Application Example

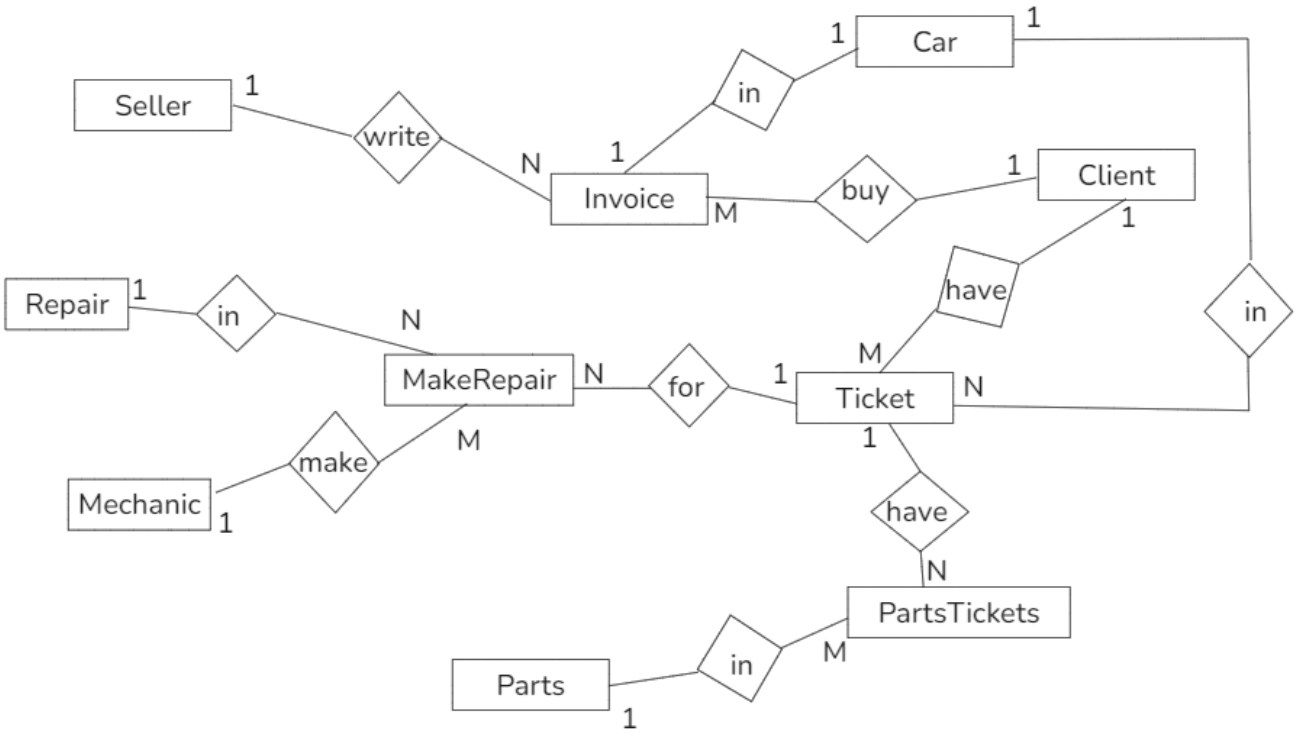
Create a Conceptual Data Model for a car dealership that sells new and used cars and operates a service shop.

The design is based on the following business rules:

- A **seller** can sell multiple cars, but each car is sold by only one seller
- A **customer** can buy multiple cars, but each car is purchased by only one customer
- A seller writes only one **invoice** for each car he sells
- A customer receives an invoice for each car they purchase
- A customer may come in simply to have their car repaired; they do not need to buy a car to be classified as a customer
- When a customer brings in one or more cars for repair or Repair, a **repair order** (ticket) is written for each car
- The dealership maintains a Repair history for each car serviced, referenced by the car's **serial number**
- A car brought in for servicing can be worked on by several mechanics, and each mechanic can work on several cars
- A car that is maintained may or may not need **parts** (e.g., adjusting a carburetor does not require new parts)

Solution: Entity-Relationship Diagram (Conceptual Model)

For simplicity, only entity types and their associations are described in the model. Attributes are defined separately



Possible Attributes

The attributes related to the relationships between entity types are not indicated here.

1. Car

- **Car_ID** (Primary Key): Unique identifier for each car
- **Serial_Number**: Unique serial number of the car
- **Brand**: Car manufactures brand
- **Model**: Specific car model
- **Color**: Car color
- **Year**: Year of manufacture
- **Car_ON**: Indicates whether the car is new or used
- **Price**: Base price to be used for selling the car

2. Seller

- **seller_ID** (Primary Key): Unique identifier for each seller
- **Last_name**: Seller's last name
- **First_name**: Seller's first name
- **Email**: Seller's email
- **Num_Ph**: Seller's phone number
- **Address**: Seller's home address

3. Client

- **Client_ID** (Primary Key): Unique identifier for each client

- **Last_Name**: Client's last name
- **First_name**: Customer's first name
- **Num_Ph**: Customer's phone number
- **Address**: Customer's address
- **City**: City of residence
- **Country**: Country of residence
- **Postal Code**: Residence postal code
- **Email**: Client's email address

4. Invoice

- **ID_Invoice** (Primary Key): Unique identifier for each invoice
- **Date**: Invoice date
- **Selling_price**: The price charged for the sale of the car (if it differs from the car's base price)

5. Ticket (Repair Order)

- **ID_Ticket** (Primary Key): Unique identifier for each Repair voucher.
- **Date_Receipt**: Date the car was received for service.
- **Date_Return**: Date the car was returned to the customer.
- **Price_charged**: Price charged to the customer for Repair (if it differs from the calculated price).

6. MakeRepair

- **Repair_ID** (Primary Key): Unique identifier for each Repair performed by a given mechanic for a given work order.
- **Hours**: Number of hours worked on the Repair in question.

7. Mechanic

- **Mechanic_ID** (Primary Key): Unique identifier for each mechanic.
- **Last_name**: Mechanic's last name.
- **First_name**: Mechanic's first name.
- **Email**: Mechanic's email.
- **Num_Ph**: Mechanic's phone number.
- **Address**: Mechanic's home address.

8. Repair

- **Repair_ID** (Primary Key): Unique identifier for each Repair.
- **Repair_name**: Service name.
- **Hourly_Rate**: Hourly rate for Repair.

9. Parts

- **ID_Part** (Primary Key): Unique identifier for each part
- **Description**: Description of the room (part)
- **Price**: Selling price of the part
- **Quantity_in_Stock**: Quantity in the stock
- Reorder_Threshold: Qunatity value indicating a need to make a new purchase.

10. PartsTickets

- **ID_Parts_Ticket** (Primary Key): Unique identifier for each piece used for a given voucher
- **Quantity**: Number of parts used
- **Price_charged**: Invoice price for the part used

Information and Knowledge Extraction

Examples of Information to Extract

- Total sales revenue.
- Number of cars sold by each salesperson (salesperson performance).
- Bestselling car models.
- Most frequently received car models for Repair.
- Most common Repair problems.
- Mechanics' performance.
- Customer purchase history.
- Customer Repair history.

Examples of Knowledge to be Deduced

- **Market trends**: Identify popular car models, customer preferences, and emerging trends.
- **Sales performance**: Evaluate salesperson performance, identify top salespeople, and optimize sales strategies.
- **After-sales service effectiveness**: Evaluate the effectiveness of Repair operations, identify bottlenecks, and improve service delivery.
- **Customer satisfaction**: Analyze Repair history to identify areas for improvement.
- **Inventory management**: Optimize inventory levels based on sales and Repair trends.

Related SQL Statements

The Creation of the Tables

```
CREATE TABLE Client (
```

```
    Client_ID       INT             PRIMARY KEY,
    Last_Name       VARCHAR(100)    NOT NULL,
    First_name      VARCHAR(100),
    Num_Ph         VARCHAR(20),
    Address         VARCHAR(255),
    City            VARCHAR(100),
    Country         VARCHAR(100),
    Postal_Code     VARCHAR(10),
    Email           VARCHAR(100)
);



CREATE TABLE Seller (
    Seller_ID       INT             PRIMARY KEY,
    Last_name            VARCHAR(100)    NOT NULL,
    First_name      VARCHAR(100),
    Email           VARCHAR(100),
    Num_Ph         VARCHAR(20),
    Address         VARCHAR(255)
);



CREATE TABLE Mechanic (
    Mechanic_ID     INT             PRIMARY KEY,
    Last_name            VARCHAR(100)    NOT NULL,
    First_name      VARCHAR(100),
    Email           VARCHAR(100),
    Num_Ph         VARCHAR(20),
    Address         VARCHAR(255)
);



CREATE TABLE Parts (
    ID_Part         INT             PRIMARY KEY,
    Description       VARCHAR(255)    NOT NULL,
    Price             DECIMAL(10, 2),
    Quantity_in_Stock INT               NOT NULL DEFAULT 0,
    Reorder_Threshold INT               NOT NULL DEFAULT 10,
);



CREATE TABLE Repair (
    Repair_ID  INT              PRIMARY KEY,
    Repair_name VARCHAR(255)   NOT NULL,
    Hourly_Rate     DECIMAL(10, 2)
```

```
);
```

```
CREATE TABLE Car (
    Car_ID          INT             PRIMARY KEY,
    Serial_Number   VARCHAR(50)     UNIQUE NOT NULL,
    Brand           VARCHAR(100),
    Model           VARCHAR(100),
    Color           VARCHAR(50),
    Year            INT,
    Car_ON          VARCHAR(5)      CHECK (Car_ON IN ('New', 'Used')),
    Price           DECIMAL(10, 2),

);
```

```
CREATE TABLE Invoice (
    ID_Invoice      INT             PRIMARY KEY,
    Date_Invoice    DATE,
    Selling_price   DECIMAL(10, 2),


    Car_ID          INT             UNIQUE NOT NULL,
    Seller_ID       INT             NOT NULL,
    Client_ID       INT             NOT NULL,

    FOREIGN KEY (Car_ID) REFERENCES Car(Car_ID),
    FOREIGN KEY (Seller_ID) REFERENCES Seller(Seller_ID),
    FOREIGN KEY (Client_ID) REFERENCES Client(Client_ID)
);
```

```
CREATE TABLE Ticket (
    ID_Ticket          INT             PRIMARY KEY,
    Date_Receipt    DATE,
    Date_Return     DATE,
    Price_charged   DECIMAL(10, 2),


    Client_ID       INT             NOT NULL,
    Car_ID          INT             NOT NULL,

    FOREIGN KEY (Client_ID) REFERENCES Client(Client_ID),
    FOREIGN KEY (Car_ID) REFERENCES Car(Car_ID)
);
```

```
CREATE TABLE MakeRepair (

    ID_Ticket          INT             NOT NULL,
    Mechanic_ID     INT             NOT NULL,
    Repair_ID  INT             NOT NULL,

    Hours           DECIMAL(5, 2),

    PRIMARY KEY (ID_Ticket, Mechanic_ID, Repair_ID),

    FOREIGN KEY (ID_Ticket) REFERENCES Ticket(ID_Ticket),
    FOREIGN KEY (Mechanic_ID) REFERENCES Mechanic(Mechanic_ID),
    FOREIGN KEY (Repair_ID) REFERENCES Repair(Repair_ID)
);


CREATE TABLE PartsTickets (
    ID_Parts_Ticket   INT             PRIMARY KEY,
    Quantity        INT             NOT NULL,
    Price_charged   DECIMAL(10, 2),


    ID_Part       INT             NOT NULL,
    ID_Ticket          INT             NOT NULL,

    FOREIGN KEY (ID_Part) REFERENCES Parts(ID_Part),
    FOREIGN KEY (ID_Ticket) REFERENCES Ticket(ID_Ticket)
);
```

Information Extractions

1. Total Sales Revenue

This query sums the `Selling_price` from all successful car sales recorded in the `Invoice` (Invoice) table.

```
SELECT
    SUM(Selling_price) AS Total_Sales_Revenue
FROM
    Invoice;
```

2. Number of Cars Sold by Each Salesperson

This query joins `Seller` (Seller) with `Invoice` (Invoice) and counts the number of invoices (sales) associated with each salesperson, ordered by the count (performance).

```sql
SELECT
    Seller.Seller_ID,
    Seller.First_name + ' ' + Seller.Last_name AS Salesperson_Name,
    COUNT(Invoice.ID_Invoice) AS Number_of_Cars_Sold
FROM
    Seller
INNER JOIN
    Invoice ON Seller.Seller_ID = Invoice.Seller_ID
GROUP BY
    Seller.Seller_ID, Seller.First_name, Seller.Last_name
ORDER BY
    Number_of_Cars_Sold DESC;
```

3. Bestselling Car Models

This query joins `Car` (Car) with `Invoice` to find which car models have the highest number of associated invoices (sales).

```sql
SELECT
    Car.Brand,
    Car.Model,
    COUNT(Invoice.Car_ID) AS Total_Units_Sold
FROM
    Invoice
INNER JOIN
    Car ON Invoice.Car_ID = Car.Car_ID
GROUP BY
    Car.Brand, Car.Model
ORDER BY
    Total_Units_Sold DESC;
```

4. Most Frequently Received Car Models for Repair

This query joins `Car` (Car) with `Ticket` (Repair Order) to count how many Repair orders have been created for each car model.

```sql
SELECT
    Car.Brand,
    Car.Model,
    COUNT(Ticket.ID_Ticket) AS Total_Repair_Orders
FROM
    Ticket
INNER JOIN
    Car  ON Ticket.Car_ID = Car.Car_ID
GROUP BY
    Car.Brand, Car.Model
```

```
ORDER BY
    Total_Repair_Orders DESC;
```

6. Most Common Repair Problems

This query uses the `MakeRepair` association table to count how many times each type of `Repair` (Service) has been performed across all orders.

```
SELECT
    Repair.Repair_name,
    COUNT(MakeRepair.ID_Ticket) AS Times_Performed
FROM
    Repair
INNER JOIN
    MakeRepair  ON Repair.Repair_ID = MakeRepair.Repair_ID
GROUP BY
    Repair.Repair_name
ORDER BY
    Times_Performed DESC;
```

7. Mechanics' Performance

This query calculates the total hours worked by each mechanic, providing a measure of their activity or utilization.

```
SELECT
    Mechanic.Mechanic_ID,
    Mechanic.First_name + ' ' + Mechanic.Last_name AS Mechanic_Name,
    SUM(MakeRepair.Hours) AS Total_Hours_Worked
FROM
    Mechanic
INNER JOIN
    MakeRepair ON Mechanic.Mechanic_ID = MakeRepair.Mechanic_ID
GROUP BY
    Mechanic.Mechanic_ID, Mechanic.First_name, Mechanic.Last_name
ORDER BY
    Total_Hours_Worked DESC;
```

8. Customer Purchase History

This query retrieves a detailed history of every car purchased by every customer, including the invoice details.

```
SELECT
    Client.Client_ID,
    Client.First_name + ' ' + Client.Last_Name AS Customer_Name,
    Invoice.ID_Invoice,
```

```
    Invoice.Date_Invoice,
    Invoice.Selling_price,
    Car.Brand,
    Car.Model,
    Seller.First_name + ' ' + Seller.Last_name AS Salesperson
FROM
    Client
INNER JOIN
    Invoice  ON Client.Client_ID = Invoice.Client_ID
INNER JOIN
    Car  ON Invoice.Car_ID = Car.Car_ID
INNER JOIN
    Seller  ON Invoice.Seller_ID = Seller.Seller_ID
ORDER BY
    Client.Client_ID, Invoice.Date_Invoice DESC;
```

Additional Queries and Knowledge Deduction

1. Inventory Management Queries

a. Parts Usage and per Order

This query calculates the total cost and of parts used for *each specific Repair order* (`Ticket`), providing the data needed to check against stock.

```
SELECT
    PartsTickets.ID_Ticket,
    SUM(PartsTickets.Quantity) AS Total_Parts_Used_Count,
    SUM(PartsTickets.Quantity * PartsTickets.Price_charged) AS
Total_Parts_Revenue_Per_Ticket
FROM
    PartsTickets
GROUP BY
    PartsTickets.ID_Ticket
ORDER BY
    PartsTickets.ID_Ticket;
```

b. Total of Each Part Used

This query helps the inventory manager determine which parts need reordering by summing the total used for *each unique part* across all Repair orders.

```
SELECT
    Parts.ID_Part,
    Parts.Description AS Part_Name,
    Parts.Price AS Part_Base_Price,
    SUM(PartsTickets.Quantity) AS Total_Quantity_Used
FROM
```

```
    Parts
INNER JOIN
    PartsTickets  ON Parts.ID_Part= PartsTickets.ID_Part
GROUP BY
    Parts.ID_Part, Parts.Description, Parts.Price
ORDER BY
    Total_Quantity_Used DESC;
```

c. Total Revenue Generated from Parts

This query calculates the total gross revenue generated from the sale of parts across all Repair orders.

```
SELECT
    SUM(PartsTickets.Quantity * PartsTickets.Price_charged) AS
Grand_Total_Parts_Revenue
FROM
    PartsTickets;
```

d. Inventory Management: Low Stock Alert

This query identifies all parts where the current `Quantity_in_Stock` is at or below the defined `Reorder_Threshold`, indicating an immediate need to purchase more.

```
SELECT
    ID_Part,
    Description AS Part_Name,
    Quantity_in_Stock,
    Reorder_Threshold,
    'REORDER REQUIRED' AS Status
FROM
    Parts
WHERE
    Quantity_in_Stock <= Reorder_Threshold
ORDER BY
    Quantity_in_Stock ASC;
```

2. Analyzing After-Sales Service Effectiveness

We will perform two key calculations:

1. **Total Labor Cost vs. Total Charged Price (per Order):** See if the price charged to the customer for the Repair order (`Ticket.Price_charged`) covers the internal cost of labor.
2. **Mechanic Utilization and Revenue:** Analyze how much labor revenue each mechanic is contributing.

a. Labor Cost and Profitability per Service Order

This query calculates the estimated internal **cost of labor** (Mechanic Hours × Repair Hourly Rate) for each Repair order (`Ticket`) and compares it to the final `Price_charged` to determine the service shop's profit margin on labor.

```
SELECT
    Ticket.ID_Ticket,
    Ticket.Date_Receipt,
    Car.Brand + ' ' + Car.Model AS Car_Serviced,
    Ticket.Price_charged AS Total_Price_Charged,


    SUM(MakeRepair.Hours * Repair.Hourly_Rate) AS Estimated_Labor_Cost,


    Ticket.Price_charged - SUM(MakeRepair.Hours * Repair.Hourly_Rate) AS
Labor_Profit_Margin
FROM
    Ticket
INNER JOIN
    Car ON Ticket.Car_ID = Car.Car_ID
INNER JOIN
    MakeRepair  ON Ticket.ID_Ticket = MakeRepair.ID_Ticket
INNER JOIN
    Repair ON MakeRepair.Repair_ID = Repair.Repair_ID
GROUP BY
    Ticket.ID_Ticket, Ticket.Date_Receipt, Car.Brand, Car.Model,
Ticket.Price_charged
ORDER BY
    Labor_Profit_Margin;
```

Derived Knowledge:

- **Profitability Check:** A negative `Labor_Profit_Margin` indicates the total price charged on that service order failed to cover the internal cost of the time spent, potentially pointing to underpricing or inefficient work.
- **Pricing Strategy:** Orders with a high margin might suggest opportunities to raise prices on specific services or that the estimated hours were generous.

b. Mechanic Contribution to Labor Value

This query calculates the total value of labor (based on the defined `Hourly_Rate` for the services they performed) contributed by each mechanic. This measures their productivity in terms of billable time value.

```
SELECT
    Mechanic.Mechanic_ID,
    Mechanic.First_name + ' ' + Mechanic.Last_name AS Mechanic_Name,
```

```
    SUM(MakeRepair.Hours) AS Total_Hours_Worked,


    SUM(MakeRepair.Hours * Repair.Hourly_Rate) AS
Total_Labor_Value_Contributed
FROM
    Mechanic
INNER JOIN
    MakeRepair  ON Mechanic.Mechanic_ID = MakeRepair.Mechanic_ID
INNER JOIN
    Repair  ON MakeRepair.Repair_ID = Repair.Repair_ID
GROUP BY
    Mechanic.Mechanic_ID, Mechanic.First_name, Mechanic.Last_name
ORDER BY
    Total_Labor_Value_Contributed DESC;
```

Derived Knowledge:

- **Performance Evaluation:** This directly quantifies the financial value each mechanic generates for the shop, which is essential for reviews, Ticketuses, and resource allocation.
- **Skill Utilization:** If a highly-paid mechanic is spending too much time on low-rate services, it might suggest a need to optimize scheduling based on skill and service type.

This set of queries allows you to move beyond simple counts (like "total hours worked") to analyze the **financial impact** of your service operations and the **efficiency** of individual employees.

3. Market Trends Analysis Queries

a. Bestselling Car Models (Total Sales)

This query identifies the *most popular* models by sheer sales volume.

```
SELECT
    Car.Brand,
    Car.Model,
    COUNT(Invoice.Car_ID) AS Total_Units_Sold
FROM
    Invoice
INNER JOIN
    Car  ON Invoice.Car_ID = Car.Car_ID
GROUP BY
    Car.Brand, Car.Model
ORDER BY
    Total_Units_Sold DESC;
```

b. Emerging Trends: Sales of New vs. Used Cars

This query tracks the preference between new and used vehicles, which is vital for optimizing inventory mix and marketing efforts.

```
SELECT
    Car_ON AS Condition,
    COUNT(Invoice.Car_ID) AS Total_Sales,
    SUM(Invoice.Selling_price) AS Total_Revenue
FROM
    Car
INNER JOIN
    Invoice ON Car.Car_ID = Invoice.Car_ID
GROUP BY
    Car_ON
ORDER BY
    Total_Sales DESC;
```

c. Customer Preference by Car Segment (Model Year Analysis)

This query helps identify whether customers prefer newer or older vehicles within the dealership's inventory, which guides purchasing and trade-in decisions.

```
SELECT
    Car.Year,
    COUNT(Invoice.Car_ID) AS Total_Sales,
    SUM(Invoice.Selling_price) AS Total_Revenue
FROM
    Car
INNER JOIN
    Invoice  ON Car.Car_ID = Invoice.Car_ID
GROUP BY
    Car.Year
ORDER BY
    Car.Year DESC;
```

d. Top Performing Salespeople by Revenue

Tracking revenue is critical for understanding the profitability of sales strategies and individual salesperson effectiveness.

```
SELECT
    Seller.Seller_ID,
    Seller.First_name + ' ' + Seller.Last_name AS Salesperson_Name,
    SUM(Invoice.Selling_price) AS Total_Revenue_Generated
FROM
    Seller
```

```
INNER JOIN
    Invoice ON Seller.Seller_ID = Invoice.Seller_ID
GROUP BY
    Seller.Seller_ID, Seller.First_name, Seller.Last_name
ORDER BY
    Total_Revenue_Generated DESC;
```

Knowledge Derived from Market Trends

By these queries, you can deduce the following points :

- **Market trends:** Identify popular models, preferences (New vs. Used), and emerging trends (e.g., whether older models are popular for reasons).
- **Sales performance:** Evaluate performance based on individual salesperson effectiveness allowing you to optimize sales .

References

- Watt, Adrienne, and Nelson Eng. *Database Design*. 2nd Edition. BCcampus, 2014