**Car Rental Final Report**

**INTRODUCTION**

To handle customer data, rental transactions, and vehicle availability, the automobile rental sector needs an effective database system. The main goal of our project is to create and implement a structured database that guarantees smooth operations, accurate data retrieval, and peak performance. The design process, implementation details, and overall assessment of our database system are all thoroughly examined in this report.

**Background of the organization: Rentalcars.com**

Rentalcars.com is a globally recognized leader in the car rental industry and a key subsidiary of Booking Holdings Inc., which also owns major travel platforms such as Kayak, Booking.com, and Priceline. Founded in 2004 and headquartered in Manchester, UK, the company has grown into one of the most trusted car rental aggregators, serving customers in over 160 countries through partnerships with more than 60,000 rental locations. Rentalcars.com simplifies the car rental process by offering a transparent, user-friendly booking platform with competitive pricing, multilingual customer support, and financial protection. The platform caters to a wide range of travelers, from budget-conscious tourists to business professionals seeking premium vehicles, ensuring a seamless rental experience worldwide.

**Objective**

The primary objective of Rentalcars.com is to deliver a hassle-free, cost-effective, and reliable car rental service that meets the diverse needs of global customers. The platform aims to provide an extensive selection of vehicles, including economy cars, SUVs, luxury models, and specialty vehicles, with clear pricing and no hidden fees. By enabling advance bookings with flexible cancellation policies, Rentalcars.com ensures convenience and peace of mind for travelers. Additionally, the company prioritizes customer security through verified payment methods, insurance options, and fraud protection. To enhance accessibility, the platform supports multiple languages and offers 24/7 customer assistance. Another critical goal is maintaining real-time inventory tracking to prevent overbooking and optimize fleet management, ensuring that customers always have access to available vehicles. Through these efforts, Rentalcars.com strives to uphold its reputation as the leading car rental aggregator in the travel industry.

**Scope**

The proposed database system for Rentalcars.com will streamline operations and enhance customer experience with secure user authentication, allowing customers to register, save payment details, and input rental preferences like locations, dates, and vehicle type. It generates real-time search results with dynamic pricing, enabling a smooth checkout process where users review terms, select insurance, and complete payments automatically sending confirmations and reminders. The backend monitors fleet availability in real time, prevents overbooking, tracks maintenance, and flags low inventory. A feedback module collects reviews, while loyalty rewards and analytics tools track trends and supplier performance, ensuring data driven decisions. This integrated system delivers a seamless experience for users and administrators, supporting Rentalcars.com’s commitment to efficient, high quality service.

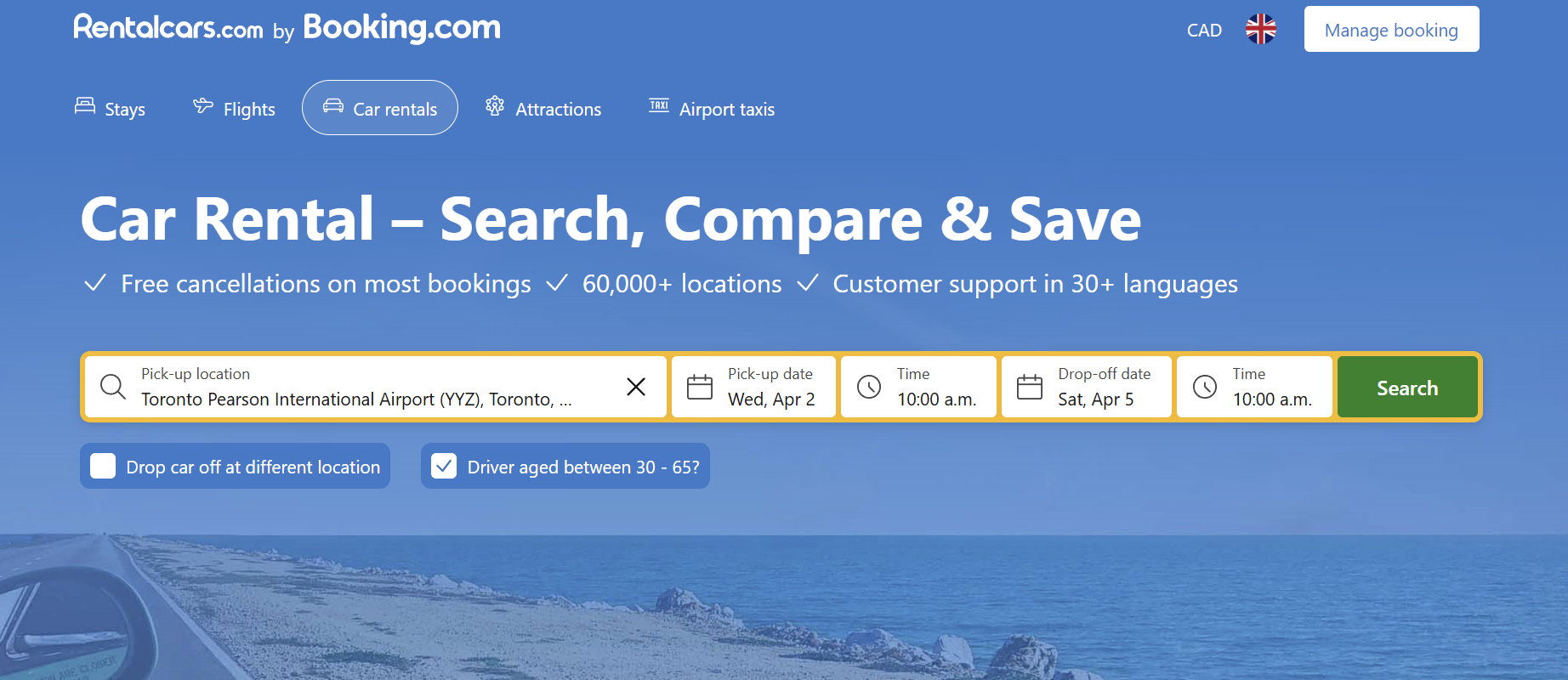
**Functional requirements of the application**

The Rentalcars.com database system was designed to fulfill three core functional requirements: search/retrieve, book/purchase, and manage operations. For search and retrieval, the system enables users to query real-time vehicle inventory based on location, date, and vehicle type while displaying dynamic rental pricing and accessing customer profile information. The booking and purchase functionality features a secure checkout process with integrated payment processing, optional insurance add-ons, and automated email/SMS confirmation notifications to ensure a seamless transaction experience. On the administrative side, the management capabilities allow Rentalcars.com staff to monitor fleet availability across all locations, track maintenance schedules, and analyze customer feedback through comprehensive analytics. Together, these interconnected functions create a robust platform that serves both customer needs and business operations efficiently.

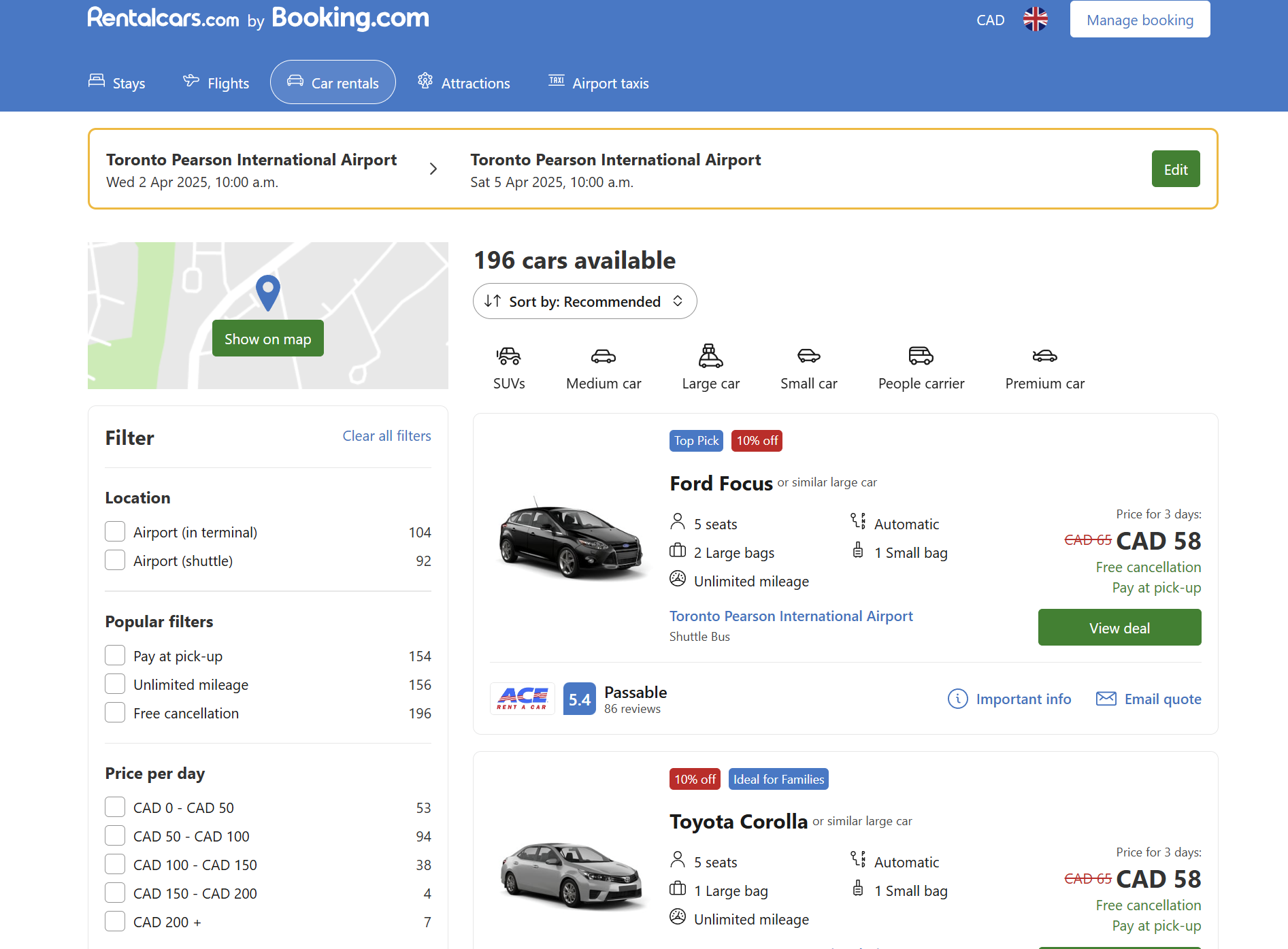
**DATABASE MODELLING**

**External Views**

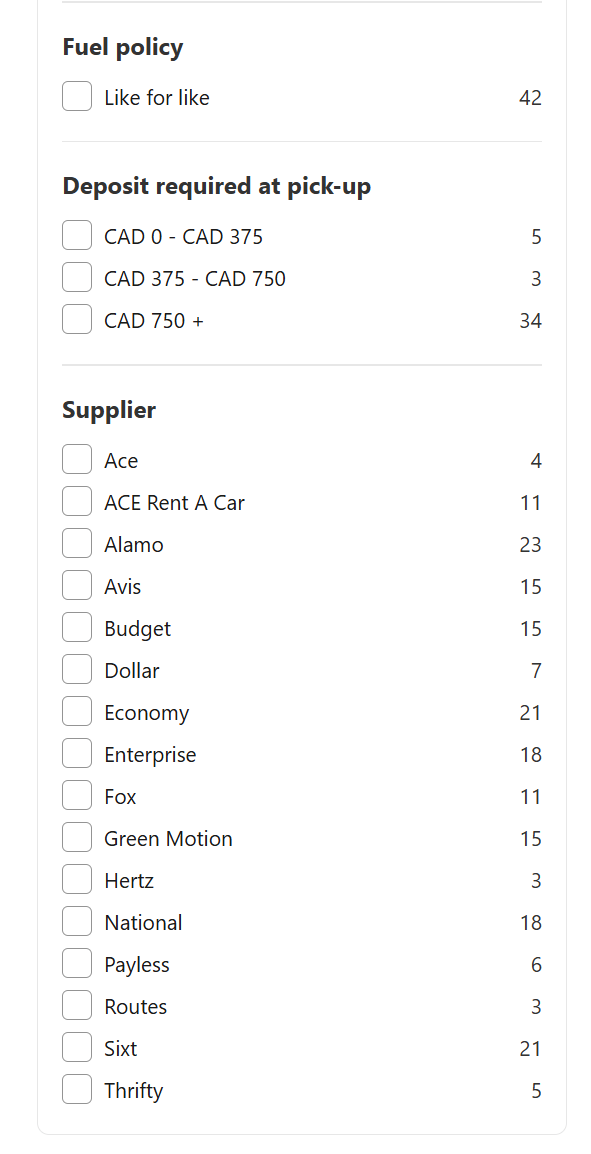
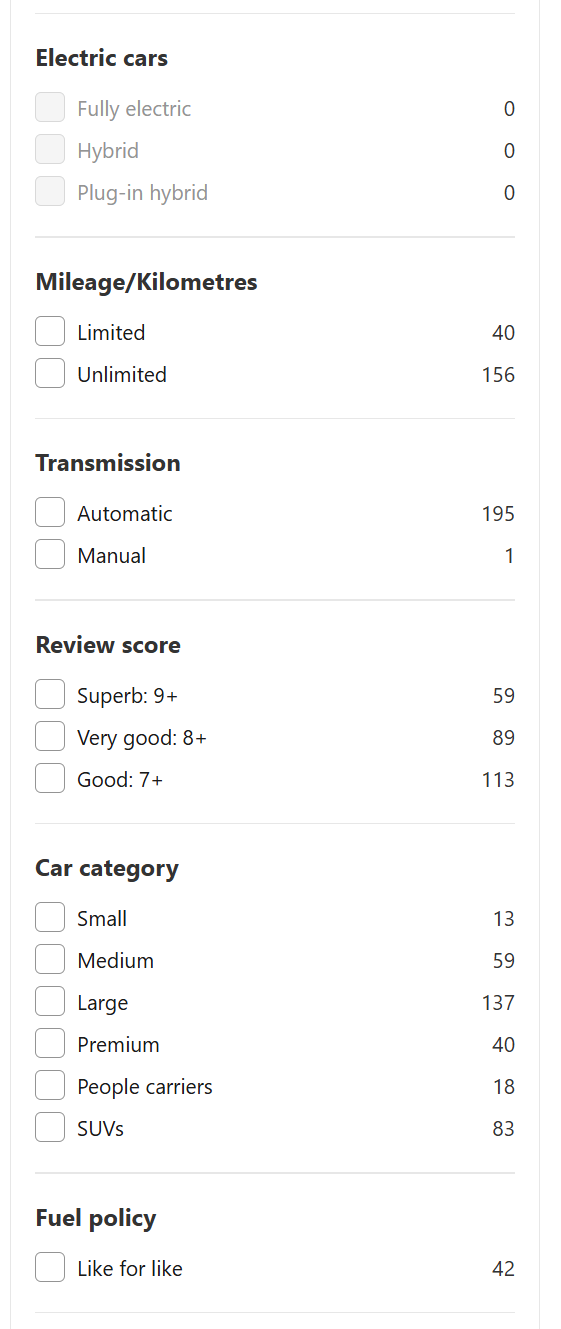
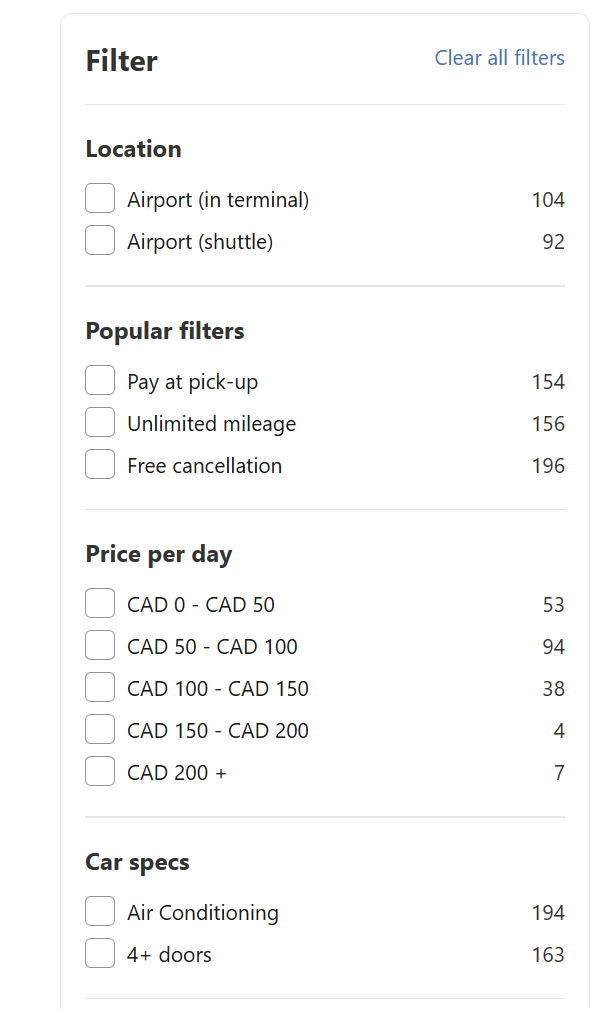
Below are some external views of the site, rentalcars.com as we provide images of the entire booking process. We start with selecting where and when a customer would like to book, for this example we selected Toronto Pearson Airport, from April 3rd to April 5th:



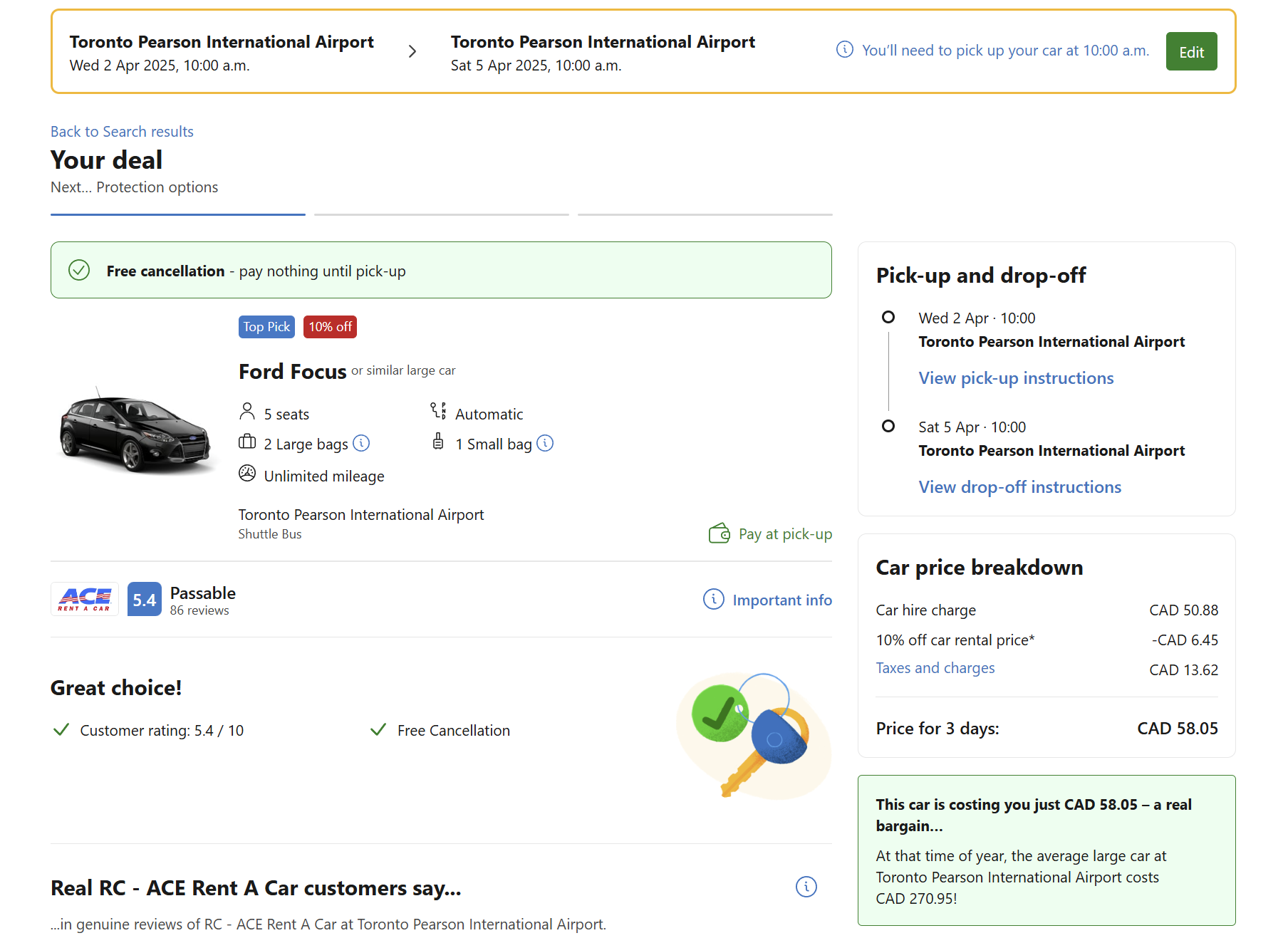
Result of the above search:

****

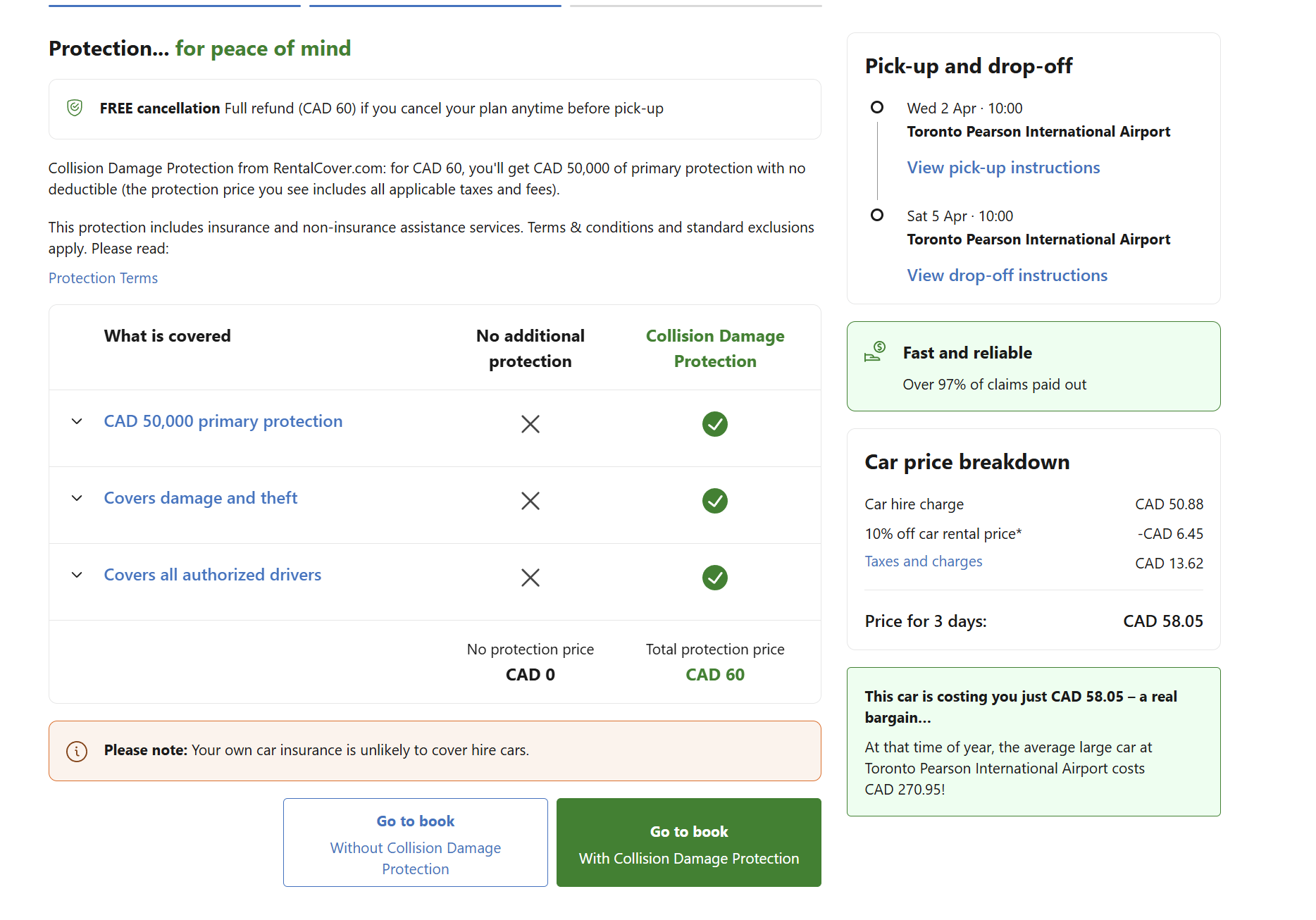
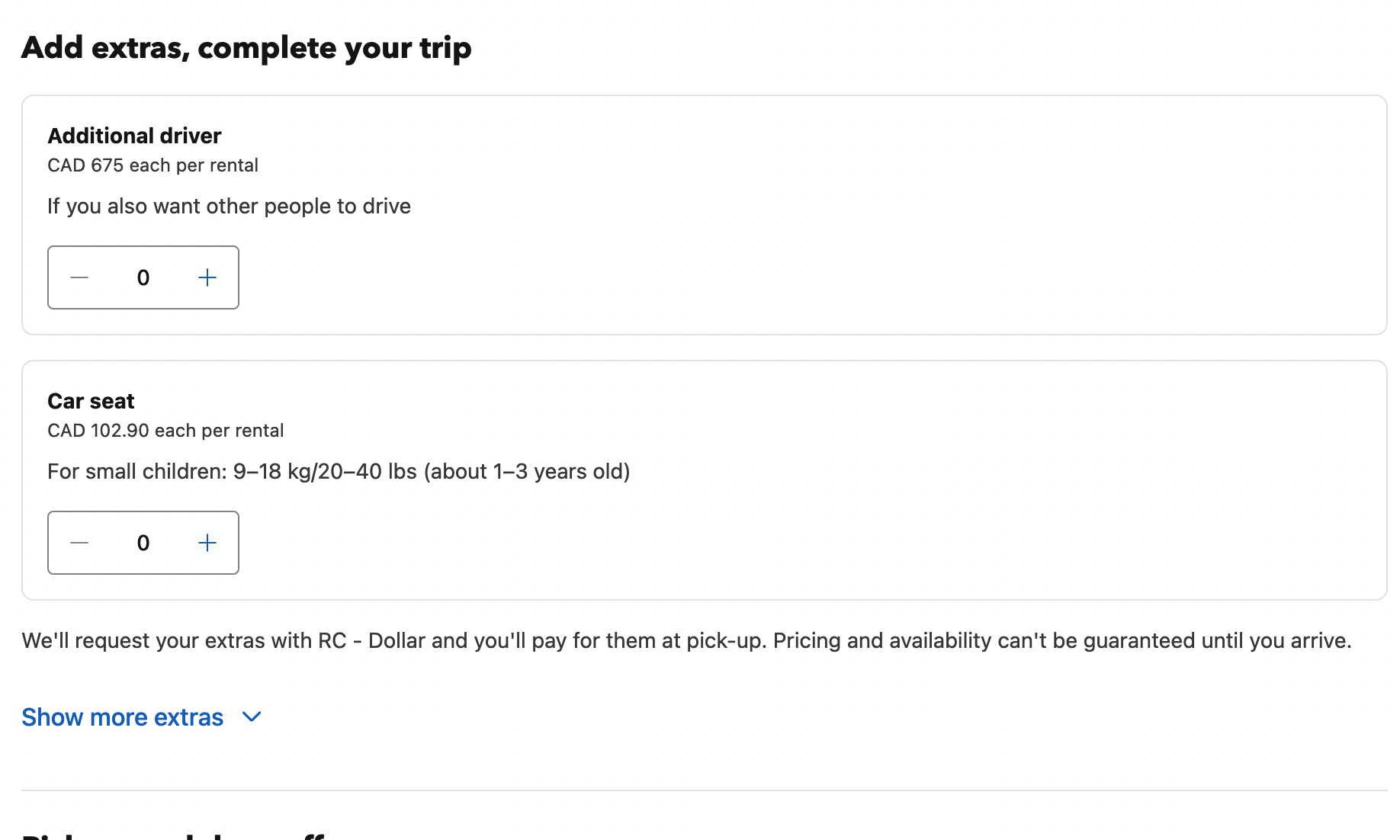
All available vehicle filtering options:

****

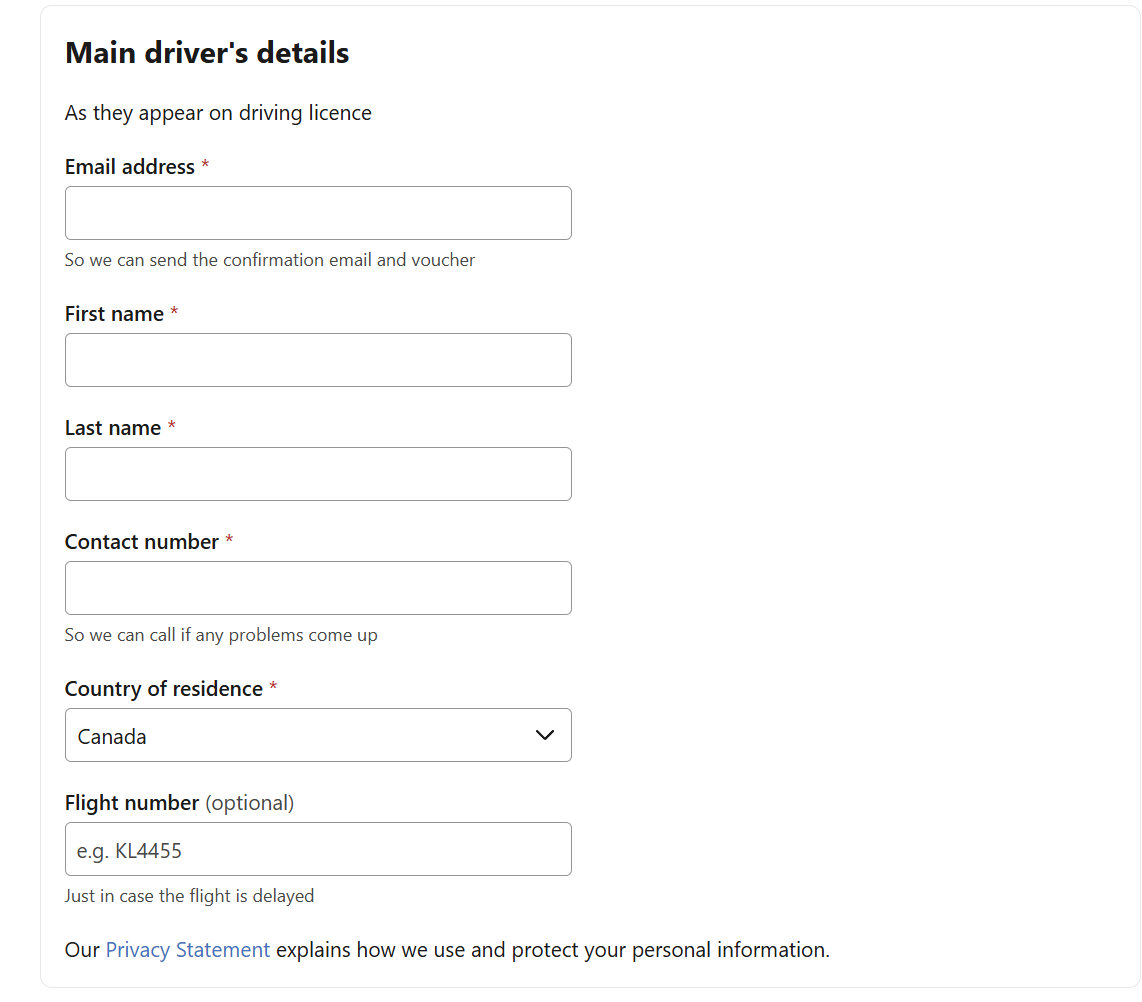
Here we have selected the vehicle we would like to rent:

****

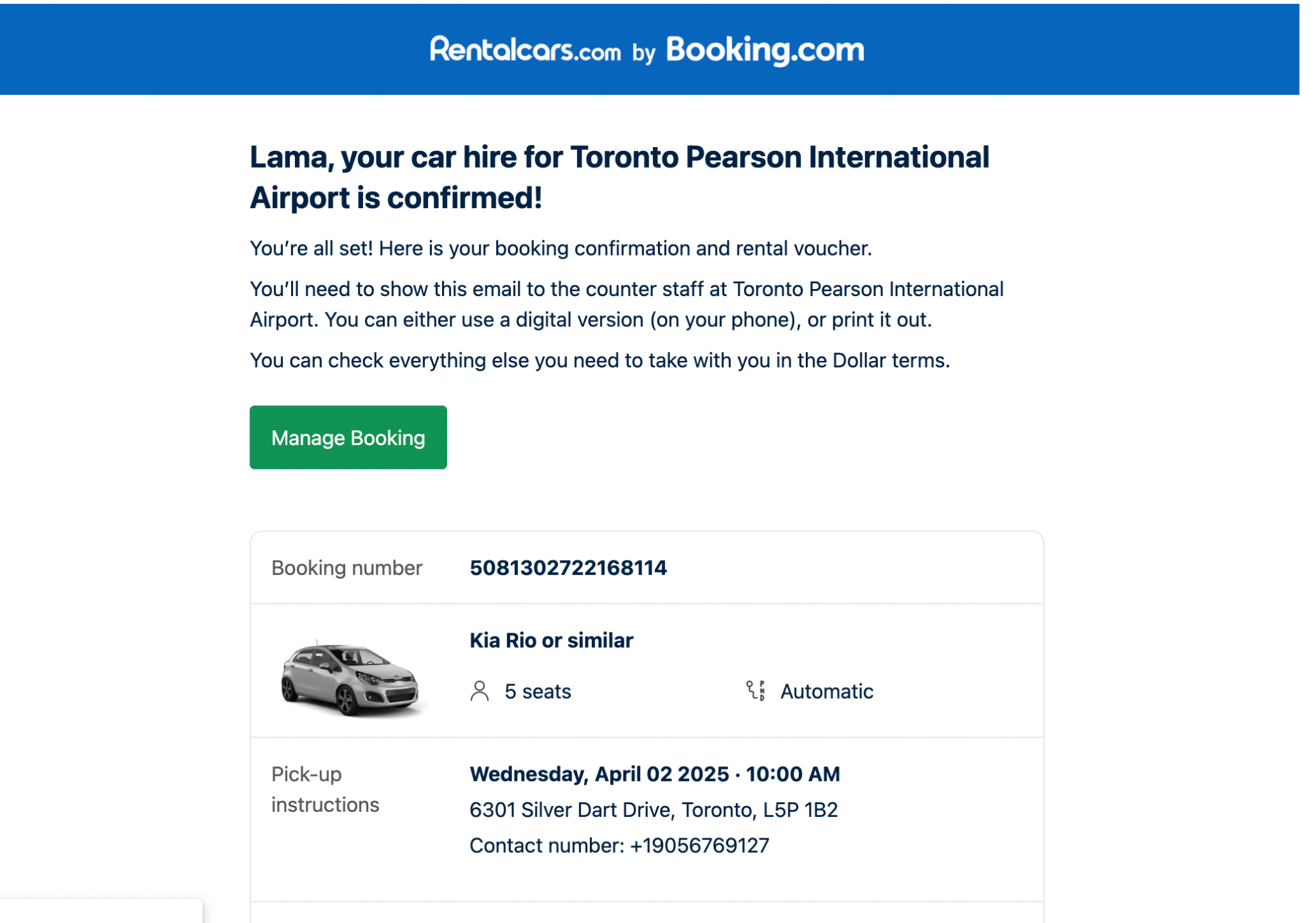
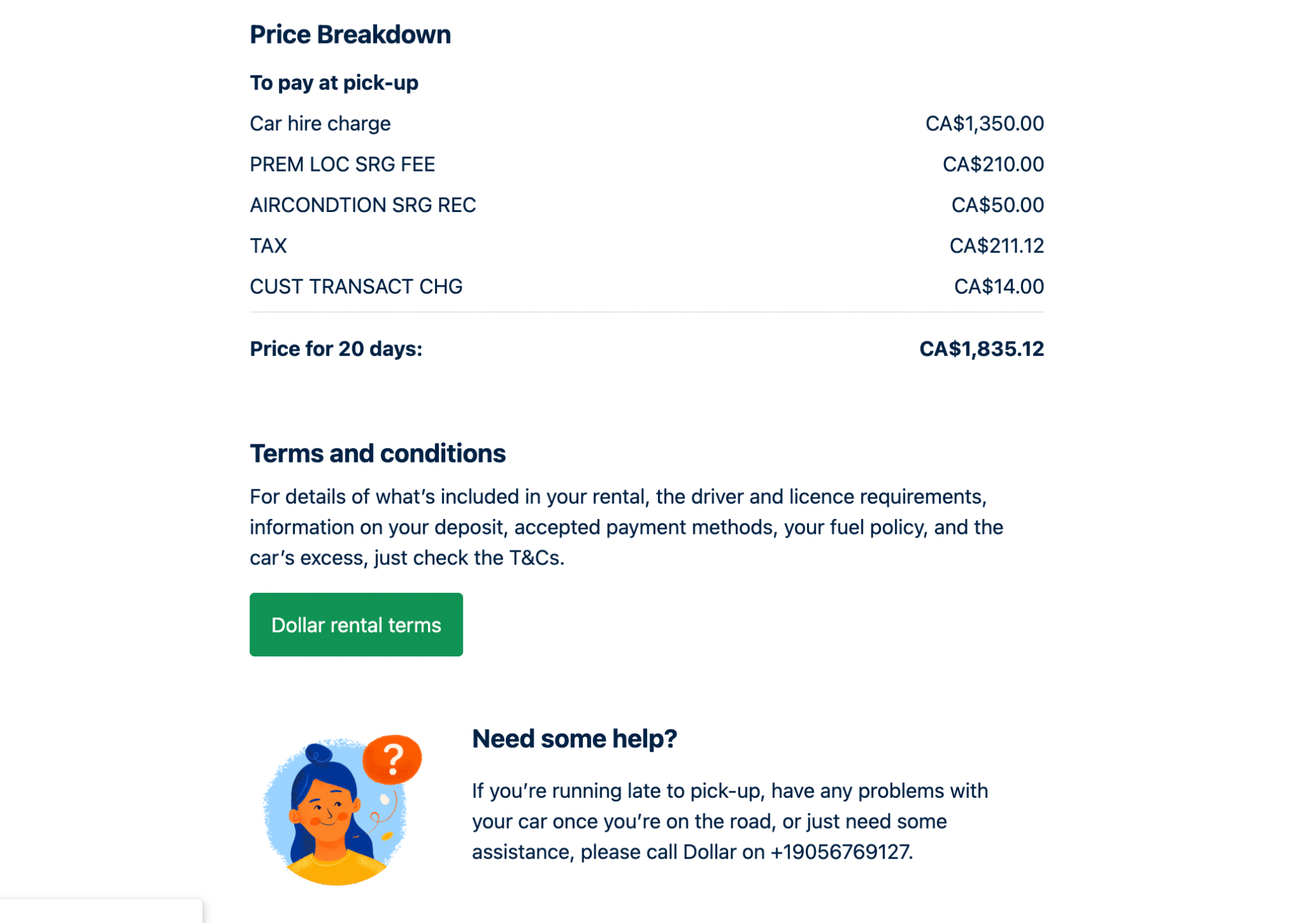
Available options to add to the vehicle reservation before checkout:

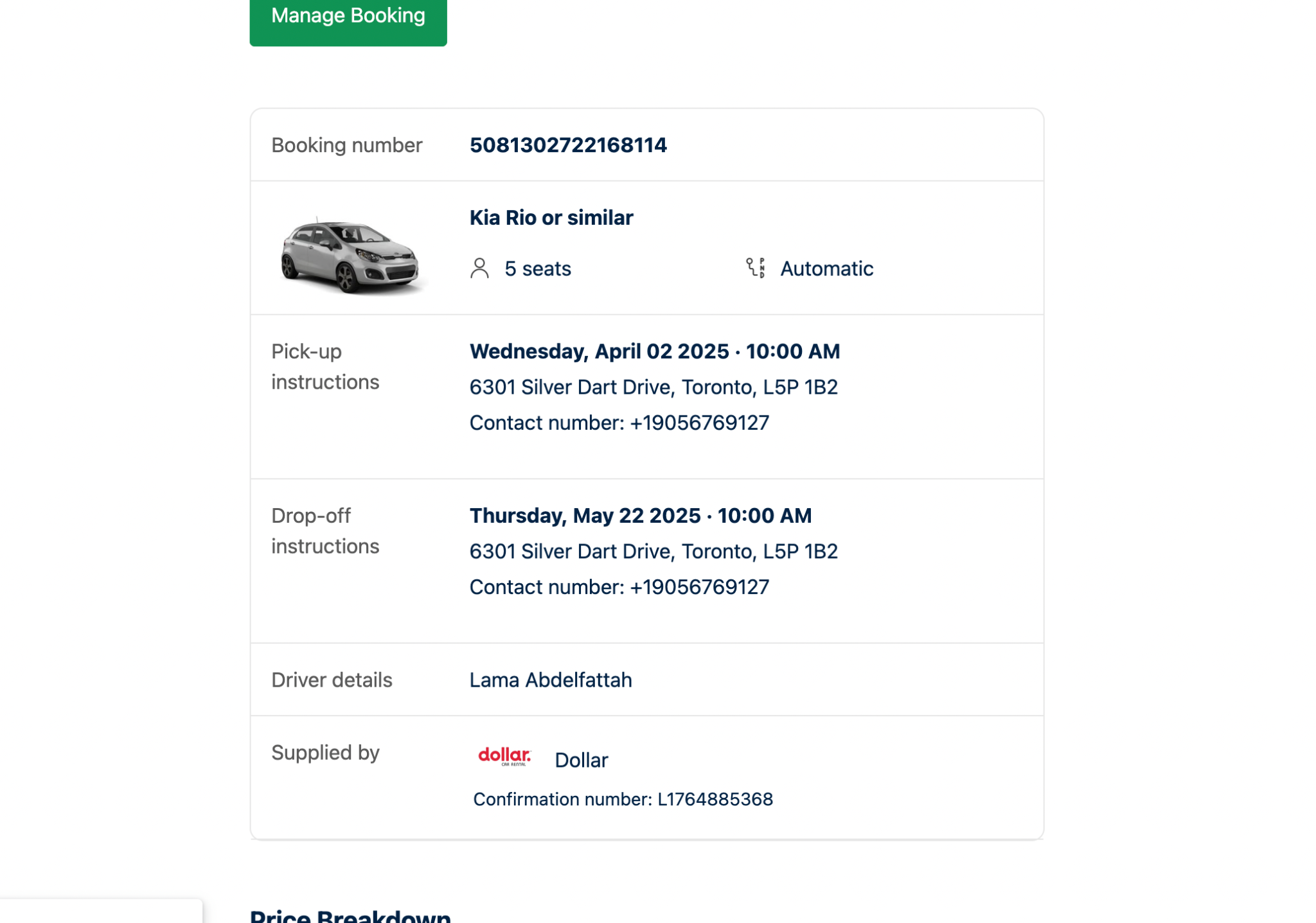
****

Necessary information the customer must fill out during the booking process:

****

After booking the vehicle, we can see the confirmation and booking details:

****

****

**Business Rules and Assumptions**

Key Simplifying Assumptions for Database Modelling:

1. All rental transactions are assumed to be one-way, with vehicles being picked up and dropped off at different branches of the same rental company(e.g., Enterprise, Avis, Hertz).
2. Each vehicle is assigned to a single rental company and maintains a “current location” that updates after every rental.
3. Vehicle details (e.g., make, model, transmission) and extra options (e.g., waivers, car seats) are considered static for the purpose of the reservation period.
4. Vehicle categories (e.g., premium, SUVs, minivan) are predefined and a vehicle can belong to multiple categories.

Other Rules/Assumptions:

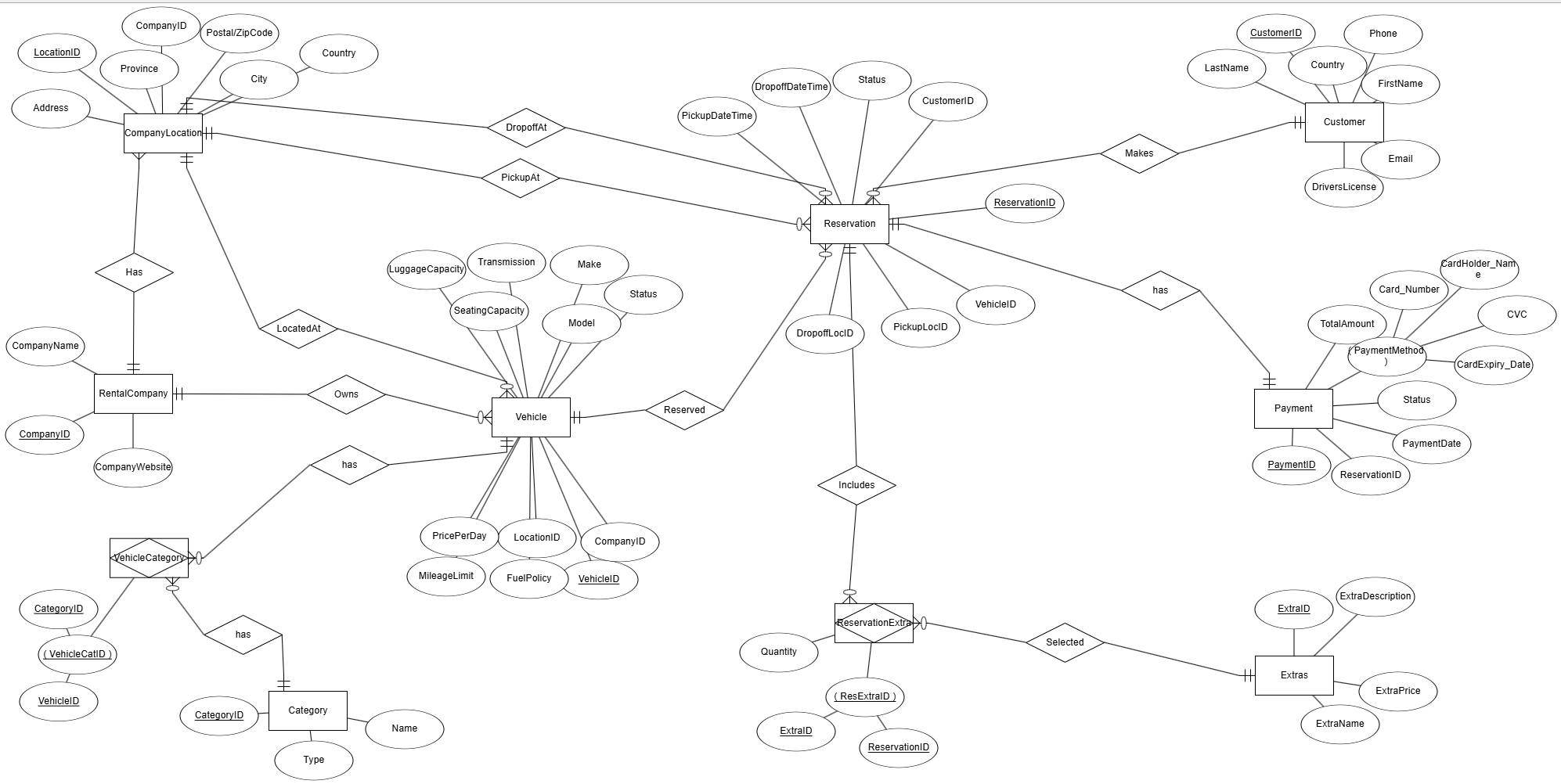
1. A reservation must include a valid customer, a selected vehicle, a pickup location, a dropoff location, and both pickup and dropoff date/time values.
2. A vehicle is available for rental only if its status is “available” and its current location matches the chosen pickup location.
3. One customer can create multiple reservations over time.
4. A single reservation can include multiple extra options, each with a specified quantity (e.g., multiple car seats).
5. Payment details are directly tied to a reservation and must be completed for the reservation to be confirmed.
6. The only Payment Method assumed here is Credit Card.
7. Rental pricing is determined by the vehicle’s price per day and may include additional charges for selected extra options.
8. Company locations are distinct and exclusively associated with a single rental company; thus, pickup and dropoff locations must belong to the same company.
9. All key attributes (such as CompanyID, VehicleID, CustomerID, etc.) are unique and non-null to ensure data integrity.
10. The system does not model vehicle maintenance or repair events as part of the rental process.
11. Transactions (reservations and payments) are recorded as separate audit trails for operational reporting and data integrity.
12. Date and time information for pickup and dropoff is critical for scheduling and is assumed to be provided in a consistent format.
13. Vehicle availability is updated in real-time based on active reservations.

**Relationship between Entities:**

1. A Rental Company owns one or more Company Locations, with each Company Location associated with exactly one Rental Company. (**One-to-Many**)
2. A Rental Company may have multiple vehicles or none at all, but each vehicle is assigned to only one Rental Company. (**One-to-Many**)
3. A vehicle can be classified under multiple categories, and each category can include multiple vehicles. (**Many-to-Many**)
4. A Company Location may have several vehicles or none, while each vehicle is stationed at exactly one Company Location. (**One-to-Many**)
5. A vehicle can have multiple reservations or none, whereas each reservation is linked to a single vehicle. (**One-to-Many**)
6. A Company Location may facilitate numerous reservations for both pickup and drop-off, but each reservation is tied to only one Company Location for pickup or drop-off. (**One-to-Many**)
7. A reservation can include multiple extras, and an extra may be associated with multiple reservations. (**Many-to-Many**)
8. A customer may make several reservations or none, while each reservation is linked to a single customer. (**One-to-Many**)
9. Each reservation is associated with exactly one payment, and each payment corresponds to only one reservation. (**One-to-One**)

**DATABASE MODELLING**

**E-R Model**

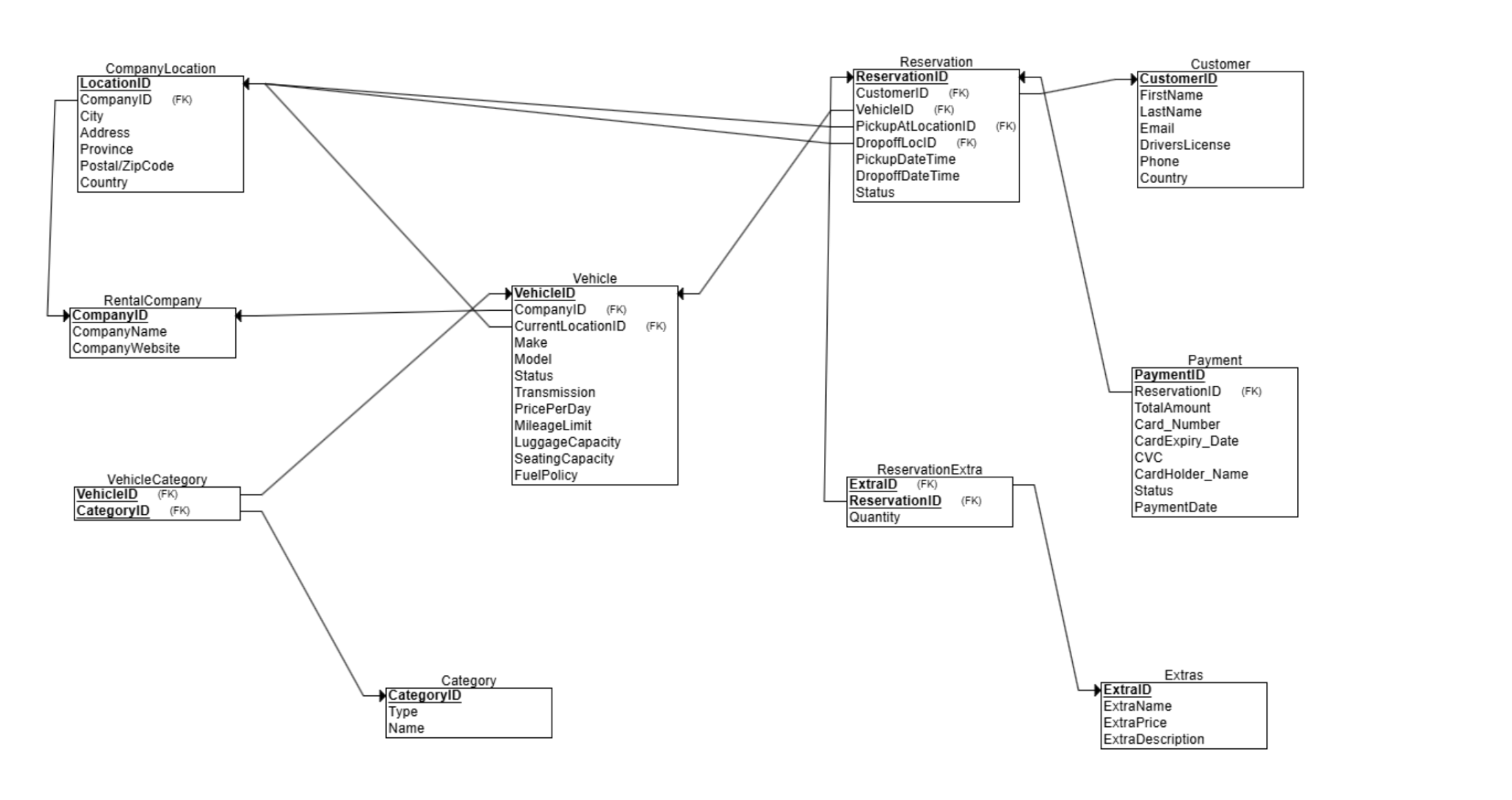
****

In the feedback on our Project Progress Report A, it was suggested that "Payment" should be modelled as a weak entity. However, the professor later confirmed via email that "Payment" is inherently a strong entity; therefore, we did not make any changes in this regard. Additionally, we received feedback stating that the relationship between "Vehicle" and "Category" is not many-to-many, implying that a separate entity called "VehicleCategory" is unnecessary. However, upon revisiting the site, we observed that their relationship is indeed many-to-many, a finding we subsequently verified with the professor in class. To further support this, we have attached screenshots demonstrating that a single category can include multiple cars, and the same car can belong to multiple categories.

****

As observed in all three images, a car, such as the "Vauxhall Grandland," belongs to multiple categories, including "Large Car," "SUVs," and "Medium Car". Similarly, the "Renault Kadjar" is classified under both the "Large Car" and "SUVs" categories. Additionally, it is evident that each category contains multiple cars. This confirms that the relationship between "Vehicle" and "Category" is many-to-many.

**Relational Schema**

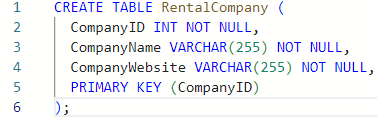
****

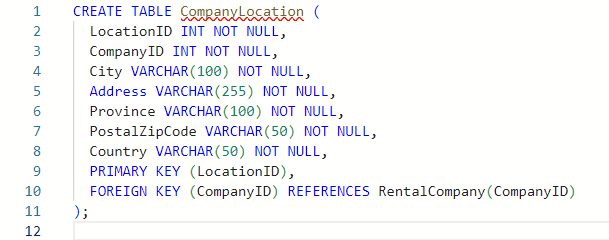
**DATABASE DESIGN AND IMPLEMENTATION**

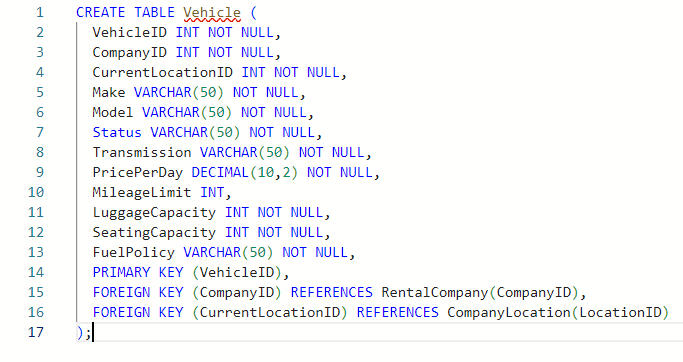
**Data Integrity Control**

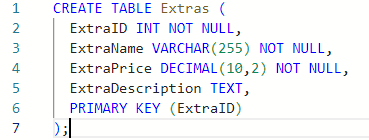
| **Table Name** | **Column Name** | **Key** | **Data Type** | **Allow Nulls** | **Foreign Key** |
| --- | --- | --- | --- | --- | --- |
| **RentalCompany** | CompanyID | PK | INT | No | - |
|  | CompanyName |  | VARCHAR(255) | No | - |
|  | CompanyWebsite |  | VARCHAR(255) | No | - |
| **CompanyLocation** | LocationID | PK | INT | No | - |
|  | CompanyID |  | INT | No | RentalCompany(CompanyID) |
|  | City |  | VARCHAR(100) | No | - |
|  | Address |  | VARCHAR(255) | No | - |
|  | Province |  | VARCHAR(100) | No | - |
|  | PostalZipCode |  | VARCHAR(50) | No | - |
| **Vehicle** | VehicleID | PK | INT | No | - |
|  | CompanyID |  | INT | No | RentalCompany(CompanyID) |
|  | CurrentLocationID |  | INT | No | CompanyLocation(LocationID) |
|  | Make |  | VARCHAR(50) | No | - |
|  | Model |  | VARCHAR(50) | No | - |
|  | Status |  | VARCHAR(50) | No | - |
|  | Transmission |  | VARCHAR(50) | No | - |
|  | PricePerDay |  | DECIMAL(10,2) | No | - |
|  | MileageLimit |  | INT | Yes | - |
|  | LuggageCapacity |  | INT | No | - |
|  | SeatingCapacity |  | INT | No | - |
|  | FuelPolicy |  | VARCHAR(50) | No | - |
| **Extras** | ExtraID | PK | INT | No | - |
|  | ExtraName |  | VARCHAR(255) | No | - |
|  | ExtraPrice |  | DECIMAL(10,2) | No | - |
|  | ExtraDescription |  | TEXT | Yes | - |
| **Customer** | CustomerID | PK | INT | No | - |
|  | FirstName |  | VARCHAR(100) | No | - |
|  | LastName |  | VARCHAR(100) | No | - |
|  | Email |  | VARCHAR(255) | No | - |
|  | DriversLicense |  | VARCHAR(50) | No | - |
|  | Phone |  | VARCHAR(50) | No | - |
|  | Country |  | VARCHAR(50) | No | - |
| **Category** | CategoryID | PK | INT | No | - |
|  | Type |  | VARCHAR(50) | No | - |
|  | Name |  | VARCHAR(100) | No | - |
| **VehicleCategory** | VehicleID | PK, FK | INT | No | Vehicle(VehicleID) |
|  | CategoryID | PK, FK | INT | No | Category(CategoryID) |
| **Reservation** | ReservationID | PK | INT | No | - |
|  | CustomerID |  | INT | No | Customer(CustomerID) |
|  | VehicleID |  | INT | No | Vehicle(VehicleID) |
|  | PickupLocID |  | INT | No | CompanyLocation(LocationID) |
|  | DropoffLocID |  | INT | No | CompanyLocation(LocationID) |
|  | PickupDateTime |  | DATETIME | No | - |
|  | DropoffDateTime |  | DATETIME | No | - |
|  | Status |  | VARCHAR(50) | No | - |
| **Payment** | PaymentID | PK | INT | No | - |
|  | ReservationID |  | INT | No | Reservation(ReservationID) |
|  | TotalAmount |  | DECIMAL(10,2) | No | - |
|  | PaymentMethod |  | VARCHAR(50) | No | - |
|  | Status |  | VARCHAR(50) | No | - |
|  | PaymentDate |  | DATETIME | No | - |
|  | Card\_Number |  | VARCHAR(19) | No |  |
|  | CVC |  | VARCHAR(4) | No |  |
|  | CardExpiry\_Date |  | VARCHAR(5) | No |  |
|  | Card\_Holder \_Name |  | VARCHAR(50) | No |  |
| **ReservationExtra** | ExtraID | PK, FK | INT | No | Extras(ExtraID) |
|  | ReservationID | PK, FK | INT | No | Reservation(ReservationID) |
|  | Quantity |  | INT | No | - |

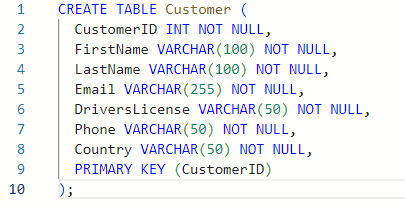
**Table Design**

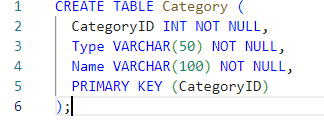
****

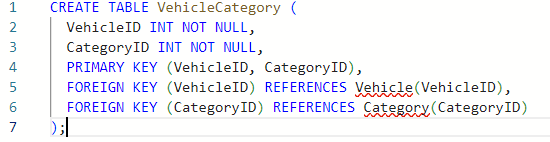
****

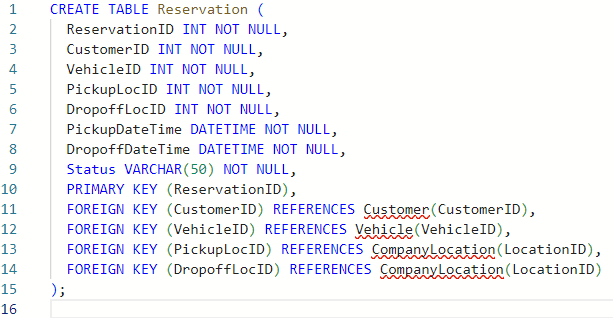
****

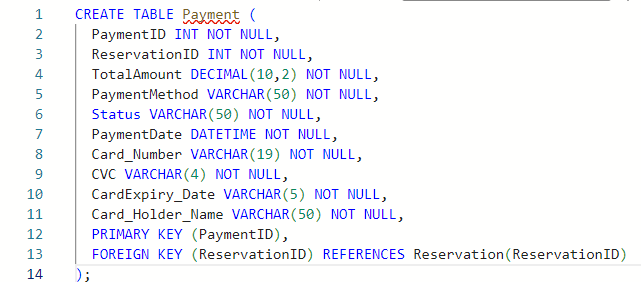
****

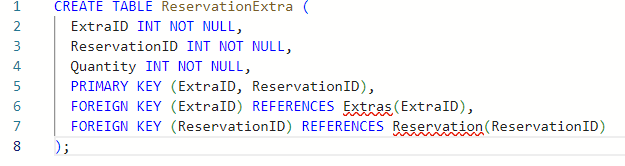
****

****

****

****

****

****

\*Please ignore the underlined tables, all queries ran successfully

### 

### **Entity Integrity**

### Entity integrity ensures that each table has a unique identifier as primary key. Each table has a unique identifier such as:

* CompanyID for RentalCompany Table.
* LocationID for CompanyLocation Table..
* VehicleID for Vehicle entity Table.
* ExtraID for Extras Table.
* CustomerID for Customer Table.
* CategoryID for Category Table.
* A composite key(VehicleID, CategoryID) for VehicleCategory Table.
* ReservationID for Reservation Table.
* PaymentID for Payment Table.
* A composite key (ExtraID, ReservationID) for ReservationExtra Table.

**Reference Integrity**

Reference integrity ensures valid relationships between tables by connecting them with Foreign Keys.

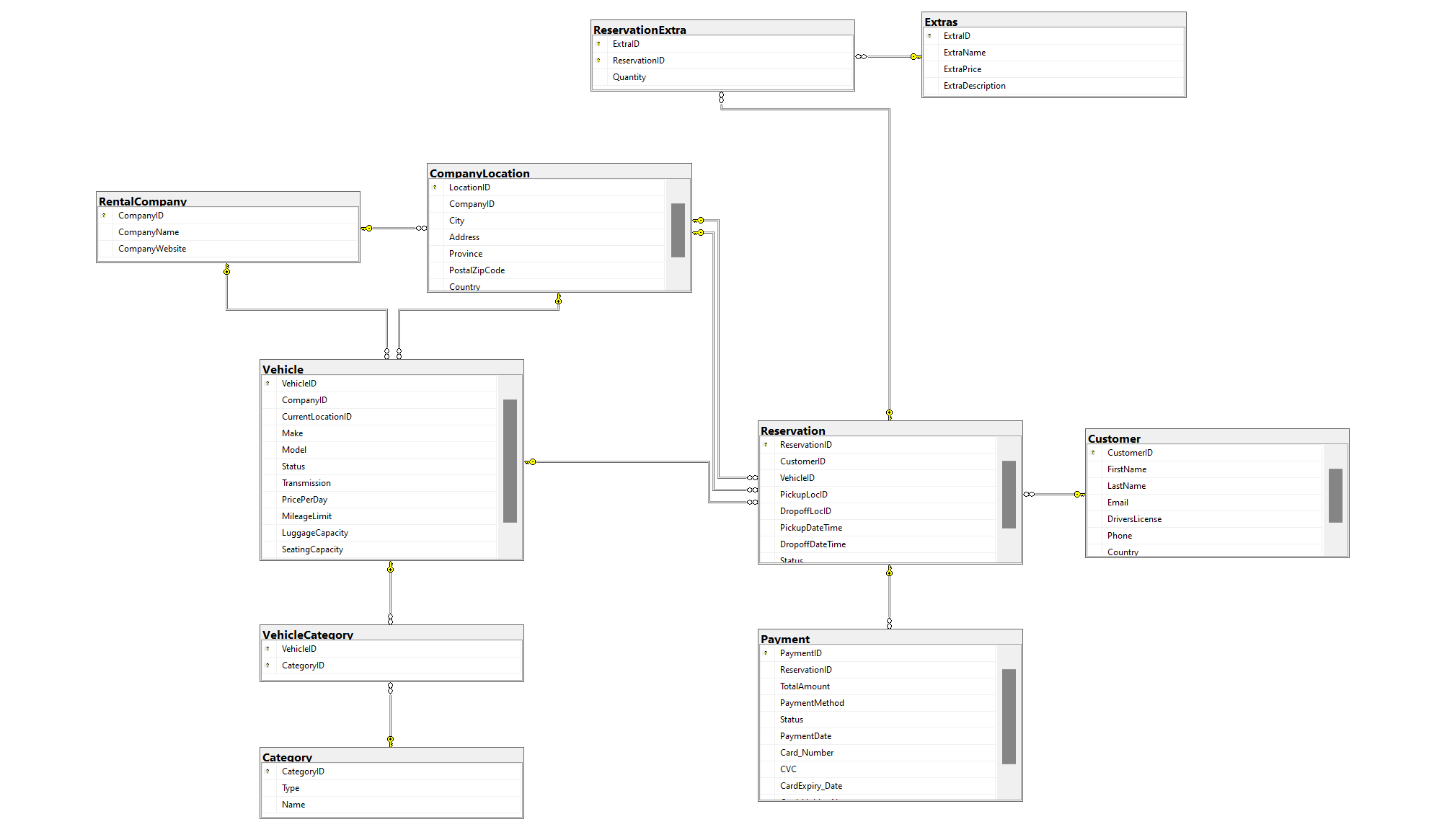
* In CompanyLocation Table:
  + CompanyID is a foreign key referencing RentalCompany.CompanyID, ensuring every location belongs to a valid company.
* In Vehicle Table:
  + CompanyID is a foreign key referencing RentalCompany.CompanyID, ensuring every vehicle belongs to a valid company.
  + CurrentLocationID is a foreign key referencing CompanyLocation.LocationID, ensuring vehicles are associated with existing locations.
* In VehicleCategory Table:
  + VehicleID is a foreign key referencing Vehicle.VehicleID, ensuring only valid vehicles are categorized.
  + CategoryID is a foreign key referencing Category.CategoryID, ensuring categories exist.
* In Reservation Table:
  + CustomerID is a foreign key referencing Customer.CustomerID, ensuring each reservation belongs to a valid customer.
  + VehicleID is a foreign key referencing Vehicle.VehicleID, ensuring only valid vehicles are reserved.
  + PickupLocID and DropoffLocID are foreign keys referencing CompanyLocation.LocationID, ensuring valid pickup/drop-off locations.
* In Payment Table:
  + ReservationID is a foreign key referencing Reservation.ReservationID, ensuring payments correspond to valid reservations
* In ReservationExtra Table:
  + ExtraID is a foreign key referencing Extras.ExtraID, ensuring extra services are valid.
  + ReservationID is a foreign key referencing Reservation.ReservationID, ensuring extras belong to valid reservations.

**Domain Integrity**

Domain integrity ensures that columns have appropriate data types and constraints to maintain data validity.

* CompanyName, Make, Model, Status, Transmission, FuelPolicy, PaymentMethod, Card\_Holder\_Name: Stored as VARCHAR, ensuring only text is entered.
* There is a NUMERIC constraint on PricePerDay DECIMAL(10,2) that ensures the maximum numbers should be10 with 2 digits after the decimal.
* PickupDateTime, DropoffDateTime, PaymentDate: Stored as DATETIME, ensuring valid date formats.
* SeatingCapacity, MileageLimit, LuggageCapacity, Quantity: Stored as INT, ensuring numeric values. Where MileageLimit allows NULL to indicate unlimited mileage.
* CVC VARCHAR(4): Ensures valid CVC length is either 3 (for most credit cards) or 4(American express).
* Card\_Number VARCHAR(19): Ensures card numbers fit standard lengths.
* CardExpiry\_Date VARCHAR(5): Ensures expiry date follows MM/YY format.

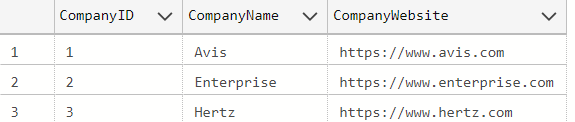
**Relational Table**

****

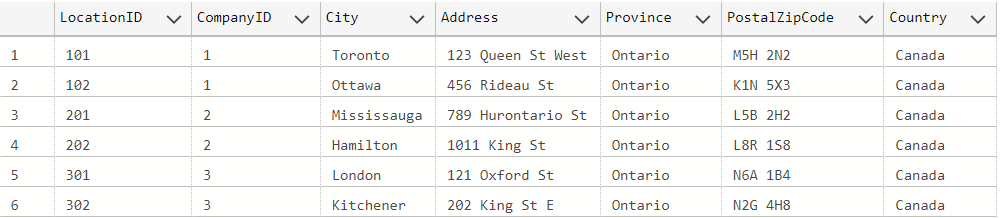
**\***Not all the column names are visible due to space constraints.

**Sample Data**

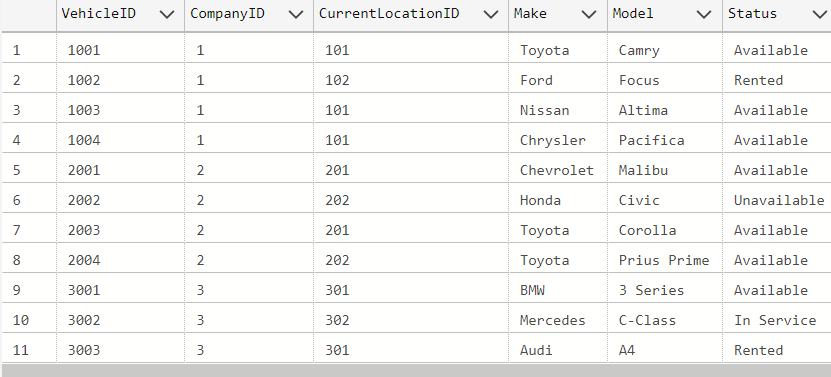
RentalCompany

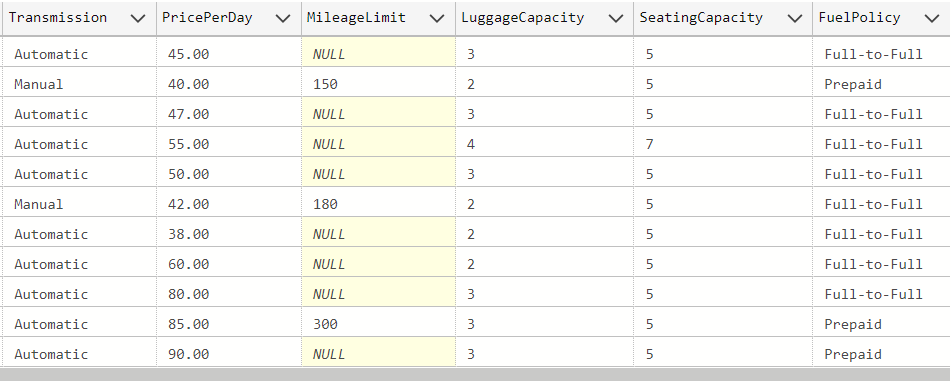


CompanyLocation



Vehicle





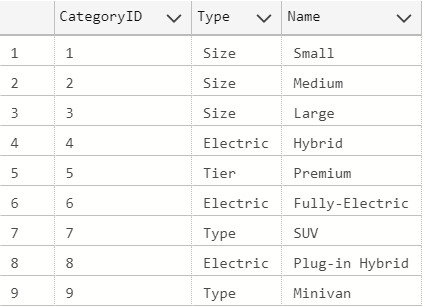
Extras



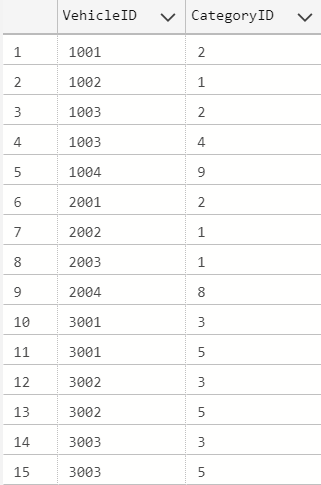
Customer



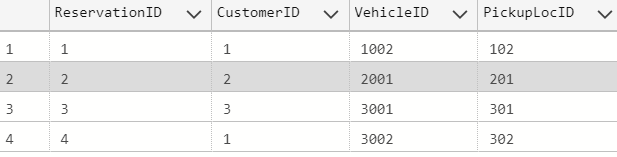
Category

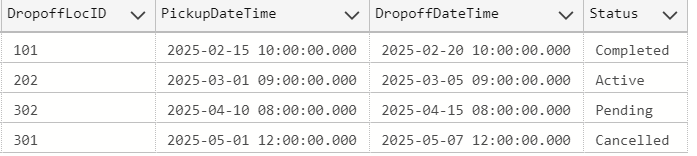


VehicleCategory

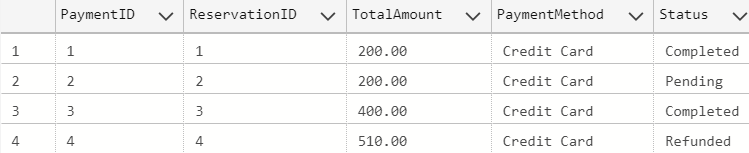


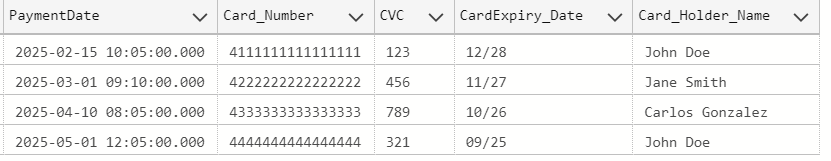
Reservation



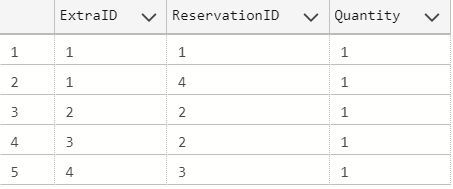


Payement





ReservationExtra



**Normalization Check:**

**1NF (First Normal Form):**

Each column contains atomic (indivisible) values. Each row is uniquely identified by a primary key. There are no multiple values in a single column or nested tables within the tables.

Therefore, All tables are in 1NF.

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

To be in 2NF, a table must be in 1NF and Have no partial dependency (i.e., no non-key column depends on just part of a composite key).

There are two such tables:

VehicleCategory (VehicleID, CategoryID)

Only includes foreign keys and no partial dependencies exist.

ReservationExtra (ExtraID, ReservationID)

Quantity depends on the whole composite key.

Therefore, All tables are in 2NF

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

To be in 3NF, a table must be in 2NF and have no transitive dependencies (non-key columns must not depend on other non key columns)

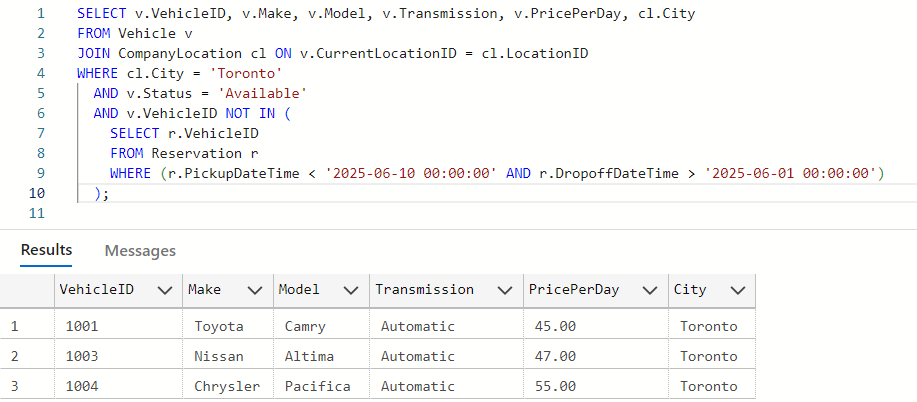
None of the tables have transitive dependencies except the Payment Table.

Card Holder Name depends on a non key column Payment Method.

**Sample SQL Queries**

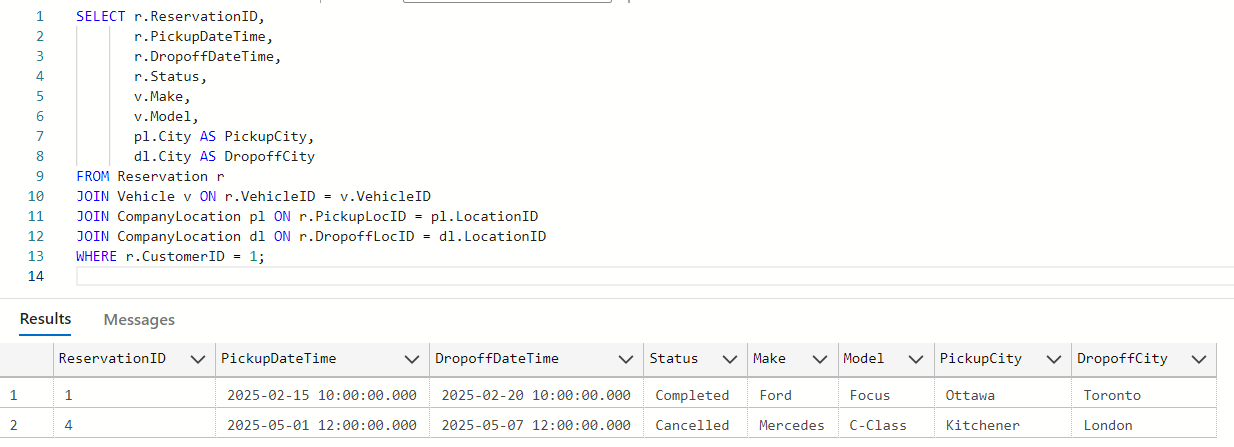
1. List Available Vehicles at a Specific Location and Date Range

This query finds vehicles that are available in a given city (in this example, Toronto) and that are not booked during a specific period.



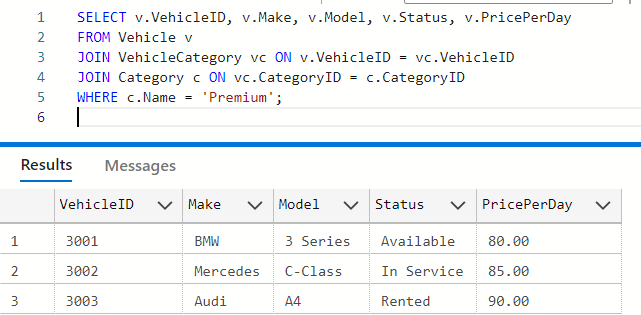
1. Retrieve All Reservations for a Specific Customer with Details

This query shows the reservations for a given customer (change the CustomerID as needed) along with vehicle and location details.



1. Search for Vehicles by Category (e.g., Premium)

This query retrieves all vehicles that belong to the Premium category.



4. Reservations that Include a Specific Extra (e.g., 'Child Seat')

This query retrieves reservations where the selected extra option is a Child Seat. It joins with customer details for context.

****

**EVALUATION AND CONCLUSION**

**Overview of Design and Implementation**

To build a scalable and efficient car rental database system, we adopted a methodical development cycle involving requirements analysis, ER modeling, schema normalization, and SQL Server based implementation. Emphasis was placed on maintaining data integrity through primary and foreign keys and enforcing consistency via domain and entity constraints. Performance was enhanced through indexing on frequently queried attributes. The final schema supports all core operations such as real-time vehicle availability tracking, secure booking, and dynamic pricing, providing a foundation for a reliable and extensible rental platform.

**Achievement of the project**

The Rentalcars.com database project successfully delivered a fully functional system that streamlines the car rental process for global users. The database enables real-time vehicle searches based on location, date, and vehicle type, presenting customers with transparent pricing and availability across 60,000+ rental partners. Key achievements include a secure booking module with integrated payment processing, automated confirmation emails/SMS, and a user-friendly interface available in multiple languages. On the administrative side, the system tracks fleet inventory, prevents overbooking, and provides analytics on rental trends. By centralizing customer profiles, payment methods, and rental histories, the database significantly enhances operational efficiency while maintaining Rentalcars.com’s commitment to hassle free service.

**Further development and improvement**

To build on this project, several enhancements could further elevate the platform. Future iterations might integrate partnerships with ride-sharing services (e.g., Uber or Lyft) to offer end-to-end travel solutions. Dynamic pricing algorithms could adjust rates based on demand fluctuations, local events, or fuel prices. Additional vehicle details such as real-time GPS tracking, emissions data, or user-submitted cleanliness ratings would help customers make informed choices. Expanding the loyalty program to include tiered rewards (e.g., free upgrades for frequent renters) and personalized discounts could boost retention. Finally, adding AI-driven features like predictive maintenance alerts for rental fleets or chatbot-assisted bookings would align with industry innovations.

### **Advantages of the Database Package (MICROSOFT SQL SERVER)**

Microsoft SQL Server provided an ideal foundation for Rentalcars.com's database system due to its strong scalability, effortlessly accommodating growing global rental transactions and user traffic. The platform's relational constraints ensure data integrity across linked tables (e.g., customer bookings to vehicle inventory), preventing inconsistencies. As an industry-standard tool, SQL Server offered seamless integration with Rentalcars.com's front-end booking interface, while its intuitive design allowed student developers to contribute effectively. The cost-effective licensing (for educational use) and compatibility with cloud deployments aligned well with project budgets. However, limitations emerged during stress testing: concurrency delays occurred during peak booking simulations, and the base version required supplemental tools for end-to-end encryption of sensitive payment data. Performance also varied based on server hardware, and initial design iterations revealed redundancy risks in vehicle availability tables until normalization was refined.

### **Limitations of the Database Package**

The database package we used, Microsoft SQL Server, comes with several limitations worth noting. It may encounter concurrency issues, leading to delays when multiple transactions are processed at the same time. The basic version also lacks some advanced, enterprise-level features, which could restrict functionality in larger-scale applications. From a security standpoint, the platform requires additional configuration or third-party tools to implement strong encryption and effective access control, especially for beginners. Its performance is highly dependent on the hardware it runs on, meaning lower-end systems may experience slowdowns. Lastly, the system does not provide warnings for poorly normalized data, which can lead to redundancy and inconsistent tables if the schema isn’t carefully designed from the outset.

### **Experience Learned**

Developing Rentalcars.com’s database was a transformative learning opportunity. The team gained hands-on expertise in relational database design, constructing ER diagrams that mapped customer accounts, rental transactions, and fleet logistics. SQL query optimization became critical for instance, indexing frequently searched fields like pickup locations and vehicle types accelerated response times. By implementing primary/foreign keys, we enforced referential integrity (e.g., ensuring a booked vehicle couldn’t be deleted from the inventory table). Practical challenges like concurrency management taught us to balance real-time bookings with system performance. Collaboratively, we divided tasks into one subgroup focused on table normalization to eliminate redundancy, while others designed sample queries for pricing algorithms. The project honed time management (meeting iterative deadlines) and debugging skills when testing edge cases (e.g., overlapping reservations). Most importantly, we learned to design for scalability structuring tables to support future features like dynamic pricing or AI-driven recommendations. These transferable skills will prove invaluable in professional database roles.

**Contribution and Time Spent**

* Introduction of the Project and company.
* Database Model - Business rules and assumptions.
* Developing E-R model- Company Location and Rental Company entity.
* Database Modeling Relational Schema.
* Table Design of Company Location and Rental Company.
* Implementation of Relational Table.
* Running Sample Query - List Available Vehicles at a Specific Location and Date Range.
* Evaluation and Conclusion - Further development and improvement
* Objective of the company.
* Data Modeling - Relationship between Entities.
* Developing E-R model- Vehicle category and Category entity.
* Database Design and Implementation of Data Integrity control.
* Table Design of Vehicle category and Category.
* Implementation of Relational Table.
* Running Sample Query - Retrieve All Reservations for a Specific Customer with Details
* Advantages of the Database Package (MICROSOFT SQL SERVER) and Limitations.
* Company scope.
* Database Model - External Views.
* Developing E-R model- Reservation and Reservation Extra entity.
* Database Modeling Relational Schema.
* Table Design of Category Reservation and Reservation Extra
* Inserting Sample Data in the table.
* Running Sample Query - Reservations that Include a Specific Extra.
* Evaluation and Conclusion - Achievement of the project.
* Functional requirements of the application.
* Data Modeling- Relationship between Entities.
* Developing E-R model- Vehicle and Extra Entity.
* Database Design and Implementation of Data Integrity control.
* Table Design of Vehicle and Extra.
* Inserting Sample Data in the table.
* Running Sample Query - Search for Vehicles by Category.
* Evaluation and Conclusion - Overview of Design and Implementation
* Functional requirements of the application.
* Database Model - External Views and Business rules and assumptions.
* Developing E-R model- Customer and Payment entity.
* Database Design and Implementation of Data Integrity control.
* Table design of Customer and Payment.
* Inserting Sample Data in the table.
* Normalisation Check for all 1,2 and 3 Normal Form.
* Evaluation and Conclusion - Experience Learned

**THANK YOU!**