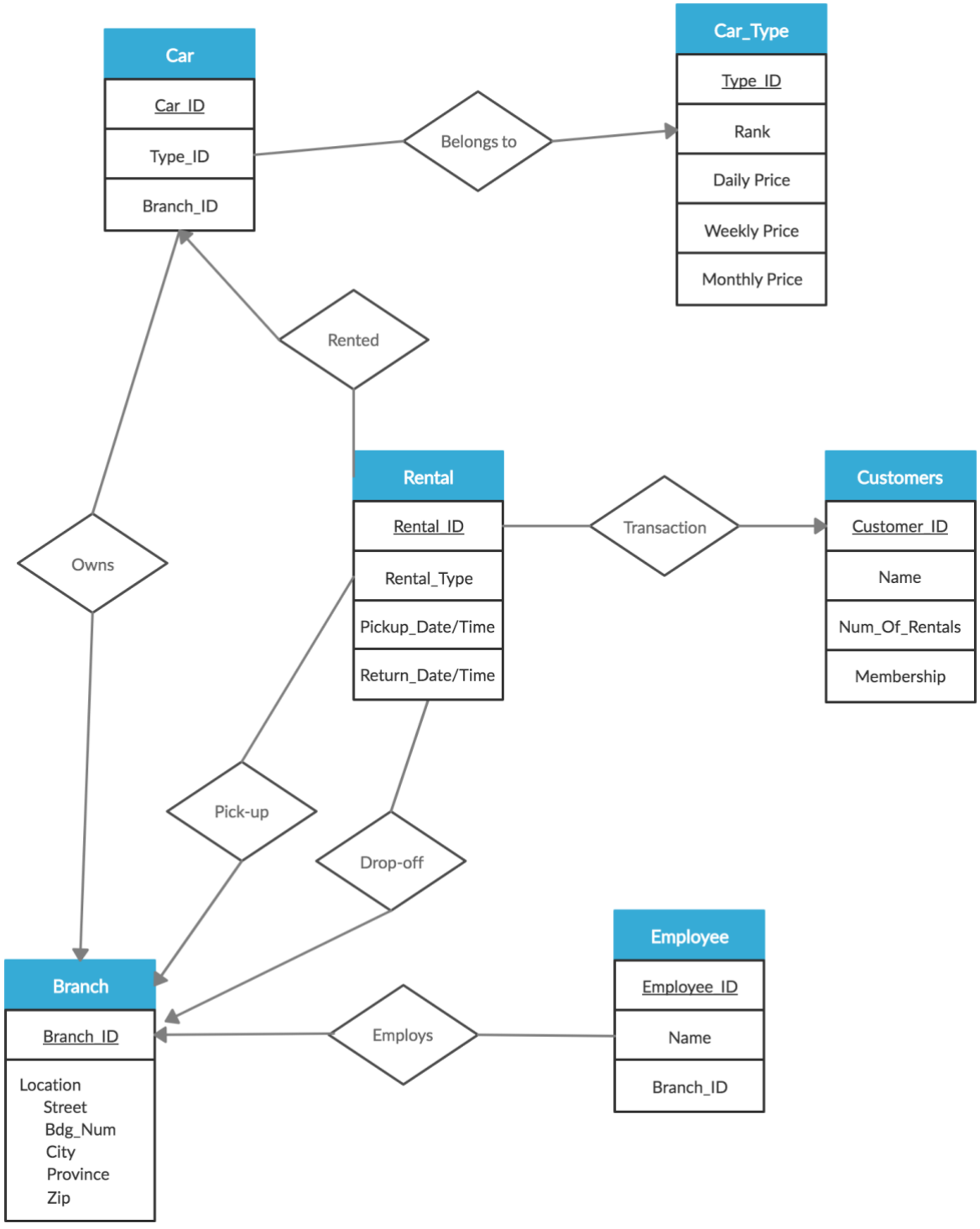
CMB Rentals

CMB Rentals – Design Document

CMB Rentals Inc.

CMPT 291 Group 11

Entity-Relationship Diagram



Design Choices

General

We decided to use Visual Studio for this project, using Windows Forms written in C#. The Windows Forms interface is user-friendly and easy to use, which is perfect for individuals who do not have much experience using computers. We used six different objects – branches, employees, customers, cars, car types, and transactions. Each instance of any of the objects has an ID which distinguishes it from other objects of its type. This made referencing objects much easier on the Visual Studios end. We also have tables to keep track of IDs the application is currently on (for increasing by 1 each time), as well as relational tables for each relation that exists on the database.

Upon opening the application, users will be presented with choosing if they are a customer or an employee. For each of the two streams, users traverse through forms by selecting easy-to-see buttons, with back buttons on each page to allow simple traversal. Each form will hide when a new one is called, which helps prevent cluttering the user's screen. Each form is simple to follow and presents information in a logical order. When users have to enter numbers, a number counter is used. A date-time picker is used for dates, and drop-down lists are used for when there are limited options for the user to pick from. This eliminates the need for error checking certain information boxes. After completing must actions, pop-up messages occur to inform the user that their actions were successful.

Customer Stream

The customer stream first has the user enter their user ID and their name, and from this information the users will be able to either set drop-off dates for ongoing rentals, as well as start a new rental. Should they choose the latter, the following form allows the customer to select based on their pick-up and location preferences. We decided to include the email to differentiate between customers, since no two customers can have the same email. The customer IDs will be based on the email they entered. This also allows quick checking if they are Gold star members or not.

Having the customers enter rental information prior to showing the list of cars allows the program to ignore vehicles that will not be available for the duration that the user selected. It uses the cars available at each branch, along with the transactions per car to identify which ones will be available, which will prevent double-booking a car. There are pop-ups which block users from not entering information, which prevents faulty queries to be made for the searches.

Employee Screen

We used one main menu screen for the employee users that allowed a clear view of each of the actions available. This includes buttons for add/delete cars, add/delete employees, add/delete customers, view/process transactions, and generate report. Each of the lists presented on each of these separate forms are formatted in clean, easy-to-view ways, with the use of checklists to delete/process multiple items at a time. For each of the add buttons, a separate form will appear which allows the user to enter the specifications that they want for the new instance.

Having add/delete cars, employees and customers on separate forms allows the program to only have to access one table per form, which eases the load on pulling from the database. For the report generator, a PDF is created which is displayed on a screen that pops up. It includes a list of each employee and car for each branch, as well as a general list of all customers and rental transactions. It first pulls information from multiple tables, formats into string arrays, and uses Sautinsoft.Document to format the PDF generation.

Referential Constraints

We made the use of IDs as the primary keys for each of the tables, except the branch table – for that we used the branch location name. Using simple numbers that are guaranteed to be different for each instance of an object helps keep the data organized and easily referenceable. We also implemented foreign keys for the car and employee objects, being the branch ID. Using these foreign keys added to the referential integrity of the data, as each employee will be working under one specific branch, and each car will be available at a branch. Should a car be returned to a different branch, it will change the foreign key to that branch ID.