Wolf Parking Management System

For Wolf Parking

CSC 540 Database Management Systems
Project Report 3
Team CC

Daniel Buchanan(dsbuchan), Ophelia Sin(oysin), Aadil Tajani(atajani), Manali Teke(mteke)
November 16th, 2023

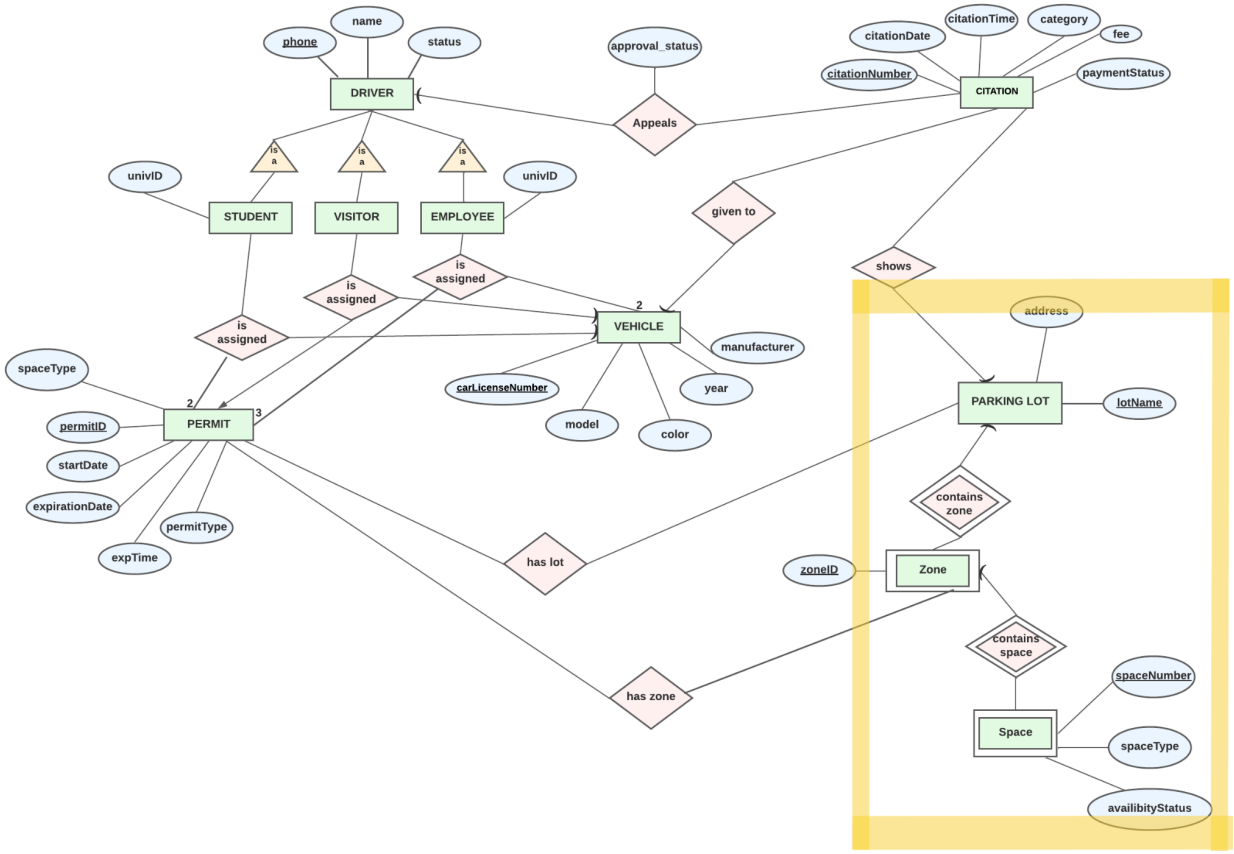
Revisions in Report 1

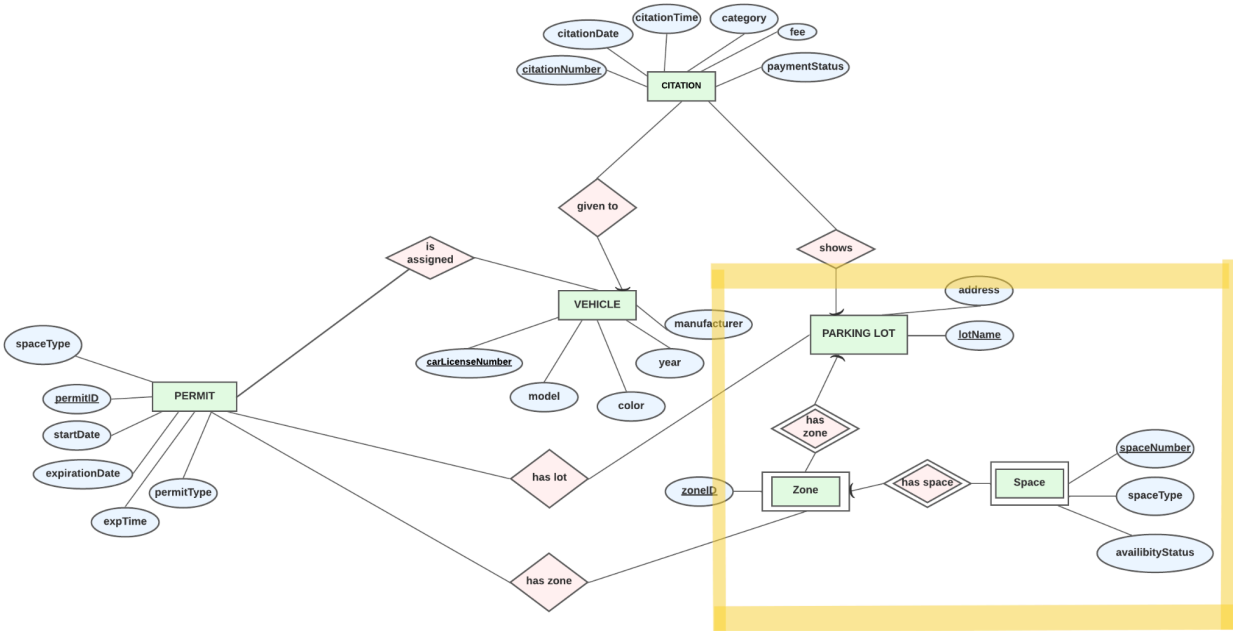
1. Q7. Local ER Diagrams (Pg. 8, Pg.9) - *Redundancy or impractical design*

Original Report Page: 8, 9

Changes:

1. Changed the Parking Lot-Zone-Space design in all views. Parking Lots have zones and each zone has spaces.

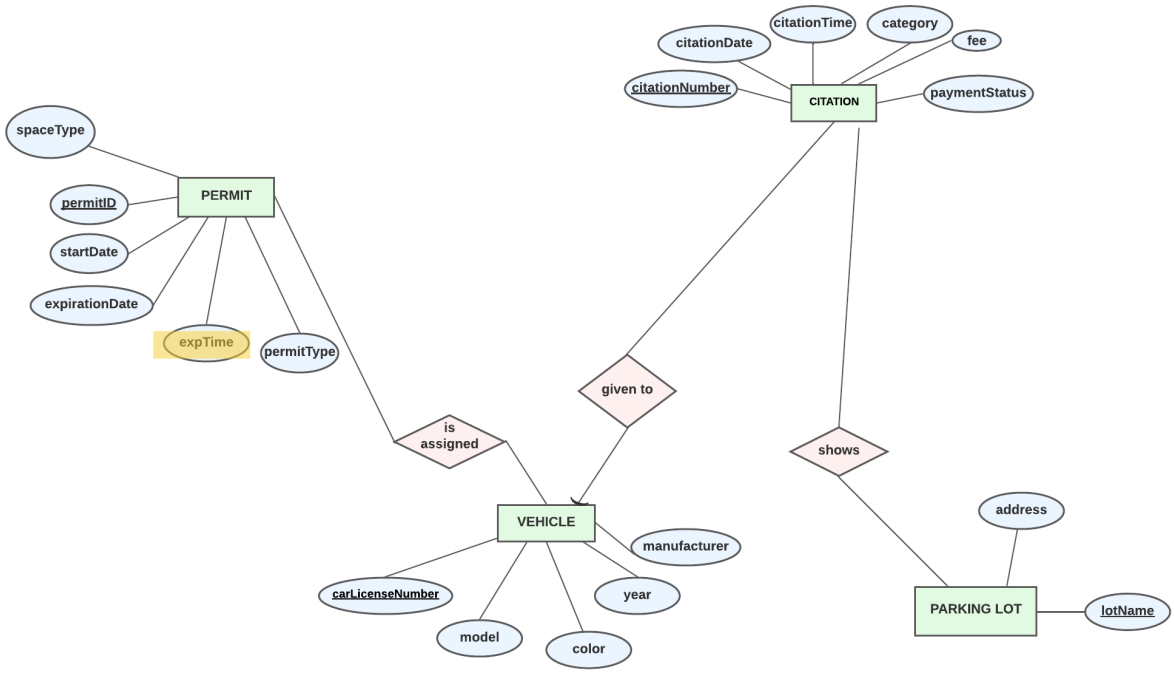




ParkingLot(<u>lotName</u>, address)
Zone(<u>zoneID</u>, <u>lotName</u>)
Space(<u>spaceNumber</u>,<u>zoneID</u>, <u>lotName</u>, spaceType, availabilityStatus)

2. Q9. Local Relational Schemas (Pg.9(edits), 11) - *Missing attributes from E/R diagram*

Original Report Page: 9, 11



Revisions in Report 2

1. Q1 Relational schemas - 3NF Does not discuss ALL non trivial FD that the instructor can reasonably assume might hold on your schemas (For ONE schema)
Original Report Page: 4

Changes: Added citationNumber → phone, appealStatus non trivial FD

Appeals(phone, citationNumber, appealStatus)
citationNumber, phone → citationNumber, phone, appealStatus holds because the citationNumber and phone make up the superkey of the relation and it identifies appeal status as well . Because they make up the superkey of the relation, this relation is in BCNF and therefore in 3NF. citationNumber → phone, appealStatus holds because given citationNumber we can determine the driver that was cited. This is true as each citationNumber is assigned with exactly one phone number.

2. Q3. Missing multiple integrity constraints as listed in Q2 - *Null Constraints do not match as mentioned in Q2*
Original Report Page: 7

Changes: Added NULL constraints to category, fee, and payment_status attributes in Citation table on Pg7

```
CREATE TABLE Citation (
    citation_number INT,
    citation_date DATE NOT NULL,
    citation_time TIME NOT NULL,
    category VARCHAR(128) NOT NULL,
    fee FLOAT(9,2) NOT NULL,
    payment_status BOOLEAN NOT NULL,
    PRIMARY KEY (citation_number)
);
```

3. **Q3. Missing multiple integrity constraints as listed in Q2 (Pg9) - *Foreign Key constraints do not match as mentioned in Q2***

Original Report Page: 6, 9

Changes: Added foreign key constraints in Q2 on Pg. 6. And as they form a primary key together and are foreign keys, we don't need to mention NOT NULL in Q3 as that enforces NOT NULL by itself.

HasZone(permitID, zoneId, lotName)

permitId, zoneID, and lotName referenced together are primary keys. zoneID, and lot_name are foreign key referenced from tables Zone and ParkingLot.

4. **Q3. Missing multiple integrity constraints as listed in Q2 (Pg9) - *Null constraints do not match as mentioned in Q2.***

Original Report Page: 6, 9

Changes: Added foreign key constraints in Q2 on Pg. 6. And as they form a primary key together and are foreign keys, we don't need to mention NOT NULL in Q3 as that enforces NOT NULL by itself.

GivenTo(citationNumber, carLicenseNumber)

citationNumber and carLicenseNumber referenced together form the primary key. citationNumber and carLicenseNumber are foreign key referenced from tables Citation and Vehicle

5. **Q4.1. One Partially correct query (Pg18) - The query should check for lot, zone, space and expired permit violations.**

Original Report Page: 17-18

Changes: Updated the query to also check expired permits on Pg 18.

- *Detect parking violations by checking if a car has a valid permit in the lot,zone and space.*

Note: The query returns tuples of parking lots,zones and spaces where the vehicle has a valid parking permit and also the permit expiration date and time. If the vehicle is not parked in the mentioned Lot, Zone and SpaceType or expired permit, it is a parking violation.

```
■    SQL> SELECT P.permit_id, space_type, expiration_date,
expiration_time, H.zone_id, H.lot_name FROM  IsAssigned IA JOIN
Permit P ON IA.permit_id = P.permit_id JOIN HasZone H ON
H.permit_id = P.permit_id WHERE NOT EXISTS(SELECT H.lot_name,
H.zone_id, P.space_type FROM HasZone H JOIN Permit P ON
H.permit_id = P.permit_id WHERE H.permit_id IN (     SELECT
permit_id     FROM IsAssigned     WHERE car_license_number =
'PQR987' )AND lot_name = 'Poulton'  AND zone_id = 'A' AND
space_type = 'Handicap' AND (CURRENT_TIMESTAMP < expiration_date
OR CURRENT_TIMESTAMP = expiration_date AND CURRENT_TIMESTAMP <
expiration_time)) AND car_license_number = 'PQR987';
+-----------+-----------+-----------------+-----------------+----
-----+----------+
| permit_id | space_type | expiration_date | expiration_time |
zone_id | lot_name |
+-----------+-----------+-----------------+-----------------+----
-----+----------+
|         6 | Handicap   | 2023-11-11      | 10:30:00        | A
| Oval     |
+-----------+-----------+-----------------+-----------------+----
-----+----------+
1 row in set (0.0069 sec)
```

6. Q4.1. Missing Operation (Pg 19) - Maintain a Citation

Original Report Page: 18-19

Changes: Added operation maintain a citation which can update all attributes of citation.

Maintain a citation

SQL> UPDATE Citation set category = 'Expired Permit' WHERE citation_number = 107;
Query OK, 1 row affected (0.0056 sec)

Rows matched: 1 Changed: 1 Warnings: 0

SQL> UPDATE Citation set citation_time = '10:23:00' WHERE citation_number = 108;
Query OK, 1 row affected (0.0042 sec)

Rows matched: 1 Changed: 1 Warnings: 0

SQL> UPDATE Citation set payment_status = 1 WHERE citation_number = 102;
Query OK, 1 row affected (0.0034 sec)

Rows matched: 1 Changed: 1 Warnings: 0

7. Q4.3. 1st Relational Algebra Expression Minor Mistake (Pg.25)
Original Report Page: 25
Changes: Changed the relational algebra query, applied delta to lot_name after projection

Specification: "Return the single name of all parking lots that have permits with a 'residential' permit type."

Relational Algebra:
$\delta(\pi_{(lot\_name)}(\sigma_{(permit\_type = 'residential')}(HasLot \bowtie Permit)))$

Assumptions:

1. Phone Number is unique for all drivers.
2. Student and Employee drivers will have a Phone Number.
3. Space Number is unique within a Parking Lot and a Zone is unique within a Parking Lot.
4. Parking Lot name is unique.
5. Parking manager is one of the administrators.
6. Any permit with a particular space type can park in a regular space. However, to access non regular space types, drivers have to hold a permit for that specific space type. For example, only a driver with a handicap parking permit can park in a handicap space.
7. A parking lot can contain multiple zones. A driver can park in the specified parking lot and zone in their permits.
8. A parking lot can have multiple types of spaces.
9. Each permit can have multiple lots and zones.
10. Administrators manage the billing processes.
11. On creation of a citation, payment status is set to FALSE(unpaid) by default and is available to the driver with the citation information and any status change reflects to all(driver, security, and administrator).
12. Security can look up if a vehicle has a valid permit to note a violation and generate a citation.
13. The citation number is unique for every citation.
14. Each permit can only have one space type.
15. paymentStatus is a boolean value where 1 is paid and 0 is unpaid.
16. availability_status is a boolean value where 1 is available and 0 is unavailable.
17. approval_status indicates the status of appealed citations. By default, it has a value set to Pending and can be Approved/Rejected by the admin.

Transaction 1: Create Citation (src → Citations.java → generateCitation())

While creating Citations, the details of citation need to be added to Citation table, details for whom the citation is, needs to be added to GivenTo, details for Lot of citation needs to be added to Shows and if the vehicle is new and not in DB, it needs to be added to Vehicle table all at once as the data is related and makes a record complete, and thus we are using transaction here.

```
// Execute insert queries in transaction in all the respective tables at
the same time to generate citation
            try{
                connection.setAutoCommit(false); // set autocommit to
false before executing statements
                // insert in citation
                try (PreparedStatement preparedStatement =
connection.prepareStatement(insertCitationQuery)) {
                    preparedStatement.setInt(1, citation_number);
                    preparedStatement.setDate(2, citation_date);
                    preparedStatement.setTime(3, citation_time);
                    preparedStatement.setString(4, category);
                    preparedStatement.setFloat(5, fee);
                    preparedStatement.setBoolean(6, payment_status);
                    preparedStatement.executeUpdate(); // execute query
                } catch (Exception e) {
                    connection.rollback(); // in case there is an issue,
rollback, return back to menu to avoid any further and go to finally
                    System.out.println("Error Occurred while inserting
citation data " + e.getMessage());
                    return;
                }
                try (PreparedStatement preparedStatement =
connection.prepareStatement(insertShowsQuery)) {
                    // insert in shows
                    preparedStatement.setInt(1, citation_number);
                    preparedStatement.setString(2, lot_name);
                    preparedStatement.executeUpdate();
                    System.out.println("Citation Lot Assigned
successfully.");
```

```
                } catch (Exception e) {
                    connection.rollback(); // in case there is an issue,
rollback, return back to menu to avoid any further and go to finally
                    System.out.println("Error Occurred while inserting
lot data " + e.getMessage());
                    return;
                }
                try (PreparedStatement preparedStatement =
connection.prepareStatement(insertGivenToQuery)) {
                    // insert in givento
                    preparedStatement.setInt(1, citation_number);
                    preparedStatement.setString(2, car_license_number);
                    preparedStatement.executeUpdate();
                } catch (Exception e) {
                    connection.rollback(); // in case there is an issue,
rollback, return back to menu to avoid any further and go to finally
                    System.out.println("Error Occurred while inserting
lot data " + e.getMessage());
                    return;
                }
                if (insertVehicle && !model.equals(null) &&
!color.equals(null)){
                    try (PreparedStatement preparedStatement =
connection.prepareStatement(insertVehicleQuery)) {
                        // insert in vehicle if not found in DB
                        preparedStatement.setString(1,
car_license_number);
                        preparedStatement.setString(2, model);
                        preparedStatement.setInt(3, 0000);
                        preparedStatement.setString(4, color);
                        preparedStatement.setString(5, "N/A");
                        preparedStatement.executeUpdate();
                    } catch (Exception e) {
                        connection.rollback(); // in case there is an
issue, rollback, return back to menu to avoid any further and go to
finally
                        System.out.println("Error Occurred while
inserting vehicle data " + e.getMessage());
```

```
                        return;
                    }
                }
                connection.commit(); // all execution went well, so
commit the transaction
                System.out.println("Citation generated successfully.");
            } catch (SQLException e) {
                System.out.println("Error Occurred while managing
transaction: " + e.getMessage());
 connection.rollback();
            } finally {
                try { // transaction is complete so set autocommit back
to true
                    connection.setAutoCommit(true);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
```

Transaction 2: Adding Permit (src → MaintainPermit.java → addPermit())

While creating and assigning permits, the details of permit need to be added to Permit table, details for which vehicle and driver the permit are given, and need to be added to IsAssigned table, details for Lot and Zone for which the permit is need to be added to the HasZone tables. If the vehicle is new and not in DB, it needs to be added to the Vehicle table. Together the data is related and makes a record complete. And that is why we are using transactions here.

```
//Using transaction to execute insert queries in all the tables at the same
time
    try{

        //start transaction if driver is eligible for the permit
    if (flag == true) {
        connection.setAutoCommit(false);  //setting auto-commit to false
before executing statements
```

```
        //insert into Permit
        if(!secondvehicle){ //dont add this stuff if we are adding a second
car on a permit for employee, just assign the car
            try (PreparedStatement preparedStatement =
connection.prepareStatement(insertPermitQuery)) {
                preparedStatement.setInt(1, permit_id);
                preparedStatement.setString(2, space_type);
                preparedStatement.setDate(3, start_date);
                preparedStatement.setDate(4, expiration_date);
                preparedStatement.setTime(5, expiration_time);
                preparedStatement.setString(6, permit_type);
                preparedStatement.executeUpdate();  //execute the query for
insertion
            } catch (Exception e) {
                //in case there is an issue in inserting data in Permit
table, rollback the transaction
                connection.rollback();
                System.out.println("Error Occurred while inserting permit
data " + e.getMessage());
                return;
            }
            //insert information of zone, lot related to the permit
            try (PreparedStatement preparedStatement =
connection.prepareStatement(insertHasZoneQuery)) {
                preparedStatement.setInt(1, permit_id);
                preparedStatement.setString(2, zone_id);
                preparedStatement.setString(3, lot_name);
                preparedStatement.executeUpdate(); //execute the query for
insertion
                System.out.println("Added permit zone");
            } catch (Exception e) {
                //in case there is an issue in inserting data in HasZone
table, rollback the transaction
                connection.rollback();
                System.out.println("Error Occurred while inserting permit
zone " + e.getMessage());
                return;
            }
        }
```

```
        //if vehicle not found in Vehicles table, add new entry into
Vehicle.
        if (insertVehicle && !model.equals(null) && !color.equals(null) &&
!manufacturer.equals(null) && year!=-1){
            try (PreparedStatement preparedStatement =
connection.prepareStatement(insertVehicleQuery)) {
                preparedStatement.setString(1, car_license_number);
                preparedStatement.setString(2, model);
                preparedStatement.setInt(3, year);
                preparedStatement.setString(4, color);
                preparedStatement.setString(5, manufacturer);
                preparedStatement.executeUpdate(); //execute the query for
insertion
                System.out.println("Vehicle data added successfully.");
            } catch (Exception e) {
                //in case there is an issue in inserting data in Vehicle
table, rollback the transaction
                connection.rollback();
                System.out.println("Error Occurred while inserting vehicle
data " + e.getMessage());
                return;
            }
        }
        //insertion into IsAssigned table
        try (PreparedStatement preparedStatement =
connection.prepareStatement(insertIsAssignedQuery)) {
            preparedStatement.setString(1, phone);
            preparedStatement.setInt(2, permit_id);
            preparedStatement.setString(3, car_license_number);
            preparedStatement.executeUpdate();//execute the query for
insertion
            System.out.println("Permit Assigned to Driver successfully.");
        } catch (Exception e) {
            //in case there is an issue in assigning permits, rollback the
transaction
            connection.rollback();
            System.out.println("Error Occurred while assigning permit " +
e.getMessage());
            return;
        }
```

```
        connection.commit(); //all queries executed successfully, so commit
the transaction
        System.out.println("Permit Added successfully.");
    }
} catch (SQLException e) {
    connection.rollback();
    System.out.println("Error Occurred while managing transaction: " +
e.getMessage());
} finally {
    try {
        //transaction is completed, so set auto-commit to true
        connection.setAutoCommit(true);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Design Decisions:

We have designed the application that starts with a Menu showing the various types of tasks that can be accomplished through this system. We have followed the 5 task design that includes Information Processing Menu, Permits and Vehicles Menu, Citations Menu, Reports Menu, Display Tables, and Exit. This allows us to encapsulate all tasks in their respective menu as described in the narrative, making testing and locating the needed tasks easier and simpler through CLI based Menu.We are also adding the demo data at the start with a script automatically with clean tables.

Selecting the different menus provides another menu to choose the tasks from and use them as required. This is accomplished by having modular code where we have separate java files for each of the 5 tasks, where we just call functions in the main file. We have decided to implement transactions at several places where multiple tables need to be updated at once together for maintaining completeness, and we are also checking for some constraints, and erroneous inputs in java.

We also make use of a ShutDownHook in Java that makes sure the connection to the database closes for graceful as well as erroneous shutdown of the application. Try catch and finally blocks at every stage make sure the program outputs the behavior and doesn't stop working abruptly.

Functional Roles:

Part 1:
Software Engineer: Aadil (Prime), Daniel(Backup)
Database Designer/Administrator: Manali(Prime), Ophelia(Backup)
Application Programmer: Ophelia(Prime), Aadil (Backup)
Test Plan Engineer: Daniel(Prime), Manali(Backup)

Part 2:
Software Engineer: Manali(Prime), Aadil(Backup)
Database Designer/Administrator: Ophelia(Prime), Daniel(Backup)
Application Programmer: Daniel(Prime), Ophelia(Backup)
Test Plan Engineer: Aadil (Prime), Manali(Backup)

Part 3:
Software Engineer: Daniel(Prime), Ophelia(Backup)
Database Designer/Administrator: Manali(Prime), Daniel(Backup)
Application Programmer: Aadil (Prime), Manali(Backup)
Test Plan Engineer: Ophelia(Prime), Aadil (Backup)