
10	Charlotte	Miller	2
11	Amelia	Davis	2
12	Lucas	Garcia	2
13	Henry	Martinez	2
14	Mason	Hernandez	2
15	Sophia	Rodriguez	2
16	Mia	Martinez	2
17	Isabella	Lopez	2
18	Evelyn	Gonzalez	2
19	Liam	Perez	2

**Query 4:** Find the best-performing salesperson for each year (by sales revenue).

**Objective:** To find the salesperson who closed the most amounts of deal.

**Query:**

```
WITH YearlySales AS (
    SELECT
        sp.salesperson_id,
        sp.first_name,
        sp.last_name,
        YEAR(s.sale_date) AS sale_year,
        SUM(s.sale_price) AS total_revenue
    FROM Sales s
    JOIN Salespersons sp ON s.salesperson_id = sp.salesperson_id
    GROUP BY sp.salesperson_id, sp.first_name, sp.last_name, YEAR(s.sale_date)
)
SELECT
    ys.sale_year,
    ys.first_name,
    ys.last_name,
    ys.total_revenue
FROM YearlySales ys
JOIN (
    SELECT
        sale_year,
        MAX(total_revenue) AS max_revenue
    FROM YearlySales
    GROUP BY sale_year
) max_revenue_per_year
ON ys.sale_year = max_revenue_per_year.sale_year AND ys.total_revenue = max_revenue_per_year.max_revenue;
```

**Output:**

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
sale_year	first_name	last_name	total_revenue
2021	Alice	Brown	60000.00
2023	Sophia	Martinez	122000.00
2024	James	Moore	51000.00
2022	William	Taylor	116000.00
2020	Daniel	Harris	26000.00



Name: **Anamika Das**

Query 1: To calculate the total sales by summing up the sales for all cars using a stored procedure

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'cardealershipdatabase' is expanded, showing various tables and views. The 'Stored Procedures' folder is selected. The main pane displays the following SQL code:

```

542
543 DELIMITER //
544 CREATE PROCEDURE CalculateTotalSales()
545 BEGIN
546     DECLARE total_sales DECIMAL(15, 2);
547     SELECT SUM(sale_price) INTO total_sales
548     FROM Sales;
549     SELECT total_sales AS TotalSales;
550 END //
551 DELIMITER ;
552
553 CALL CalculateTotalSales();
554
555

```

Below the code editor, the 'Result Grid' is shown with the following data:

TotalSales
1089500.00

The 'Schema: cardealershipdatabase' is indicated at the bottom left.

Query 2: The total amount paid for each sale, along with the finance company and interest rate, where the payment exceeds 5000

The screenshot shows the SQL Server Enterprise Manager interface with a complex SQL query entered in the query editor. The query is as follows:

```

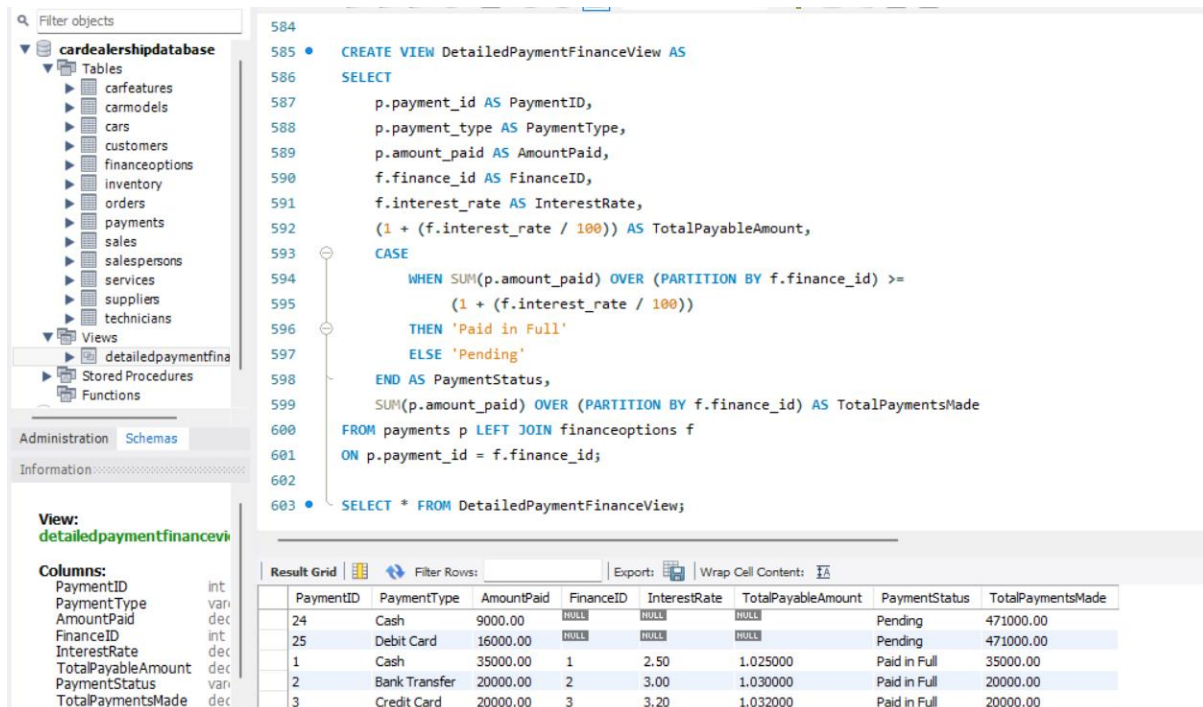
591 SELECT
592     S.sale_id,
593     SUM(P.amount_paid) AS TotalAmountPaid,
594     F.finance_company_name,
595     F.interest_rate
596 FROM
597     Sales S JOIN
598     Payments P ON S.sale_id = P.sale_id
599 JOIN
600     FinanceOptions F ON S.sale_id = F.finance_id
601 GROUP BY
602     S.sale_id, F.finance_company_name, F.interest_rate
603 HAVING
604     SUM(P.amount_paid) > 5000 -- Only include sales where total amount paid is greater than 5000
605 ORDER BY
606     TotalAmountPaid DESC;
607

```

Below the query editor, the 'Result Grid' is shown with the following data:

sale_id	TotalAmountPaid	finance_company_name	interest_rate
7	95000.00	EasyAuto Finance	4.20
1	50000.00	Bank of Auto	2.50
6	47000.00	PrimeAuto Loans	3.50
2	42000.00	Finance4U	3.00
3	38000.00	AutoCredit Union	3.20
5	30000.00	National Car Finance	3.00

**Query 3:** The query creates a view that combines detailed information about payment types, amounts paid, and financing details like pending or paid for cars purchased



The screenshot shows a database IDE with a left sidebar containing a tree view of the 'card dealership database' schema. The 'Views' section is expanded, showing 'detailedpaymentfinancview'. The main editor displays the SQL code for creating and selecting from this view. The 'Result Grid' at the bottom shows the output of the query, which includes columns for PaymentID, PaymentType, AmountPaid, FinanceID, InterestRate, TotalPayableAmount, PaymentStatus, and TotalPaymentsMade. The results are sorted by TotalPayableAmount in descending order.

```

584
585 • CREATE VIEW DetailedPaymentFinanceView AS
586 SELECT
587     p.payment_id AS PaymentID,
588     p.payment_type AS PaymentType,
589     p.amount_paid AS AmountPaid,
590     f.finance_id AS FinanceID,
591     f.interest_rate AS InterestRate,
592     (1 + (f.interest_rate / 100)) AS TotalPayableAmount,
593     CASE
594         WHEN SUM(p.amount_paid) OVER (PARTITION BY f.finance_id) >=
595             (1 + (f.interest_rate / 100))
596         THEN 'Paid in Full'
597         ELSE 'Pending'
598     END AS PaymentStatus,
599     SUM(p.amount_paid) OVER (PARTITION BY f.finance_id) AS TotalPaymentsMade
600 FROM payments p LEFT JOIN financeoptions f
601 ON p.payment_id = f.finance_id;
602
603 • SELECT * FROM DetailedPaymentFinanceView;
  
```

PaymentID	PaymentType	AmountPaid	FinanceID	InterestRate	TotalPayableAmount	PaymentStatus	TotalPaymentsMade
24	Cash	9000.00	NULL	NULL	NULL	Pending	471000.00
25	Debit Card	16000.00	NULL	NULL	NULL	Pending	471000.00
1	Cash	35000.00	1	2.50	1.025000	Paid in Full	35000.00
2	Bank Transfer	20000.00	2	3.00	1.030000	Paid in Full	20000.00
3	Credit Card	20000.00	3	3.20	1.032000	Paid in Full	20000.00

**Query 4:** Calculating the total revenue and average sale price for cars sold by each manufacturer from highest to lowest

```

SELECT
    c.make AS Manufacturer,
    COUNT(s.sale_id) AS Total_Cars_Sold,
    ROUND(SUM(s.sale_price), 2) AS Total_Revenue,
    ROUND(AVG(s.sale_price), 2) AS Average_Sale_Price
FROM
    CarDealershipDataBase.Cars c
JOIN
    CarDealershipDataBase.Sales s ON c.car_id = s.car_id
WHERE
    s.sale_date IS NOT NULL
GROUP BY
    c.make
ORDER BY
    Total_Revenue DESC;
  
```

	Manufacturer	Total_Cars_Sold	Total_Revenue	Average_Sale_Price
►	Tesla	5	271000.00	54200.00
	BMW	4	164500.00	41125.00
	Chevrolet	4	133500.00	33375.00
	Mercedes	3	131500.00	43833.33
	Audi	2	94000.00	47000.00
	Kia	3	74000.00	24666.67
	Ford	2	61000.00	30500.00
	Toyota	2	54000.00	27000.00
	Honda	2	47000.00	23500.00
	Maserati	1	23000.00	23000.00
	Nissan	1	18000.00	18000.00
	Jaguar	1	18000.00	18000.00

## Krish Sonani

### Query 1

Identify Underperforming Salespersons

Find salespersons who sold cars with an average sale price below the overall average sale price.

```
SELECT sp.salesperson_id,
       CONCAT(sp.first_name, ' ', sp.last_name) AS salesperson_name,
       AVG(s.sale_price) AS avg_sale_price
FROM Sales s
JOIN Salespersons sp ON s.salesperson_id = sp.salesperson_id
GROUP BY sp.salesperson_id, sp.first_name, sp.last_name;
```

	salesperson_id	salesperson_name	avg_sale_price
►	1	Alice Brown	23166.666667
	2	Bob Johnson	39800.000000
	3	David Clark	36125.000000
	4	Sophia Martinez	41250.000000
	5	James Moore	44500.000000
	6	William Taylor	58000.000000
	7	Ava Lee	21000.000000
	8	Daniel Harris	26000.000000
	9	Lucas Wilson	51000.000000
	10	Emma White	29000.000000

### Query 2 Cars with Most Frequent Services

Objective: Retrieve the cars that have undergone the highest number of services, along with their service types and counts.

```
SELECT c.car_id,  
       cm.make,  
       cm.model,  
       serv.service_type,  
       COUNT(serv.service_id) AS service_count  
FROM Services serv  
JOIN Cars c ON serv.car_id = c.car_id  
JOIN CarModels cm ON c.model_id = cm.model_id  
GROUP BY c.car_id, cm.make, cm.model, serv.service_type  
ORDER BY service_count DESC  
LIMIT 5;
```

	car_id	make	model	service_type	service_count
▶	1	Toyota	Corolla	Oil Change	2
	2	Ford	Mustang	Tire Rotation	1
	3	Tesla	Model S	Brake Inspection	1
	4	Honda	Civic	Battery Replacement	1
	5	Chevrolet	Camaro	Transmission Repair	1

### Query 3 Cars Performance Classification

Objective: Classify cars based on their stock status, mileage, and sales performance into categories like High Demand, Average Demand, and Low Demand



```

SELECT c.car_id,
       cm.make,
       cm.model,
       c.mileage,
       c.stock_status,
       COUNT(s.sale_id) AS total_sales,
       CASE
         WHEN c.stock_status = 'In Stock' AND c.mileage < 15000 AND COUNT(s.sale_id) > 5 THEN 'High Demand'
         WHEN c.stock_status = 'In Stock' AND c.mileage BETWEEN 15000 AND 30000 THEN 'Average Demand'
         WHEN c.stock_status = 'Out of Stock' AND COUNT(s.sale_id) = 0 THEN 'Low Demand'
         ELSE 'Moderate Demand'
       END AS performance_category
FROM Cars c
JOIN CarModels cm ON c.model_id = cm.model_id
LEFT JOIN Sales s ON c.car_id = s.car_id
GROUP BY c.car_id, cm.make, cm.model, c.mileage, c.stock_status
ORDER BY performance_category DESC, total_sales DESC;

```

	car_id	make	model	mileage	stock_status	total_sales	performance_category
▶	9	Tesla	Model 3	1500	In Stock	3	Moderate Demand
	6	BMW	X5	8000	In Stock	3	Moderate Demand
	8	Mercedes	C-Class	12000	Sold	3	Moderate Demand
	15	Kia	Sorento	11500	Sold	3	Moderate Demand
	3	Tesla	Model S	1200	In Stock	2	Moderate Demand
	5	Chevrolet	Camaro	7000	Sold	2	Moderate Demand
	7	Audi	A4	9000	In Stock	2	Moderate Demand
	12	Chevrolet	Tahoe	8000	In Stock	2	Moderate Demand
	2	Ford	Mustang	5000	Sold	1	Moderate Demand
	4	Honda	Civic	12000	In Stock	1	Moderate Demand
	17	BMW	7 Series	3000	In Stock	1	Moderate Demand
	10	Toyota	Highlan...	18000	Sold	1	Moderate Demand
	11	Ford	Escape	7500	In Stock	1	Moderate Demand
	13	Honda	Accord	10000	Sold	1	Moderate Demand
	14	Nissan	Altima	6500	In Stock	1	Moderate Demand
	24	Bentley	Contine...	1000	In Stock	1	Moderate Demand
	21	Jaguar	XJ	15000	Out of Stock	1	Moderate Demand
	16	Mercede...	S-Class	5000	In Stock	0	Moderate Demand
	69	Nissan	Altima	900	In Stock	0	Moderate Demand
	18	Audi	A8	2500	In Stock	0	Moderate Demand
	19	Porsche	Panamera	4000	In Stock	0	Moderate Demand
	20	Lexus	LS	7000	In Stock	0	Moderate Demand
	22	Land Rover	Range ...	2000	In Stock	0	Moderate Demand
	23	Maserati	Quattro...	6000	In Stock	0	Moderate Demand
	70	Dodge	Journey	1100	In Stock	0	Moderate Demand

#### Query 4 Cars with Most Profitability Over Time

Objective: Find the cars that generated the highest profit over multiple sales and include total profit per car.

```
SELECT c.car_id,  
       cm.make,  
       cm.model,  
       SUM(s.sale_price - inv.purchase_price) AS total_profit  
FROM Sales s  
JOIN Cars c ON s.car_id = c.car_id  
JOIN CarModels cm ON c.model_id = cm.model_id  
JOIN Inventory inv ON c.car_id = inv.car_id  
GROUP BY c.car_id, cm.make, cm.model  
ORDER BY total_profit DESC  
LIMIT 5;
```

	car_id	make	model	total_profit
►	9	Tesla	Model 3	62500.00
	6	BMW	X5	39500.00
	8	Mercedes	C-Class	39500.00
	15	Kia	Sorento	10000.00
	7	Audi	A4	8000.00

---

### NoSQL Scripts/Queries:

NoSQL databases are non-relational database for handling unstructured, semi-structured, or structured data. They provide high flexibility, scalability, and performance.

#### Key Features:

**Schema-less:** No fixed schema; data can evolve over time.

**Types:** Includes Document Stores, Key-Value Stores, Column Stores, and Graph Databases.

**High Performance:** Optimized for specific use cases like handling large volumes of data.

In a car dealership management system, the database is also implemented using the MongoDB platform with Python programming language. Below are the queries and connection strings:

## NoSQL Database and Collections:

```
[1]: !pip install pymongo

Requirement already satisfied: pymongo in c:\users\danam\anaconda3\lib\site-packages (4.8.0)
Requirement already satisfied: dnspython<3.0.0, >=1.16.0 in c:\users\danam\anaconda3\lib\site-packages (from pymongo) (2.4.2)

[156]: from pymongo.mongo_client import MongoClient

uri = "mongodb+srv://axd240045:e0KP5QndVMqG3u8d@cluster0.ztex8.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"
# Create a new client and connect to the server
client = MongoClient(uri)
# Send a ping to confirm a successful connection
try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
except Exception as e:
    print(e)

Pinged your deployment. You successfully connected to MongoDB!

[158]: db = client["CarDealershipDatabase"]
cars = db["cars"]
services = db["services"]
sales = db["sales"]
inventory = db["inventory"]

print(db.list_collection_names())

['cars', 'services', 'sales', 'inventory']

cars.insert_many(cars_data)

[8]: InsertManyResult([ObjectId('6750e49ba07ee9b74479edc8'), ObjectId('6750e49ba07ee9b74479edc9'), ObjectId('6750e49ba07ee9b74479edca'), ObjectId('6750e49ba07ee9b74479edcb'), ObjectId('6750e49ba07ee9b74479edcc'), ObjectId('6750e49ba07ee9b74479edcd'), ObjectId('6750e49ba07ee9b74479edce'), ObjectId('6750e49ba07ee9b74479edcf'), ObjectId('6750e49ba07ee9b74479edd0'), ObjectId('6750e49ba07ee9b74479edd1'), ObjectId('6750e49ba07ee9b74479edd2'), ObjectId('6750e49ba07ee9b74479edd3'), ObjectId('6750e49ba07ee9b74479edd4'), ObjectId('6750e49ba07ee9b74479edd5'), ObjectId('6750e49ba07ee9b74479edd6'), ObjectId('6750e49ba07ee9b74479edd7'), ObjectId('6750e49ba07ee9b74479edd8'), ObjectId('6750e49ba07ee9b74479edd9'), ObjectId('6750e49ba07ee9b74479edda'), ObjectId('6750e49ba07ee9b74479eddb'), ObjectId('6750e49ba07ee9b74479eddc'), ObjectId('6750e49ba07ee9b74479eddd'), ObjectId('6750e49ba07ee9b74479edde'), ObjectId('6750e49ba07ee9b74479eddf'), ObjectId('6750e49ba07ee9b74479ede0'), ObjectId('6750e49ba07ee9b74479ede1'), ObjectId('6750e49ba07ee9b74479ede2'), ObjectId('6750e49ba07ee9b74479ede3'), ObjectId('6750e49ba07ee9b74479ede4'), ObjectId('6750e49ba07ee9b74479ede5'), ObjectId('6750e49ba07ee9b74479ede6'), ObjectId('6750e49ba07ee9b74479ede7'), ObjectId('6750e49ba07ee9b74479ede8'), ObjectId('6750e49ba07ee9b74479ede9'), ObjectId('6750e49ba07ee9b74479edea'), ObjectId('6750e49ba07ee9b74479edeb'), ObjectId('6750e49ba07ee9b74479edec'), ObjectId('6750e49ba07ee9b74479eded'), ObjectId('6750e49ba07ee9b74479edee'), ObjectId('6750e49ba07ee9b74479edef'), ObjectId('6750e49ba07ee9b74479edf0'), ObjectId('6750e49ba07ee9b74479edf1'), ObjectId('6750e49ba07ee9b74479edf2'), ObjectId('6750e49ba07ee9b74479edf3'), ObjectId('6750e49ba07ee9b74479edf4'), ObjectId('6750e49ba07ee9b74479edf5'), ObjectId('6750e49ba07ee9b74479edf6'), ObjectId('6750e49ba07ee9b74479edf7'), ObjectId('6750e49ba07ee9b74479edf8'), ObjectId('6750e49ba07ee9b74479edf9'), ObjectId('6750e49ba07ee9b74479edfa'), ObjectId('6750e49ba07ee9b74479edfb'), ObjectId('6750e49ba07ee9b74479edfc'), ObjectId('6750e49ba07ee9b74479edfd'), ObjectId('6750e49ba07ee9b74479edfe'), ObjectId('6750e49ba07ee9b74479edff'), ObjectId('6750e49ba07ee9b74479ee00'), ObjectId('6750e49ba07ee9b74479ee01'), ObjectId('6750e49ba07ee9b74479ee02'), ObjectId('6750e49ba07ee9b74479ee03'), ObjectId('6750e49ba07ee9b74479ee04'), ObjectId('6750e49ba07ee9b74479ee05'), ObjectId('6750e49ba07ee9b74479ee06'), ObjectId('6750e49ba07ee9b74479ee07'), ObjectId('6750e49ba07ee9b74479ee08'), ObjectId('6750e49ba07ee9b74479ee09')], acknowledged=True)

[88]: # Access the 'car' collection from the database
cars = db['cars']

# Fetching all documents in the 'car' collection
for car_variable in cars.find():
    print(car_variable)

{'_id': ObjectId('6750e49ba07ee9b74479edc8'), 'car_id': 1, 'year': 2020, 'price': 20000.0, 'color': 'White', 'vin': 'JTMZK33V876009879', 'stock_status': 'Available', 'mileage': 15000, 'model': {'model_id': 1, 'make': 'Toyota', 'model': 'Corolla', 'engine_type': 'Gasoline'}}
{'_id': ObjectId('6750e49ba07ee9b74479edc9'), 'car_id': 2, 'year': 2021, 'price': 35000.0, 'color': 'Red', 'vin': '1ZVBP8CF6B5123456', 'stock_status': 'Sold', 'mileage': 5000, 'model': {'model_id': 2, 'make': 'Ford', 'model': 'Mustang', 'engine_type': 'Gasoline'}}
{'_id': ObjectId('6750e49ba07ee9b74479edca'), 'car_id': 3, 'year': 2022, 'price': 80000.0, 'color': 'Black', 'vin': '5YJSA1E25FF100001', 'stock_status': 'In Stock', 'mileage': 1200, 'model': {'model_id': 3, 'make': 'Tesla', 'model': 'Model S', 'engine_type': 'Electric'}}
{'_id': ObjectId('6750e49ba07ee9b74479edcb'), 'car_id': 4, 'year': 2019, 'price': 18000.0, 'color': 'Blue', 'vin': '19XFC2F59J216546', 'stock_status': 'In Stock', 'mileage': 12000, 'model': {'model_id': 4, 'make': 'Honda', 'model': 'Civic', 'engine_type': 'Gasoline'}}
{'_id': ObjectId('6750e49ba07ee9b74479edcc'), 'car_id': 5, 'year': 2021, 'price': 40000.0, 'color': 'Yellow', 'vin': '1G1F51R63J0108156', 'stock_status': 'Sold', 'mileage': 7000, 'model': {'model_id': 5, 'make': 'Chevrolet', 'model': 'Camaro', 'engine_type': 'Gasoline'}}
{'_id': ObjectId('6750e49ba07ee9b74479edcd'), 'car_id': 6, 'year': 2021, 'price': 60000.0, 'color': 'Silver', 'vin': 'WBAH5C59F0K12345', 'stock_status': 'In Stock', 'mileage': 8000, 'model': {'model_id': 6, 'make': 'BMW', 'model': 'X5', 'engine_type': 'Gasoline'}}
{'_id': ObjectId('6750e49ba07ee9b74479edce'), 'car_id': 7, 'year': 2020, 'price': 45000.0, 'color': 'Black', 'vin': 'WAUZZZF48HA123456', 'stock_status': 'In Stock', 'mileage': 9000, 'model': {'model_id': 7, 'make': 'Audi', 'model': 'A4', 'engine_type': 'Gasoline'}}
{'_id': ObjectId('6750e49ba07ee9b74479edcf'), 'car_id': 8, 'year': 2019, 'price': 55000.0, 'color': 'White', 'vin': 'WDBUF65J13B123456', 'stock_status': 'Sold', 'mileage': 12000, 'model': {'model_id': 8, 'make': 'Mercedes', 'model': 'C-Class', 'engine_type': 'Gasoline'}}
```

## NoSQL Queries (7) by each team member:

Name: Krish Sonani

## Query: Update a Single Document of Cars Collection by its unique VIN number

```

•[164]: # Query 1: Update a Single Document of Cars Collection by its unique VIN number
cars.update_one(
    {"vin": "JTMZK33V876009879"}, # Filter to find the document with vin
    {"$set": {"mileage": "24000"}} # Update the 'mileage' field 15000 to "24000"
)

[164]: UpdateResult({'n': 1, 'electionId': ObjectId('7fffffff0000000000000001ba'), 'opTime': {'ts': Timestamp(1733425510, 98), 't': 442}, 'nModified': 1, 'ok': 1.0, '$clusterTime': {'clusterTime': Timestamp(1733425510, 98), 'signature': {'hash': b'\xd3\x9c\xe8[V\x16\x92\xbf\xe1w\xa1WA\xbd\xe0+7p\xcc', 'keyId': 7389728499646332949}}, 'operationTime': Timestamp(1733425510, 98), 'updatedExisting': True), acknowledged=True)

[166]: # Fetch the updated document
updated_car = cars.find_one({"vin": "JTMZK33V876009879"})
print(updated_car)

{'_id': ObjectId('6750e49ba07ee9b74479edc8'), 'car_id': 1, 'year': 2020, 'price': 20000.0, 'color': 'White', 'vin': 'JTMZK33V876009879', 'stock_status': 'Available', 'mileage': '24000', 'model': {'model_id': 1, 'make': 'Toyota', 'model': 'Corolla', 'engine_type': 'Gasoline'}}

```

Name: Ishan Waghmare

## Query: Delete one document specified by objectID from Cars Collection

```

•[106]: # Query 2: Delete one document specified by objectID from Cars Collection
from bson import ObjectId # Ensure ObjectId is imported

# Specify the _id of the car document to delete
car_id = ObjectId("6750e49ba07ee9b74479edca")

result = cars.delete_one({"_id": car_id})

# Check if the document was deleted
if result.deleted_count > 0:
    print("Car deleted successfully.")
else:
    print("No car found with the specified _id.")

```

Car deleted successfully.

Name: Ninad Patil

## Query: Filters records where the location is "Lot 6", groups them according to the same location, and sums the total quantity for that location

```

•[178]: # Query 3: Filters records where the location is "Lot 6", groups them according to the same location, and sums the total quantity for that location

pipeline = [{"$match": {'location': 'Lot 6'}},
            {'$group': {'_id': '$location', 'total_quantity': {'$sum': '$quantity'}}}]

result = inventory.aggregate(pipeline)
for doc in result:
    print(doc)

{'_id': 'Lot 6', 'total_quantity': 10}

```

Name: Nihal Sharma

## Query: Count the total number of cars by year in ascending order using the Aggregate function



•[168]: # Query 4: Count the total number of cars by year in ascending order using the Aggregate function

```
pipeline = [
    {"$group": {"_id": "$year", "count": {"$sum": 1}}}, # Group by year and count cars
    {"$sort": {"_id": 1}} # Optional: Sort the results by year in ascending order
]

result = cars.aggregate(pipeline)

# Print the results
for doc in result:
    print(f"Year: {doc['_id']], Count: {doc['count']}]")
```

```
Year: 2017, Count: 1
Year: 2018, Count: 2
Year: 2019, Count: 4
Year: 2020, Count: 4
Year: 2021, Count: 27
Year: 2022, Count: 27
```

Name: **Vijay Venkatesan**

Query: Retrieve car data along with their associated sales information, and include cars even if they have no sales

•[170]: # Query 5: Retrieve car data along with their associated sales information, and include cars even if they have no sales using \$lookup

```
pipeline = [
    {
        "$lookup": {
            "from": "sales", # The name of the collection to join (sales)
            "localField": "car_id", # Field in the cars collection
            "foreignField": "car_id", # Field in the sales collection that references car_id
            "as": "sales_info" # The name of the new array field in the cars collection containing matching sales data
        }
    },
    {
        "$unwind": {
            "path": "$sales_info", # Path to the array field created by $lookup
            "preserveNullAndEmptyArrays": True # If no matching sales, include the car document with null sales
        }
    }
]

result = cars.aggregate(pipeline)

# Print the result of the join
for doc in result:
    print(doc)
```

```
{'_id': ObjectId('6750e49ba07ee9b74479edc8'), 'car_id': 1, 'year': 2020, 'price': 20000.0, 'color': 'White', 'vin': 'JTMZK33V876009879', 'stock_status': 'Available', 'mileage': '24000', 'model': {'model_id': 1, 'make': 'Toyota', 'model': 'Corolla', 'engine_type': 'Gasoline'}, 'sales_info': {'_id': ObjectId('6750e5bba07ee9b74479ee1d'), 'sale_id': 3, 'car_id': 1, 'customer_id': 4, 'sale_date': '3/18/23', 'price': 20000.0, 'payment_method': 'Cash', 'salesperson': {'salesperson_id': 1, 'first_name': 'Alice', 'last_name': 'Brown', 'email': 'alice.brown@example.com', 'phone': '555-1111', 'hire_date': '5/1/19', 'total_sales': 50}}}
{'_id': ObjectId('6750e49ba07ee9b74479edc9'), 'car_id': 2, 'year': 2021, 'price': 35000.0, 'color': 'Red', 'vin': '1ZVBP8CF6B5123456', 'stock_status': 'Sold', 'mileage': 5000, 'model': {'model_id': 2, 'make': 'Ford', 'model': 'Mustang', 'engine_type': 'Gasoline'}, 'sales_info': {'_id': ObjectId('6750e49ba07ee9b74479ee1d'), 'sale_id': 3, 'car_id': 1, 'customer_id': 4, 'sale_date': '3/18/23', 'price': 20000.0, 'payment_method': 'Cash', 'salesperson': {'salesperson_id': 1, 'first_name': 'Alice', 'last_name': 'Brown', 'email': 'alice.brown@example.com', 'phone': '555-1111', 'hire_date': '5/1/19', 'total_sales': 50}}}
```

Name: **Anamika Das**

Query: Calculate the average service cost for each car, including the location, based on data from the services and inventory collection

```

•[172]: # Query 6: Calculate the average service cost for each car, including the location, based on data from the services na inventory collection
pipeline = [
    { "$lookup": {
        "from": "services",          # Join with the 'services' collection
        "localField": "car_id",      # Match 'car_id' in 'inventory'
        "foreignField": "car_id",    # Match 'car_id' in 'services'
        "as": "service_details"     # Add matching services as an array
    }},
    { "$unwind": {
        "path": "$service_details",  # Flatten the service details array
        "preserveNullAndEmptyArrays": False # Exclude cars with no services
    }},
    { "$group": {
        "_id": "$car_id",            # Group by car_id
        "avg_service_cost": { "$avg": "$service_details.cost" }, # Calculate average cost
        "location": { "$first": "$location" } # Include the car's location from inventory
    }},
    { "$sort": { "avg_service_cost": -1 } # Sort by average cost (descending)
  }]
result = inventory.aggregate(pipeline)
for doc in result:
    # Print the results
    print(f"Car ID: {doc['_id']], Location: {doc['location']], Avg Service Cost: {doc['avg_service_cost']}")

```

```

Car ID: 14, Location: Lot 11, Avg Service Cost: 700.0
Car ID: 6, Location: Lot 10, Avg Service Cost: 600.0
Car ID: 13, Location: Lot 14, Avg Service Cost: 500.0
Car ID: 5, Location: Lot 5, Avg Service Cost: 400.0
Car ID: 2, Location: Lot 3, Avg Service Cost: 265.0
Car ID: 15, Location: Lot 15, Avg Service Cost: 200.0
Car ID: 9, Location: Lot 8, Avg Service Cost: 180.0
Car ID: 7, Location: Lot 7, Avg Service Cost: 150.0

```

Name: **Yao-Hui Tseng**

Query: filters records based on 2 criteria: Price > 100 and records containing "Oil Change" or "Battery Replacement"

```

•[176]: # Query 8: It filters records based on 2 criteria: Price > 100 and records containing "Oil Change" or "Battery Replacement"
# as a service type

services_find = db.services.find({"$or": [{"service_type": {"$in": ["Oil Change", "Battery Replacement"]}}, {"cost": {"$gt": 100}}]})

for car in services_find:
    service_type = car["service_type"]
    price = car["cost"]
    print(f"Services: {service_type}, Price: {price}")

```

```

Services: Oil Change, Price: 50
Services: Battery Replacement, Price: 150
Services: Transmission Repair, Price: 400
Services: Oil Change, Price: 50
Services: Engine Repair, Price: 500
Services: Tire Replacement, Price: 150
Services: Battery Replacement, Price: 180
Services: Brake Service, Price: 120
Services: Engine Repair, Price: 600
Services: Transmission Repair, Price: 700
Services: Suspension Repair, Price: 500
Services: Coolant Flush, Price: 200

```