**Project team 3 - Motor-Vehicle Rental Service System**

**Team Members: Ashwin Venkatesh Prabhu, Ashwini Kadam, Koushik Kashojjula, Salman Mujtaba**

***PROJECT PROPOSAL***

**Content, Scope and Objectives**

Motor-Vehicle Rental Service System is a database-driven application which helps in maintaining the data for a motor-vehicle rental store. Registered users have the option to rent a vehicle of their choice. The basic functionalities of this system includes, allowing the user to view the vehicle availability for the duration selected, option for self-drive or chauffeur, option for pickup or delivered. The systems administrator should be able to monitor all the activities (future bookings, current bookings, past bookings, cancellations, etc.) on the system.

***PROJECT ENVIRONMENT***

➔ Visual Studio 2015 using C# and WinForm application to create the UI.

➔ Project hosted locally.

➔ MySQL workbench 6.3 for back-end database.

***HIGH LEVEL REQUIREMENTS***

**Initial user roles**

1) Registered User: A user who has a profile created in the system and is eligible to book a car.

2) Admin: A user who maintains the system.

**Initial user story descriptions**

1) As an Admin, I want to log in to the system.

2) As an Admin, I want to logout of the system.

3) As a Registered User, I want to log in to the system.

4) As a Registered User, I want to logout of the system.

5) As a Registered User, I want to check the availability of cars so that I can book one.

6) As a Registered User, I want to book a car.

7) As a Admin, I want to change car availability.

***HIGH LEVEL CONCEPTUAL DESIGN***

Entity:

1) Registered User

2) Vehicle

3) Admin

*Registered user* books a *Vehicle*

*Admin* manages the *Vehicle*

***SPRINT 1***

User Stories:

1. As a user, I want to sign up/create an account
2. As an admin, I want to log in to the system
3. As an admin, I want to logout of the system
4. As an admin, I want to add new vehicles to the system
5. As an admin, I want to delete vehicles from the system
6. As a registered user, I want to log in to the system
7. As a registered user, I want to logout of the system
8. As a registered user, I want to view the available vehicles

Notes:

1. The tables required for the above user stories are User and vehicle.
2. User table is required for user stories 1, 2, 5, 6 and 7
3. Vehicle table is required for user stories 3, 4 and 8
4. ‘User’ is the entity here and ‘admin’ and ‘registered user’ are specialisation to that entity

**Conceptual design**

Entity: **User**

Attributes:

user\_id

name [composite]

last\_name

first\_name

date\_of\_birth

license\_no

address [composite]

Address\_line\_1

Address\_line\_2

City

State

zip\_code

email

contact [multi-valued]

Entity: **VehicleDetails**

Attributes:

vehicle\_id

vehicle\_number

registration\_no

model

year

color

seat\_capacity

is\_available

cost\_per\_day

cost\_per\_mile

category

conditioned

Entity: **UserCredentials**

Attributes:

user\_name

password

type\_of\_user

Relationship: **Admin** adds to **VehicleDetails**

Cardinality: One to many

Participation:

Admin has partial participation

VehicleDetails has total participation

Relationship: **Admin** deletes from **VehicleDetails**

Cardinality: One to many

Participation:

Admin has partial participation

VehicleDetails has total participation

**Logical design**

Table: **user**

Columns:

user\_id

last\_name

first\_name

date\_of\_birth

license\_no

address\_line\_1

address\_line\_2

city

state

zi\_code

email

contact\_number\_1

contact\_number\_2

Table: **vehicledetails**

Columns:

vehicle\_id

vehicle\_number

registration\_no

model

year

color

seat\_capacity

is\_available

cost\_per\_day

cost\_per\_mile

category

conditioned

Table: **usercredentials**

Columns:

user\_name

password

user\_id [foreign key; references to user\_id of user table]

type\_of\_user

*Justification* : We are using **foreign key approach** to represent *‘User logs into the system’* scenario. This is one to one relationship between **user** and **usercredentials** - **usercredentials** will store only user login information, and user table stores all the other info related to the user.

***SPRINT 2***

User Stories:

1. As an admin, I want to update vehicles details like cost per mile, cost per day, conditioned, availability of vehicle, etc.
2. As a registered user, I want to view available vehicles for chosen future date
3. As a registered user, I want to view available vehicles at a particular pickup location
4. As a registered user, I want to book/reserve a vehicle for a particular date
5. As a registered user, I want to update my profile details
6. As a registered user, I want to add card information
7. As a registered user, I want to update card information
8. As an admin, I want to view all the user information

**Conceptual design**

Entity: **User**

Attributes:

user\_id

name [composite]

last\_name

first\_name

date\_of\_birth

license\_no

address [composite]

address\_line\_1

address\_line\_2

city

state

zip\_code

email

contact [multi-valued]

Entity: **VehicleDetails**

Attributes:

vehicle\_id

vehicle\_number

registration\_no

model

year

color

seat\_capacity

is\_available

cost\_per\_day

cost\_per\_mile

category

conditioned

Entity: **UserCredentials**

Attributes:

user\_name

password

type\_of\_user

Entity: **PickupLocation**

Attributes:

location\_id

address

state

zipcode

phone\_number

email

Entity: **CardDetails**

Attribute:

user\_id [foreign key references to user\_id from user table]

card\_no

name\_on\_card

expiry\_date

cvv

billing\_address

Relationship: **Registered\_user** reserves **Vehicle(VehicleDetails)**

Cardinality: One to Many

Participation:

Registered\_user has partial participation

Reservation has partial participation

Relationship: **User** has **UserCredentials**

Cardinality: One to One

Participation:

**User** has total participation

**UserCredentials** has total participation

Relationship: **User** has **CardDetails**

Cardinality: One to Many

Participation:

**User** has partial participation

**CardDetails** has total participation

**Logical design**

Table: **user**

Columns:

user\_id

last\_name

first\_name

date\_of\_birth

license\_no

address\_line\_1

address\_line\_2

zip\_code

email

contact\_number\_1

contact\_number\_2

Highest normalization level: 4NF

Table: **zipcodes**

Columns:

zipcode

city

State

Highest normalization level: 4NF

Table: **vehicledetails**

Columns:

vehicle\_id

vehicle\_number

registration\_no

model

year

color

seat\_capacity

is\_available

cost\_per\_day

cost\_per\_mile

category

conditioned

location\_id [foreign key reference from pickup locations]

Highest normalization level: 4NF

Table: **usercredentials**

Columns:

user\_name

password

user\_id [foreign key; references to user\_id of user table]

type\_of\_user

*Justification* : We are using **foreign key approach** to represent *‘User logs into the system’* scenario. This is one to one relationship between **user** and **usercredentials** - **usercredentials** will store only user login information, and user table stores all the other info related to the user.

Highest normalization level: 4NF

Table: **pickuplocation**

Columns:

location\_id

address

zipcode [Foreign key references to zipcode from zipcodes table]

phone\_number

email

Highest normalization level: 4NF

Table: **reservation**

Columns:

reservation\_id

start\_date

end\_date

rental\_amount

deposit\_amount

user\_id [foreign key references to user\_id of user table]

vehicle\_id [foreign key references to vehicle\_id from vehicledetails table]

total\_amount\_paid

location\_id [foreign key references to location\_id from pickuplocation]

Highest normalization level: 4NF

Table: **carddetails**

Columns:

user\_id [foreign key references to user\_id of user table]

card\_no

name\_on\_card

expiry\_date

cvv

address\_line\_1

address\_line\_2

Zipcode\_cd [foreign key references to zipcode of zipcodes table]

Highest normalization level: 4NF

***SPRINT 3***

User Stories:

1. As a registered user, I want to pay for reservation using my stored card details
2. As a registered user, I want to view my booking history
3. As an admin, I want to add offer discounts to vehicle
4. As a registered user, I want to purchase insurance for the booking

**Conceptual design**

Entity: **User**

Attributes:

user\_id

name [composite]

last\_name

first\_name

date\_of\_birth

license\_no

address [composite]

address\_line\_1

address\_line\_2

city

state

zip\_code

email

contact [multi-valued]

Entity: **VehicleDetails**

Attributes:

vehicle\_id

vehicle\_number

model

year

color

seat\_capacity

is\_available

cost\_per\_day

cost\_per\_mile

category

conditioned

Entity: **UserCredentials**

Attributes:

user\_name

password

type\_of\_user

Entity: **PickupLocation**

Attributes:

location\_id

address\_line\_1

address\_line\_2

zipcode

phone\_number

email

Relationship: **Registered\_user** reserves **Vehicle(VehicleDetails)**

Cardinality: One to Many

Participation:

**Registered\_user** has full participation

**Vehicle** has full participation

Relationship: **User** has **UserCredentials**

Cardinality: One to One

Participation:

**User** has total participation

**UserCredentials** has total participation

Relationship: **Vehicle (VehicleDetails)** has **PickupLocation**

Cardinality: Many to One

Participation:

**Vehicle (VehicleDetails)** has total participation

**PickupLocation** has partial participation

**Logical design**

Table: **user**

Columns:

user\_id

last\_name

first\_name

date\_of\_birth

license\_no

address\_line\_1

address\_line\_2

zip\_code[foreign key references to zipcode attribute of zipcode table]

email

contact\_number\_1

contact\_number\_2

Highest normalization level: 4NF

Index on: **user\_id**

Type: **Clustered**

Justification: In order to search for any user of the system, it makes sense to make use of the user\_id since it uniquely identifies the user.

The index is clustered because it depends upon the sorted ordering of user\_id. Also, it is highly recommended that we create indices on columns of a table from which you often query the data.

Index on: **zipcode**

Type: **Unclustered**

Justification: Zipcode is queried to get the city and state part of the address of the user. So, we created an unclustered index.

Table: **zipcodes**

Columns:

zipcode

city

State

Highest normalization level: 4NF

Index on: **zipcode**

Type: **Clustered**

Justification: In order to search for any city or state, the zipcode would always be queried first. It makes sense to create a clustered index on zipcode.

Table: **insurance**

Columns:

insurance\_type

bodily\_coverage

medical\_coverage

collision\_coverage

insurance\_price

Justification:

The above table stores the insurance information pertaining to an insurance type. As part of our users tories, the insurance is required for a booking of a vehicle so, the table is linked to the vehicledetails through reservation. Yet again, the separate table is there just to make the database simpler and attain highest normal forms of the related tables.

Highest normalization level: 4NF

Index on: **insurance\_type**

Type: **Clustered**

Justification: Anything related to the insurance in our model, is fetched via the insurance\_type. So, it makes sense to have insurance\_type as a clustered index as it matches the order of the file.

Table: **discountdetails**

Columns:

promo\_code

description

discount\_percentage

Justification: Discount is offered for a particular promo code. This is reflected in the total amount paid for the reservation. The above table(discountdetails) is created to keep the other related tables in their highest normal form.

Highest normalization level: 4NF

Index on: **promo\_code**

Type: **Clustered**

Justification: The discount is offered via a promocode which is the primary key here. So, it makes sense to create an index for promo\_code here

Table: **vehicledetails**

Columns:

vehicle\_id

vehicle\_number

model

year

color

seat\_capacity

is\_available

cost\_per\_day

cost\_per\_mile

category

conditioned

location\_id [foreign key reference to location\_id of pickup locations]

Highest normalization level: 4NF

Index on: **vehicle\_id**

Type: **Clustered**

Justification: The justification is pretty much similar. In our project, we often search for any vehicle by using vehicle\_id. So we create a clustered index on vehicle\_id.

Index on: **location\_id**

Type: **Non-Clustered**

Justification: We query the vehicledetails table to get the vehicle availability according to the vehicle location. Hence, it makes sense to create an index over the location\_id.

Table: **usercredentials**

Columns:

user\_name

password

user\_id [foreign key; references to user\_id of user table]

type\_of\_user

*Justification* : We are using **foreign key approach** to represent *‘User logs into the system’* scenario. This is one to one relationship between **user** and **usercredentials** - **usercredentials** will store only user login information, and user table stores all the other info related to the user.

Highest normalization level: 4NF

Index on: **user\_name**

Type: **Clustered**

Justification: user\_name is required every time the user wants to log into the system. Also, we might want to search for the user by it’s name.

Index on: **user\_id**

Type: **Non-Clustered**

Justification: In order to identify a user, we fetch the user\_id of the usercredentials table. Since we have queried the user\_id many times, it is feasible to create a non-clustered index on user\_id.

Table: **pickuplocation**

Columns:

location\_id

address\_line\_1

address\_line\_2

zipcode [Foreign key references to zipcode from zipcodes table]

phone\_number

email

Justification: We need a separate table that tracks all the available locations. A vehicle may or may not be present at the location but the location info must be available

Highest normalization level: 4NF

Index on: **location\_id**

Type: **Clustered**

Justification: In order to search for any location, we need to query the location id. Therefore, a clustered index is created on the location\_id.

Index on: **zipcode**

Type: **Non Clustered**

Justification: The city and state of the pickup location is fetched for a location using the zipcode. Hence, we can create a non-clustered index on the zipcode.

Table: **reservation**

Columns:

reservation\_id

start\_date

end\_date

rental\_amount

deposit\_amount

total\_amount\_paid

user\_id [foreign key references to user\_id of user table]

vehicle\_id [foreign key references to vehicle\_id from vehicledetails table]

insurance\_type [foreign key references to insurance\_type from insurance table]

insurance\_amount

promo\_code [foreign key references to promo\_code of discountdetails table]

*Justification* : This is table is used to represent the relationship between **Registered**\_**user** reserves **Vehicle(as VehicleDetails)**.

Highest normalization level: 2NF

Justification:As the reservation table holds the information related to the reservation, it makes sense to include everything related to the reservation in one table. Note that there would not really be any insertion, deletion or updation anomalies because all the important things are derived in this table. On the contrary, if we try and split the tables, it would simply act as unnecessary overheads. Therefore, we’d like to keep it in 2NF.

Index on: **start\_date, end\_date, vehicle\_id**

Type: **Clustered**

Justification: In order to search for any reservation, we need to query the start date, end date and vehicle id. Therefore, a clustered index is created on the these three primary keys.

Index on: **user\_id**

Type: **Non-Clustered**

Justification: It is the user who makes a reservation and in order to address that reservation, we user the user’s user\_id. Therefore, we created a non-clustered index over the user\_id.

Index on: **vehicle\_id**

Type: **Non-Clustered**

Justification: The reservation table is all about storing the booked vehicle corresponding to a user\_id. The booked vehicle is queried through the vehicle\_id hence we created a non clustered index over the vehicle\_id.

Index on: **promo\_code**

Type: **Non-Clustered**

Justification: Promo code is used to calculate the discount being offered. Hence, we created a non clustered index.

Table: **carddetails**

Columns:

card\_id

user\_id [foreign key references to user\_id of user table]

card\_no

name\_on\_card

expiry\_date

cvv

address\_line\_1

address\_line\_2

zipcode\_cd[foreign key references to the column zipcode of zipcode table]

Highest normalization level: 4NF

Index on: **card\_id**

Type: **Clustered**

Justification: The card id identifies the card of the user. We use the card id wherever the card is involved. So, we created an index over the card\_id which is a clustered one.

Index on: **zipcode\_cd**

Type: **Non-Clustered**

Justification: The zipcode\_cd identifies the zip code, state, and city where the card of the user is registered. The zip code will be used whenever a user would add a card for the payment. Therefore, we created a non clustered index over zipcode\_cd.

### **Stored programs**

**Stored procedure**: LoginUser

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Parameter Type** | **Variable Name** | **Variable Type** |
| IN | usrname | VARCHAR(45) |
| IN | psswd | VARCHAR(45) |

**Goal:** To login the user into the system using the username and password entered by the user.

CREATE PROCEDURE `LoginUser`(IN usrname VARCHAR(45), IN passwd VARCHAR(45))

BEGIN

SELECT type\_of\_user

FROM usercredentials where user\_name

LIKE usrname and password like passwd;

END

**Stored procedure**: ViewAvailableVehicles

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Parameter Type** | **Variable Name** | **Variable Type** |
| IN | locationid | VARCHAR(45) |
| IN | startdate | DATE |
| IN | enddate | DATE |

**Goal:** To get the vehicles available for reservation during the start date and end date entered by the user. The if conditions takes care of the fact whether we want to view the vehicles of a specific location or simply view all the vehicles.

CREATE PROCEDURE `ViewAvailableVehicles`(IN locationid VARCHAR(45), IN startdate DATE, IN enddate DATE)

BEGIN

IF locationid = "NOLOCATION" THEN

SELECT DISTINCT model, seat\_capacity,

cost\_per\_day, cost\_per\_mile,

category, conditioned,

location\_id, vehicle\_id FROM vehicledetails

WHERE vehicle\_id NOT IN (select vehicle\_id from reservation where start\_date = startdate AND end\_date = enddate);

ELSE

SELECT DISTINCT model, seat\_capacity, cost\_per\_day, cost\_per\_mile, category,

conditioned, location\_id, vehicle\_id

FROM vehicledetails

WHERE location\_id = locationid AND

vehicle\_id NOT IN (select vehicle\_id from reservation WHERE start\_date = startdate AND end\_date = enddate);

END IF;

END

**Stored procedure**: DeleteVehicle

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Parameter Type** | **Variable Name** | **Variable Type** |
| IN | vehicleid | INT(11) |

**Goal:** To delete the vehicle info from the vehicledetails table.

CREATE PROCEDURE `DeleteVehicle`(IN vehicleid INT(11))

BEGIN

DELETE FROM vehicledetails WHERE vehicle\_id = vehicleid;

END

**Stored procedure**: UpdateVehicle

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Parameter Type** | **Variable Name** | **Variable Type** |
| IN | mdl | VARCHAR(10) |
| IN | yr | INT(5) |
| IN | clr | VARCHAR(45) |
| IN | seatcapacity | INT(11) |
| IN | isavailable | ENUM(‘yes,’no’) |
| IN | costperday | INT(11) |
| IN | costpermile | INT(11) |
| IN | ctgry | VARCHAR(50) |
| IN | contnd | ENUM(‘ac’,’non-ac’) |
| IN | vehclid | INT(11) |

**Goal:** The procedure is used to update vehicle details. This can be used when the admin logs in.

CREATE PROCEDURE `UpdateVehicle`(

IN mdl VARCHAR(10), IN yr INT(5), IN clr VARCHAR(45),

IN seatcapacity INT(11), IN isavailable ENUM('yes', 'no'),

IN costperday INT(11), IN costpermile INT(11),

IN ctgry VARCHAR(50), IN cndtnd ENUM('ac', 'non-ac'),

IN vehclid INT(11))

BEGIN

UPDATE vehicledetails SET

model = mdl,

year = yr,

color = clr,

seat\_capacity = seatcapacity,

is\_available = isavailable,

cost\_per\_day = costperday,

cost\_per\_mile = costpermile,

category = ctgry,

conditioned = cndtnd

WHERE vehicle\_id = vehclid;

END

**Stored procedure**: **BookVehicle**

**Parameters**:

|  |  |  |
| --- | --- | --- |
| **Parameter Type** | **Variable Name** | **Variable Type** |
| IN | startdate | DATE |
| IN | enddate | DATE |
| IN | rentalamount | INT(11) |
| IN | depositamount | INT(11) |
| IN | totalamountpaid | INT(11) |
| IN | vehicleid | INT(11) |
| IN | userid | INT(11) |
| IN | promocode | VARCHAR(15) |
| IN | insurancetype | VARCHAR(15) |

**Goal:** This procedure is called when a vehicle is reserved by a user.

CREATE PROCEDURE `BookVehicle`(IN startdate DATE, IN enddate DATE, IN rentalamount INT(11), IN depositamount INT(11),

IN totalamountpaid INT(11), IN vehicleid INT(11), IN userid INT(11), IN promocode VARCHAR(15), IN insurancetype VARCHAR(15))

BEGIN

INSERT INTO reservation (start\_date, end\_date, rental\_amount, deposit\_amount, total\_amount\_paid, vehicle\_id, user\_id, promo\_code, insurance\_type)

VALUES (startdate, enddate, rentalamount, depositamount, totalamountpaid, vehicleid, userid, promocode, insurancetype);

END

**Stored Procedure: ViewBookingHistory**

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Parameter Type** | **Variable Name** | **Variable Type** |
| IN | userid | INT(11) |

**Goal:** Procedure to view the booking history.

CREATE PROCEDURE `ViewBookingHistory`(IN userid INT(11))

BEGIN

SELECT start\_date, end\_date, total\_amount\_paid, vehicle\_id

FROM reservation

WHERE user\_id = userid

ORDER BY start\_date DESC;

END

**Stored Function :** GetInsuranceAmount

**Parameters:**

|  |  |
| --- | --- |
| **Variable Name** | **Variable Type** |
| insurancetype | VARCHAR(15) |
| startdate | DATE |
| enddate | DATE |

**Goal:** The function is used to obtain the insurance amount for the car.

CREATE FUNCTION `GetInsuranceAmount`(insurancetype VARCHAR(15), startdate DATE, enddate DATE) RETURNS float(8,2)

BEGIN

DECLARE insuranceamount FLOAT(8, 2);

SELECT (insurance\_price \* datediff(enddate, startdate)) INTO insuranceamount

FROM insurance

WHERE insurance\_type = insurancetype;

RETURN insuranceamount;

END

**Stored Function :** GetDiscountPercentage

**Parameters:**

|  |  |
| --- | --- |
| **Variable Name** | **Variable Type** |
| promo | VARCHAR(15) |

**Goal:** The function is used to obtain the discount percentage for the car.

CREATE FUNCTION `GetDiscountPercentage`(promo VARCHAR(15)) RETURNS int(11)

BEGIN

DECLARE discount INT(11);

SELECT discount\_percentage

INTO discount

FROM discountdetails

WHERE promo\_code = promo;

IF discount = NULL

THEN SET discount = 0;

END IF;

RETURN discount;

END

**Trigger :**

**Goal:** Update total\_amount\_paid in ‘reservation’ table

DROP TRIGGER IF EXISTS before\_reservation\_insert;

DELIMITER $$

CREATE TRIGGER before\_reservation\_insert

BEFORE INSERT ON reservation

FOR EACH ROW

BEGIN

SET NEW.insurance\_amount = GetInsuranceAmount(NEW.insurance\_type, NEW.start\_date, NEW.end\_date),

NEW.total\_amount\_paid = NEW.rental\_amount \* (100 - GetDiscountPercentage(NEW.promo\_code)) + NEW.deposit\_amount + NEW.insurance\_amount;

END $$

DELIMITER ;