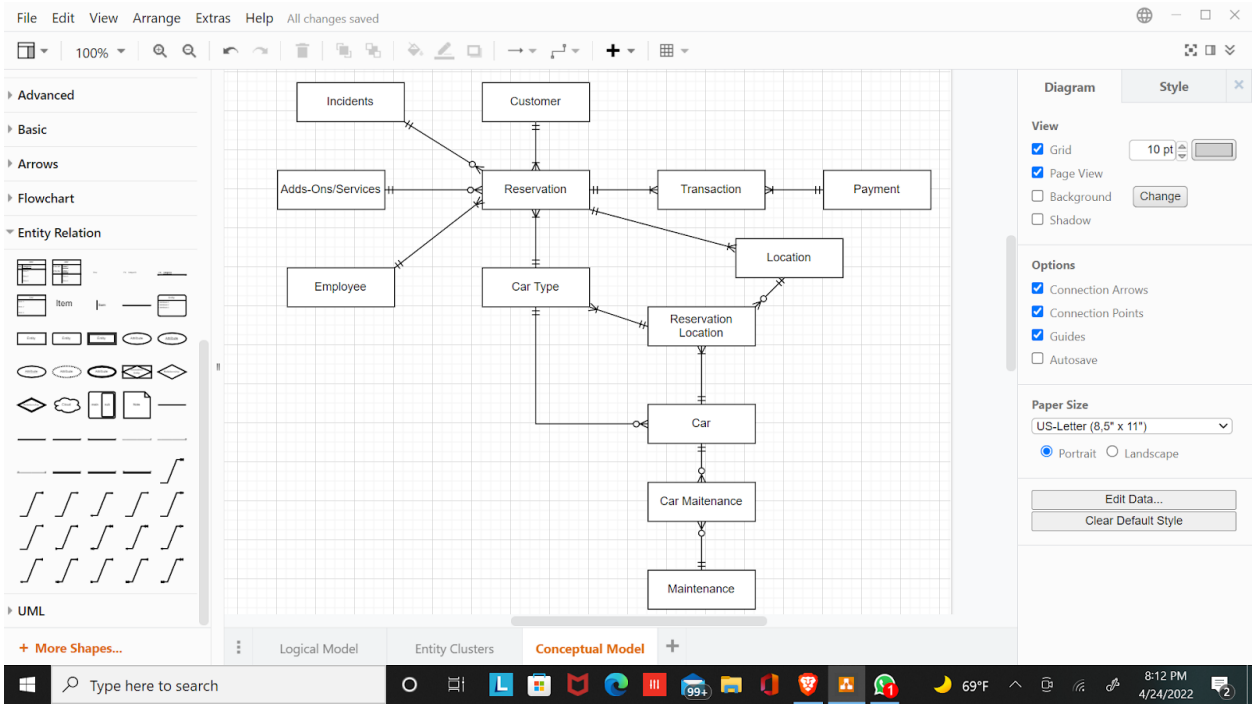
CAR RENTAL SYSTEM

By Priyanka Malla

Introduction: The older conventional system is a manually driven system that requires paperwork to be managed by the employees when a car is sent out by the company to a customer. This requires doing the paperwork for the booking process which is very arduous to manage because it is in the physical form, and it is not a very trusted format. The manual work done by the employee needed to be automated because it is requiring much time and effort of the employees ergo the proposed system is engendered keeping in mind all the functions and quandaries faced by the customers while dealing in renting out the car. The car rental system is a well-equipped method to handle the overall requisites of the current people's cull. Certain responsibilities can be managed by the system if not compulsorily done by other people is system provides an opportunity to the entities who need to manage their work through the system and avail the ground authenticity where the rental car data is optimized in a format sustaining the accounts of employees, customers.

In the design of the database, we have included a conceptual model, a logical model, a data dictionary, the physical design of the database, the description of some of the tasks expected from the database, and related SQL queries.

A. Conceptual Model

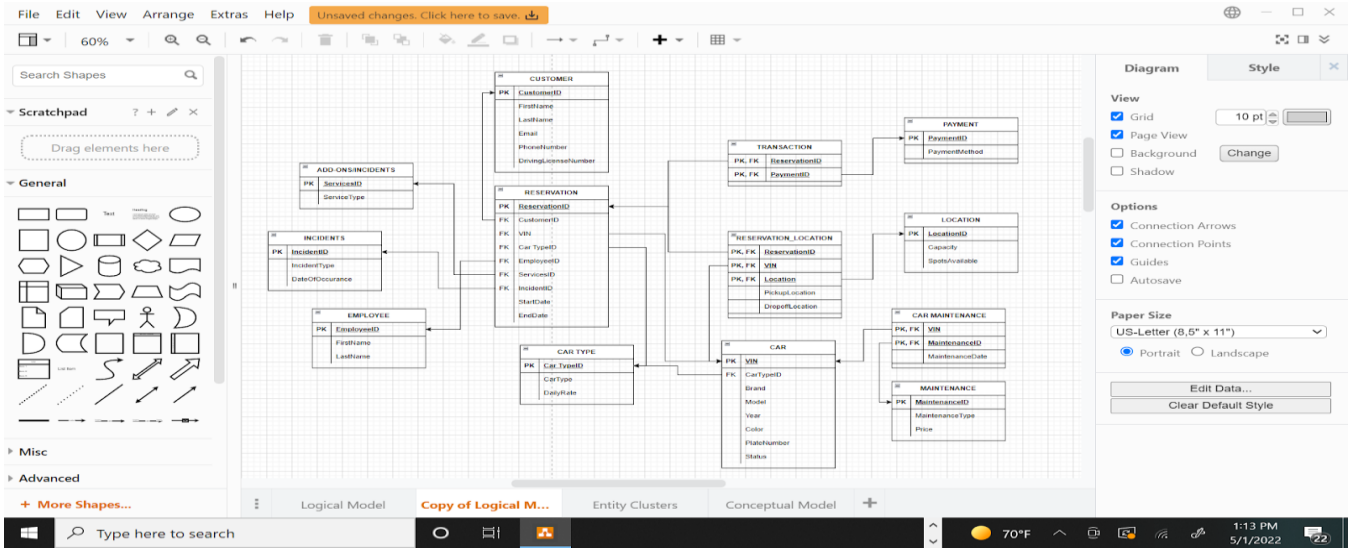


The conceptual model has 13 entities with about 14 different relationships consisting of either One to Many or Many to Many relationships. The relationship types include mandatory one, optional one, mandatory many, and optional many. The foundation of the conceptual model was the Reservation Entity since it played a key role in the reservation, payment, and pick up/drop off process.

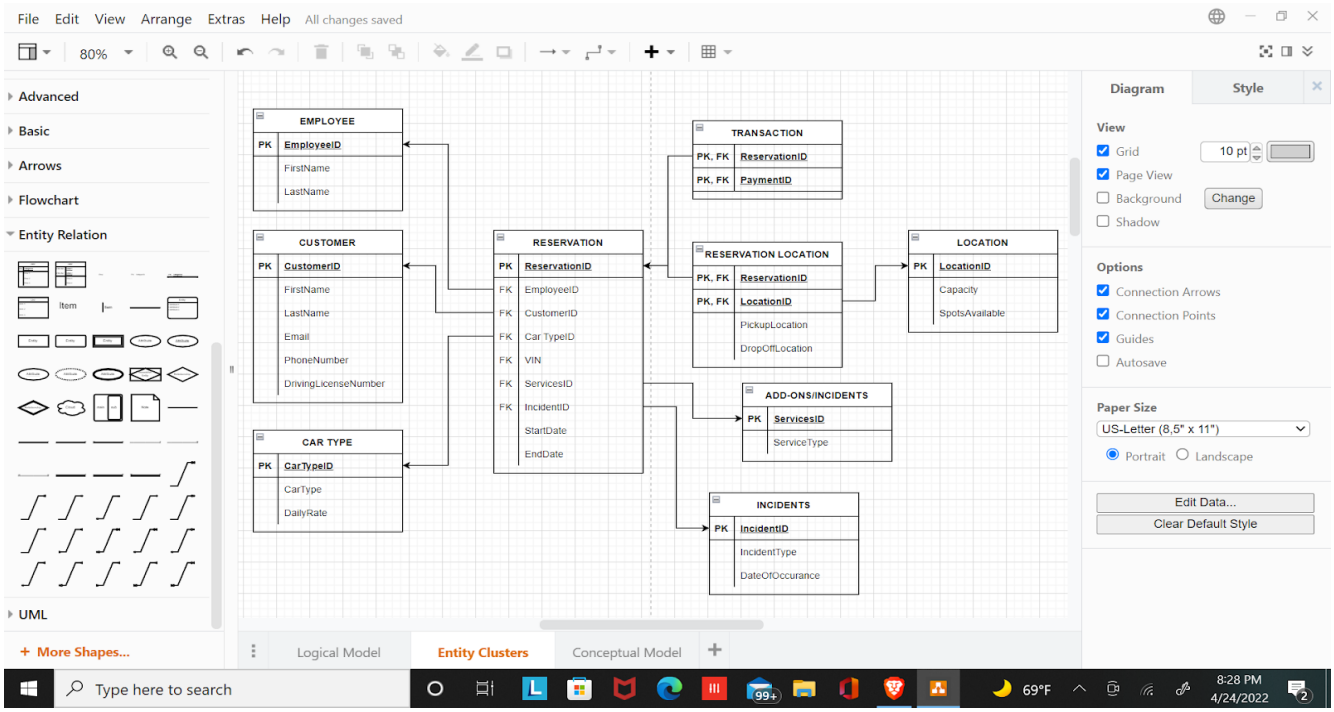
There were 5 supertype/subtype relationships in this database project. The supertype/subtype relationships consisted of Payment method, Incident, Car Type, Maintenance, and Add-Ons/Services. The purpose of the supertype/subtype relationship is to identify entities that could be broken down into further pieces. By doing so, you create a sense of simplicity within your models. The supertypes are a generic entity type that has a relationship with one or more subtypes. Subtypes are a subgrouping of the entities in an entity type that has attributes distinct from those in other subgroupings. The relationships at the supertype level indicate that all subtypes will participate in the relationship and the instances may participate in a relationship unique to that subtype. It is important to know that there are constraints in supertype/subtype relationships. These constraints are called completeness and disjointness constraints. The completeness constraint specifies whether an instance of a supertype must also be a member of at least one subtype. An example of this would be the Car Type subtype since a car must belong to at least one car type. The disjointness constraint is whether an instance of a supertype may simultaneously be a member of two (or more) subtypes. An example of this would be the Maintenance subtype since the maintenance on a car can include different things.

A. Logical Model

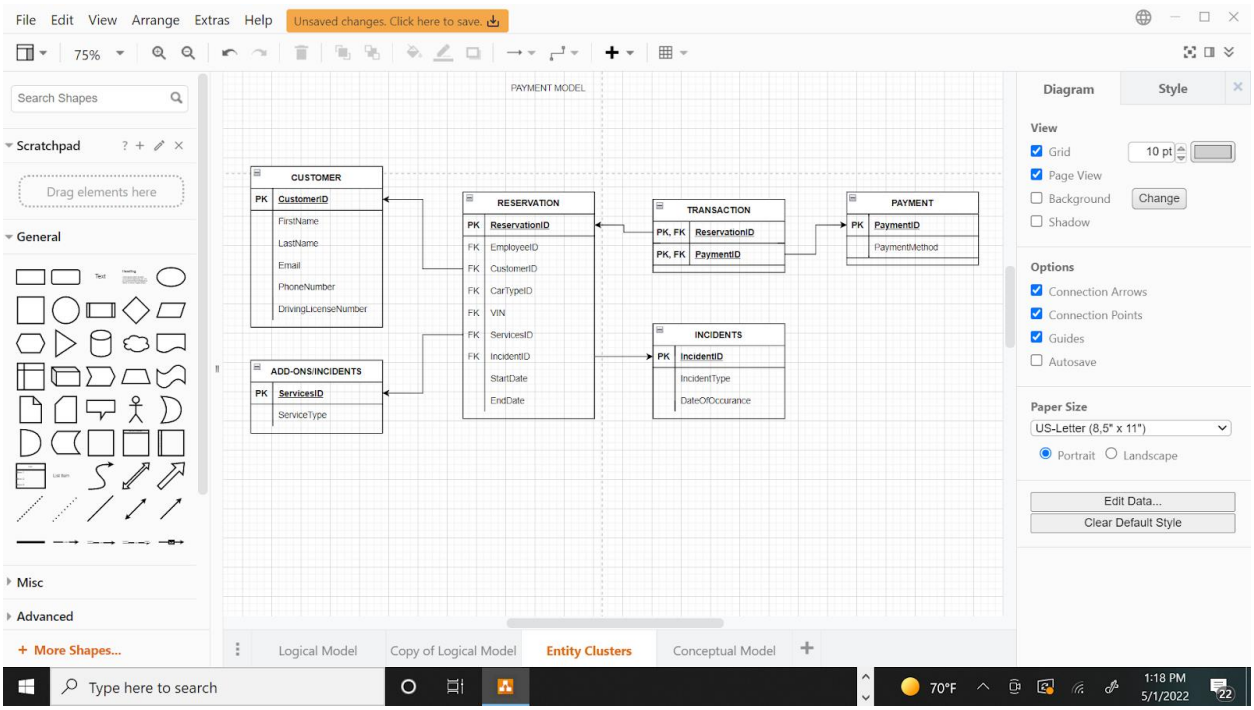
The logical model created was split into three different entity clusters to represent different parts of the process: the reservation process, the payment process, and the pickup/drop-off process.



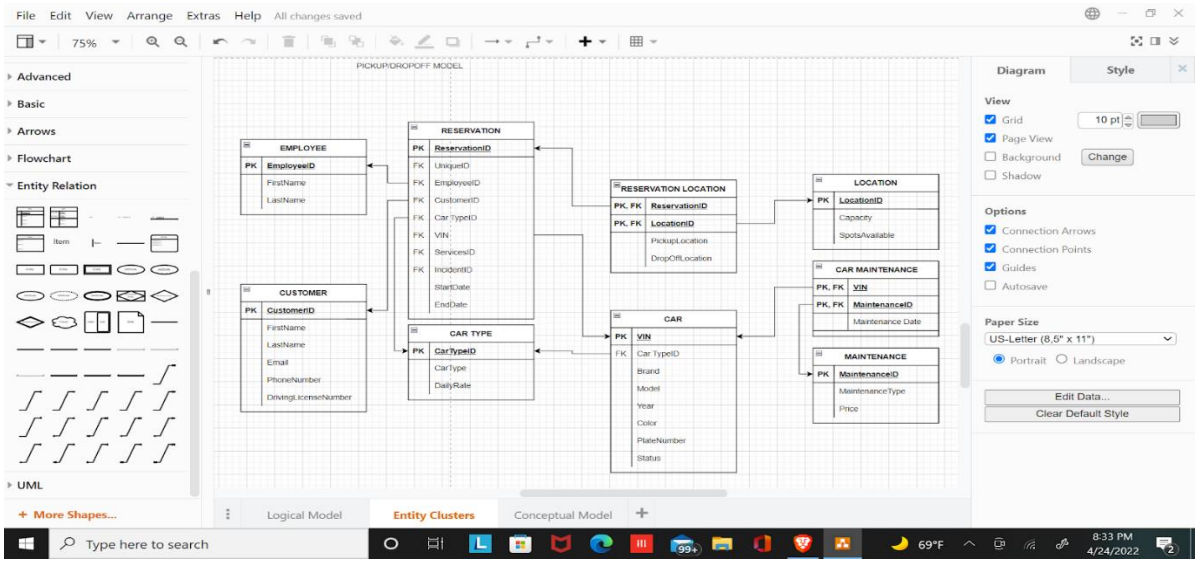
For the “reservation process” cluster, we identified 9 entities that are key to this process. The purpose of the reservation process is to identify and collect important pieces of information about the reservation the customer is making such as who the customer is, the employee who helped them, the car type selected, the car chosen, the start of the transaction, the service requested, the incidents recorded, and the location specified. The key entity in this cluster was the “Reservation” cluster as it housed the most foreign keys.



The “payment process” cluster is for identifying what the customer has and will pay for. This cluster holds the customer, reservation, reservation details, add-on/services, transaction, and payment entities. The key entities in this cluster would be the reservation, transaction, and payment entities. The reservation entity will hold and pass all the necessary information about the customer and the car that they rented to the transaction entity, and the payment entity will hold and pass all of the necessary information about how the customer will pay for the rental service to the transaction service.



The “pick up / drop off process” cluster’s purpose is to record all information about how the pickup and drop off process will be handled. The process begins with the customer checking in with an employee to help them pick up the car they specified. All this information as well as where they picked up the car is recorded to the “Reservation” entity. When the customer brings back the car, the drop-off process begins and it starts by recording the drop-off process in the “Reservation” entity. This will then lead to a maintenance check on the car, and any problems identified on the car during the check will be recorded in the “Maintenance” entity.



A. Data Dictionary

The purpose of the database is to identify the vital pieces of the rental service. In a service that has seen much stagnation, it is important to create a database to identify how each location can improve to gain new customers, deepen their loyalty with us, and keep hold of existing customers. This can also help identify ways to optimize the experience customers have in renting cars and finding to rent cars that are done by introducing new forms of technology or introducing more electric cars to drive down the costs spent on fuel, the car rental service can be successful in achieving goals like these if they process and efficiently use data with an appropriate database.

ENTITY: Customer

COLUMNS	DATATYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
CustomerID	INT	The primary key to identifying the customer	000001-999999	Y	PK
First Name	VARCHAR	First name of the customer		Y	
Last Name	VARCHAR	Last name of the customer		Y	
Email	VARCHAR	Email of the customer		Y	
Phone Number	INT	Phone number of the customer		Y	
Driving License Number	INT	The license number of the customer		Y	

ENTITY: Employee

COLUMNS	DATATYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
EmployeeID	INT	The primary key to identifying the employee	000001-999999	Y	PK
First Name	VARCHAR	First name of the employee		Y	
Last Name	VARCHAR	Last name of the employee		Y	

ENTITY: Reservation

COLUMNS	DATATYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
ReservationID	INT	The primary key to identifying the reservation	000001-999999	Y	PK
CustomerID	INT	The foreign key identifying the customer	000001-999999	Y	FK
VIN	INT	key identifying the car	000001-999999	Y	FK
CarTypeID	INT	Tnnnnnnnnnn key identifying the car type	000001-999999	Y	FK

EmployeeID	INT	The foreign key identifying the employee	000001-999999	Y	FK
ServicesID	INT	The foreign key identifying the ADD-ONS	000001-999999	Y	FK
IncidentID	INT	Tnnnnnnnnnn key identifying the incidents	000001-999999	Y	FK
StartDate	DATE	Date of when the car was rented		Y	
EndDate	DATE	Date of when the car was returned		Y	

ENTITY: Car

COLUMNS	**DATA TYPE**	**DESCRIPTION**	**RANGE**	**REQUIRED**	**PK OR FK**
VIN	INT	The primary key to identifying the car	000001-999999	Y	PK
Car TypeID	INT	The foreign key identifying the car type	000001-999999	Y	FK
Brand	VARCHAR	Company that made the car		Y	
Model	VARCHAR	Model of the car		Y	

Year	YEAR	Year the car was made	0001-9999	Y	
Color	CHAR	Color of the car		Y	
Plate number	VARCHAR	Plate number assigned to the car		Y	
Status	VARCHAR	Availability of the car		Y	

ENTITY: CarType

COLUMNS	DATA TYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
CarTypeID	INT	The primary key to identifying the car type	000001-999999	Y	PK
CarType	INT	Type of car		Y	
DailyRate	FLOAT	The daily price paid to rent the car		Y	

ENTITY: Payment

COLUMNS	DATA TYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
PaymentID	INT	The primary key to identifying the payment	000001-999999	Y	PK
Payment Method	VARCHAR	How the customer will pay		Y	

ENTITY: Add-Ons/Service

COLUMNS	DATA TYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
ServicesID	INT	The primary key to identifying the services provided	000001-999999	Y	PK
ServiceType	CHAR	service type provided		N	

ENTITY: Transaction

COLUMNS	DATA TYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
ReservationID	INT	The composite key identifying the reservation	000001-999999	Y	PK/FK (Composite Key)
PaymentID	INT	The composite key identifying the payment	000001-999999	Y	PK/FK (Composite Key)

ENTITY: Location

COLUMNS	DATA TYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
LocationID	INT	The primary key to identifying the location	000001-999999	Y	PK
Capacity	INT	The number of cars the location can hold		Y	
SpotsAvailable	INT	Spaces available for the cars		Y	

ENTITY: Maintenance

COLUMNS	DATATYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
MaintenanceID	INT	The primary key to identifying the maintenance done	000001-999999	Y	PK
Maintenance Type	CHAR	Type of maintenance performed		N	
Price	FLOAT	The cost of maintenance		N	

ENTITY: ReservedLocation

COLUMNS	DATA TYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
ReservationID	INT	The composite key identifying the reservation	000001-999999	Y	PK/FK(Composite Key)
VIN	INT	The composite key identifying the car	000001-999999	Y	PK/FK(Composite Key)
LocationID	INT	identifying the location		Y	PK/FK(Composite Key)

PickupLocation	VARCHAR	Location of where the car was picked up		Y	
DropoffLocation	VARCHAR	Location of where the car was dropped off		Y	

ENTITY: Car Maintenance

COLUMNS	DATA TYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
VIN	INT	The composite key identifying the car	000001-999999	Y	PK/FK(Composite Key)
MaintenanceID	INT	The composite key identifies the maintenance done	000001-999999	Y	PK/FK(Composite Key)
MaintenanceDate	DATE	To identify the date of maintenance		Y	

ENTITY: Incidents

COLUMNS	DATATYPE	DESCRIPTION	RANGE	REQUIRED	PK OR FK
IncidentID	INT	The primary key to identifying the incident	000001-999999	Y	PK
Incident Type	CHAR	Type of incident occurred		N	
Dateofoccurance	DATE	Date of the incident		N	
Servicename (Delete)	VARCHAR	Service needed for the incident			

D. DESCRIPTION OF THE PHYSICAL DESIGN

A) Denormalization is the process of merging two relations into a single new one. It is the process of altering table structures in a normalized database to allow for controlled redundancy and improved database efficiency. The denormalization is conducted as follows:

1) Adding Columns That Aren't Necessary. 2) Incorporating Derived Columns 3) Table Combination 4) Groups That Recur 5) Creating tables for extracts 6) Relationships of Partition

For any queries, adding redundant columns eliminates joins; however, this solution has the following drawbacks such as it necessitates the maintenance of a new column, and all modifications must be done to two tables, one of which may have too many rows.

An alternative to combining relations is to break them into several smaller and more manageable partitions. There are two types of partitions, Horizontal and Vertical partitioning. Horizontal partitioning involves distributing relation tuples among several (smaller) partitioning relations. Vertical partitioning

involves distributing a relationship's attributes across several (smaller) partitioning relationships (the primary key is duplicated to allow the original relationship to be recreated). Because of the nature of our logical design,

B) Constraints are a crucial aspect of a relational model. The relational model backs up a well-defined theory of attribute or table constraints. Constraints are useful because they enable a database designer to establish the semantics of data. Constraints are constraints that DBMSs must follow to ensure that data meets the semantics.

The object that would have access to the database would be the Employees entity with the Reservation process as the subject. The action of the object in this situation would be to insert and read data. This way they can check customers who come into their locations and set up a reservation with their choice of car. Another entity that can act as an object is the Transaction entity with the Payment process as the subject. The actions of the Transaction object would be to read data so that it can process how the payment method will be conducted and for which reservation it will be attached.

c) Data types that were used in creating the database consist of DATE, FLOAT, CHAR, VARCHAR, and INT.

The DATE data type is used to store year, month, and day values. It was used for the Maintenance Date attribute from the Car Maintenance entity, the Year attribute from the Car entity, the Start Date and End Date attribute from the Reservation entity, and the Date of Occurrence attribute from the Incidents entity.

The FLOAT data type is an approximate numerical value that can be used for numbers with decimals. FLOAT data types were used for the DailyRate attribute in the Car Type entity and the Price attribute in the Maintenance entity.

The CHAR data type is a string with a seat length that contains letters, numbers, and special characters. The size argument determines the character length of the column, which can range from 0 to 255. 1 is the default value. The attributes that have CHAR data type are Incident Type from the Incidents entity, Maintenance Type from the Maintenance entity, Service Type from the Add Ons entity, and Color from the Car entity.

The VARCHAR data type is a strain of variable length that contains letters, numbers, and special characters. The size option sets the maximum length of a column in characters, which can range from 0 to 65535. VARCHAR data types are identified in attributes such as the Pickup location and Drop Off location from the Reserved Location entity, the First and Last name from the Employee entity, and the Brand and Model from the Car entity.

The INT is a medium-sized number. The range of signed numbers is -2147483648 to 2147483647. From 0 to 4294967295 is the unsigned range. The maximum display width is specified by the size option (which is 255). The INT data type was identified in all the attributes that were labeled as primary keys such as CustomerID, ReservationID, VIN, LocationID, and EmployeeID.

D. For file organization, we use the hashing technique. The computation of the hash function on some fields of the records is used by Hash File Organization. The result of the hash function defines the position of the disk block where the records will be written.
When a record is requested using the hash key columns, an address is generated, and the entire record is fetched using that address. When a new record needs to be inserted, the hash key is used to generate the address, and the record is then directly placed. In the case of removing and updating, the same procedure is followed. There is no effort involved in searching and categorizing the full file using this method. Each record will be processed individually in this approach.
e. The sort of failure that happened, the structures that were damaged, and the type of recovery that we do all influence the recovery process. Recovery may be as simple as restarting an instance if no files are lost or destroyed. If data has been lost, more actions must be taken to recover it.
There are two components to disaster recovery: disaster tolerance and backup.
i. Backup entails creating one or more copies of critical data generated by application systems, as well as the original critical data. For example, in our project, we use an external device drive as a backup.
ii. We make sure our database is constantly on a failover cluster, which uses the window server failover cluster (WSFC) nodes framework to give instance-level protection. SQL Server Agent jobs and other instance and database objects are linked servers. Connecting applications are not affected by failover because it is automatic and transparent.
iii. SQL Server Replication: Our database relies on replication, which allows us to transfer and distribute data and database objects from one database to another while keeping them synchronized.

E. SQL Queries:

After completing the design of the database, for proper functioning, we need to implement the database into SQL. We have used MySQL for the implementation. Firstly, we need to create the tables with the entities mentioned in the logical models shown in the design and also establish their respective data types. Our database has been designed with a total of 13 classes, so we will make 13 different tables. When we are creating our tables, it is important to also establish the relationship that an entity might have with another entity, which is portrayed by primary keys and foreign keys.

After this, we must add sample data to the tables. This is done to check the functionality of the system. The data need not be legitimate. WE just need data so that we can run queries on the database. Once we have finally inserted our dummy data, we can now come up with a variety of different queries to fetch certain things that are expected from our database and perform common tasks that one customer or user might do.