

Database Management System

Final Project Report

Section 2 - Group 10

The Team

Aditya Shah - 201801008

Rishi Shivani - 201801073

Pavan Kotak - 201801120

Bhrugu Dave - 201801401

Table of Contents

Revision History	4
Product Description	6
Purpose	6
Intended Audience and Reading Suggestions	6
Product Scope	6
Description	6
Fact-Finding Phase	11
Background reading	11
Interviews	13
Griefingdor: (Role Play) Interview Plan For CEO	13
Griefingdor: (Role Play) Interview Summary For CEO	14
Griefingdor: (Role Play) Interview Plan For Branch Managers	15
Griefingdor: (Role Play) Interview Summary For Branch Managers	16
Griefingdor: (Role Play) Interview Plan For Accountants	17
Griefingdor: (Role Play) Interview Summary For Accountants	18
Griefingdor: (Role Play) Interview Plan For Clerks	19
Griefingdor: (Role Play) Interview Summary For Clerks	20
Questionnaires	21
Summary of Responses	22
Griefingdor: Observations	26
Fact-Finding Chart	27
Expected Deliverables	28
Temporary Database Design	29
User Classes and Characteristics	30
Operating Environment	32

Product Function and User Privileges	34
Assumptions	36
Business Constraints	36
Nouns and Verbs	37
Accepted Nouns and Verbs List	40
Rejected Nouns	41
Final ER Diagram	44
Functional Dependencies	45
1NF(First Normal Form)	46
2NF(Second Normal Form)	46
3NF(Third Normal Form)	46
Final Schema	47
CSV Files	47
Final Relational Schema Diagram	48
DDL Scripts	49
Rate Table	49
Branch Record	50
Customer	51
Branch Record Phone Number	52
Customer Phone Number	53
Customer City	54
Salary	55
Car Record	56

Maintenance Record	57
Employee Record	58
Rents	59
Employee Record Phone Number	60

Revision History

Name	Date	Reason for change	Version
Car rental service	26/09/2020		1.0
Car rental service	30/09/2020	Finalizing the document after relevant steps.	2.0
Car rental service	03/10/2020	Revision of the document and updating the background reading	3.0
Car rental service	28/11/2020	Updating the document with current schemas	4.0

Product Description

1. Purpose

The purpose of this product is to provide a solution for the problems faced by local car rental service providers like better connectivity among the multiple outlets, proper management of staff and cars within the outlet, availability of instantaneous records for the same, and an overall secure and adequate fund management system.

2. Intended Audience and Reading Suggestions

Stakeholders like the owner of the car rental service should go through the sections of product descriptions, different privileges are available for different users of this database, the fact-finding section, and the business constraint of this database. While staff members, who will be the administrators of this database, should go through the sections following the user classes and characteristics section. The tester of this database should go through the whole document starting from the fact-finding section. The developers should look in details mentioned in sections after the Fact Finding Chart which will give insights about the design and implementation of the required database.

3. Product Scope

This product mainly provides convenience to different car rental service providers who still maintain their records on paper or using excel sheets. This software offers convenient, secure, and reliable services for maintaining, updating, and deleting records from the database. Also, this product provides a relatively better interface for handling the records for the managerial staff. Thus, the scope of this database extends to services looking for a better database management system.

4. Description

In this modern world, everyone needs a form of transportation that is fast, convenient, and easy to access. This form of transportation isn't static for everyone, it will change person to person and even day to day. But for a while, now a good way to combine multiple forms of transportations i.e. having a personal car, hiring a taxi, etc has been to rent a car. Now, as more people are inclined towards this sort of ideology and since it gets a boost from start-ups advertising about the same thing, we end up in a place where the local rental

industry gets a boost. But there is a hindrance in the form of the convenience factor as most of the local rental places haven't yet migrated to an online platform so it's hard for the management to keep track of their cars and for the customers to browse their selection of cars before visiting the place. So, for digitizing the simple implementation of their workflow into a product that can be accessed by various types of users without a restriction of geography is required.

Now, since we are designing this product for the current client which is Ghost Car Rental Services, we must first properly understand the problems that they are facing and what their current workflow is so that we can provide a solution for them that is the most optimal. Therefore, firstly we need a detailed description of the current workflow and then speculate what solutions for the interface are possible.

The primary thing that any car rental place would need is cars, as without it there the place isn't really a car rental place. So, currently, GCRS maintains the records of cars in an excel spreadsheet, which is updatable by only the main branch as it is stored locally there. Now, this spreadsheet contains information along the lines of the license plate and chassis number of the cars. And any new cars that are purchased can only be managed by the main branch. From this there is a clear problem of the need to update the records at each and every branch as these updates, in some scenarios, can overwhelm the people doing these tasks at the main branch as the company expects the all the branches to have an updated record so that the customer has the most recent list of cars. This feature of concurrency is an important aspect, in the company's eye, is an important aspect of fast and efficient customer service.

Speaking of customer service, let's discuss the people who are the most important, nay the only ones who provide it, the clerical staff and the drivers. Currently, the management of drivers at GCRS can most aptly be termed as haphazard. The records of employed drivers and staff are maintained on books managed by the main branch using a unique employee ID. So, even if a branch wanted to hire a temp worker it has to coordinate with the main branch so that the main branch can have updated information on the workers, even updating personal information requires one to work with the main branch and of course, maintaining salary records is a nightmare. All the information on how much to pay each member of the staff is only available to view at the end of the month and all of this is sent by the main branch. The most obvious solution to this problem is to provide an easy to grasp and user-friendly UI, database synchronization between different franchises, and the main branch. This will lay

off some manual work done by the clerks of different franchises and also the main branch.

As it is obvious the central theme of all the discussions until this point relates to the staff of each and every branch. The records of staff are to be maintained branch-wise at each and every individual branch and as well as should be made available to the main branch. Thus for this purpose, branch managers are held responsible. It's the job of the branch manager to maintain records of the fleet of cars under that branch, staff employed and answering any main issues caused at the branch. Also, they need to maintain concurrency with the main branch.

It is quite obvious that all the individual branches depend too much on the main branch and the owners also have to depend on the main branch staff for any sort of data. Thus, in the case of any sort of malfunction at the main branch, the company is very much vulnerable to important data loss. Also, let's not forget the point that all these records are maintained on either books or locally stored spreadsheets. Thus, the theft of data is also a very serious possibility. Thus, an integrated database would be the aptest solution. It would provide a concurrent database that could have limited access as well as it would give full access to the owners to the statistical data at any given time.

Up till now, we have just discussed the staff and the fleet of cars at a branch. Let us also see how the data of customers is being managed. At the moment, any customer can acquire the service of GCRS by either calling the branch or on spot booking. Before the car is handed over, the branch staff, more precisely the clerk, takes all the necessary information about the customer and proof ID before handing over the car. These records are maintained by a unique id which is given to the customer also, so that the next time the customer comes, he need not fill up the details form again. This system eliminates the redundancy at individual branch levels but causes problems across different branches. Suppose a customer registered at a particular branch and obtained a unique id for identification. But this id won't work if he chooses to obtain services from a different branch. As such this data is branch limited, also, synchronization among the branches is not an option under the current system implemented for handling data. Thus, databases would maintain and provide synchronized data across branches about customers and also maintain uniqueness using customer ID given to each customer at time of registration.

So far we have discussed how different important data about individuals is being managed. But, we haven't discussed how the trip records as well as maintenance records are being handled. Suppose, a customer decides to acquire services of GCRS via a phone call. First of all, he is asked whether he has a unique customer id or not. If not he is asked to give all the required information over the phone call or is asked to come over to the branch for the same. Then he is asked, whether he requires a driver or not. If yes, then the call attendee checks whether a driver is available or not and assigns one for the same if available. Also, the customer is asked about what sort of car he wants, and it's the job of the staff to determine whether a car fulfilling the requirements is available or not. Thus, the staff has to do tedious work of going through for each and every booking request. Once this is fulfilled, the clerk has to jot down each and every trip in a record book describing which car is rented, which driver is allotted, and all sorts of required details. This record is maintained in books, thus, for calculating profit at any given time one has to calculate a significant amount of data. Once all these details are filled, the car is handed over and the pickup destination and time period with drop off destination is noted down after the trip is over. If the rented car is dropped on a different outlet, then it also causes some serious trouble for the clerks to synchronize the data for each trip and how to distribute the money between the outlets. Thus, intra and inter-city trip data updation is also a major issue.

One of the problems is that when a customer wants to rent a car, a clerk needs to show the available options, if a customer is booking via phone then the clerk describes the available cars according to the requirement of the customer, if a customer rents via on spot booking, then the clerk shows him the best option available. A customer would like to browse all the available options by himself and choose the best for him rather than the clerk showing him the car or describing the vehicle over the phone. So, we intend to develop a user-friendly interface for customers such that a customer can browse over the available options and filter the options according to his choice and rates of individual cars are also displayed which is maintained using a rate table.

We can maintain the records for all the franchises including the main branch by implementing an efficient database management system that will contain the staff data, customer data, fleet management, and records of the trips. This will provide an efficient solution to the concurrency of the data and all the databases can be synchronized to a cloud among the various outlets. This would result in a better and efficient way of inserting, updating, and also deleting the data into the

database. This will also help to ease the maintenance of the database among the various outlets.

The integration of the database will provide users of different classes with different privileges. This means that owners will have options for providing limited access to the database. Customers will have access to only the database of available cars on a particular franchise. Staff will also have limited roles to the database according to their job position such as clerks, branch managers, accountants, and drivers.

One feature that the company so far has failed to incorporate is the rating system for the vehicles and drivers given by the customers. We intend to implement such a system so that a customer can give a rating in the stars and if a customer wishes can also leave written comments along with the rating. This will help other customers to choose the vehicle and a driver considering the previous ratings of the respective car and driver. Implementing this system will also help the customer filter the available results according to the previous ratings and reviews. Also, the company can improve its facilities and services according to the reviews and ratings are given by the customers after the completion of the trip. This feature shall be integrated with the UI and displayed when asked.

Fact-Finding Phase

A. Background reading

To get a proper understanding of what customers of GCRS need and what the company owners require in order to run their business effectively we referred to different books and blogs.

Firstly, to get the best understanding of the company's most important aspect requirement we went through a blog that stated what a customer expects from a car rental service provider. For further insight, we circulated forms among different classes of customers the company had as well as on the internet.

Secondly, to gain insight into what exactly the company exactly needs for better management and effective handling of data, a couple of blogs and books were referred to list out, what various options the company has going forward and which options can the company afford.

To build a proper design to meet company's requirements, we studied in depth about existing software namely Rent-a-car Manager, Rent Centric and Rent Syst available for managing car rental services. The first two software provides a solution which implements a centralized server maintaining database. While the third one implements cloud based database managing software.

The first Rent-a-car Manager provides several facilities such as fleet management, activity planners, cost management, client management, statistical reports, and web integration. We aim to implement fleet management, cost management, client management and web integration for easy to use and customer satisfaction. We aim to make the User Interface adaptive to different environments with its responsive design.¹

The second Rent Centric provides a better fleet management system which extends to the vehicle tracking, maintenance and history record collecting,

¹ <http://rentacar-manager.com/en/>

financial management and fleet reporting. We intend to add GPS ID to each car and a customer as well as the staff can track the vehicle. Fleet management feature provides a complete overview of the state of the fleet - current location, physical condition, engagement, costs and income of each vehicle individually. This is overall a better software for fleet management compared to Rent-a-car Manager software.²

The last one Rent Syst is a software to automate business processes with full control of car rental and convenient order processing. Suitable for both large fleets and owners of one or more cars. With its help, order management is greatly simplified and also increases the revenue and profitability of each car. Our software allows you to take a different look at managing the auto business. With its help you can track individual cars and keep track of the available cars and help the customers keep track of their past rentals as well for a better experience.³

Sources:

1. <https://fleetroot.com/blog/car-rental-software-the-best-guide-2020/>
2. Dean Leffingwell and Don Widring, "Managing Software Requirements: A Use Case"
3. <https://www.rentcentric.com/products/car-rental-software>
4. <https://rentsyst.com/>
5. <http://rentacar-manager.com/en/>
6. https://www.researchgate.net/publication/325253983_Online_Car_Rental_System_using_Web-Based_and_SMS_Technology

² <https://rentsyst.com/>

³ <https://www.rentcentric.com/products/car-rental-software>

B. Interviews

Mainly three interviews were held for developing a proper design of system management, one with the owners, one with the branch manager, and one with the employees of various branches. The interview plan and summary of all interviews are attached here.

Griefingdor: (Role Play) Interview Plan For CEO

System: Ghost car rental service

Project Reference: SF/SJ/2020/9

Interviewee:

1) Mr.Reeves (Role Play) **Designation:** CEO

Contact Details: gcrs_owner@gmail.com

Organization Details: Ghost car rental service

Interviewer:

1) Rishi Shiwani **Designation:** Business Development Executive

2) Bhrugu Dave **Designation:** Developer

Date: 29/09/2020

Time: 16:30

Duration: 1 hour

Place: WebEx Meeting

Purpose of Interview:

Preliminary meeting to identify problems, bottlenecks, and requirements related to the current system of operation at Ghost car rental service.

Agenda:

Gaining ideas about the budget of the company.

Other business and implementation constraints.

Working of different outlets and data synchronization.

Features expected in the database.

Documents to be brought to the interview:

Rough plan of the UI and database

Any documents relating to its usage

Griefingdor: (Role Play) Interview Summary For CEO

System: Ghost car rental service

Project Reference: SF/SJ/2020/9

Interviewee:

- 1) Mr.Reeves (Role Play) **Designation:** CEO

Contact Details: gcrs_owner@gmail.com

Organization Details: Ghost car rental service

Interviewer:

- 1) Rishi Shiwani **Designation:** Business Development Executive
2) Bhrugu Dave **Designation:** Developer

Date: 29/09/2020

Time: 16:30

Duration: 1 hour

Place: WebEx Meeting

Results of Interview:

- The company is willing to allot a fair budget for effective implementation.
- Data synchronization among different outlets is absolutely necessary.
- There are no technical or business constraints expected to occur.
- Simple UI must be implemented for customers.
- Staff also shouldn't find it difficult to use.
- Different privileges allocations power should be provided.
- Data security is a major concern.

Griefingdor: (Role Play) Interview Plan For Branch Managers

System: Ghost car rental service

Project Reference: SF/SJ/2020/9

Interviewee:

- 1) Hemant Chanda(Role Play)
- 2) Manish Kumar

Designation: Branch Manager
Designation: Branch Manager

Organization Details: Ghost car rental service

Interviewer:

- 1) Pavan Kotak
- 2) Aditya Shah

Designation: Business Development Executive
Designation: Developer

Date: 29/09/2020

Time: 18:30

Duration: 2 hours

Place: WebEx Meeting

Purpose of Interview:

To get insight into the inner working of an individual branch and how they collaborate with other branches for data synchronization.

Agenda:

Different positions offered in every branch.

Different user classes, expected in the database.

Data synchronization.

Staff's technical skills.

Documents to be brought to the interview:

Rough plan of database and different functionalities of the database.

Any documents relating to its usage.

Griefingdor: (Role Play) Interview Summary For Branch Managers

System: Ghost car rental service

Project Reference: SF/SJ/2020/9

Interviewee:

- 1) Hemant Chanda(Role Play)
- 2) Manish Kumar

Designation: Branch Manager
Designation: Branch Manager

Organization Details: Ghost car rental service

Interviewer:

- 1) Pavan Kotak
- 2) Aditya Shah

Designation: Business Development Executive
Designation: Developer

Date: 29/09/2020

Time: 18:30

Duration: 2 hours

Place: WebEx Meeting

Results of Interview:

- At the moment, all reservations are made through calls and on-spot bookings.
- In every branch, 3 types of position have access to the records namely, clerk, accountant, and manager, and the rest of the staff is for technical and physical support
- The clerk is responsible for making proper entries into the record and the accountant handles all the billings and expenditure of the branch.
- The manager overlooks all the records and billings and in case of any data entry mistake, the manager is responsible for data updation.
- Most of the staff, who will access the database have a fair bit of software skills and are willing to adapt to the new norm.
- Thus, the staff should have relatively easy access to the database.

Griefingdor: (Role Play) Interview Plan For Accountants

System: Ghost car rental service

Project Reference: SF/SJ/2020/9

Interviewee:

- 1) Nidhi Kesaria(Role Play)
- 2) Priyanka Sameja

Designation: Accountant
Designation: Accountant

Organization Details: Ghost car rental service

Interviewer:

- 1) Pavan Kotak
- 2) Bhrugu Dave

Designation: Business Development Executive
Designation: Developer

Date: 23/09/2020

Time: 18:30

Duration: 1.5 hours

Place: WebEx Meeting

Purpose of Interview:

To get insight into the accounting aspects of the company and to know the requirements from the accounts side.

Agenda:

The different functions that the accountants use and what they require to perform their job.

The technical expertise of the accounting department.

The frequency of accessing the data.

Documents to be brought to the interview:

Rough plan of database and different functionalities of the database.

Any documents relating to its usage.

Griefingdor: (Role Play) Interview Summary For Accountants

System: Ghost car rental service

Project Reference: SF/SJ/2020/9

Interviewee:

- 1) Nidhi Kesaria(Role Play)
- 2) Priyanka Sameja

Designation: Accountant
Designation: Accountant

Organization Details: Ghost car rental service

Interviewer:

- 1) Pavan Kotak
- 2) Bhrugu Dave

Designation: Business Development Executive
Designation: Developer

Date: 23/09/2020

Time: 18:30

Duration: 1.5 hours

Place: WebEx Meeting

Results of Interview:

- At the moment all the relevant data is sent to them at the end of the month.
- The staff has a basic idea of using this kind of software.
- Accountants are asked for monthly reports and occasionally asked financial reports.
- Accountants must have access to the salaries of all staff members of their own branch.
- If possible, instead of writing functions, they would like a UI for the same.

Griefingdor: (Role Play) Interview Plan For Clerks

System: Ghost car rental service

Project Reference: SF/SJ/2020/9

Interviewee:

- | | |
|------------------------------|---------------------------|
| 1) Prashant Verma(Role Play) | Designation: Clerk |
| 2) Harsha Goyal | Designation: Clerk |
| 3) Sanjay Dutta | Designation: Clerk |

Organization Details: Ghost car rental service

Interviewer:

- | | |
|----------------|--|
| 1) Pavan Kotak | Designation: Business Development Executive |
| 2) Bhruqu Dave | Designation: Developer |

Date: 23/09/2020

Time: 18:30

Duration: 2 hours

Place: WebEx Meeting

Purpose of Interview:

To get insights into how records are handled by the clerks and what problems are faced in synchronization across branches.

Agenda:

Different functionality expect in the new system

Software skills and technical abilities of the staff

Training sessions

Loopholes of the current system and expected solutions.

Documents to be brought to the interview:

Rough plan of database and different functionalities of the database.

Any documents relating to its usage.

Griefingdor: (Role Play) Interview Summary For Clerks

System: Ghost car rental service

Project Reference: SF/SJ/2020/9

Interviewee:

- 1) Prashant Verma(Role Play)
- 2) Harsha Goyal
- 3) Sanjay Dutta

Designation: Clerk
Designation: Clerk
Designation: Clerk

Organization Details: Ghost car rental service

Interviewer:

- 1) Pavan Kotak
- 2) Bhrugu Dave

Designation: Business Development Executive
Designation: Developer

Date: 23/09/2020

Time: 18:30

Duration: 2 hours

Place: WebEx Meeting

Results of Interview:

- The different functions that clerks need to make entries into the database.
- They also need access to the list of cars and drivers available at any particular moment.
- Aside, from this, the current system works mostly on excel sheets and sometimes becomes inefficient for entering records.
- Thus, a compact data entry system user interface is expected.
- The staff has sufficient knowledge about the software used for implementation. Thus, no extra training will be required.
- The most known problem was entering data at the same moment in the same branch resulted in data inconsistency.
- Thus, proper handling of race conditions is required.

These are the plans and summaries of each and every interview conducted. The main concerns of the company before implementing a new system is the data synchronization and better user interface at each and every level of usage of the database.

C. Questionnaires

Griefingdor – Car Rental Survey for Ghost Car Rental Services

Please circle your answers to the following questions:

1. How often do you rent a car to meet your needs?

Never / once a year / two or three times a year / every month / no regular pattern

2. Factors Car Rentals Customers use to select a Car rental agency?

Price / Reach /Availability of Cars/ Quality of Services / Fast Check out of the car

3. What is the most important factor when you book your own personal car hire?

Price/ Number of seats/ Discounts/ Driver/ Comfort

4. How often do you need to use your car for work during the day?

No regular pattern/Every week/ 2-3 times a month/ once a month

5. Will you like a User Interface which would help you to book efficiently?

Yes / No

6. If you face problems, would you like to report it?

Yes / No

7. If Yes, How would you like to report it?

Message/ Call/ Website

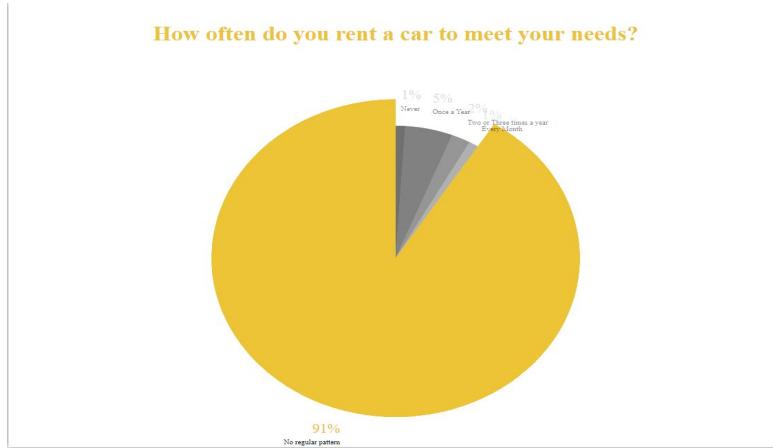
Your Name

Thank you for completing this questionnaire

D. Summary of Responses

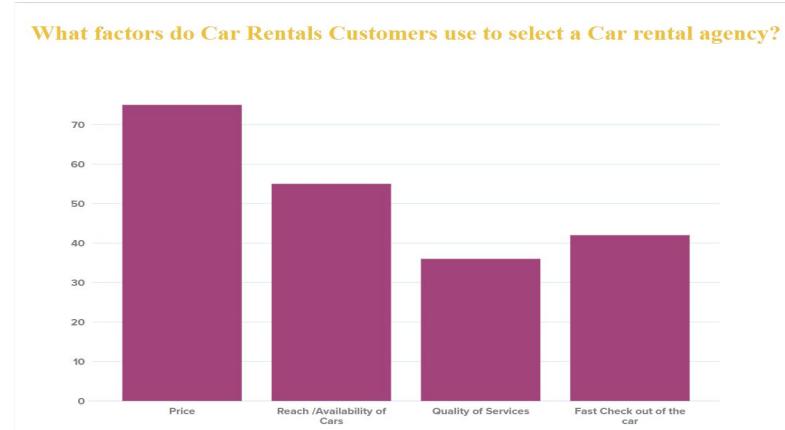
1. How often do you rent a car to meet your needs?

R: 91 percent of the audience responded with no regular pattern since most of them feel that there can be a need for a rented car at any time. Customers want to book a car any time they want so they want a 24/7 availability option for their immediate and urgent needs.



2. What factors do Car Rentals Customers use to select a Car rental agency?

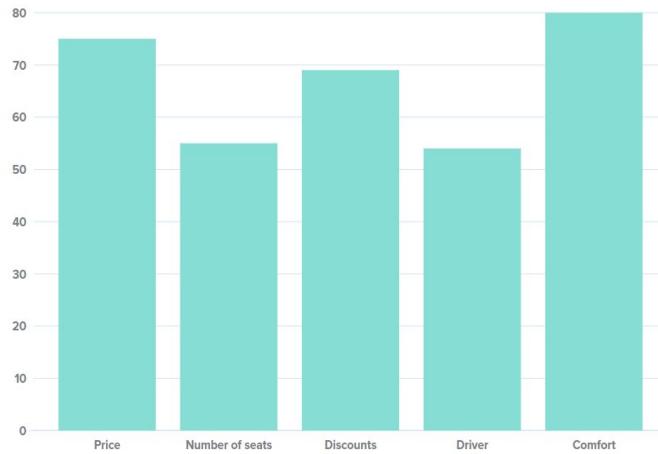
R: 75 percent of the respondents opted for the price since they wanted a fair price for their rented car and opted for discounted cars while 60 percent of respondents wanted fast check out of the car to minimize their time usage in the whole process. 55 percent of respondents preferred the availability of cars as a major factor for customer satisfaction.



3. What is the most important factor when you book your own personal car hire?

R: 80 percent of the respondents preferred comfort as they feel a car is worth renting when it is comfortable. 75 percent of respondents feel the price is the other dominating factor in this business.

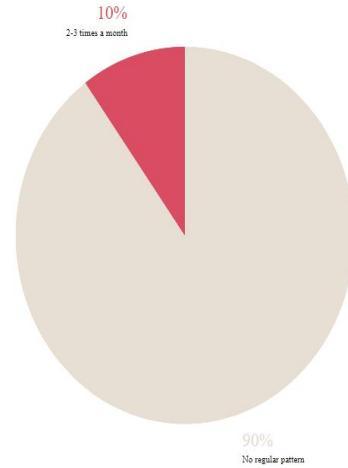
What are the important factor when you book your own personal car hire?



4. How often do you need to use your car for work during the day?

R: 90 percent of the respondents feel there is no regular pattern in how often they use their car for work. 10 percent use it 2-3 times a month for work.

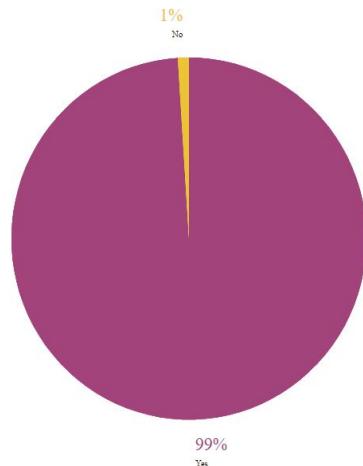
How often do you need to use your car for work during the day?



5. Will you like a User Interface which would help you to book efficiently?

R: 99 percent of respondents opted for yes as its time-efficient to book from a user interface.

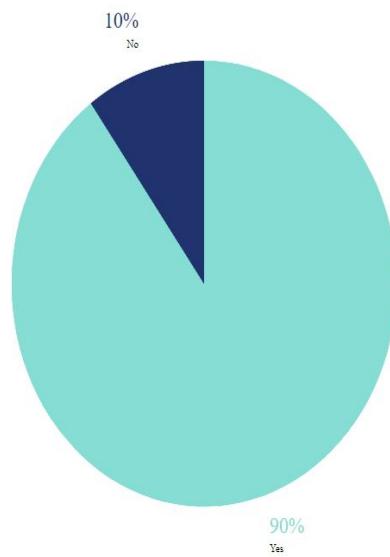
Will you like a User Interface which would help you to book efficiently?



6. If you face problems, would you like to report it?

R: 90 percent of respondents wanted to report the problems faced. They wanted to report the issues and wanted them to be resolved.

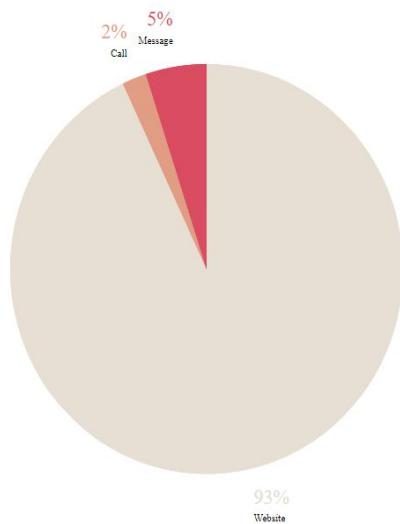
If you face problems, would you like to report it?



7. If Yes, How would you like to report it?

R: 93 percent of the respondents wanted to report the problem on a website as it is the most convenient way.

If you face problems, would you like to report it?



Thus, we can clearly see that the customers have a clear opinion about having a compact UI for acquiring services is the utmost requirement. A platform for providing feedback should also exist.

Griefingdor: Observations

System: Ghost car rental service

Project Reference: SF/SJ/2020/9

Observers:

1) Pavan Kotak

Designation: Business Development Executive

Date: 23/09/2020

Time: 18:30

Duration: 45 mins

Place: WebEx Meeting

Observations:

1. Customers prefer price over other facilities as they feel the price is more important for customer satisfaction. Price is the most dominating factor in the car rental business.
2. The second most important factor is the comfort as Customers crave for a comfortable rental car. Comfort is the biggest demand after a nominal price.
3. Customers seek the most comfortable vehicle according to their budget.
4. Customers prefer vehicles that have been maintained and cleaned regularly.
5. For an efficient booking service, customers prefer to have a user interface to book a car of their choice.
6. Customers like to go over several choices before making a decision.
7. Customers are eager to provide feedback on their rented vehicles/driver so it's a good marketing strategy to satisfy their demands and ask for their feedback to improve our services.
8. Lot of problems are caused and require much man work for synchronizing data across various outlets.

Thus, we observed that data integration is absolutely essential because current software is unable to provide an efficient solution.

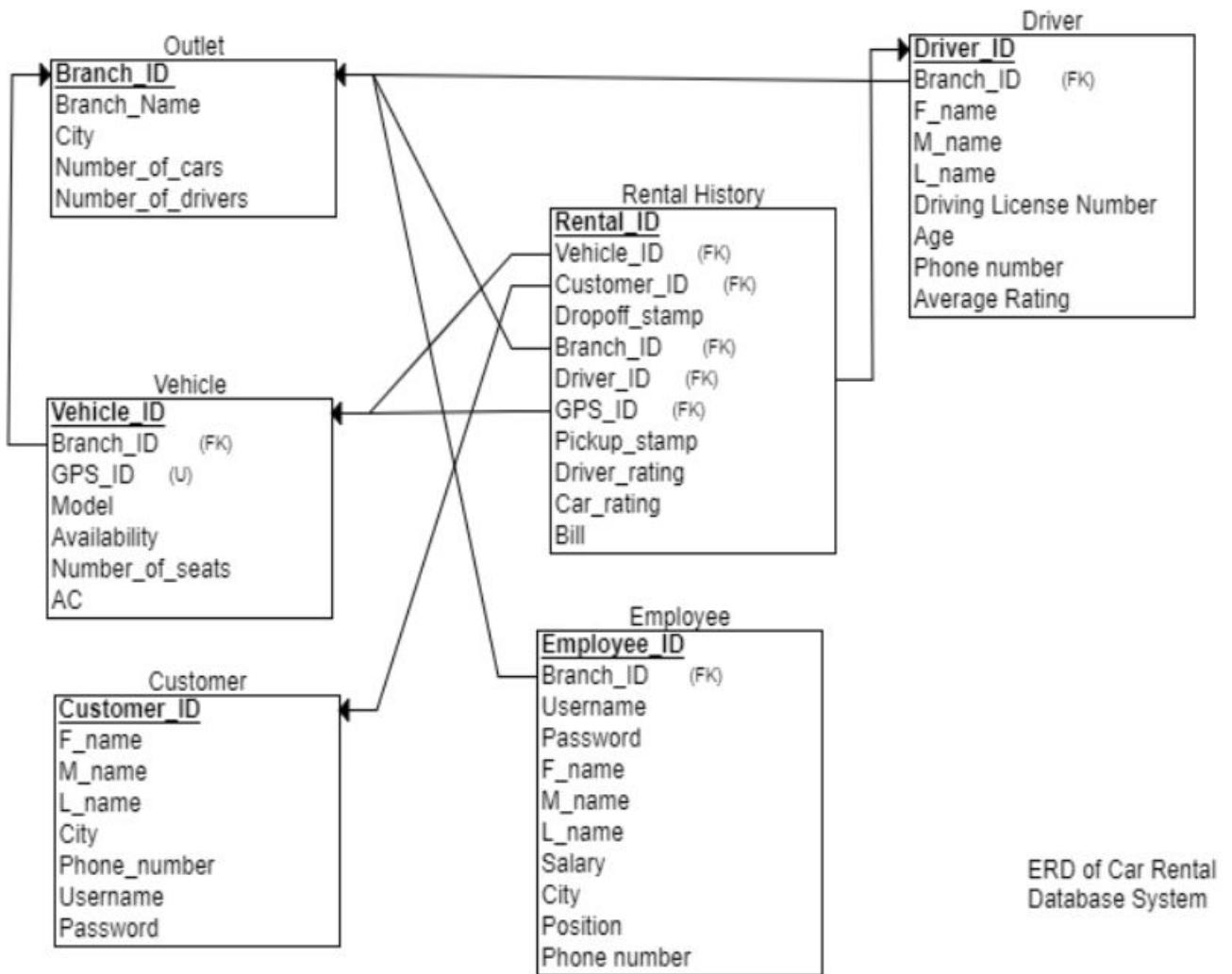
Fact-Finding Chart

Objective	Technique	Subject	Time Commitment
To get the background on the company	Background Reading	Company reports, Blogs, Wikipedia	1 day
To establish the company objective	Interview	CEO	1 hour
To get an insight into each branch and management	Interview	2 Branch Manager	2 hour 1 hour each
To find out the core business operations and what records and resources are kept	Interview	3 Clerk	2 hours
To find out the current accounting system works and what to establish the accounting for the new model	Interview	2 Accountant	1:30 hours 45 mins each
To follow up development of business understanding and features	Observations	-----	45 mins

Expected Deliverables

- The most important deliverable expected from the new system is data synchronization among all the branches of the company.
- The second important thing is limited access to the database. The owner can give access to the manager as required.
- Handling of data entries made at the same time i.e. race condition should be taken care of.
- To provide a better feedback system as well as display the same feedbacks efficiently on request.
- The owner although gave no constraints on the budget but expects a relatively affordable new management system for their business.
- Also, abstraction of implementation is also absolutely necessary.

Temporary Database Design



- This is a rough entity relation design of the database required. Here, there are mainly 6 tables and the underlined attribute denotes the primary key of the table, and symbol (FK) determines the foreign key of the table while (U) determines that the given attribute must be unique.

User Classes and Characteristics

The main people who have been identified as regular system users as follow

1. CEO
2. Branch Manager
3. Accountant
4. Clerk
5. Customers

The users can be classified according to their frequency of use, a subset of product functions used technical expertise, security or privilege levels, educational level, or experience they have to the Car Rental System (CRS).

- CEO is supposed to have the following characteristics
 - CEO will have access to all the features and all the functions
- Branch managers are supposed to have the following characteristics
 - On the whole, the Branch manager and administrator spend comparatively lesser time than others on the system. Usually, they would access the system when they are required to make reports or something else.
 - They should have high technical expertise. They have to be fluent in the system usage and should be well aware of all the system functionalities.
 - Branch managers have access to all security levels as they have to access and maintain the system occasionally.
 - The branch manager and administrator should have experience in handling a system of this domain. He must be able to identify errors caused due to improper data entries.
- Clerk and Accountant are supposed to have the following characteristics
 - They spend the maximum amount of time on the system in the form of entering information, or updating the information.
 - They are not required to have technical expertise. The most they would need is to be computer literate to understand the working of the system and use it efficiently.
 - They have very low-security access. They cannot access quite a few functionalities of the system except the ones they really have to use.
 - They should have good communication skills and even if they have no experience in system usage, it's not a big problem.
 - They should have experience in this area with a minimum diploma in IT.

- Customers are supposed to have the following characteristics
 - They have a minimal amount of access to the database.
 - They can just request their own profile.
 - They can also see how many cars and drivers are available on a particular franchise.
 - They can request a car and/or driver from the UI provided.
 - They are also provided with a GPS ID to track down the car at the time of booking services.
 - They can also see details of all the available drivers in order to facilitate overall better rental services.

Operating Environment

- Reliability:
 - The system shall be available for 90% of 24 hrs.
 - The system shall not be subject to failure for 20 hrs.
 - The system shall be repaired in at least one hour.
 - The system shall accurately deal with decimal numbers.
- Performance:
 - The system response for operations shall be at most 5 seconds.
 - The system shall handle five users per department.
- Supportability:
 - The system shall accommodate changes and enhancements that the development team is going to do in at most 5 days.
 - Also, easy management of authorization access is required.
 - A good, compact and simple user interface is required at each and every level of data access to the database.
- Design Constraints:
 - The system shall be designed according to IEEE Standards.
 - The system shall replace the existing system.
 - The system shall be programmed using JAVA.
- Connectivity Requirements:
 - For data synchronization, we have two options either to integrate our database into the cloud database system or we need to connect all database servers of each and every individual branch through physical links.
 - A centralized database system is required at both level inter and intra level for synchronizing data.
- Hardware Requirements:

- The software should be run on any sort of desktop or laptop environment, regardless of the operating system.
 - The software also has the potential of running on tablets, but with a more simplified version.
 - Essential input/output devices are keyboards, mouse, and printers; nothing else is required but can be recommended if desired.
-
- Software Requirements:
 - To store all the records efficiently in our servers, we need at least 100 GB of server memory at each and every outlet.
 - Also, time to time backup of data is mandatory for extra safety.
 - RAM up to 16 GB for fast processing.
 - Server and domain host by Amazon

Product Function and User Privileges

1. Get Customer Information.
 - This function will fetch the detailed customer profile from the servers
 - Accessed by: Customers
2. Choose a package with vehicle model
 - This function will filter the data list according to the input parameters and display it to the user.
 - Accessed by: Customers
3. View Rental history
 - This function will display all the vehicle bookings made within the given time frame.
 - Accessed by: All user classes except customers
4. View car or driver average rating
 - This function returns the current average rating of a car or driver.
 - Accessed by: All user classes.
5. Create a final invoice
 - This function will generate the final invoice.
 - Accessed by: Accountant, Clerk
6. Collect payment
 - This function will calculate and collect the payable amount from the user
 - Accessed by: Clerk, Accountant, Customer
7. Get inventory report
 - This function will be accessible by the employee to check the number of vehicles and drivers available on the franchise.
 - Accessed by: All user classes except customer
8. Add vehicles
 - This function will allow employees to add the new vehicles and drivers into the system
 - Accessed by: Branch manager, Clerk

9. Get vehicle information

- This function will display the information of the rented car to the user
- Accessed by: All user classes.

10. Generate backup

- This function will create the backup of the current database.
- Accessed by: Branch manager

11. Get Tracking ID

- This function will fetch GPS_ID associated with the booked car which allows one to track down the car.
- Accessed by: All user classes except accountant.

12. Displaying Driver Details

- This function will fetch details of the driver using the entered driver ID.
- Accessed by: All user classes except accountant.

13. Displaying reviews

- This function will display all car/driver reviews of previous trips.
- Accessed by: All user classes except accountant.

Assumptions

- Each customer can book only one vehicle and/or driver.
- The rental price of cars is fixed and the same at every branch.
- Each customer who has rented the car has returned the car.
- The rate of the vehicle/driver will be per hour of usage.
- Each customer has returned the vehicle with no damage.
- Any damages on the trip are not accounted for by the company.
- Any car is tracked by just entering the GPS id.
- The salary of the driver doesn't depend on the number of fares completed.

Business Constraints

- The company hasn't specified any money related or space related constraint. Although, the only constraint mentioned was a Proper User Interface is absolutely necessary.
- For centralizing the database, the physical connection was checked out and cloud integration was recommended with minimal cost to the company.

Nouns and Verbs

Noun	Verb
Transportation	Digitize
Everyone	Change
Client	Visit
License Plate	Contains
Cars	Purchase
Chassis Number	Require
Records	Rate
Car List	Updated
Customer Service	Picking
Platform	Understand
Driver	Overwhelm
Clerical Staff	Provide
Worker	Hire
Salary Record	Acquire
Branch Manager	Provide
Accountant	Limited
Individual Branch	Eliminates
Proof ID	Handing
Employee ID	Coordinate
Unique ID	Pay
Details Form	Fill

Trip Record	Grasp
Maintenance Record	Synchronized
Customer	uses
Call Attendee	Assigns
City	Combine
Privileges	Integration
Rating	According
Review	Relate
Profit Record	Rent
Pickup Destination	Migrate
Dropoff Destination	Speculate
Time Period	Store
Money	Except
Modern World	Manage
Ideology	Delete
Start-up	Update
Local rental industry	Filter
Geography	Depends
Customer ID	Access
Excel Spreadsheet	Maintain
Task	Forget
Member	Discuss
Month	Fill
Problem	Implement

User Friendly UI	Handle
Franchise	Obtain
Owners	Check
Name	Determine
Data	Fulfil
Phone Call	Alot
Outlet	Calculate
Rate	Show
Rate Table	Books
Login Details	Works
	Parked
	Log in

Accepted Nouns and Verbs List

Candidate Entity Set	Candidate Attribute Set	Candidate Relationship Set
Customer Record	License plate	Maintaining
Car Record	Chassis number	Rate
Employee Record	Job position	Books
Maintenance Record	Proof ID	Parked
Rental Record	Employee ID	Works
Branch Record	Customer ID	Log in
Rate Table	Rating	
Login Details	Dropoff point	
	Pickup point	
	Time period	
	Salary	
	Rate	
	Phone number	
	City	
	Review	
	Branch ID	
	Car Type	
	Password	
	Access Type	

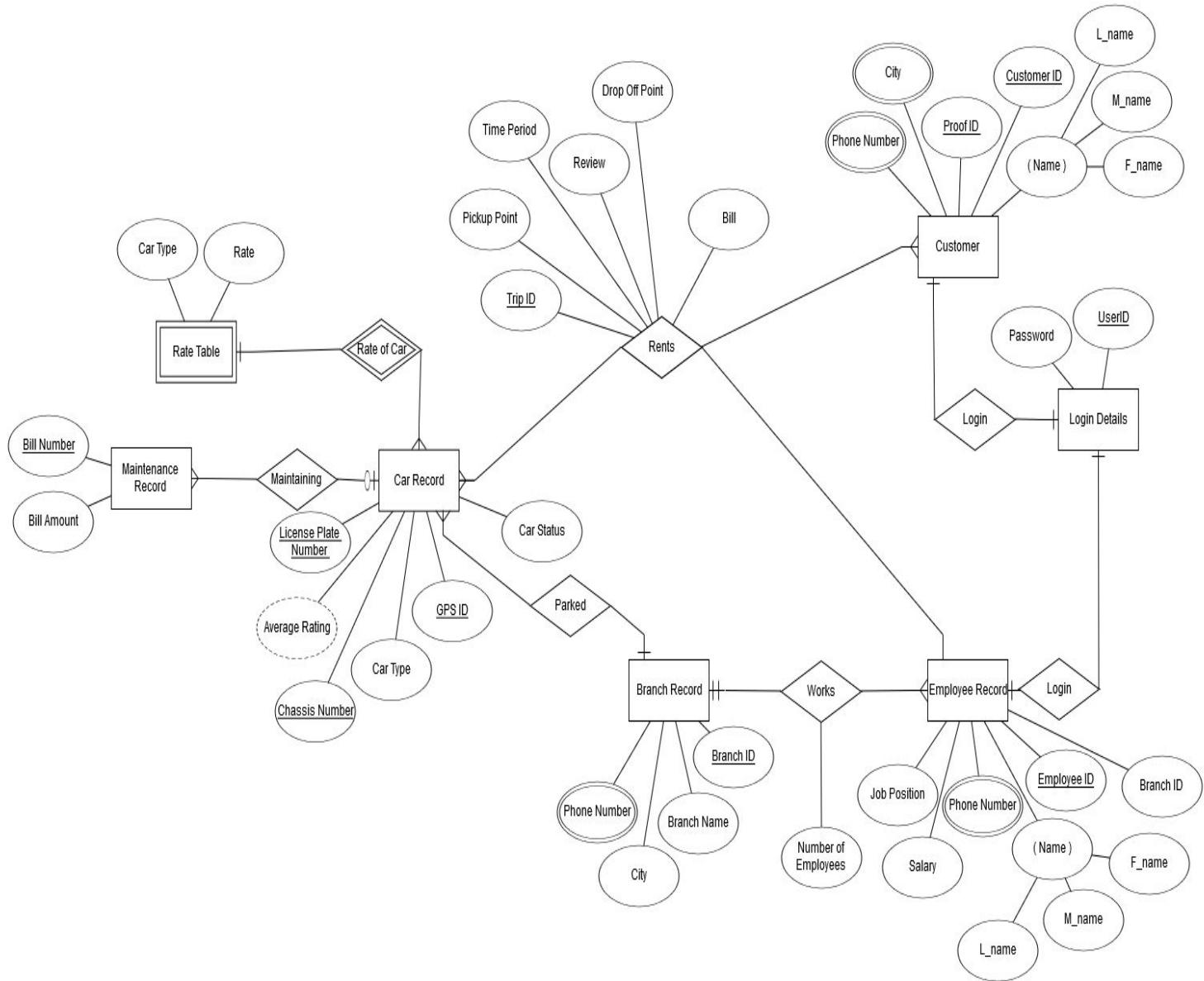
Rejected Nouns

Noun	Rejected Reason
Everyone	Vague
Transportation	Irrelevant
Customer Service	Association
Platform	General
Modern World	Vague
Call Attendee	Irrelevant
Details Form	Irrelevant
Ideology	Vague
Startup	Vague
Local Rental Industry	Vague
Geography	Vague
Excel Spreadsheet	Irrelevant
Task	Duplicate
Member	Duplicate
Month	Irrelevant
Solution	General
Problem	General
User Friendly UI	Irrelevant
Franchise	Duplicate
Owners	Duplicate
Name	Duplicate

Data	General
Client	Duplicate
Phone Call	Irrelevant
Outlet	Duplicate
Worker	Duplicate
Individual Branch	Duplicate
Unique ID	Duplicate
Money	Duplicate
Branch Manager	Attribute
Car List	Duplicate
Customer Service	Attribute
Platform	Generic
Driver	Attribute
Clerical Staff	Attribute
Worker	Duplicate
Branch Manager	Attribute
Accountant	Attribute
Individual Branch	Duplicate
Details Form	Irrelevant
Trip	Attribute
Customer	Attribute
City	Attribute
Privileges	Attribute
Rating	Attribute

Review	Attribute
Pickup Destination	Attribute
Dropoff Destination	Attribute
Time Period	Attribute
Money	Attribute
Rate	Attribute

Final ER Diagram



Functional Dependencies

1. Branch Record
 - o (Branch ID) -> (BranchName, City) (Primary Key Dependencies)
2. Branch Record_PhoneNumber
 - o (Branch ID, Phone Number) -> (Branch ID, Phone Number)
3. Customer_city
 - o (City, Customer ID) -> (City, Customer ID)
4. Employee Record
 - o Job position -> Salary
 - o Employee ID -> (Branch ID, User ID, F_name, M_name, L_name, Job Position, Salary)
 - o User ID -> (Employee ID, Branch ID, F_name, M_name, L_name, Job Position, Salary)
5. Employee Record_PhoneNumber
 - o (Phone Number, Employee ID) -> (Phone Number, Employee ID)
6. Login Details
 - o (User Id) -> (Password)
7. Customer
 - o (Customer ID) -> (User ID, Proof ID, F_name, M_name, L_name)
 - o (User ID) -> (Customer ID, Proof ID, F_name, M_name, L_name)
8. Customer_PhoneNumber
 - o (Phone Number, Customer ID) -> (Phone Number, Customer ID)
9. Car Record
 - o (Chassis Number) -> (Chassis Number, Car Status, Car Type, Branch ID, License Plate Number, GPS ID, Average Rating)
 - o (GPS ID) -> (Chassis Number, Car Status, Car Type, Branch ID, License Plate Number, GPS ID, Average Rating)
 - o (License Plate Number) -> (Chassis Number, Car Status, Car Type, Branch ID, License Plate Number, GPS ID, Average Rating)
10. Rents
 - o Trip Id -> (Employee ID, Customer ID, Chassis Number, Time Period, Drop Off Point, Pick Up Point, Bill, Review)
11. Maintenance Record
 - o (Bill Number) -> (Bill Amount, Chassis Number)
12. Rate Table
 - o (Car Type) -> (Rate)

1NF(First Normal Form)

1. All tables are in 1NF form. Because all multivalued attributes are already decomposed into other tables like employerecord_phonenumber, etc.

2NF(Second Normal Form)

1. There is no partial dependency.

3NF(Third Normal Form)

In the employee table, a job position which is a non prime attribute determines another non prime attribute salary. Thus a transitive dependency is formed. Thus, another table is formed consisting of job position and salary.

Final Schema

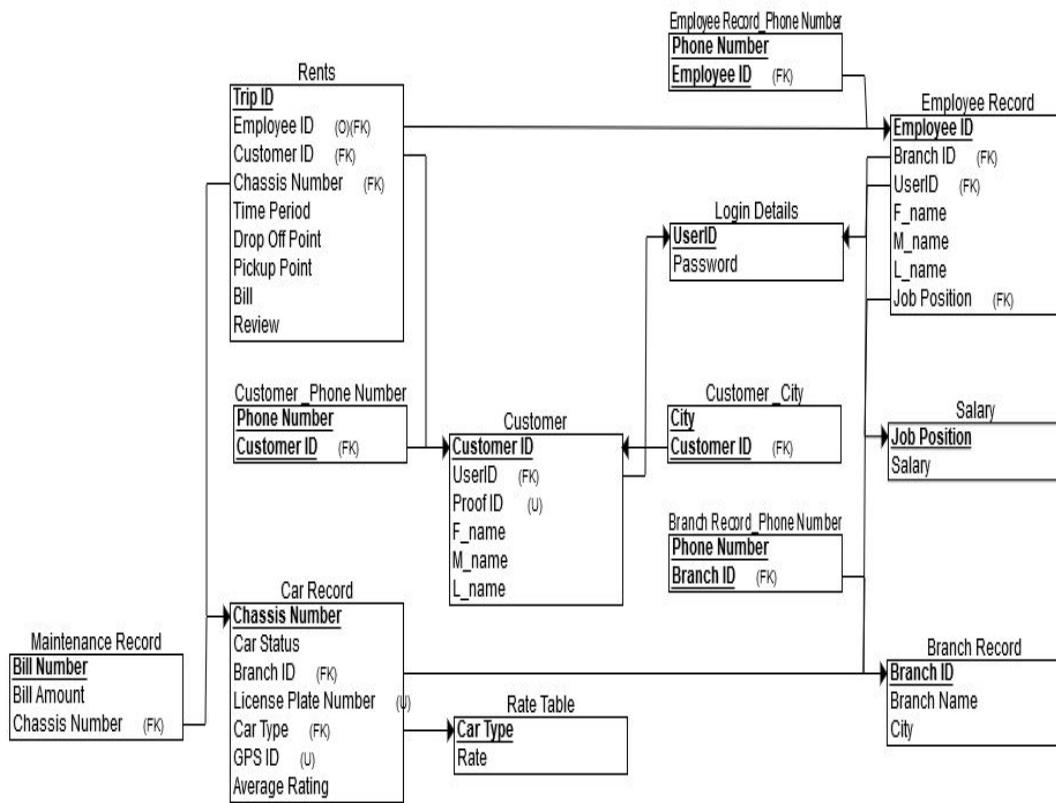
- Rate Table(Rate, Car Type)
- Maintenance Record(Bill Number, Bill Amount, Chassis Number)
- Rents(Trip ID, Employee ID, Customer ID, Chassis Number, Time Period, Drop Off Point, Pick Up Point, Bill, Review)
- Customer_ Phone Number(Phone Number, Customer ID)
- Car Record(Chassis Number, Car Status, Car Type, Branch ID, License Plate Number, GPS ID, Average Rating)
- Customer(Customer ID, Proof ID, F_name, M_name, L_name)
- CustomerID_UserID(Customer ID, User ID)
- Employee Record_Phone Number(Phone Number, Employee ID)
- Login Details(User ID, Password)
- Employee Record(Employee ID, Branch ID, User ID, F_name, M_name, L_name, Job Position)
- EmployeeID_Salary(Employee ID, User ID)
- Customer_City(City, Customer ID)
- Branch Record_Phone Number(Phone Number, Branch ID)
- Branch Record(Branch ID, Branch Name, City)

CSV Files

For the data population, we have to import data using csv files. Here is the link for the same.

<https://drive.google.com/file/d/1Vux8Msw3g2yjRtfSE0GQ1GtDs6iFF2LS/view?usp=sharing>

Final Relational Schema Diagram



DDL Scripts

Rate Table

CREATE TABLE Rate_Table

(

Car_Type VARCHAR(10) NOT NULL,
Rate INT NOT NULL,
PRIMARY KEY (Car_Type)

);

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under the 'Tables' section, 'rate_table' is selected. The main pane displays a query editor with the following SQL code:

```
1 SELECT * FROM gcrs.rate_table
2 ORDER BY car_type ASC
```

Below the query editor is a data grid titled 'Data Output' showing the following data:

car_type	rate
HATCHBACK	13
MINIVAN	7
SEDAN	15
SUV	9

A green message bar at the bottom right indicates: 'Successfully run. Total query runtime: 295 msec. 4 rows affected.'

Number of records = 4

Branch Record

CREATE TABLE Branch_Record

```
(  
    Branch_Name VARCHAR(20) NOT NULL,  
    Branch_ID INT NOT NULL,  
    City VARCHAR(20) NOT NULL,  
    PRIMARY KEY (Branch_ID)  
);
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects, with 'Tables (12)' expanded and 'branch_record' selected. In the main pane, the 'Query Editor' tab is active, showing the following SQL code:

```
1 SELECT * FROM gcrs.branch_record  
2 ORDER BY branch_id ASC
```

The 'Data Output' tab is selected, displaying the results of the query:

	branch_name	branch_id	city
1	Kataria	1	Ahmedabad
2	Ghost	2	Vadodara
3	Famas	3	Mumbai
4	Negav	4	Goa
5	Mag	5	Rajkot
6	AWP	6	Hyderabad
7	USPS	7	Surat
8	Scout	8	Delhi
9	Kreig	9	Kanpur
10	Tec	10	Jaipur

Number of records = 10

Customer

CREATE TABLE Customer

```
(  
    Proof_ID VARCHAR(20) NOT NULL,  
    Customer_ID INT NOT NULL,  
    F_name VARCHAR(15) NOT NULL,  
    M_name VARCHAR(15) NOT NULL,  
    L_name VARCHAR(15) NOT NULL,  
    Pass_word VARCHAR(20) NOT NULL,  
    PRIMARY KEY (Customer_ID),  
    UNIQUE (Proof_ID)  
);
```

The screenshot shows the pgAdmin 4 interface with the 'Customer' table selected in the left sidebar. The main area displays the table's data with 14 rows. A message at the bottom right indicates the query was successfully run.

customer_id	f_name	m_name	l_name	pass_word
1001	Aditya	Arpit	Shah	Ydjjuk
1002	Rishi	Raj	Shivani	FZBzmH2DE
1003	Rushi	Ruchit	Patel	AYomneW
1004	Pavan	Karan	Kotak	SQVnXMYEO
1005	Sanket	Anil	Jain	8TZRDOC
1006	Nishit	Vasant	Jagetia	QmRgP1eP3
1007	Vedant	Dipen	Thakkar	3vV7qagY
1008	Sarthak	Tarak	Patel	LSLhteo
1009	Rahul	Narain	Khatri	ZtxQbgDjfW9
1010	Bhagyesh	Ashok	Ganatra	Dw3KVe
1011	Neel	Kunj	Makadia	moRSgb
1012	Dhruvil	Atmaram	Bhatt	PB3BgX
1013	Hemant	Tukaram		
1014	Hitesh	Kavan		

Number of records = 80

Branch Record Phone Number

```
CREATE TABLE Branch_Record_Phone_Number
(
    Branch_ID INT NOT NULL,
    Phone_Number NUMERIC NOT NULL,
    PRIMARY KEY (Branch_ID, Phone_Number),
    FOREIGN KEY (Branch_ID) REFERENCES Branch_Record(Branch_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects, including FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions (1), Materialized Views, Sequences, and Tables (12). The 'branch_record_phone_number' table is selected. The main pane shows a 'Query Editor' with the following SQL code:

```
1 SELECT * FROM gcrs.branch_record_phone_number
2 ORDER BY branch_id ASC, phone_number ASC
```

The results are displayed in a table titled 'Data Output' with columns 'branch_id' and 'phone_number'. The data consists of 20 rows:

branch_id	phone_number
1	6796497786
2	8968439382
3	6389758166
4	9485317979
5	3995389375
6	6267918172
7	5617933228
8	9375293016
9	7239957584
10	9461577761
11	5118103399
12	7637886912
13	1783692442
14	8196978711

A green message bar at the bottom right indicates: 'Successfully run. Total query runtime: 218 msec. 20 rows affected.'

Number of records = 20

Customer Phone Number

```
CREATE TABLE Customer_Phone_Number
(
    Phone_Number NUMERIC NOT NULL,
    Customer_ID INT NOT NULL,
    PRIMARY KEY (Phone_Number, Customer_ID),
    FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

The screenshot shows the pgAdmin 4 interface. On the left is a tree view of database objects under 'Browser'. In the center is the 'Query Editor' window with the following content:

```
gcrs.customer_phone_number/carserv@PostgreSQL_10
Query History
1 SELECT * FROM gcrs.customer_phone_number
2 ORDER BY phone_number ASC, customer_id ASC
```

Below the query history is a table titled 'Data Output' showing 14 rows of data from the 'customer_phone_number' table. The columns are 'phone_number' and 'customer_id'. The data is as follows:

	phone_number	customer_id
1	1084848354	1043
2	1126413960	1058
3	1159264950	1046
4	1266957286	1071
5	1336732663	1078
6	1636280096	1011
7	1658732702	1015
8	1749466815	1056
9	1757126993	1032
10	1779135382	1036
11	1942380289	1001
12	1971680567	1007
13	2042970388	1069
14	2067248474	1059

A green message bar at the bottom right of the query editor says 'Successfully run. Total query runtime: 323 msec. 100 rows affected.'

Number of records = 100

Customer City

```
CREATE TABLE Customer_City
(
    City VARCHAR(20) NOT NULL,
    Customer_ID INT NOT NULL,
    PRIMARY KEY (City, Customer_ID),
    FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

The screenshot shows the pgAdmin 4 interface. On the left is the 'Browser' pane, which lists various database objects like FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, and Tables (12). The 'customer_city' table is selected. The main area is the 'Query Editor' showing the following SQL query:

```
SELECT * FROM gcrs.customer_city
ORDER BY city ASC, customer_id ASC
```

The results are displayed in a table titled 'Data Output' with columns 'city' and 'customer_id'. The data consists of 14 rows:

	city	customer_id
1	Ahmedabad	1003
2	Ahmedabad	1004
3	Ahmedabad	1012
4	Ahmedabad	1017
5	Ahmedabad	1035
6	Ahmedabad	1036
7	Ahmedabad	1053
8	Ahmedabad	1071
9	Bengaluru	1008
10	Bengaluru	1033
11	Bengaluru	1034
12	Bengaluru	1041
13	Bengaluru	1042
14	Bengaluru	1044

A green message bar at the bottom right indicates: 'Successfully run. Total query runtime: 253 msec. 90 rows affected.'

Number of records = 90

Salary

CREATE TABLE Salary

```
(  
    Job_Position VARCHAR(15) NOT NULL,  
    Salary INT NOT NULL,  
    PRIMARY KEY (Job_Position)  
);
```

The screenshot shows the pgAdmin 4 interface. On the left, the sidebar lists various database objects: FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions (1), Materialized Views, Sequences, and Tables (12). The 'salary' table is selected and highlighted in blue. The main pane displays the 'Query Editor' with the following SQL query:

```
1 SELECT * FROM gcrs.salary  
2 ORDER BY job_position ASC
```

The results of the query are shown in a table titled 'Data Output':

job_position	salary
Accountant	80000
CEO	1000000
Clerk	45000
Driver	20000
Manager	100000

A green message bar at the bottom right indicates: "Successfully run. Total query runtime: 277 msec. 5 rows affected."

Number of records = 5

Car Record

```
CREATE TABLE Car_Record
```

```
(
```

```
    Chassis_Number VARCHAR(20) NOT NULL,  
    License_Plate_Number VARCHAR(10) NOT NULL,  
    GPS_ID VARCHAR(15) NOT NULL,  
    Average_Rating FLOAT NOT NULL,  
    Branch_ID INT NOT NULL,  
    Car_Type VARCHAR(10) NOT NULL,  
    Car_Status VARCHAR(15) NOT NULL,  
    PRIMARY KEY (Chassis_Number),  
    FOREIGN KEY (Branch_ID) REFERENCES Branch_Record(Branch_ID)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
    FOREIGN KEY (Car_Type) REFERENCES Rate_Table(Car_Type)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
    UNIQUE (License_Plate_Number),  
    UNIQUE (GPS_ID)
```

```
);
```

The screenshot shows the pgAdmin 4 interface with the 'car_record' table selected in the left sidebar. The main area displays the table's data with 80 rows. A message at the bottom right indicates the query was successfully run with a runtime of 285 msec and 80 rows affected.

chassis_number	license_plate_number	gps_id	average_rating	branch_id	car_type	car_st
1 9XF8Z2CE77569	GJ01RK5478	957LYUDROI	5.16	7	SEDAN	Wor ^
2 1C4NJCB4FD805592	UP12UP1231	6990QB2AFE	4.47	5	HATCHBACK	Not!
3 1C4RDHAG6EC09743	GA05NM3144	840GSMBRLV	4.97	10	SEDAN	Wor
4 1C6RD6GP5CS845993	HP810W9023	065TDCX0X7	3.22	6	HATCHBACK	Wor
5 1C6RD7JTCS807152	UP78IG4859	805ATEZ46I	8.17	1	MINIVAN	Wor
6 1D4PT4CK2AH601271	MH5ELO2004	031BQUSXUF	1.46	9	MINIVAN	Wor
7 1D4PU7GXIAW233689	GJ04ZAE6293	869ETWA502	7.97	7	SUV	Not!
8 1D7RE3BKXBS297848	GJ01JB7493	524WSE770K	7.27	8	SUV	Wor
9 1D7RE3GKCBS279594	HP10HO9491	767HK9C17	6.02	6	MINIVAN	Wor
10 1FMJU1J51AE14093	UP78ALS555	918OLKQ2VK	8.04	5	MINIVAN	Wor
11 1FTEW1CFXFK978719	UP43JS9403	349IBNTW3H	2.91	7	SEDAN	Wor
12 1FTWW3A50AE676535	UP85SL4738	331KOZFOC9				
13 1G6AK5S32F0913017	UP75HJ3141	793NLN8JWZ				
14 1G6AX5SXRF0M153708	GJN47A6292	697XRWW0AI				

Number of records = 80

Maintenance Record

CREATE TABLE Maintenance_Record

(

```
Bill_Number INT NOT NULL,  
Bill_Amount FLOAT NOT NULL,  
Chassis_Number VARCHAR(20) NOT NULL,  
PRIMARY KEY (Bill_Number),  
FOREIGN KEY (Chassis_Number) REFERENCES Car_Record(Chassis_Number)  
ON DELETE CASCADE  
ON UPDATE CASCADE
```

);

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database schema with various objects like FTS Configurations, Functions, and Tables. The main area shows the results of a query run against the 'maintenance_record' table.

Query Editor:

```
1 SELECT * FROM gcrs.maintenance_record  
2 ORDER BY bill_number ASC
```

Data Output:

	bill_number	bill_amount	chassis_number
1	1	13697.05	JTDK0TB83D1119831
2	2	10207.34	5GALRBED9AJ030833
3	3	5082.37	KNAFU6A27C554545
4	4	12593.95	5TFBW5F14CX581924
5	5	12638.86	WAUDG9E26A181698
6	6	13123.05	JNTBJ0HP2DM096626
7	7	8757.87	JTDKUD32ED813982
8	8	15960.49	1G8RD7J19CS087152
9	9	9087.21	WAULV94E29N644198
10	10	13347.48	1GDGX670970149724
11	11	4091.93	WBANF33557B024819
12	12	1542.31	1FMUJUJ51AE140936
13	13	15779.46	2G61RSS3XF9649515
14	14	15525.17	3LN6L2LUODR120619

Notifications: Successfully run. Total query runtime: 226 msec. 20 rows affected.

Number of records = 20

Employee Record

```
CREATE TABLE Employee_Record
(
    Employee_ID INT NOT NULL,
    F_name VARCHAR(15) NOT NULL,
    M_name VARCHAR(15) NOT NULL,
    L_name VARCHAR(15) NOT NULL,
    Pass_word VARCHAR(20) NOT NULL,
    Job_Position VARCHAR(15) NOT NULL,
    Branch_ID INT NOT NULL,
    PRIMARY KEY (Employee_ID),
    FOREIGN KEY (Branch_ID) REFERENCES Branch_Record(Branch_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (Job_Position) REFERENCES Salary(Job_Position)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

The screenshot shows the pgAdmin 4 interface with the 'Employee_Record' table selected. The left sidebar displays various database objects like FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, and Tables (12). The main pane shows the table structure with columns: employee_id [PK] integer, f_name character varying (15), m_name character varying (15), l_name character varying (15), pass_word character varying (20), job_position character varying (15), and branch_id integer. Below the structure, a query editor window contains the following SQL code:

```
1 SELECT * FROM gcrs.employee_record
2 ORDER BY employee_id ASC
```

The results pane displays 14 rows of data:

employee_id	f_name	m_name	l_name	pass_word	job_position	branch_id
1	Gustav	Man	Jean	GAp8oUnQe1	Clerk	1
2	Dora	Grace	Cars	FygeNBPS	Manager	2
3	Paula	Andris	Whitewood	7PiOCAE7NP24	Clerk	3
4	Derk	Shamus	Steward	fpM9uh	Accountant	4
5	Tades	Nilson	Akitt	SJ0f6TRK	Clerk	5
6	Missy	Vladamir	Espadater	2iETN6lf	Clerk	6
7	Thom	Basile	Titford	TqLKHHP	Clerk	7
8	Dorolsa	Rolfe	Orhrt	QVJOFpNU	Manager	8
9	Vincent	Orson	Tratearn	uTXeqBm3p	Clerk	9
10	Katlin	Denis	Tolumello	o1GQOeqeOG	Manager	10
11	Lura	Pat	McCahey	DV9xvtoS8gi5	Clerk	11
12	Janot	Vanya	Gipp	L2ELJE2NFrUN	Accountant	12
13	Arlie	Calhoun	Petrusz			13
14	Mariia	Hamish	Anfrink			14

A green message bar at the bottom right of the results pane states: "Successfully run. Total query runtime: 293 msec. 80 rows affected."

Number of records = 80

Rents

CREATE TABLE Rents

(

```
Time_Period FLOAT NOT NULL,  
Drop_Off_Point VARCHAR(30) NOT NULL,  
Pickup_Point VARCHAR(30) NOT NULL,  
Bill FLOAT NOT NULL,  
Review VARCHAR(15) NOT NULL,  
Trip_ID INT NOT NULL,  
Chassis_Number VARCHAR(20) NOT NULL,  
Customer_ID INT NOT NULL,  
Employee_ID INT,  
PRIMARY KEY (Trip_ID),  
FOREIGN KEY (Chassis_Number) REFERENCES Car_Record(Chassis_Number)  
ON DELETE CASCADE  
ON UPDATE CASCADE,  
FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)  
ON DELETE CASCADE  
ON UPDATE CASCADE,  
FOREIGN KEY (Employee_ID) REFERENCES Employee_Record(Employee_ID)  
ON DELETE CASCADE  
ON UPDATE CASCADE
```

);

The screenshot shows the pgAdmin 4 interface with the 'rents' table selected in the 'gcrs.rents' schema. The table structure includes columns: time_period, drop_off_point, pickup_point, bill, review, trip_id [PK], and chassis_number. The data output shows 150 rows of sample data from various cities like Lucknow, Gangtok, Surat, Vadodara, etc., with reviews ranging from Bad to Excellent and bills ranging from 5180.37 to 8442.6.

time_period	drop_off_point	pickup_point	bill	review	trip_id [PK]	chassis_number
562.84	Lucknow	Kanpur	8442.6	Bad	10001	WAU4GBFB9 ^
733.24	Gangtok	Bengaluru	5132.68	Excellent	10002	WDDGF4HB1
451.24	Surat	Vadodara	3158.68	Fast	10003	1N6AD0CU6I
773.12	Vadodara	Hyderabad	11596.8	Bad	10004	WBAVA3359
282.66	Ahmedabad	Delhi	3674.58	Bad	10005	JTDKTUJ03B
895.37	Gangtok	Vadodara	13430.55	Slow	10006	1XFBZ2E2C
872.17	Surat	Bengaluru	13082.55	Bad	10007	WBAWC3354
829.19	Mumbai	Bengaluru	7462.71	Average	10008	JTDKD7B3JL
991.92	Surat	Delhi	6943.44	Fast	10009	5GALRBED9/
399.49	Goa	Kanpur	5180.37	Slow	10010	KNAFU6A27i
18.72	Bengaluru	Vadodara	131.04	Average	10011	5TFBW5F14C
439.05	Mumbai	Vadodara	5707.65	Fast	10012	JN1CV6FE1F
524.13	Lucknow	Goa				
779.57	Bengaluru	Hyderabad				

Number of records = 150

Employee Record Phone Number

```
CREATE TABLE Employee_Record_Phone_Number
```

```
(
```

```
    Employee_ID INT NOT NULL,  
    Phone_Number NUMERIC NOT NULL,  
    PRIMARY KEY (Phone_Number, Employee_ID),  
    FOREIGN KEY (Employee_ID) REFERENCES Employee_Record(Employee_ID)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE
```

```
);
```

The screenshot shows the pgAdmin 4 interface. On the left, the browser tree displays various database objects like FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, and Tables (12). The 'Tables (12)' section is expanded, showing tables such as branch_record, branch_record_phone_number, car_record, customer, customer_city, customer_phone_number, employee_record, maintenance_record, rate_table, rents, salary, and trigger functions. The 'employee_record_phone_number' table is selected.

In the center, the Query Editor window contains the following SQL code:

```
1 SELECT * FROM gcrs.employee_record_phone_number  
2 ORDER BY employee_id ASC, phone_number ASC
```

Below the query editor is the Data Output pane, which displays the results of the query. The results are presented in a table with two columns: 'employee_id' and 'phone_number'. The data is as follows:

employee_id	phone_number
1	6672432653
2	3705417902
3	8253262914
4	5751513925
5	9987403853
6	9361299896
7	8525608524
8	3102125378
9	3378735783
10	4798798170
11	3085049311
12	4047805821
13	3637829479
14	3492551484

A green success message at the bottom right of the data output pane states: "Successfully run. Total query runtime: 204 msec. 80 rows affected."

Number of records = 80

SQL Queries and Results

1. Select all records from customer

```
SELECT * FROM CUSTOMER
```

The screenshot shows the pgAdmin 4 interface. On the left, the object browser displays various database objects like FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, and Tables. The 'customer' table is selected. The main pane shows a query editor with the following SQL code:

```
1 SET SEARCH_PATH TO gcrs
2 1 SELECT * FROM CUSTOMER
3
4 2 SELECT CAR_TYPE, COUNT(*)
5   FROM CAR_RECORD
6   GROUP BY CAR_TYPE
7
8 3 SELECT CHASSIS_NUMBER,GPS_ID,LICENSE_PLATE_NUMBER,CAR_TYPE
9   FROM CAR_RECORD
10 WHERE CAR_TYPE='LICDML'
```

The results pane displays the data from the CUSTOMER table:

proof_id	customer_id	f_name	m_name	l_name	pass_word	
1	WBAVW535XB6P635761	1001	Aditya	Arpit	Shah	Ydjjuk
2	WAUCFAFH9CNCN867237	1002	Rishi	Raj	Shivani	FZBzmH2DE
3	WBAAV33421F963287	1003	Rushi	Ruchit	Patel	AYomneW
4	JM1NC2JF6E0511179	1004	Pavan	Karan	Kotak	SQVnXMYEO
5	WWWED7AJ3BW848853	1005	Sanket	Anil	Jain	8TZRDOC
6	SCFAD05D69G269775	1006	Nishit	Vasant	Jagetia	Qm6Rp1eP3
7	3D4PG6FV0AT544828	1007	Vedant	Dipen	Thakkar	3vV7qagY
8	WAU14AE36N123592	1008	Sarthak	Tarak	Patel	LSLteo
9	WA1LGBFE3CD565292	1009	Rahul	Narain	Khatri	ZtxQ8gDjfW9
10	WBAYB6C58DC994289	1010	Bhagesh	Ashok	Ganatra	Dw3KVe
11	WBAFR1CS56BC724366	1011	Neel	Kunj	Atmaran	✓ Successfully run. Total query runtime: 181 msec. 80 rows affected.
12	1GYS4HEF6BR512959	1012	Dhruvil			

2 . Get count of number of cars category wise

```
SELECT CAR_TYPE, COUNT(*)  
FROM CAR_RECORD  
GROUP BY CAR_TYPE
```

The screenshot shows the pgAdmin 4 interface. On the left is a tree view of database objects under 'carserver/postgres@PostgreSQL 10'. The 'Tables' node is expanded, showing 'customer', 'branch_record', 'branch_record_phone_number', 'car_record', and 'customer'. The 'customer' table is selected. On the right is a 'Query Editor' window with the following SQL code:

```
1 SET SEARCH_PATH TO gcrs  
2 1 SELECT * FROM CUSTOMER  
3  
4 2 SELECT CAR_TYPE, COUNT(*)  
5     FROM CAR_RECORD  
6     GROUP BY CAR_TYPE  
7  
8 3 SELECT CHASSIS_NUMBER,GPS_ID,LICENSE_PLATE_NUMBER,CAR_TYPE  
9     FROM CAR_RECORD  
10    WHERE CAR_TYPE='SEDAN'
```

Below the code, the 'Data Output' tab is selected, displaying a table with the results:

car_type	count
MINIVAN	18
HATCHBACK	17
SEDAN	24
SUV	21

A green success message at the bottom right of the query editor says: "Successfully run. Total query runtime: 110 msec. 4 rows affected."

3 Get chassis number, license plate number and gps id where car category is sedan.

```
SELECT CHASSIS_NUMBER,GPS_ID,LICENSE_PLATE_NUMBER,CAR_TYPE  
FROM CAR_RECORD  
WHERE CAR_TYPE='SEDAN'
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser Tab:** Shows multiple tabs including "Meet - jvn-bqtn-apy", "IT214 - lab 10 - Google Docs", and "SQL Queries - Google Docs".
- pgAdmin Header:** Includes "File", "Object", "Tools", and "Help" menus.
- Left Sidebar (Browser):** Lists database objects:
 - FTS Templates
 - Foreign Tables
 - Functions (1)
 - customer_details(a integer)
 - Materialized Views
 - Sequences
 - Tables (12)
 - branch_record
 - branch_record_phone_number
 - car_record
 - customer
 - customer_city
 - customer_phone_number
 - employee_record
 - employee_record_phone_number
 - maintenance_record
 - rate_table
 - rents
 - Columns (9)
 - time_period
 - drop_off_point
 - pickup_point
 - bill
 - review
 - trip_id
 - chassis_number
 - customer_id
 - employee_id
 - Constraints
- Query Editor:** Displays the SQL query:

```
6   GROUP BY CAR_TYPE  
7  
8 3  SELECT CHASSIS_NUMBER,GPS_ID,LICENSE_PLATE_NUMBER,CAR_TYPE  
9    FROM CAR_RECORD  
10   WHERE CAR_TYPE='SEDAN'  
11  
12 4  SELECT * FROM EMPLOYEE_RECORD  
13    WHERE JOB_POSITION = 'Accountant'  
14  
15 5  SELECT * FROM CUSTOMER_CITY
```
- Data Output:** Shows the results of the query:

	chassis_number	gps_id	license_plate_number	car_type
1	WAUAGBF99AN057316	880TVYSP1	GJ06HL6090	SEDAN
2	WBAVA33597F895214	709EHCSSZ7	GJ01HG8919	SEDAN
3	19XPB2E22CE775695	957LVUDROI	GJ01RK5478	SEDAN
4	WBAWC33547P036766	100BIASC7Z	GJ01RK4418	SEDAN
5	WAUDG98E26A181698	853ZH09P70	MH56LR4890	SEDAN
6	KNADMA4A34C6384112	838UPB96AA	MH29TU4839	SEDAN
7	JN1BJ0HP2DM096626	384MGEGBH	MH46RE2360	SEDAN
8	WAULV94E29N644198	9750TYC159	HP48AS5893	SEDAN
9	JN8AZ1MUSEW736485	1372ORV568	UP73CH6090	SEDAN
10	JTJBMTFX3A599568	620LHK7Y00	GJ13KL1111	SEDAN
11	3C3CFCCR8DT916916	496BHAXULD	HP11GK2121	
12	WBA3A5C52EF023019	875VTWORHS	WB58LA8492	
- Message Bar:** Shows a green message: "Successfully run. Total query runtime: 179 msec. 24 rows affected."

4 Select employees whose job position is accountant.

```
SELECT * FROM EMPLOYEE_RECORD  
WHERE JOB_POSITION = 'Accountant'
```

The screenshot shows the pgAdmin 4 interface. On the left, the object browser displays various database objects like FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, Tables, and Views. The 'customer' table is currently selected. In the center, the Query Editor window contains the following SQL code:

```
11  
12 4 SELECT * FROM EMPLOYEE_RECORD  
13 WHERE JOB_POSITION = 'Accountant'  
14  
15 5 SELECT * FROM CUSTOMER_CITY  
16 WHERE CITY = 'Ahmedabad'  
17  
18 6 SELECT COUNT(*) FROM RENTS  
19 WHERE DROP_OFF_POINT='Kanpur'  
20
```

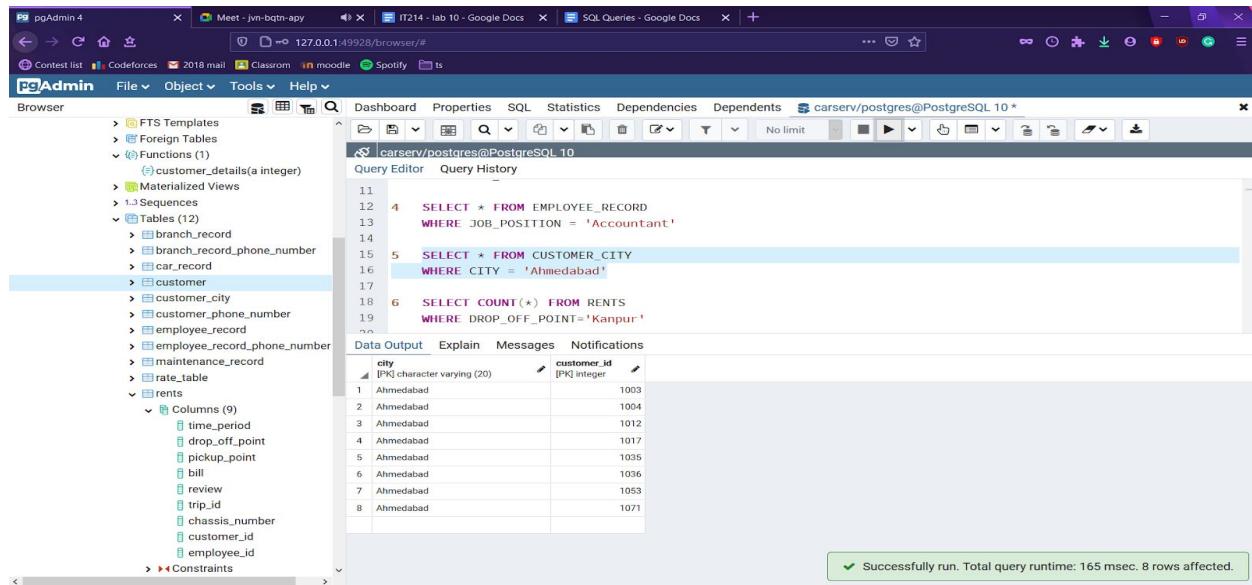
Below the code, the Data Output tab is active, showing the results of the first query (SELECT * FROM EMPLOYEE_RECORD WHERE JOB_POSITION = 'Accountant'). The results are as follows:

employee_id	f_name	m_name	l_name	pass_word	job_position	bra
1	Derk	Shamus	Steward	fpM9uh	Accountant	inte
2	Janot	Vanya	Gipp	12ELIE2NfUN	Accountant	
3	Ailee	Calhoun	Petruszka	zGIOHGdT0yf	Accountant	
4	Marita	Hamish	Andrick	VHIpve9JhKUj	Accountant	
5	Iver	Willi	Kemermann	60Yk2	Accountant	
6	Brinna	Leigh	Millican	n11ZK	Accountant	
7	Hercule	Gardy	Sperring	97RnSG	Accountant	
8	Leone	Andrej	Dennant	zeINR7vJ4	Accountant	
9	Rikki	Felic	Penn	43o9ef8PSIK	Accountant	
10	Orv	Hoyt	Scalhill	p8mMp5	Accountant	
11	Virgie	Pierre	Bellson			

At the bottom right of the Data Output tab, a green message box indicates: "Successfully run. Total query runtime: 169 msec. 22 rows affected."

5 Get a customer id who lives in Ahmedabad.

```
SELECT * FROM CUSTOMER_CITY  
WHERE CITY = 'Ahmedabad'
```



The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects, including FTS Templates, Foreign Tables, Functions (1), Materialized Views, Sequences, Tables (12), and various tables like branch_record, car_record, customer, customer_city, customer_phone_number, employee_record, and maintenance_record. The 'rents' table is currently selected. On the right, the 'Query Editor' pane contains the following SQL code:

```
11  
12 4   SELECT * FROM EMPLOYEE_RECORD  
13   WHERE JOB_POSITION = 'Accountant'  
14  
15 5   SELECT * FROM CUSTOMER_CITY  
16   WHERE CITY = 'Ahmedabad'  
17  
18 6   SELECT COUNT(*) FROM RENTS  
19   WHERE DROP_OFF_POINT='Kanpur'  
20
```

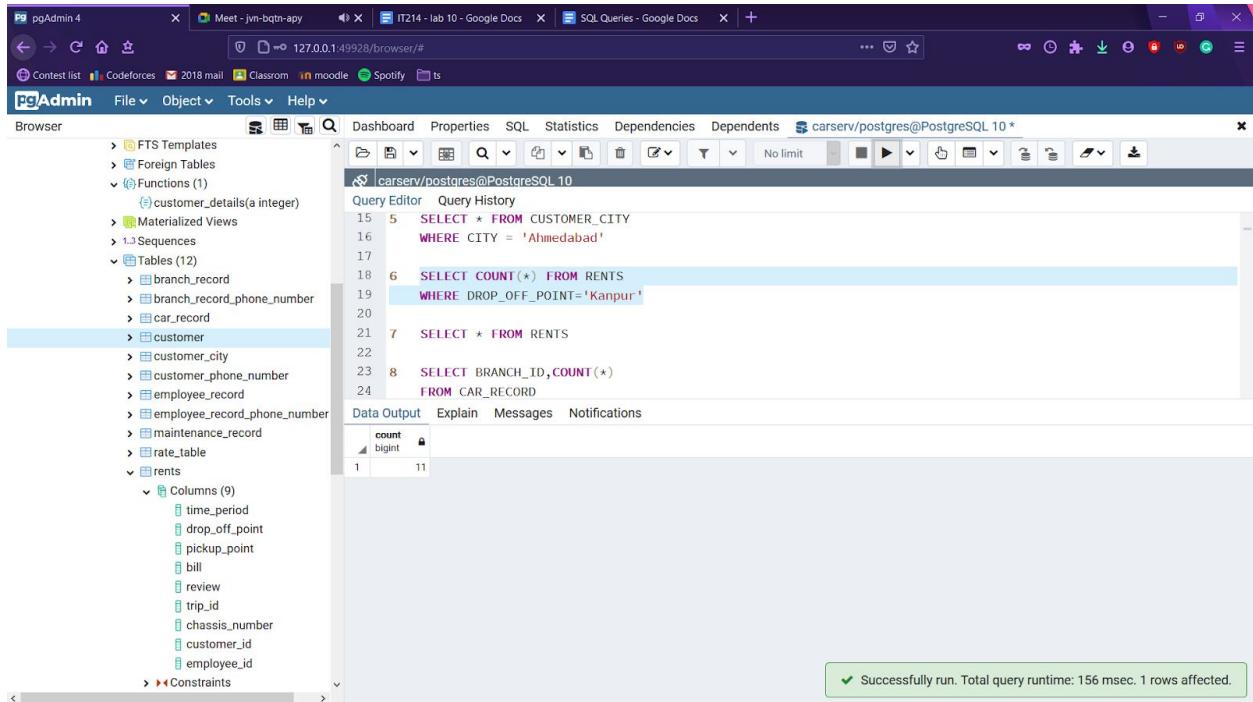
The 'Data Output' pane shows the results of the fifth query (SELECT * FROM CUSTOMER_CITY WHERE CITY = 'Ahmedabad'). The output is a table with two columns: 'city' and 'customer_id'. The data is as follows:

city	customer_id
Ahmedabad	1003
Ahmedabad	1004
Ahmedabad	1012
Ahmedabad	1017
Ahmedabad	1035
Ahmedabad	1036
Ahmedabad	1053
Ahmedabad	1071

A green message bar at the bottom right of the Data Output pane says: "Successfully run. Total query runtime: 165 msec. 8 rows affected."

6 Count the number of rent histories whose drop-off point is Kanpur.

```
SELECT COUNT(*) FROM RENTS  
WHERE DROP_OFF_POINT='Kanpur'
```



The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the database schema structure on the left, including FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, Tables (12), branch_record, branch_record_phone_number, car_record, customer, customer_city, customer_phone_number, employee_record, employee_record_phone_number, maintenance_record, rate_table, rents, and their respective columns and constraints.
- Query Editor:** Displays the SQL query:

```
15 5  SELECT * FROM CUSTOMER_CITY  
16  WHERE CITY = 'Ahmedabad'  
17  
18 6  SELECT COUNT(*) FROM RENTS  
19  WHERE DROP_OFF_POINT='Kanpur'  
20  
21 7  SELECT * FROM RENTS  
22  
23 8  SELECT BRANCH_ID,COUNT(*)  
     FROM CAR_RECORD
```
- Data Output:** Shows the result of the query:

count	bigint
1	11
- Status Bar:** Shows a green message: "Successfully run. Total query runtime: 156 msec. 1 rows affected."

7 Display rental history.

SELECT * FROM RENTS

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane lists database objects: FTS Templates, Foreign Tables, Functions (1), Materialized Views, Sequences, Tables (12), branch_record, branch_record_phone_number, car_record, customer, customer_city, customer_phone_number, employee_record, employee_record_phone_number, maintenance_record, rate_table, rents, and its columns (time_period, drop_off_point, pickup_point, bill, review, trip_id, chassis_number, customer_id, employee_id). The 'Query Editor' pane contains the following SQL code:

```
15 5  SELECT * FROM CUSTOMER_CITY
16 WHERE CITY = 'Ahmedabad'
17
18 6  SELECT COUNT(*) FROM RENTS
19 WHERE DROP_OFF_POINT='Kanpur'
20
21 7  SELECT * FROM RENTS
22
23 8  SELECT BRANCH_ID,COUNT(*)
24 FROM CAR_RECORD
```

The 'Data Output' pane displays the results of the last query (SELECT * FROM RENTS) as a table:

time_period	drop_off_point	pickup_point	bill	review	trip_id	chassis_number
895.37	Gangtok	Vadodara	13430.55	Slow	10006	19XFB2E22C
872.17	Surat	Bengaluru	13082.55	Bad	10007	WBAWC3354
829.19	Mumbai	Bengaluru	7462.71	Average	10008	JTDKDTB33L
991.92	Surat	Delhi	6943.44	Fast	10009	5GALRBED9/
398.49	Goa	Kanpur	5180.37	Slow	10010	KNAFU6A27I
18.72	Bengaluru	Vadodara	131.04	Average	10011	5TFBW5F14C
439.05	Mumbai	Vadodara	5707.65	Fast	10012	JN1CV6E1F1
524.13	Lucknow	Goa	4717.17	Average	10013	4T3BA3BB4E
779.57	Bengaluru	Hyderabad	10134.41	Fast	10014	WVGFF9BP6
442.1	Jaipur	Hyderabad	6631.5	Worst	10015	WAUDG98E2
135.95	Lucknow	Bengaluru				

A green success message at the bottom right of the data output pane reads: "Successfully run. Total query runtime: 151 msec. 150 rows affected."

8 Find the number of cars in a particular branch.

```
SELECT BRANCH_ID,COUNT(*)
FROM CAR_RECORD
GROUP BY BRANCH_ID
ORDER BY BRANCH_ID
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects, including FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, Tables (12), and various schema-level objects like customer, customer_city, customer_phone_number, employee_record, employee_record_phone_number, maintenance_record, rate_table, rents, and constraints. The main area is the Query Editor, which contains the following SQL code:

```
20
21 7   SELECT * FROM RENTS
22
23 8   SELECT BRANCH_ID,COUNT(*)
24     FROM CAR_RECORD
25     GROUP BY BRANCH_ID
26     ORDER BY BRANCH_ID
27
28 9   SELECT COUNT(*)
```

Below the Query Editor is the Data Output pane, which displays a table with three columns: branch_id, count, and a lock icon. The data is as follows:

branch_id	count	lock
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	10	

A green success message at the bottom right of the Data Output pane states: "Successfully run. Total query runtime: 164 msec. 10 rows affected."

9 Count the number of rent histories whose pick-up point is Rajkot.

```
SELECT COUNT(*)
FROM RENTS
WHERE PICKUP_POINT='Rajkot'
```

The screenshot shows the pgAdmin 4 interface. On the left is a tree view of database objects under 'Browser'. In the center is the 'Query Editor' window with the following SQL code:

```
24 FROM CAR_RECORD
25 GROUP BY BRANCH_ID
26 ORDER BY BRANCH_ID
27
28 9 SELECT COUNT(*)
29 FROM RENTS
30 WHERE PICKUP_POINT='Rajkot'
31
32 10 SELECT TRIP_ID FROM RENTS
33 WHERE RENTS.TIME_PERIOD > 300
```

Below the code is a 'Data Output' table:

count	bigint
1	13

A green message bar at the bottom right says: "Successfully run. Total query runtime: 150 msec. 1 rows affected."

10 Display the rent id who used the car for more than 300 min

```
SELECT TRIP_ID FROM RENTS  
WHERE RENTS.TIME_PERIOD > 300
```

The screenshot shows the pgAdmin 4 interface. The left pane is the Browser, displaying a tree view of database objects including FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, Tables (12), and various record and customer tables. The 'rents' table is currently selected. The right pane contains the Query Editor and Data Output panes.

Query Editor:

```
20  SELECT COUNT(*)
21  FROM RENTS
22  WHERE PICKUP_POINT='Rajkot'
23
24
25  10  SELECT TRIP_ID FROM RENTS
26  WHERE RENTS.TIME_PERIOD > 300
27
28
29  11  SELECT * FROM CAR_RECORD
30  WHERE CAR_TYPE = 'SEDAN' AND CAR_STATUS = 'Working'
31
32
33
34
35
36
37
```

Data Output:

trip_id	[PK] integer
1	10006
2	10007
3	10008
4	10009
5	10010
6	10012
7	10013
8	10014
9	10015
10	10018
11	10019
12	10021

Message bar at the bottom: ✓ Successfully run. Total query runtime: 254 msec. 105 rows affected.

11 Select cars which are working and car type is sedan.

```
SELECT * FROM CAR_RECORD  
WHERE CAR_TYPE = 'SEDAN' AND CAR_STATUS = 'Working'
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser Tab:** pgAdmin 4
- Query Editor:** carserver/postgres@PostgreSQL_10
- SQL Query:**

```
33   WHERE RENTS.TIME_PERIOD > 300  
34  
35 11  SELECT * FROM CAR_RECORD  
36    WHERE CAR_TYPE = 'SEDAN' AND CAR_STATUS = 'Working'  
37  
38 12  SELECT * FROM CAR_RECORD  
39    WHERE AVERAGE_RATING > 4.00  
40  
41 13 select customer_id, count(*)  
42   from rents
```
- Data Output:** A table showing results from the query. The columns are: chassis_number, license_plate_number, gps_id, average_rating, branch_id, car_type, and car_status. The table contains 19 rows of data.
- Status Bar:** Successfully run. Total query runtime: 260 msec. 19 rows affected.

chassis_number	license_plate_number	gps_id	average_rating	branch_id	car_type	car_status
WAU4G6BF99AN057316	GJ06HL6090	880TVVS8P1	4.98	6	SEDAN	Wor ^
WBVAVA33597895214	GJ01HG8919	709EHCSZ7	9.97	5	SEDAN	Wor
19XPB2E22CE77569	GJ01RK5478	957LYUDROI	4.44	7	SEDAN	Wor
WBAWC33547P036766	GJ01RK4418	100BIASC7Z	4.21	5	SEDAN	Wor
WAUDG98E26A181698	MHSGLR4890	853ZH09P70	9.33	3	SEDAN	Wor
KNADM4A34C6384112	MH29TU4839	838UP896AA	8.08	5	SEDAN	Wor
JN18J0HP2D0M906626	MH46RE2360	384MGEG3BH	6.84	7	SEDAN	Wor
WAULV94E29N644198	HP48AS5893	975OTYC159	2.45	2	SEDAN	Wor
JNBIAZ1MUSEW736485	UF73CH6090	137ZORV568	1.48	2	SEDAN	Wor
JLIBM7FX3A5995658	GJ13KL1111	620LUK7Y00	0.73	2	SEDAN	Wor
3C3FFCRBDT916916	HP11GK2121	496HAXULD				
WAU4G6BF99AN057316	GJ06HL6090	880TVVS8P1				
WBVAVA33597895214	GJ01HG8919	709EHCSZ7				
19XPB2E22CE77569	GJ01RK5478	957LYUDROI				
WBAWC33547P036766	GJ01RK4418	100BIASC7Z				
WAUDG98E26A181698	MHSGLR4890	853ZH09P70				
KNADM4A34C6384112	MH29TU4839	838UP896AA				
JN18J0HP2D0M906626	MH46RE2360	384MGEG3BH				
WAULV94E29N644198	HP48AS5893	975OTYC159				
JNBIAZ1MUSEW736485	UF73CH6090	137ZORV568				
JLIBM7FX3A5995658	GJ13KL1111	620LUK7Y00				
3C3FFCRBDT916916	HP11GK2121	496HAXULD				

12 Select all cars with average rating higher than 4.00

```
SELECT * FROM CAR_RECORD  
WHERE AVERAGE_RATING > 4.00
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects, including FTS Templates, Foreign Tables, Functions (1), Materialized Views, Sequences, and Tables (12). The 'rents' table is currently selected. The 'Query Editor' pane at the top contains the following SQL code:

```
12 SELECT * FROM CAR_RECORD  
WHERE AVERAGE_RATING > 4.00  
13 select customer_id, count(*)  
from rents  
group by customer_id  
order by customer_id  
14 select count(*)
```

The 'Data Output' pane below shows the results of the query, which lists 57 rows of data from the 'rents' table. The columns are: chassis_number, license_plate_number, gps_id, average_rating, branch_id, car_type, and car_st. The results are ordered by customer_id. A green message bar at the bottom right indicates that the query was successfully run and affected 57 rows.

customer_id	count(*)
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1
21	1
22	1
23	1
24	1
25	1
26	1
27	1
28	1
29	1
30	1
31	1
32	1
33	1
34	1
35	1
36	1
37	1
38	1
39	1
40	1
41	1
42	1
43	1
44	1
45	1
46	1
47	1
48	1
49	1
50	1
51	1
52	1
53	1
54	1
55	1
56	1
57	1

13 Find the number of bookings done by each customer.

```
select customer_id,count(*)  
from rents  
group by customer_id  
order by customer_id
```

The screenshot shows the pgAdmin 4 interface. On the left is the 'Browser' pane, which lists various database objects like FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, Tables (including branch_record, car_record, customer, etc.), and Constraints. The 'rents' table is currently selected. The main area is the 'Query Editor' where the following SQL code is written:

```
12 SELECT * FROM CAR_RECORD  
WHERE AVERAGE_RATING > 4.00  
13 select customer_id, count(*)  
from rents  
group by customer_id  
order by customer_id  
14 select count(*)
```

Below the editor is the 'Data Output' tab, which displays the results of the third query:

customer_id	count
1	1001
2	1002
3	1003
4	1005
5	1006
6	1007
7	1008
8	1010
9	1011
10	1012
11	1013
12	1016

A green message bar at the bottom right indicates: "Successfully run. Total query runtime: 122 msec. 70 rows affected."

14 Find the number of customers who gave reviews as good.

```
select count(*)  
from rents  
where rents.review = 'Good'
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects, including FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, Tables (12), and a selected 'rents' table. The 'rents' table has 9 columns: time_period, drop_off_point, pickup_point, bill, review, trip_id, chassis_number, customer_id, and employee_id. On the right, the 'Query Editor' pane contains the following SQL code:

```
42     from rents  
43     group by customer_id  
44     order by customer_id  
45  
46 14 select count(*)  
47     from rents  
48     where rents.review = 'Good'  
49  
50 15 select count(*)  
51     from customer
```

The 'Data Output' pane shows the results of the query:

count	bigint
1	24

A green status bar at the bottom right indicates: **Successfully run. Total query runtime: 146 msec. 1 rows affected.**

15 List the number of customers with the surname " Shah ".

```
select count(*)  
from customer  
where customer.l_name = 'Shah'
```

The screenshot shows the pgAdmin 4 interface. On the left is a tree view of database objects under 'carserv/postgres@PostgreSQL 10'. The 'customer' table is selected. The main area contains a query editor with the following code:

```
14 select count(*)  
    from rents  
   where rents.review = 'Good'  
15 select count(*)  
    from customer  
   where customer.l_name = 'Shah'  
16 select f_name,m_name,l_name,time_period  
    from customer
```

Below the query editor is a 'Data Output' tab showing a single row of results:

count	bigint
1	8

A green success message at the bottom right says: "Successfully run. Total query runtime: 144 msec. 1 rows affected."

16 Find the customer details of who rented the car from 200 to 500 minutes.

```
select f_name,m_name,l_name,time_period  
from customer  
join rents  
on customer.customer_id = rents.customer_id  
where rents.time_period > 200 and rents.time_period<500
```

The screenshot shows the pgAdmin 4 interface. On the left, the object browser displays a database structure with tables like FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, and various tables including branch_record, car_record, customer, customer_city, customer_phone_number, employee_record, employee_record_phone_number, maintenance_record, rate_table, rents, and trips. The rents table is currently selected. The main pane shows a query editor with the following SQL code:

```
from customer  
where customer.l_name = 'Shah'  
16 select f_name,m_name,l_name,time_period  
from customer  
join rents  
on customer.customer_id = rents.customer_id  
where rents.time_period > 200 and rents.time_period<500  
17 select count(*) from rents
```

Below the query editor is a data output table showing 12 rows of customer information. The columns are f_name, m_name, l_name, and time_period. The data is as follows:

	f_name	m_name	l_name	time_period
1	Saketh	Nilesh	Ram	398.49
2	Devansh	Girish	Vyas	439.05
3	Rishi	Raj	Shivani	442.1
4	Devyani	Aditya	Panchal	374.34
5	Hashtil	Kaplesh	Joshi	469.6
6	Khushi	Kanu	Dave	388.43
7	Vyom	Rajesh	Modi	387.73
8	Heta	Aditya	Ahir	254.17
9	Vanshika	Amit	Kochhar	227.12
10	Faizan	Arbaz	Shaikh	255.33
11	Ved	Pankaj	Patel	
12	Tulsi	Das	Khan	

A green success message at the bottom right of the data output area states: "Successfully run. Total query runtime: 160 msec. 39 rows affected."

17 Find the number of customers who opted for drivers.

```
select count(*) from rents  
where employee_id is not NULL
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects, including FTS Templates, Foreign Tables, Functions (1), Materialized Views, Sequences, and Tables (12). The 'rents' table is selected. The 'Query Editor' pane contains two queries:

```
17 select count(*) from rents  
      where employee_id is not NULL.  
  
18 select *  
      from employee_record  
      where employee_id in (  
          select employee_id  
          from rents  
          where employee_id is not NULL)  
  
Data Output Explain Messages Notifications
```

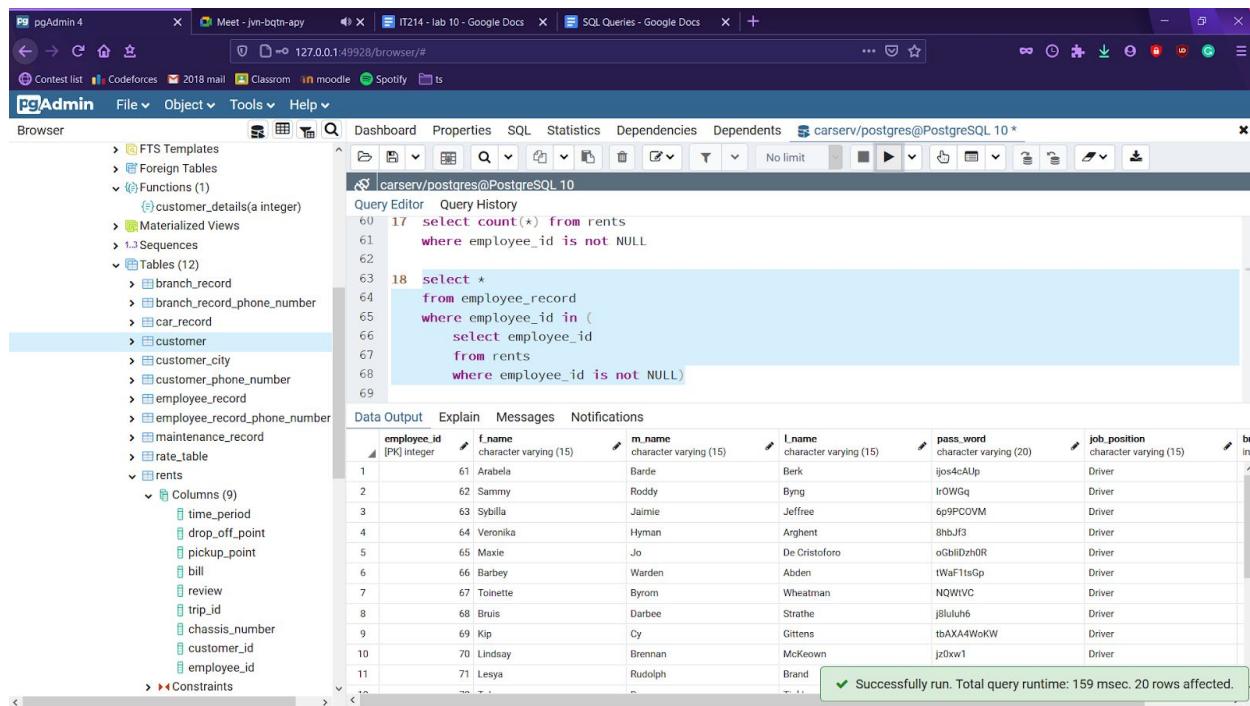
The 'Data Output' tab shows the result of the first query:

count	bigrnt
1	116

A green status bar at the bottom right indicates: ✓ Successfully run. Total query runtime: 134 msec. 1 rows affected.

18 Find the employee details who drove the car for the customers.

```
select *
from employee_record
where employee_id in (
    select employee_id
    from rents
    where employee_id is not NULL)
```



The screenshot shows the pgAdmin 4 interface with a query editor window. The browser pane on the left lists various database objects like FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, and Tables. The 'rents' table is selected. The main window displays the SQL query:

```
17 select count(*) from rents
       where employee_id is not NULL
18 select *
       from employee_record
       where employee_id in (
           select employee_id
           from rents
           where employee_id is not NULL)
```

The 'Data Output' tab shows the results of the query:

employee_id	f_name	m_name	l_name	pass_word	job_position	
1	61	Anabela	Barde	Berk	jos4cAUp	Driver
2	62	Sammy	Roddy	Byng	RoWGq	Driver
3	63	Sybilla	Jaimie	Jefree	6p9PCOVM	Driver
4	64	Veronika	Hyman	Arhent	8hbJf3	Driver
5	65	Maxie	Jo	De Cristoforo	oGbiLzh0R	Driver
6	66	Barbey	Warden	Abden	tWaF1tsGp	Driver
7	67	Toinette	Byrom	Wheatman	NQWIVC	Driver
8	68	Bruis	Darbee	Strathe	j8lulu6	Driver
9	69	Kip	Cy	Gittens	tbAXA4WoKW	Driver
10	70	Lindsay	Brennan	McKeown	jZdxw1	Driver
11	71	Lesya	Rudolph	Brand		

A green success message at the bottom right of the results pane says "Successfully run. Total query runtime: 159 msec. 20 rows affected."

19 Find the total cost of all maintenance bills.

```
select sum(bill_amount)
from maintenance_record
```

The screenshot shows the pgAdmin 4 interface. On the left is the 'Browser' pane, which contains a tree view of database objects: FTS Templates, Foreign Tables, Functions (1), Materialized Views, Sequences, Tables (12), and Rents. Under 'Tables (12)', 'customer' is selected. Under 'Rents', 'Columns (9)' are listed: time_period, drop_off_point, pickup_point, bill, review, trip_id, chassis_number, customer_id, and employee_id. The main area is the 'Query Editor' with the following SQL code:

```
65 where employee_id in (
66     select employee_id
67     from rents
68     where employee_id is not NULL)
69
70 19 select sum(bill_amount)
71 from maintenance_record
72
73 20 select *
```

The 'Data Output' tab shows the result of the query:

sum
double precision
1 231216.81

A green message bar at the bottom right indicates: **Successfully run. Total query runtime: 153 msec. 1 rows affected.**

20 Find car details of cars registered in Uttar Pradesh.

```
select *
from car_record
where license_plate_number like 'UP%'
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** pgAdmin 4
- Connections:** carserv/postgres@PostgreSQL 10*
- Query Editor:** The query is being run against the carserv database.
- Query:**

```
19  select sum(bill_amount)
  from maintenance_record
20
21  select *
  from car_record
  where license_plate_number like 'UP%'
22
23  with t as (
  select v.chassis_number as rowid, s.rate as calculatedvalue
  from rents r
  left join rate_table s on r.trip_id = s.trip_id
  left join car_record v on r.car_id = v.id
  where v.license_plate_number like 'UP%'
```
- Data Output:** A table showing 22 rows of car details registered in Uttar Pradesh (UP). The columns include chassis_number, license_plate_number, gps_id, average_rating, branch_id, car_type, and car_stu.

chassis_number	license_plate_number	gps_id	average_rating	branch_id	car_type	car_stu
JTDKDTB3D1119831	UP72CL7509	764HFLXII	4.53	5	SUV	Wor ^
5GALRBED9AJ030833	UP72DB5896	645SVA5ZW6	4.84	6	MINIVAN	Wor
JN1CV6FE1FM649864	UP68CU2906	391C0DGZSM	9.17	1	HATCHBACK	Wor
4T3BA3BB4EU011060	UP25CL5914	259EEJDW6	7.67	8	SUV	Wor
1GYFK6688R011833	UP82FO4852	194IKSDQRA	7.31	2	HATCHBACK	Wor
WAUFAFL3CA969738	UP780B2991	187FXM182I	8.79	6	MINIVAN	Wor
5UXFHDC50AL581489	UP78KJ3491	331ER012QS	6.55	1	SUV	Wor
1C6RD7J19CS087152	UP78HG4859	805ATEZ46I	3.22	6	HATCHBACK	Wor
JNBAZ1MUSEW736485	UP73CH6090	197ZORV568	1.48	2	SEDAN	Wor
WBANF33557B024819	UP73C26254	315QAWKEHU	6.61	7	MINIVAN	Wor
1FMJU1J51AE140936	UP78AL5555	918OLKQ2VK				

- Message:** Successfully run. Total query runtime: 175 msec. 22 rows affected.

21 Update the bill for all rental records.

```
with t as (
    select v.chassis_number as rowid, s.rate as calculatedvalue
    from car_record as v
    join rate_table s on v.car_type = s.car_type
)
update rents
set bill = rents.time_period * t.calculatedvalue
from t
where rents.chassis_number = t.rowid
```

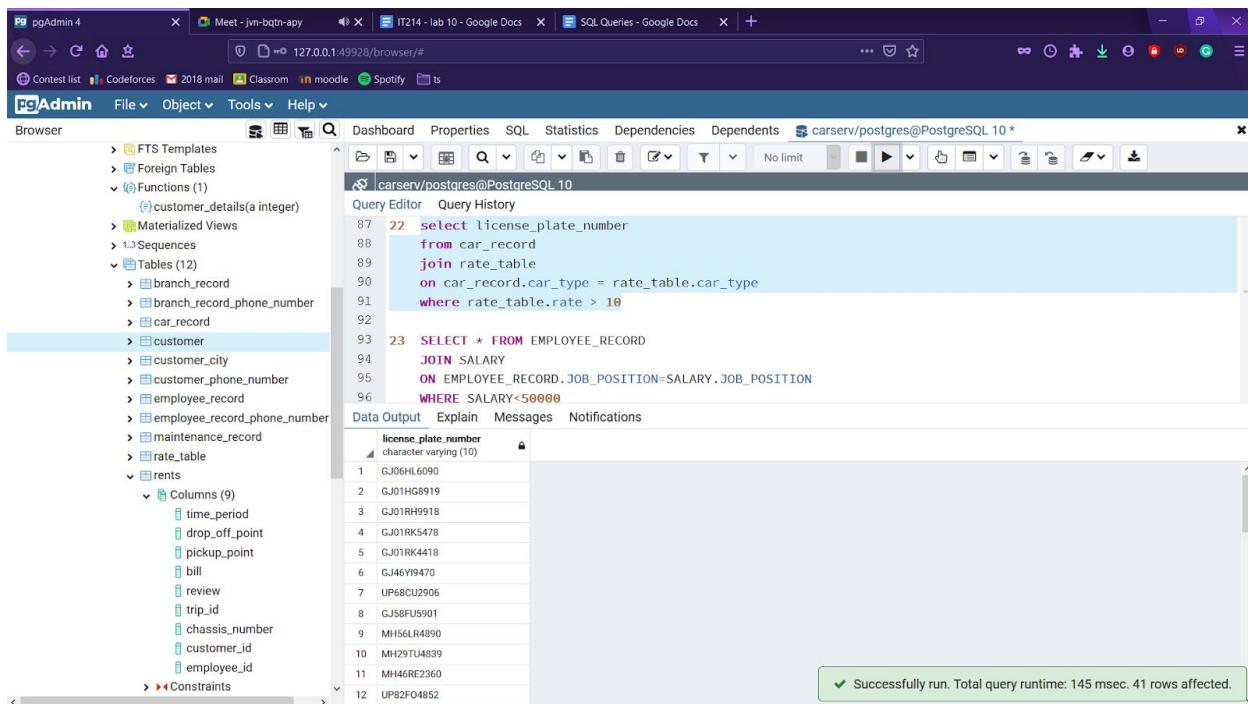
The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the database schema structure on the left, including FTS Templates, Foreign Tables, Functions (1), Materialized Views, Sequences, Tables (12), branch_record, branch_record_phone_number, car_record, customer, customer_details(a integer), customer_city, customer_phone_number, employee_record, employee_record_phone_number, maintenance_record, rate_table, rents, and rents Constraints.
- Query Editor:** Displays the SQL query being run:

```
update rents
set bill = rents.time_period * t.calculatedvalue
from t
where rents.chassis_number = t.rowid
22  select license_plate_number
from car_record
join rate_table
on car_record.car_type = rate_table.car_type
where rate_table.rate > 10
```
- Data Output:** Shows the result of the query: "UPDATE 150".
- Message Bar:** A green bar at the bottom right indicates "Query returned successfully in 161 msec."

22 Display car's license plate number whose rent is greater than 10.

```
select license_plate_number
from car_record
join rate_table
on car_record.car_type = rate_table.car_type
where rate_table.rate > 10
```



The screenshot shows the pgAdmin 4 interface with a query editor window. The left sidebar displays a tree view of database objects, including tables like customer, branch_record, and rents. The main window shows the following SQL code:

```
22 select license_plate_number
  from car_record
  join rate_table
    on car_record.car_type = rate_table.car_type
   where rate_table.rate > 10

23 SELECT * FROM EMPLOYEE_RECORD
      JOIN SALARY
        ON EMPLOYEE_RECORD.JOB_POSITION=SALARY.JOB_POSITION
       WHERE SALARY<50000
```

The results pane shows a table with one column, license_plate_number, containing 12 rows of data:

license_plate_number
GJ06HL6090
GJ01HG8919
GJ01RH9918
GJ01RK5478
GJ01RK4418
GJ46Y9470
UP68CU12906
GJ58FU5901
MH56LR4890
MH29TU4839
MH46RE2360
UP92FO4852

A green success message at the bottom right indicates: "Successfully run. Total query runtime: 145 msec. 41 rows affected."

23 Find the employee details whose salary is less than 50000.

```
SELECT * FROM EMPLOYEE_RECORD
JOIN SALARY
ON EMPLOYEE_RECORD.JOB_POSITION=SALARY.JOB_POSITION
WHERE SALARY<50000
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the database structure with tables like customer, branch_record, car_record, and rents.
- Query Editor:** Displays two queries:
 - Query 23: `SELECT * FROM EMPLOYEE_RECORD JOIN SALARY ON EMPLOYEE_RECORD.JOB_POSITION=SALARY.JOB_POSITION WHERE SALARY<50000`
 - Query 24: `SELECT CAR_TYPE,COUNT(*) FROM CAR_RECORD JOIN RENTS ON CAR_RECORD.CHAFFIS_NUMBER=RENTS.CHAFFIS_NUMBER`
- Data Output:** Shows the results of Query 23, which lists employees with salaries less than 50000. The columns are employee_id, f_name, m_name, l_name, pass_word, job_position, and bra. The results are as follows:

employee_id	f_name	m_name	l_name	pass_word	job_position	bra
1	Gustav	Man	Jean	GAp8oUnNe1	Clerk	inte
2	Paula	Andris	Whitewood	7PiOCAE7NP24	Clerk	
3	Tades	Nilson	Akitt	SJ0f6TRK	Clerk	
4	Missy	Vladamir	Espadater	2IE7N6lf	Clerk	
5	Thom	Basile	Titford	TiqljKH	Clerk	
6	Vincent	Orson	Trahearn	u1XeqBm3p	Clerk	
7	Lura	Pat	McCahey	DV9vxtoS8gj5	Clerk	
8	Niki	Foss	Anning	cfzB9nheR	Clerk	
9	Dieter	Ramon	Kelemen	VvKFxu6ThEM	Clerk	
10	Rora	Brad	Goodley	FV12p4	Clerk	
11	Winnifred	Freeman	Lalley			

Messages: A green message bar at the bottom right indicates "Successfully run. Total query runtime: 202 msec. 40 rows affected."

24 Select number of bookings done as per car type.

```
SELECT CAR_TYPE,COUNT(*) FROM CAR_RECORD  
JOIN RENTS  
ON CAR_RECORD.CHASSIS_NUMBER=RENTS.CHASSIS_NUMBER  
GROUP BY CAR_TYPE
```

The screenshot shows the pgAdmin 4 interface. On the left is a tree view of database objects under 'Browser'. In the center is a 'Query Editor' window titled 'carserv/postgres@PostgreSQL 10'. The editor contains two queries:

```
24 SELECT CAR_TYPE,COUNT(*) FROM CAR_RECORD  
JOIN RENTS  
ON CAR_RECORD.CHASSIS_NUMBER=RENTS.CHASSIS_NUMBER  
GROUP BY CAR_TYPE  
  
25 SELECT COUNT(DISTINCT CUSTOMER_ID)  
FROM RENTS  
WHERE RTI > 5000
```

Below the queries is a 'Data Output' tab showing the results of the first query:

car_type	count
MINIVAN	35
HATCHBACK	34
SEDAN	41
SUV	40

A green message bar at the bottom right indicates: 'Successfully run. Total query runtime: 143 msec. 4 rows affected.'

25 Select number of bookings done as per car type.

```
SELECT COUNT(DISTINCT CUSTOMER_ID)
FROM RENTS
WHERE BILL>5000
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the database structure with tables like FTS Templates, Foreign Tables, Functions, Sequences, Materialized Views, and various tables including branch_record, car_record, customer, and rents.
- Query Editor:** Displays two queries:
 - Query 25: `SELECT COUNT(DISTINCT CUSTOMER_ID) FROM RENTS WHERE BILL>5000`
 - Query 26: `SELECT CUSTOMER_ID FROM CUSTOMER_CITY GROUP BY CUSTOMER_ID HAVING COUNT(*)>1`
- Data Output:** Shows the result of Query 25, which has 1 row with a count of 58.
- Messages:** A green message box at the bottom right indicates "Successfully run. Total query runtime: 138 msec. 1 rows affected."

26 Count the number of customers who provided multiple cities as their residence.

```
SELECT CUSTOMER_ID FROM CUSTOMER_CITY  
GROUP BY CUSTOMER_ID  
HAVING COUNT(*)>1  
ORDER BY CUSTOMER_ID
```

The screenshot shows the pgAdmin 4 interface. On the left is the 'Browser' pane, which lists various database objects like FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, Tables, and several specific tables such as branch_record, car_record, and customer. The 'customer' table is currently selected. On the right is the 'Query Editor' pane, containing two queries. The first query is the one from the question, and the second is a related query. Below the Query Editor is the 'Data Output' pane, which displays a table with 10 rows of data. At the bottom right of the Data Output pane, there is a green success message. The top of the screen shows a browser window with tabs for Meet, Google Docs, and SQL Queries, and a system tray with various icons.

```
105 WHERE BILL>5000  
106  
107 26 SELECT CUSTOMER_ID FROM CUSTOMER_CITY  
108 GROUP BY CUSTOMER_ID  
109 HAVING COUNT(*)>1  
110 ORDER BY CUSTOMER_ID  
111  
112 27 SELECT chassis_number FROM maintenance_record  
113 GROUP BY chassis_number  
114 HAVING COUNT(*)>1
```

customer_id	
1	1071
2	1072
3	1073
4	1074
5	1075
6	1076
7	1077
8	1078
9	1079
10	1080

✓ Successfully run. Total query runtime: 134 msec. 10 rows affected.

27 Display the cars which have multiple maintenance records.

```
SELECT chassis_number FROM maintenance_record  
GROUP BY chassis_number  
HAVING COUNT(*)>1  
ORDER BY chassis_number
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects, including FTS Templates, Foreign Tables, Functions (1), Materialized Views, Sequences, Tables (12), branch_record, branch_record_phone_number, car_record, customer, customer_city, customer_phone_number, employee_record, employee_record_phone_number, maintenance_record, rate_table, and rents. The rents table is currently selected. On the right, the 'Query Editor' pane contains the following SQL code:

```
27  SELECT chassis_number FROM maintenance_record  
     GROUP BY chassis_number  
     HAVING COUNT(*)>1  
     ORDER BY chassis_number  
  
28  SELECT * FROM RENTS  
     WHERE DROP_OFF_POINT=PICKUP_POINT
```

The status bar at the bottom right indicates: "Successfully run. Total query runtime: 148 msec. 0 rows affected."

28 Display all the rental records where drop-off points and pick-up points are in the same city.

```
SELECT * FROM RENTS  
WHERE DROP_OFF_POINT=PICKUP_POINT
```

The screenshot shows the pgAdmin 4 interface with a query editor window. The query is:

```
114     HAVING COUNT(*)>1  
115     ORDER BY chassis_number  
116  
117 28  SELECT * FROM RENTS  
118 WHERE DROP_OFF_POINT=PICKUP_POINT  
119  
120 29  select license_plate_number  
121   from maintenance_record  
122   join car_record  
123   on maintenance_record.chassis_number = car_record.chassis_number
```

The results table has the following columns:

time_period	drop_off_point	pickup_point	bill	review	trip_id	chassis_number
1	562.85 Rajkot	Rajkot	8442.75	Excellent	10019	JN1B1J0HP2D
2	656.69 Kanpur	Kanpur	9850.35	Slow	10026	JN8AZT1MUSE
3	831.74 Gangtok	Gangtok	5822.18	Good	10029	WVWAP7AN8E
4	596.61 Bengaluru	Bengaluru	8949.15	Worst	10030	JTJBMTFX3A5
5	48.06 Goa	Goa	432.54	Slow	10043	1GKS1KE04ER
6	821.66 Delhi	Delhi	12327.9	Good	10046	1C6RD6GP5CS
7	898.48 Lucknow	Lucknow	8086.32	Average	10067	3N1BC1AP7BL
8	766.12 Mumbai	Mumbai	9959.56	Average	10080	KNAFU6A2ZC
9	991.92 Mumbai	Mumbai	8927.28	Average	10108	5J8TB4H34GL
10	892.63 Surat	Surat	8033.67	Average	10124	1N4AA5APXBC
11	398.49 Kanpur	Kanpur				

Message bar: ✓ Successfully run. Total query runtime: 178 msec. 16 rows affected.

29 Find all cars who have a maintenance bill.

```
select license_plate_number  
from maintenance_record  
join car_record  
on maintenance_record.chassis_number = car_record.chassis_number
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows a tree view of database objects under the schema "carserv/postgres@PostgreSQL 10". The "customer" table is currently selected.
- Query Editor:** Displays the SQL query from step 29. The results of the previous query (step 28) are shown above it.

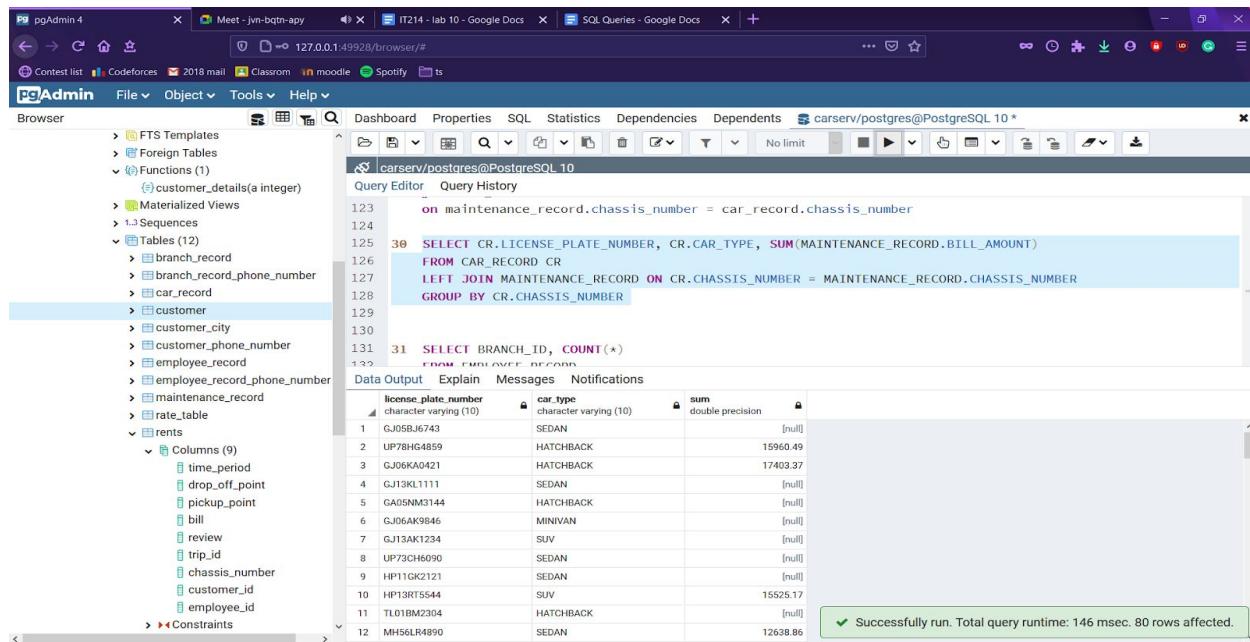
```
118 WHERE DROP_OFF_POINT=PICKUP_POINT  
119  
120 29 select license_plate_number  
121 from maintenance_record  
122 join car_record  
123 on maintenance_record.chassis_number = car_record.chassis_number  
124  
125 30 SELECT CR.LICENSE_PLATE_NUMBER, CR.CAR_TYPE, SUM(MAINTENANCE_RECORD.BILL_AMOUNT)  
126 FROM CAR_RECORD CR  
127 LEFT JOIN MAINTENANCE_RECORD ON CR.CHASSIS_NUMBER = MAINTENANCE_RECORD.CHASSIS_NUMBER
```
- Data Output:** A table titled "license_plate_number" showing 20 rows of data. The columns are character varying (10). The data is as follows:

license_plate_number
1 UP72CL7509
2 UP72DB5896
3 GJ46Y9470
4 GJ35DK3679
5 MH56LR4890
6 MH46RE2360
7 MH73OP2959
8 UP78HG4859
9 HP4BAS5893
10 HP20AJ4923
11 UP73CZ6254
12 UP78AL5555

A green message bar at the bottom right indicates: "Successfully run. Total query runtime: 278 msec. 20 rows affected."

30 Find maintenance cost of each car.

```
SELECT CR.LICENSE_PLATE_NUMBER, CR.CAR_TYPE,
SUM(MAINTENANCE_RECORD.BILL_AMOUNT)
  FROM CAR_RECORD CR
LEFT JOIN MAINTENANCE_RECORD ON CR.CHASSIS_NUMBER =
MAINTENANCE_RECORD.CHASSIS_NUMBER
 GROUP BY CR.CHASSIS_NUMBER
```



The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the database schema structure on the left, including FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, Tables (12), and various tables like customer, branch_record, car_record, etc.
- Query Editor:** Displays the SQL query from the question above. The line numbers 30, 31, and 32 are highlighted in blue.
- Data Output:** Shows the results of the query in a tabular format. The columns are license_plate_number, car_type, and sum. The data consists of 12 rows of car information and their maintenance costs.
- Messages:** A green message at the bottom right indicates "Successfully run. Total query runtime: 146 msec. 80 rows affected."

license_plate_number	car_type	sum
GJ05BL6743	SEDAN	[null]
UP78HG4859	HATCHBACK	15960.49
GJ06KA0421	HATCHBACK	17403.37
GJ13KL1111	SEDAN	[null]
GA05NM3144	HATCHBACK	[null]
GJ06AK9846	MINIVAN	[null]
GJ13AK1234	SUV	[null]
UP73CH6090	SEDAN	[null]
HP11GK2121	SEDAN	[null]
HP13RT5544	SUV	15525.17
TL01BM2304	HATCHBACK	[null]
MH56LR4890	SEDAN	12638.86

31 Find the number of employees branch wise.

```
SELECT BRANCH_ID, COUNT(*)
FROM EMPLOYEE_RECORD
GROUP BY BRANCH_ID
ORDER BY BRANCH_ID
```

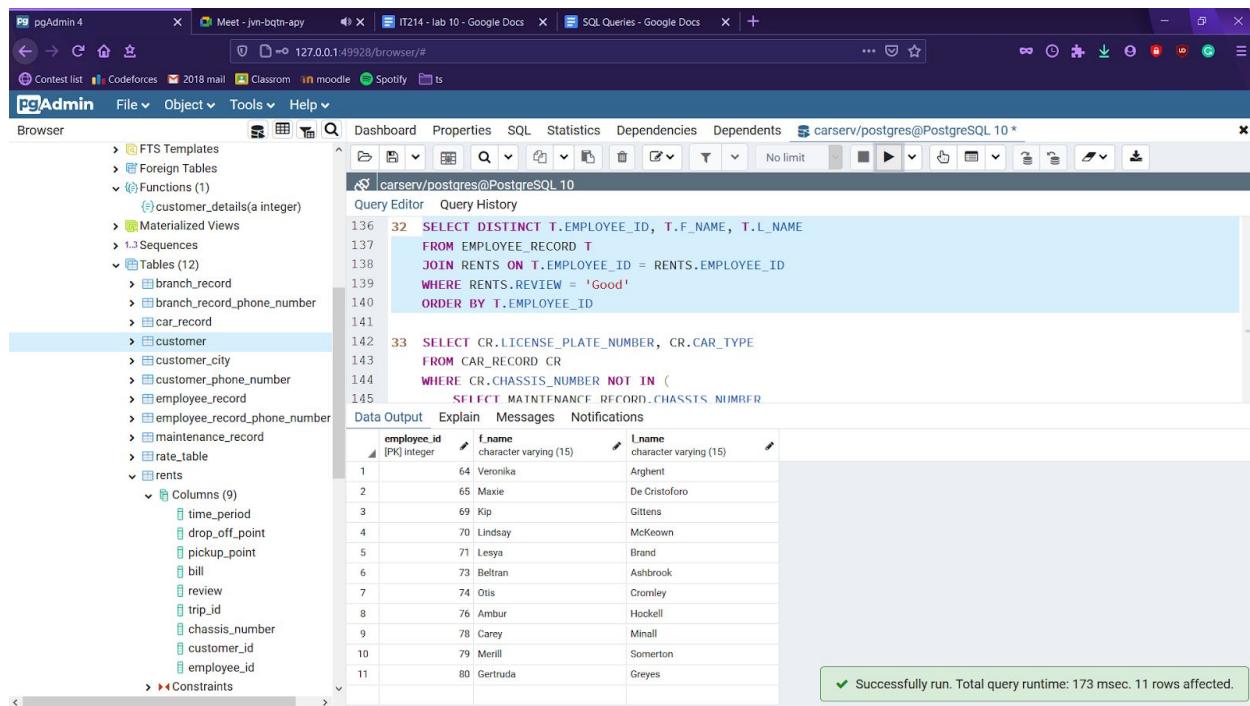
The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** pgAdmin 4
- Query Editor:** carserv/postgres@PostgreSQL 10*
- Query History:** Contains the executed query and its results.
- Data Output:** Displays the results of the query in a tabular format.
- Table Browser:** Shows the schema structure with tables like customer, branch_record, car_record, etc.
- Status Bar:** Shows a green success message: "Successfully run. Total query runtime: 136 msec. 10 rows affected."

branch_id	count
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

32 Find driver details who were reviewed as good.

```
SELECT DISTINCT T.EMPLOYEE_ID, T.F_NAME, T.L_NAME
FROM EMPLOYEE_RECORD T
JOIN RENTS ON T.EMPLOYEE_ID = RENTS.EMPLOYEE_ID
WHERE RENTS.REVIEW = 'Good'
ORDER BY T.EMPLOYEE_ID
```



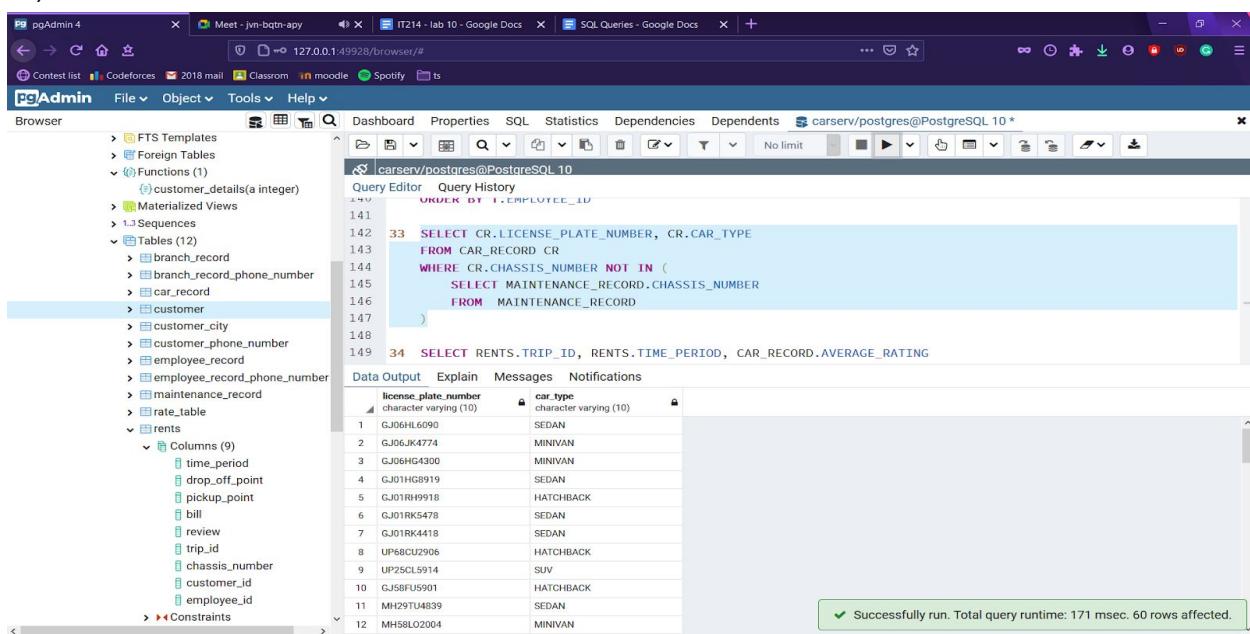
The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the database structure with tables like customer, branch_record, car_record, and rents.
- Query Editor:** Displays the SQL query from step 32.
- Data Output:** Shows the results of the query, which are 11 rows of driver names and IDs.
- Status Bar:** Shows a green message: "Successfully run. Total query runtime: 173 msec. 11 rows affected."

employee_id	f_name	l_name
1	64	Veronika
2	65	Maxie
3	69	Kip
4	70	Lindsay
5	71	Lesya
6	73	Beltran
7	74	Otis
8	76	Ambur
9	78	Carey
10	79	Merrill
11	80	Gertruda

33 Display all cars who don't have a maintenance bill.

```
SELECT CR.LICENSE_PLATE_NUMBER, CR.CAR_TYPE
FROM CAR_RECORD CR
WHERE CR.CHASSIS_NUMBER NOT IN (
    SELECT MAINTENANCE_RECORD.CHASSIS_NUMBER
    FROM MAINTENANCE_RECORD
)
```



The screenshot shows the pgAdmin 4 interface with a query editor window. The query is identical to the one above. The results show 12 rows of data, each containing a license plate number and a car type. A green message at the bottom right indicates the query was successfully run and affected 60 rows.

license_plate_number	car_type
GJ06HL6090	SEDAN
GJ06JK4774	MINIVAN
GJ06HG4300	MINIVAN
GJ01HG8919	SEDAN
GJ01RH9918	HATCHBACK
GJ01RK5478	SEDAN
GJ01RK4418	SEDAN
UP86CU2906	HATCHBACK
UP25CL5914	SUV
GJ58FU5901	HATCHBACK
MH29TU4839	SEDAN
MH58L02004	MINIVAN

Successfully run. Total query runtime: 171 msec. 60 rows affected.

34 Display rent id who rented the car for less than 300 minutes and rating is higher than 2.5.

```
SELECT RENTS.TRIP_ID, RENTS.TIME_PERIOD,
CAR_RECORD.AVERAGE_RATING
FROM RENTS
JOIN CAR_RECORD ON RENTS.CHASSIS_NUMBER =
CAR_RECORD.CHASSIS_NUMBER
WHERE CAR_RECORD.AVERAGE_RATING > 2.5 AND RENTS.TIME_PERIOD < 300
ORDER BY RENTS.TRIP_ID
```

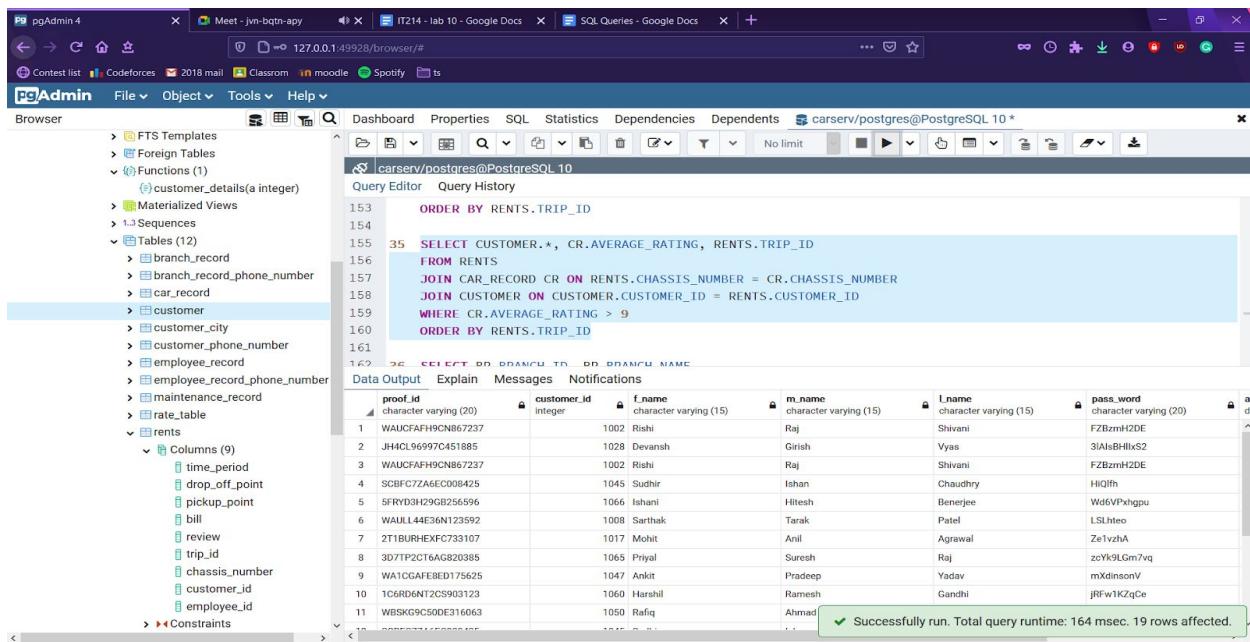
The screenshot shows the pgAdmin 4 interface. On the left is a tree view of database objects under 'Browser'. In the center is the 'Query Editor' tab, which contains two queries labeled 34 and 35. Query 34 is the one from the previous text. Query 35 is a alternative query using a different join syntax. Below the queries is a 'Data Output' tab showing the results of query 34. The results are as follows:

trip_id	time_period	average_rating
1	10005	282.66
2	10011	18.72
3	10016	135.95
4	10017	17.02
5	10020	42.81
6	10022	86.89
7	10038	254.17
8	10039	227.12
9	10042	275.19
10	10043	48.06
11	10056	298.44
12	10063	28.29

A green message bar at the bottom right says 'Successfully run. Total query runtime: 143 msec. 35 rows affected.'

35 Display all the customer details who traveled in a car having average ratings higher than 9.

```
SELECT CUSTOMER.*, CR.AVERAGE_RATING, RENTS.TRIP_ID
FROM RENTS
JOIN CAR_RECORD CR ON RENTS.CHASSIS_NUMBER = CR.CHASSIS_NUMBER
JOIN CUSTOMER ON CUSTOMER.CUSTOMER_ID = RENTS.CUSTOMER_ID
WHERE CR.AVERAGE_RATING > 9
ORDER BY RENTS.TRIP_ID
```



The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the database schema with tables like CUSTOMER, CAR_RECORD, and RENTS.
- Query Editor:** Displays the SQL query from step 35.
- Data Output:** Shows the results of the query, which are 19 rows of customer details and their average ratings.
- Message Bar:** At the bottom right, it says "Successfully run. Total query runtime: 164 msec. 19 rows affected."

customer_id	f_name	m_name	l_name	pass_word
1002	Rishi	Raj	Shivani	FZBzmH2DE
1028	Devansh	Girish	Vyas	3IAisBrHkS2
1002	Rishi	Raj	Shivani	FZBzmH2DE
1045	Sudhir	Ishan	Chaudhry	HiQfh
1066	Ishani	Hitesh	Benerjee	Wd6VPxhgpu
1008	Sarthak	Tarak	Patel	LShlteeo
1017	Mohit	Anil	Agrawal	ZelvzAh
1065	Priyal	Suresh	Raj	zcYk9Lgm7vq
1047	Ankit	Pradeep	Yadav	mxXldinsonV
1060	Harshil	Ramesh	Gandhi	jRfwIKZqCe
1050	Refiq	Ahmed		

36 Display branch name which has more than 5 vehicles.

```
SELECT BR.BRANCH_ID, BR.BRANCH_NAME
FROM BRANCH_RECORD BR
JOIN CAR_RECORD CR ON BR.BRANCH_ID = CR.BRANCH_ID
GROUP BY BR.BRANCH_ID
HAVING COUNT(*) > 8
ORDER BY BR.BRANCH_ID
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser Tab:** Shows the PostgreSQL connection information: carserv/postgres@PostgreSQL_10*.
- Query Editor:** Displays the SQL query from step 36.
- Data Output:** Shows the results of the query, which are:

branch_id	branch_name
1	2 Ghost
2	5 Mag
3	6 AWP
4	8 Scout
5	9 Kreig
- Status Bar:** Shows a green success message: "Successfully run. Total query runtime: 151 msec. 5 rows affected."

37 Create function which takes customer_id as input and returns customer details.

```
create or replace function customer_details ( a int )
returns table (
    fname character varying (30),
    mname character varying (30),
    lname character varying (30),
    customerid int
)
language plpgsql
as $$

begin
    return query
        select
            f_name,
            m_name,
            l_name,
            customer_id
        from customer
        WHERE customer_id = a;
end;
$$;
```

```
select * from customer_details(1001)
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays the database schema, including tables like 'customer', 'branch_record', and 'rents'. In the center, the 'Query Editor' pane shows the SQL code for creating a function named 'customer_details' that takes an integer parameter 'a' and returns a table of customer details. The function selects from the 'customer' table where the customer ID matches the input 'a'. Below the code, a 'Data Output' pane shows the result of executing the function with the argument '1001'. The result is a single row with columns: fname ('Aditya'), mname ('Arpit'), lname ('Shah'), and customerid (1001). A green success message at the bottom right indicates the query was run successfully with a runtime of 162 msec and 1 row affected.

fname	mname	lname	customerid
Aditya	Arpit	Shah	1001

Successfully run. Total query runtime: 162 msec. 1 rows affected.

38 Create function which takes two time periods as input and displays all rental records where the car was rented for more than the first time period and less than the second time period.

```
create or replace function tfun2 ( a float, b float )
  returns table (
    tripid int,
    customerid int,
    chassisnumber character varying(15),
    billamt float,
    timeperiod float
  )
language plpgsql
as $$

begin
  return query
    select
      trip_id,
      customer_id,
      chassis_number,
      bill,
      time_period
    from rents
    WHERE time_period between a AND b;
end;
$$;

select * from tfun2(100,200)
```

pgAdmin 4

Meet - jvn-bqtn-apy

IT214 - lab 10 - Google Docs

SQL Queries - Google Docs

127.0.0.1:4992/browser/#

Contest list Codeforces 2018 mail Classroom moodle Spotify ts

PgAdmin File Object Tools Help

Browser

Query Editor Query History

```

211      WHERE time_period between a AND b;
212      end;
213      $$;
214
215      select * from tfun2(100,200)
216
217  39  SELECT CR.LICENSE_PLATE_NUMBER, CR.CAR_TYPE, MR.BILL_AMOUNT
218  FROM CAR_RECORD CR
219  JOIN MAINTENANCE_RECORD MR ON CR.CHASSIS_NUMBER = MR.CHASSIS_NUMBER
220  WHERE MR.BILL_AMOUNT > 10000
  
```

Data Output Explain Messages Notifications

tripid	customerid	chassisnumber	billamt	timeperiod
			double precision	double precision
1	10016	1064 KNA0M4A434C6384112	2039.25	135.95
2	10023	1026 WAULV94E29NG44198	2612.4	174.16
3	10068	1029 3GTU2YEJ2DG087073	2360.55	157.37
4	10074	1021 WVGF9BP3BD667143	2597.7	173.18
5	10079	1023 SGALR8ED9AJ030833	929.46	132.78
6	10099	1073 WBANF33557B024819	1219.12	174.16
7	10115	1073 1GK51KE04ER874739	1223.55	135.95
8	10122	1061 1GD12ZCGXCF414458	1567.44	174.16
9	10138	1051 WPA1AE2A2XBL804045	951.65	135.95
10	10145	1080 SGALR8ED9AJ030833	1219.12	174.16
11	10150	1057 WVGF9BP6BD023894	1913.86	
12	10092	1018 WAUFFAFL3CA969738	951.65	

Successfully run. Total query runtime: 152 msec. 12 rows affected.

39. Display car records whose maintenance bill is greater than 10000.

```
SELECT CR.LICENSE_PLATE_NUMBER, CR.CAR_TYPE, MR.BILL_AMOUNT
FROM CAR_RECORD CR
JOIN MAINTENANCE_RECORD MR ON CR.CHASSIS_NUMBER =
MR.CHASSIS_NUMBER
WHERE MR.BILL_AMOUNT > 10000
```

The screenshot shows the pgAdmin 4 interface with a database browser on the left and a query editor on the right. The browser pane lists various tables and their columns. The query editor contains the SQL code provided above, with line numbers 39 and 40 highlighted. The results pane displays a table with 14 rows, each containing license plate number, car type, and bill amount. A message at the bottom right indicates the query was successfully run.

license_plate_number	car_type	bill_amount
UP72CL7509	SUV	13697.05
UP72DB5896	MINIVAN	10207.34
GJ35DK3679	MINIVAN	12593.95
MH56LR4890	SEDAN	12638.86
MH46RE2360	SEDAN	13123.05
UP78HC4859	HATCHBACK	15960.49
HP20AJ4923	SUV	13347.48
UP78DD5555	SUV	15779.46
HP13RT5544	SUV	15525.17
WB58LA8492	SEDAN	10183.91
HP92KA8320	HATCHBACK	12406.94
GA06HV9324	SUV	17420.91

Successfully run. Total query runtime: 179 msec. 14 rows affected.

40 Display the customer id and number of cars rented by that customer.

```
SELECT CUSTOMER.CUSTOMER_ID, CUSTOMER.F_NAME,
CUSTOMER.L_NAME,COUNT(*)
FROM CUSTOMER
JOIN RENTS ON RENTS.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
GROUP BY CUSTOMER.CUSTOMER_ID
ORDER BY CUSTOMER.CUSTOMER_ID
```

The screenshot shows the pgAdmin 4 interface with a query editor window. The browser pane on the left lists database objects like FTS Templates, Foreign Tables, Functions, Materialized Views, Sequences, and Tables (including branch_record, car_record, customer, etc.). The main window displays a query in the Query Editor:

```
219 FROM CAR_RECORD CR
220 JOIN MAINTENANCE_RECORD MR ON CR.CHASSIS_NUMBER = MR.CHASSIS_NUMBER
221 WHERE MR.BILL_AMOUNT > 10000
222
223 ④0 SELECT CUSTOMER.CUSTOMER_ID, CUSTOMER.F_NAME, CUSTOMER.L_NAME,COUNT(*)
224 FROM CUSTOMER
225 JOIN RENTS ON RENTS.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
226 GROUP BY CUSTOMER.CUSTOMER_ID
227 ORDER BY CUSTOMER.CUSTOMER_ID
```

The Data Output tab shows the results of the query:

customer_id	f_name	l_name	count	
1	1001	Aditya	Shah	2
2	1002	Rishi	Shivani	3
3	1003	Rushi	Patel	1
4	1005	Sanket	Jain	2
5	1006	Nishit	Jagetia	3
6	1007	Vedant	Thakkar	1
7	1008	Sarthak	Patel	3
8	1010	Bhayyesh	Ganatra	3
9	1011	Neel	Makadia	1
10	1012	Dhruvil	Bhatt	2
11	1013	Hemant	Jain	
12	1016	Mitesh	Koradia	

A green message bar at the bottom right indicates: "Successfully run. Total query runtime: 187 msec. 70 rows affected."