Candidate Keys for R1: (can alternatively be listed as FDs)

1. ContractId

2. {DriverName, DriverBirthday, VehicleModelNo, BeginDate}

3. {DriverName, DriverBirthday, VehicleModelNo, EndDate}

4. {DriverName, DriverBirthday, VehiclePlate, BeginDate}

5. {DriverName, DriverBirthday, VehiclePlate, EndDate}

6. {DriverLicenceNr, VehicleModelNo, BeginDate}

7. {DriverLicenceNr, VehicleModelNo, EndDate}

8. {DriverLicenceNr, VehiclePlate, BeginDate}

9. {DriverLicenceNr, VehiclePlate, EndDate}

Assumption here: a particular car can be rented by a particular driver only once a day; the driver can, however, rent more than one car, and the car can be rented by more than one driver on a certain day (e.g., if the first driver rents the car in the morning and returns it already at noon). The stronger assumption that a car can only be rented by exactly one driver on a certain day leads to a different set of candidate keys (the above attribute sets without the determinant for a driver).

Candidate Keys for R2

1. OfficeId

FDs on the initial R1 (CarRental): (without key constraints)

F1 : {DriverName, DriverBirthday} -> {DriverLicenceNr, DriverStreet, DriverCity, DriverZip, DriverCountry, DriverCustomerCategory,ExtraCharges, DiscountPercent}

F1 Violates 2NF ,3NF,BCNF

F2 : DriverLicenceNr -> {DriverLicenceNr, DriverStreet, DriverCity, DriverZip, DriverCountry, DriverCustomerCategory,ExtraCharges,DiscountPercent}

F2 Violates 2NF,3NF,BCNF

F3 : VehiclePlate -> {VehicleModelNo, VehicleBrand, VehicleCategory, VehicleNoSeats, VehicleHomeOffice, VehicleFreeKm, PricePerKm}

F3 Violates 2NF,3NF,BCNF

F4 : VehicleModelNo -> {VehiclePlate, VehicleBrand, VehicleCategory, VehicleNoSeats, VehicleHomeOffice, VehicleFreeKm, PricePerKm}

F4 Violates 2NF,3NF,BCNF

F5 : {VehicleBrand, VehicleCategory, VehicleNoSeats} -> {VehiclePricePerDay, VehicleFreeKm, PricePerKm}

F5 violates 3NF,BCNF

F6 : DriverCustomerCategory -> DiscountPercent

F6 violates 3NF,BCNF

F7 : {DriverCustomerCategory, DriverBirthday} -> ExtraCharges

F7 violates 3NF, BCNF

FDs on the initial R2 (Office):

F8 : OfficeRegionCode -> OfficeManager

F8 violates 3NF,BCNF

The highest/greatest normal form satisfied by R1 and R2 are 1NF and 2NF respectively.

Now, we begin to normalize R1 and R2 by a stepwise decomposition of relations "along functional dependencies". As a consequence, each of the decompositions will be loss-less join. Proof: When decomposing a relation R along a functional dependency A -> B (where $A \cap B = \emptyset$) into relations R1 = R \ B and R2 = AB, the intersection of R1 and R2 equals the left side of FD, i.e. A, and A will become the key of R2 since FD now holds over R2 and expresses a key constraint.

Step1: According to F1, decompose R1 into R3 and R4, where

R3= {ContratID , BeginDate, EndDate, Distance, FromOffice, ToOffice, DriverName, DriverBirthday, VehiclePlate,VehicleBrand, VehicleModelNo, VehicleCategory, VehicleNoSeats, VehicleHomeOffice, VehicleFreeKm, VehiclePricePerDay, PricePerKm}

R4={DriverName, DriverBirthday, DriverLicenceNr, DriverStreet, DriverCity, DriverZip, DriverCountry, DriverCustomerCategory, ExtraCharges, DiscountPercent}

Now, F3, F4, and F5 hold over R3; F3, and F4 violate 2NF,3NF and BCNF, F5 violates 3NF and BCNF.
F6, F7 hold over R4, violate 3NF and BCNF

The decomposition is loss-less join (see the general justification above).
The decomposition is also dependency preserving. Proof: The only functional dependencies that, after the decomposition, cannot be stated for any of the tables R3 and R4 are the key constraints for the table R1. It is sufficient to prove that these functional dependencies are contained in (F_R3 ∪ F_R4)+, where F_R3 and F_R4 are the projections of the functional dependencies on R3 and R4 respectively, and + denotes the closure of a set of functional dependencies.
The functional dependency ContractId->R1 is preserved since FD' = ContractId->R3 is in F_R3 and FD'' = {DriverName, DriverBirthday} -> R4 is in F_R4. Therefore, ContractId->R1 must be in (F_R3 ∪ F_R4)+: from FD' and FD'' it follows by transitivity that FD''' = ContractId -> R4 is in (F_R3 ∪ F_R4)+, and then, by applying the union rule to FD' and FD''' it follows that also (ContractId -> F3 ∪ F4), which is equal to ContractId->R1, must be in (F_R3 ∪ F_R4)+.
Since the above argument makes only use of the fact that the previous key for R1 is still a key for R3 after the decomposition, the argument applies to every key constraint on a decomposed table R if the decomposition along A -> B does not "destroy" the candidate key, i.e. if the candidate key for R is still a candidate key for R \ B after the decomposition (which is trivially true for a key that consist of only 1 attribute). For the given relation R1 it is also true for the key constraints corresponding to the candidate keys 2 to 5 for R1, they are preserved by analogous arguments.
We have to show that also the key constraints corresponding to the candidate keys 6 to 9 for R1 are in (F_R3 ∪ F_R4)+ (these candidate keys are not candidate keys for R3!). We show this for the key constraint {DriverLicenceNr, VehicleModelNo, BeginDate} -> R1, corresponding to candidate key 6 for R1; the proof for candidate keys 7 to 9 is completely analogous:
It holds that FD' = {DriverName, DriverBirthday, VehicleModelNo, BeginDate} -> R3 is in F_R3, and FD'' = {DriverLicenceNr} -> R4 is in F_R4. By decomposition of FD'', FD''' = {DriverLicenceNr} -> {DriverName, DriverBirthday} is also in F_R4. Then, FD'''' = {DriverLicenceNr, VehicleModelNo, BeginDate } ->
{DriverName, DriverBirthday, VehicleModelNo, BeginDate }
is in (F_R3 ∪ F_R4)+ by augmenting FD''' with {VehicleModelNo, BeginDate}. Then, by transitivity from FD'''' and FD' it follows that
(*) {DriverLicenceNr, VehicleModelNo, BeginDate} -> R3 is in (F_R3 ∪ F_R4)+.

By augmentation of FD'', it also holds that
(**) {DriverLicenceNr, VehicleModelNo, BeginDate} ->
R4 ∪ {VehicleModelNo, BeginDate} is in (F_R3 ∪ F_R4)+. Then by applying the union rule to (*) and (**): {DriverLicenceNr, VehicleModelNo, BeginDate} ->
R3 ∪ R4 ∪ {VehicleModelNo, BeginDate}
is in (F_R3 ∪ F_R4)+. Since R3 ∪ R4 ∪ {VehicleModelNo, BeginDate} = R1, {DriverLicenceNr, VehicleModelNo, BeginDate} -> R1 is in (F_R3 ∪ F_R4)+.
q.e.d.

Step 2: Decompose R4 into R5 and R6 along F6, we get:

R5= {<u>DriverName, DriverBirthday</u>, DriverLicenceNr, DriverStreet, DriverCity, DriverZip, DriverCountry, DriverCustomerCategory, ExtraCharges}

R6={<u>DriverCustomerCategory</u>, DiscountPercent}

F7 holds over R5, violates 3NF, R6 satisfies BCNF.

The decomposition is loss-less join (see the general justification above).
The decomposition is dependency preserving: the only FDs, which can no longer be stated over a single relation, are the key constraints for R4. The decomposition does not destroy their key property for relation R5. Hence, by the arguments given in step 1, the key constraints {DriverName, DriverBirthday} -> R4, and {DriverLicenceNr } -> R4 are in (F_R5 ∪ F_R6)+.

Step3: Decompose R5 into R7 and R8 along F7, we get:

R7={<u>DriverName, DriverBirthday</u>, DriverLicenceNr, DriverStreet, DriverCity, DriverZip, DriverCountry, DriverCustomerCategory}

R8={DriverBirthday,DriverCustomerCategory, ExtraCharges)

Both R7 and R8 satisfy BCNF.

The decomposition is loss-less join (see the general justification above).
The decomposition is dependency preserving: the only FDs, which can no longer be stated over a single relation, are the key constraints for R5. By the arguments given in step 1, the key constraints {DriverName, DriverBirthday} -> R5, and {DriverLicenceNr } -> R5 are in (F_R7 ∪ F_R7)+.

Step4: Decompose R3 into R9 and R10 along F3, we get:

R9={ContratID , BeginDate, EndDate, Distance, FromOffice, ToOffice, DriverName, DriverBirthday, VehiclePlate}

R10={VehiclePlate,VehicleBrand,VehicleModelNo, VehicleCategory, VehicleNoSeats, VehicleHomeOffice, VehicleFreeKm, VehiclePricePerDay, PricePerKm}

R9 satisfies BCNF.
F5 holds on R10, violates 3NF and BCNF.

The decomposition is loss-less join and dependency preserving (by similar arguments as above).

Step5: Decompose R10 into R11 and R12 along F5, we get:

R11 ={VehiclePlate,VehicleBrand,VehicleModelNo, VehicleCategory, VehicleNoSeats, VehicleHomeOffice}
R12={VehicleBrand, VehicleCategory, VehicleNoSeats VehicleFreeKm, VehiclePricePerDay, PricePerKm}

Both R11 and R12 satisfy BCNF.

The decomposition is loss-less join and dependency preserving (by similar arguments as above).

Step6: Decompose R2 into R13 and R14 along F8, we get:

R13={officeID ,OfficeId, OfficeStreet, OfficeCity, OfficeZip, OfficeCountry, OfficePhone, OfficeFax, OfficeRegionCode}

R14={OfficeRegionCode OfficeManager }

The decomposition is loss-less join and dependency preserving (by similar arguments as above).

So the **final result** is:

R9= {ContratID , BeginDate, EndDate, Distance, FromOffice, ToOffice,
DriverName, DriverBirthday, VehiclePlate}
Foreign key FromOffice,ToOffice *references* R13
Foreign key DriverName,DriverBirthday r*eferences* R7.
Foreign key VehiclePlate r*eferences* R11.

R7={DriverName, DriverBirthday, DriverLicenceNr, DriverStreet, DriverCity,
DriverZip, DriverCountry, DriverCustomerCategory}
Foreign key DriverCustomerCategory *references* R6.

R8={DriverBirthday,DriverCustomerCategory, ExtraCharges)

R6={DriverCustomerCategory, DiscountPercent}

R11={VehiclePlate,VehicleBrand,VehicleModelNo, VehicleCategory,
VehicleNoSeats, VehicleHomeOffice}
Foreign key VehicleHomeOffice *references* R13

R12={VehicleBrand, VehicleCategory, VehicleNoSeats VehicleFreeKm,
VehiclePricePerDay, PricePerKm}

R13={officeID ,OfficeId, OfficeStreet, OfficeCity, OfficeZip, OfficeCountry,
OfficePhone, OfficeFax, OfficeRegionCode}
Foreign Key OfficeRegionCode r*eferrences* R14

R14={OfficeRegionCode, OfficeManager}