Ingoude Company

# Introduction

At Effortless Car Rentals, we rely on advanced database technology to keep everything running smoothly and to give our customers the best possible experience. Our database is the backbone of our operations, helping us manage everything from bookings to billing.

# Project Scope & Objectives

- Fleet Management: Our system keeps track of every car—where it is, when it's available, and when it needs maintenance. This means our customers always have reliable options.
- Customer Profiles: We store customer preferences and rental history so we can make their experience more personal, like remembering their favorite car or preferred pickup location.
- Real-Time Bookings: Our database powers a live reservation system, so customers always see accurate availability and pricing—no surprises.
- Billing and Payments: All transactions are secure and accurate, ensuring a seamless checkout process for our customers.
- Analytics and Insights: We use data to understand trends, improve services, and make smarter decisions for the future.

# Describing Problem Statement

The Key Problems:

1. Data Overload: Our systems struggle to manage and process the growing volume of data from bookings, fleet tracking, and customer preferences, leading to delays and inefficiencies.
2. Scalability: As we expand our fleet and customer base, the current system cannot handle the increased load without performance issues.
3. Lack of Integration: Disconnected systems for bookings, payments, and fleet management create inconsistencies and make real-time updates difficult.
4. High Costs: Maintaining outdated systems increases operational costs and reduces efficiency.

# Requirement Analysis

Core Entities:

- Customers: Storing customer information such as personal details, preferences, and rental history.
- Vehicles: Information on the fleet, including vehicle types, availability, condition, and maintenance records.
- Reservations: Data related to bookings, including pick-up and drop-off locations, rental duration, and pricing.
- Payments: Secure transaction data, such as payment methods, amounts, and billing history.
- Employee Management: Details about employees who manage the fleet, customer service, and operations.

Relationships Between Entities:

- Customers are linked to Reservations, which track which vehicle is rented.
- Reservations are associated with Vehicles, indicating which car was booked.
- Vehicles have maintenance schedules and records linked to each car’s ID.
- Payments are linked to Reservations to track transactions.

# Entity

Entities represent objects or concepts in the real world that have data stored about them in the system. In a car rental system, the main entities are:

- Rental Location
- Car
- Vehicle Details
- Car User
- Reservation
- Payment

# Attributes

Attributes are the characteristics / properties of the entities the real world objects In a car rental system, the main attributes are:

Car User

1. Attributes:

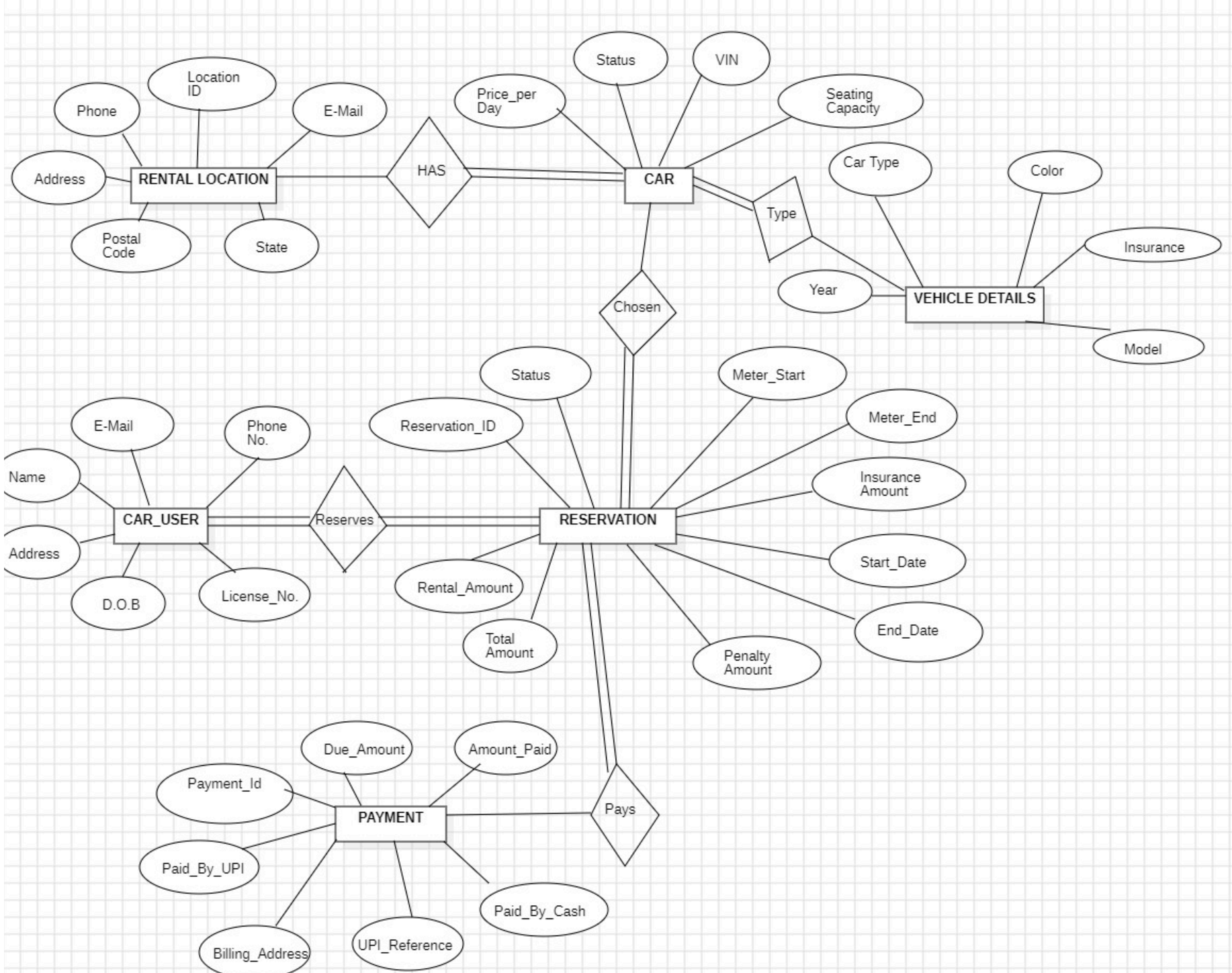
- Name
- Address
- E-Mail
- Phone No.
- License No.
- DOB

# Relationship

Relationships define how entities are connected to each other.

1. Customer and Rental:
    - Type: One-to-Many
    - A customer can make multiple rentals, but each rental is linked to only one customer.
2. Rental and Payment:
    - Type: One-to-One
    - Each rental transaction has exactly one payment, and each payment is tied to a single rental.

# ER Diagram


Location ID
Phone
E-Mail
Address
RENTAL LOCATION
Postal Code
State
HAS
Price_per Day
Status
VIN
Seating Capacity
CAR
Car Type
Color
Type
Insurance
Year
VEHICLE DETAILS
Model
Chosen
Status
Meter_Start
Reservation_ID
Meter_End
E-Mail
Phone No.
Name
CAR_USER
Reserves
RESERVATION
Insurance Amount
Address
Start_Date
D.O.B
License_No.
Rental_Amount
End_Date
Total Amount
Penalty Amount
Due_Amount
Amount_Paid
Payment_Id
PAYMENT
Pays
Paid_By_UPI
Paid_By_Cash
Billing_Address
UPI_Reference

# Creating a Database

```
Create Database db Effortless Car Rental
```

```
Use db_Effortless_Car_Rental
```

# Creating Tables

```sql
-- Create the CAR table
CREATE TABLE CAR (
    VIN VARCHAR(20) PRIMARY KEY,
    Status VARCHAR(50),
    Price_per_Day DECIMAL(10, 2),
    Seating_Capacity INT,  -- Moved here
    Location_ID INT,
    FOREIGN KEY (Location_ID) REFERENCES RENTAL_LOCATION(Location_ID)
);
```

```sql
-- Create the RESERVATION table
CREATE TABLE RESERVATION (
    Reservation_ID INT PRIMARY KEY,
    License_No VARCHAR(20),
    VIN VARCHAR(20),
    Start_Date DATE,
    End_Date DATE,
    Meter_Start INT,
    Meter_End INT,
    Rental_Amount DECIMAL(10, 2),
    Insurance_Amount DECIMAL(10, 2),
    Penalty_Amount DECIMAL(10, 2),
    Total_Amount DECIMAL(10, 2),
    Status VARCHAR(50),
    FOREIGN KEY (License_No) REFERENCES CAR_USER(License_No),
    FOREIGN KEY (VIN) REFERENCES CAR(VIN)
);
```

```sql
-- Create the VEHICLE_DETAILS table
CREATE TABLE VEHICLE_DETAILS (
    VIN VARCHAR(20) PRIMARY KEY,
    Car_Type VARCHAR(50),
    Year INT,
    Color VARCHAR(30),
    Model VARCHAR(50),
    Insurance DECIMAL(10, 2),
    FOREIGN KEY (VIN) REFERENCES CAR(VIN)
);
```

```sql
-- Create the RENTAL_LOCATION table
CREATE TABLE RENTAL_LOCATION (
    Location_ID INT PRIMARY KEY,
    Phone VARCHAR(15),
    Address VARCHAR(255),
    Postal_Code VARCHAR(10),
    State VARCHAR(50),
    Email VARCHAR(100)
);
```

# Inserting data into tables

```sql
-- Insert values into CAR_USER table
INSERT INTO CAR_USER (License_No, Name, Address, DOB, Phone_No, Email) VALUES
('LIC001', 'John Doe', '123 Main St', '1980-05-15', '123-456-7890', 'john.doe@example.com'),
('LIC002', 'Jane Smith', '456 Elm St', '1992-08-22', '987-654-3210', 'jane.smith@example.com'),
('LIC003', 'Alice Johnson', '789 Oak St', '1985-11-30', '555-123-4567', 'alice.johnson@example.com'),
('LIC004', 'Bob Brown', '321 Maple St', '1975-04-10', '555-987-6543', 'bob.brown@example.com'),
('LIC005', 'Charlie Davis', '654 Pine St', '1990-07-25', '111-222-3333', 'charlie.davis@example.com'),
('LIC006', 'David Wilson', '876 Cedar St', '1988-02-17', '444-555-6666', 'david.wilson@example.com'),
('LIC007', 'Emma White', '543 Birch St', '1995-09-19', '777-888-9999', 'emma.white@example.com'),
('LIC008', 'Fiona Black', '678 Spruce St', '1983-03-07', '888-999-0000', 'fiona.black@example.com'),
('LIC009', 'George Green', '987 Willow St', '1978-10-12', '999-000-1111', 'george.green@example.com'),
('LIC010', 'Hannah Gray', '432 Poplar St', '1993-06-28', '000-111-2222', 'hannah.gray@example.com');
```

```sql
-- Insert values into RENTAL_LOCATION table
INSERT INTO RENTAL_LOCATION (Location_ID, Phone, Address, Postal_Code, State, Email) VALUES
(1, '123-456-7890', '123 Main St', '12345', 'New York', 'location1@example.com'),
(2, '987-654-3210', '456 Elm St', '54321', 'California', 'location2@example.com'),
(3, '555-123-4567', '789 Oak St', '67890', 'Texas', 'location3@example.com'),
(4, '555-987-6543', '321 Maple St', '98765', 'Florida', 'location4@example.com'),
(5, '111-222-3333', '654 Pine St', '13579', 'Nevada', 'location5@example.com'),
(6, '444-555-6666', '876 Cedar St', '24680', 'Arizona', 'location6@example.com'),
(7, '777-888-9999', '543 Birch St', '11223', 'Illinois', 'location7@example.com'),
(8, '888-999-0000', '678 Spruce St', '33445', 'Georgia', 'location8@example.com'),
(9, '999-000-1111', '987 Willow St', '55667', 'Ohio', 'location9@example.com'),
(10, '000-111-2222', '432 Poplar St', '77889', 'Virginia', 'location10@example.com');
```

# T-SQL

What is T-SQL?

T-SQL (Transact-SQL) is a programming language used to manage and interact with Microsoft SQL Server databases. It extends standard SQL (Structured Query Language) by adding extra features, such as:

- Variables
- Conditional logic (e.g., IF, WHILE)
- Error handling
- Advanced querying capabilities

It helps you not only query the data but also perform complex tasks, automate processes, and manage the database efficiently

# T-SQL Functions and Their Types

In T-SQL, functions are pre-defined or user-defined operations that perform calculations or manipulate data and return a value. Functions in T-SQL are broadly categorized based on their use cases and return types.

Types of Functions in T-SQL

## 1. System Functions

These are built-in functions provided by T-SQL.

2. User-Defined Functions (UDFs)

Functions created by users to perform specific tasks.

# Implementing T-SQL Functions

```sql
CREATE FUNCTION CalculateTotalRentalAmount(@StartDate DATE, @EndDate DATE, @Total_Amount DECIMAL(10, 2))
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @NetAmount DECIMAL(10, 2);
    SET @NetAmount = DATEDIFF(DAY, @StartDate, @EndDate) * @Total_Amount;
    RETURN @NetAmount;
END
```

```sql
--2Get Car Availability by VIN
CREATE FUNCTION IsCarAvailable(@ID VARCHAR)
RETURNS BIT
AS
BEGIN
    DECLARE @Available BIT;
    IF EXISTS (SELECT 1 FROM CAR WHERE VIN = @ID)
        SET @Available = 0;
    ELSE
        SET @Available = 1;
    RETURN @Available;
END
```

# T-SQL Procedure

A stored procedure is a saved set of SQL commands that can be run with a single call. Think of it as a "recipe" for SQL queries that you can reuse whenever you need, without rewriting the code each time.

Key Benefits of Stored Procedures:

Reusable: Once you create a stored procedure, you can run it anytime, which saves time and effort.

Faster: Stored procedures are pre-compiled, meaning they are quicker to execute than writing SQL queries every time.

Secure: You can control who can run the stored procedure, instead of giving people direct access to your data.

Organized: Stored procedures help you organize and simplify complex tasks into manageable chunks.

Error Handling: They can handle errors smoothly, so your system doesn't break unexpectedly.

# How does it work

1. Create: You create a stored procedure using the CREATE PROCEDURE command. It contains your SQL code.
2. Execute: Once created, you can "run" the procedure using the EXEC command.
3. Input Parameters: You can pass information into stored procedures (like a car ID to fetch car details).
4. Output: Some stored procedures can give you back results or even calculated values.

# Implementing T-SQL Procedure

```
--1Get All Cars
CREATE PROCEDURE GetAllCars
AS
BEGIN
    SELECT * FROM CAR;
END
EXEC GetAllCars;
```

```sql
--2Get Cars by Location
CREATE PROCEDURE GetCarsByLocation
@LocationID INT
AS
BEGIN
    SELECT * FROM CAR WHERE Location_ID = @LocationID;
END
EXEC GetCarsByLocation @LocationID = 1;
-----------------------------------------

--3Update Car Daily Rate
CREATE PROCEDURE UpdateCarDailyRate
@ID VARCHAR,
@NewDailyRate DECIMAL(18, 2)
AS
BEGIN
BEGIN TRY
    UPDATE CAR
    SET Price_per_Day = @NewDailyRate
    WHERE VIN = @ID;
    PRINT 'Daily rate updated successfully';
END TRY
BEGIN CATCH
    PRINT 'An error occurred while updating the daily rate';
END CATCH
END
EXEC UpdateCarDailyRate @ID = VIN001, @NewDailyRate = 75.00;
---------------------------------------
```

# T-SQL Cursors

A cursor in T-SQL is a tool that allows you to process and handle each row of a result set individually, one by one. It's like going through a list of items and doing something with each item, one at a time.

Cursors are used when you need to perform operations on each row individually, rather than on the whole set of data at once.

When to Use Cursors?

- When you need to loop through rows in a result set and perform actions for each row.
- For complex row-by-row operations where set-based operations (which work with all rows at once) are not enough.

# Types of cursors

There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

- Implicit Cursors: Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
- Explicit Cursors: Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

# Implementing T-SQL Cursors

```
--4Iterate Over Rental Locations
DECLARE LocationCursor CURSOR FOR
SELECT Location_ID, State FROM RENTAL_LOCATION;
OPEN LocationCursor;
FETCH NEXT FROM LocationCursor INTO @LocationID, @State;
WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'Location ID: ' + CAST(@LocationID AS NVARCHAR(10)) + ', City: ' + @State;
    FETCH NEXT FROM LocationCursor INTO @LocationID, @State;
END
CLOSE LocationCursor;
DEALLOCATE LocationCursor;
```

```sql
--3Iterate Over Payments don't use it fro now :work needed
DECLARE PaymentCursor CURSOR FOR
SELECT Payment_ID, Amount, Status FROM PAYMENT;
OPEN PaymentCursor;
FETCH NEXT FROM PaymentCursor INTO @TransactionID, @Amount, @Status;
WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'Transaction ID: ' + CAST(@TransactionID AS NVARCHAR(10)) + ', Amount: ' + CAST(@Amount AS NVARCHAR(10)) + ', Status: ' + @Status;
    FETCH NEXT FROM PaymentCursor INTO @TransactionID, @Amount, @Status;
END
CLOSE PaymentCursor;
DEALLOCATE PaymentCursor;

------------------------------
--2Iterate Over Users :--prints basic user details
DECLARE UserCursor CURSOR FOR
SELECT License_No, Name, Email FROM CAR_USER;
OPEN UserCursor;
FETCH NEXT FROM UserCursor INTO @License_No, @Name, @Email;
WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'User ID: ' + CAST(@License_No AS NVARCHAR(10)) + ', Name: ' + @Name + ', Email: ' + @Email;
    FETCH NEXT FROM UserCursor INTO @License_No, @Name, @Email;
END
CLOSE UserCursor;
DEALLOCATE UserCursor;
```

# Conclusion

The Car Rental Database Project efficiently manages car rental operations by organizing data on customers, vehicles, reservations, and payments. It streamlines processes, improves customer service, and ensures reliable transactions. The system is scalable, secure, and provides a strong foundation for future enhancements, supporting business growth and operational efficiency.

# Thank You