NC STATE UNIVERSITY

Wolf Parking Management System

CSC 540 Database Management Systems - Project 3

PROJECT TEAM A

Aditi Vakeel	avakeel
Boscosylvester Chittilapilly	bchitti
Janani Pradeep	jpradee
Nahed Abu Zaid	naabuzai

Assumptions:

1. Drivers will not have access to this database. They must go to the parking office for any parking-related concerns or requests.
2. A driver goes to the billing staff to raise an appeal or make a payment for a citation.
3. To fetch any data related to the driver other than name, UID external databases are to be used.
4. Admin gets to know (this knowledge is not concerned with this DB) about the availability status of spaces (lots, Zones, types) and updates data accordingly.
5. The only payments recorded in this database are the ones made for citations, and no other fee is applicable for getting a permit.
6. Billing staff will update payment status based on payments and the appeal table.
7. A new entity called 'Payments' is maintained solely by the billing staff to store information about the citation payment information.
8. Billing Staff's decisions/actions on appeal approvals are based on external knowledge.
9. The appeal table would be used to store information about the appeal description. and appeal status. It takes values 'In Progress,' Approved,' and 'Rejected.' The default value 'In Progress'.
10. Each citation can be challenged by at most one appeal and can have at most one payment.
11. To deal with deletion anomalies and reduce data redundancy, we maintain separate entities for 'SpaceType,' 'PermitType,' and 'CitationCategory'.
12. The Year of the Vehicles is assumed to be in the range 1900-2300.
13. Drivers with Status 'S' and 'E' have UID_Phone a 9-digit number and Drivers with Status 'V' have 10-digit phone numbers.
14. The Availability status in Spaces takes two values - 'Available', 'Unavailable' with the default value as 'Available'.
15. The Payment status in Citations takes three values - 'Paid,' 'Unpaid,' and 'Waived Off' with the default value as 'Unpaid.'
16. Apart from the three categories mentioned in the narrative for Citation Category, three additional categories have been added to handle 50% Handicap Discount. Whenever a new category is added, one additional entry for Handicap Discount needs to be entered.
17. The operation "generate a report for the total number of citations given in all zones in the lot for a given time range (e.g., monthly or annually)" was implemented with the assumption that a report is being generated for a lotwise count of citations for a particular date range.
18. No two Parking Lots have the same combination of name and address.
19. We would be storing invalid parking lots, invalid zones, or invalid space details in the Citations table, which would help in reporting citations registered in a specific location. The Citations table would be linked to the Spaces table by One-Many relationships on the ParkinglotID, ZoneID, and SpaceNumber.
20. Report for returning the available space number given a space type, and the parking lot returns only one space number even though there could be multiple spaces available because the narrative says *"Return an available space number given a space type in a given parking lot."*
21. Parking enforcement personnel will be relied upon to check if a vehicle is physically parked in the correct space or not. The Enforcement personnel will rely on the database to check if the vehicle has a valid permit.
22. Any two parking lots cannot have the same names.

1. Corrections:

Report-1:

5. Application Program Interfaces:

Operation 4: Reports
Report 1 – pg 7

- Generate a report for all citations:
  generateReportForCitations()
  - return all citations generated.

Operations 3: Generating and Maintaining Citations
Report 1 – pg 7

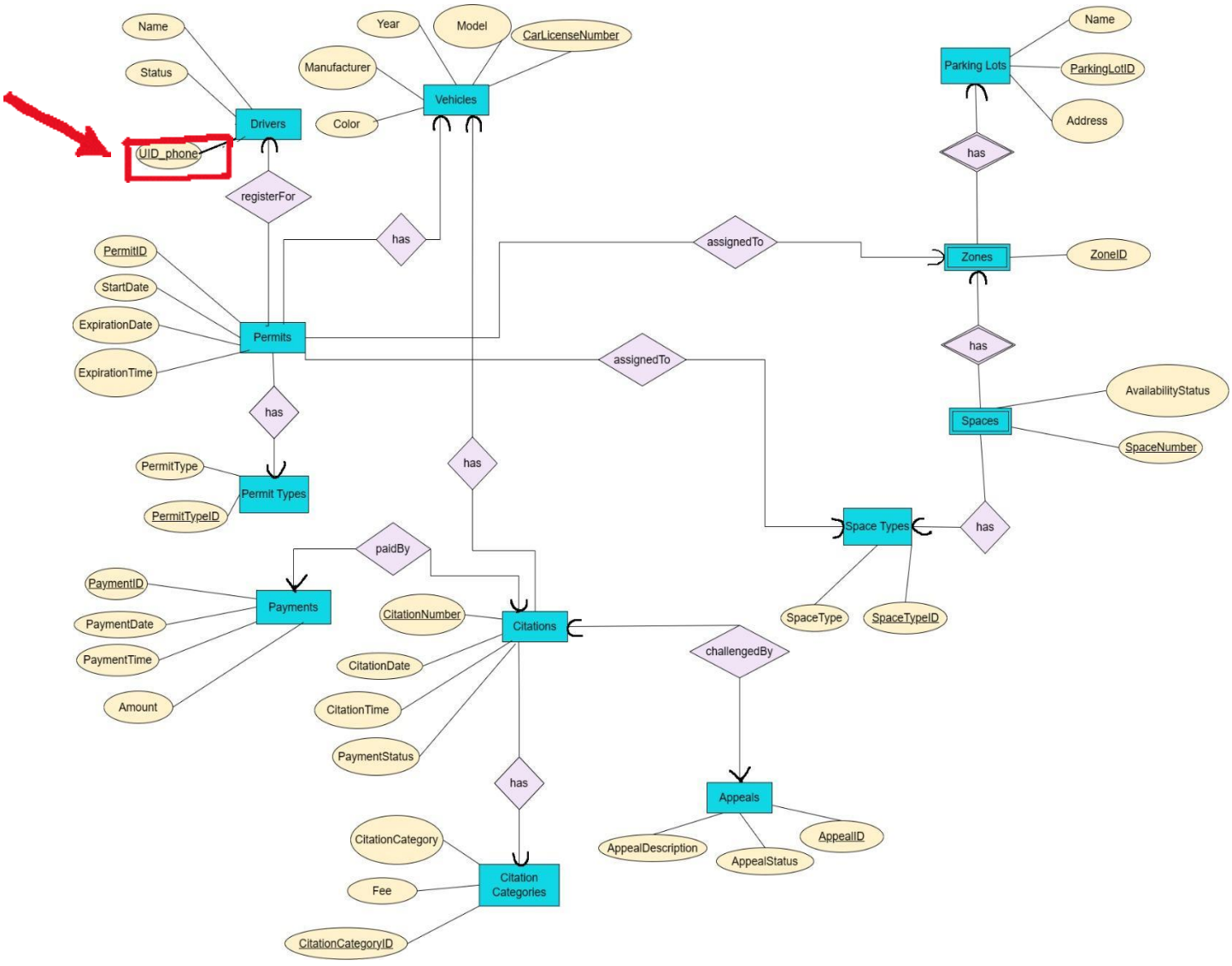
- Generate a report to get an available space number in a given parking lot for a given space type.
  generateAvailableSpaceInLot()
  - return the available space number.

P.T.O

7. Local E/R Diagrams:

Reporting Staff View

Report 1 pg- 10



9. Local Relational Schemas:

Report 1- pg 14, 15

9.1 Admin's View:

Vehicles (CarLicenceNumber, Model, Year, Manufacturer, Color)
Drivers (Name, Status, UID_Phone)
ParkingLots (Name, ParkingLotID, Address)
Zones (ZoneID, ParkingLotID)
Spaces (SpaceNumber, ZoneID, ParkingLotID, AvailabilityStatus, SpaceTypeID)
SpaceTypes (SpaceType, SpaceTypeID)
Permits (PermitID, StartDate, ExpirationDate, ExpirationTime, PermitTypeID, ParkingLotID, ZoneID, SpaceTypeID, UID_Phone, CarLicenseNumber)
PermitTypes (PermitType, PermitTypeID)

9.2 . Reporting Staff View:

Vehicles (CarLicenceNumber, Model, Year, Manufacturer, Color)
Drivers (Name, Status, UID_Phone)
ParkingLots (Name, ParkingLotID, Address)
Zones (ZoneID, ParkingLotID)
Spaces (SpaceNumber, ZoneID, ParkingLotID, AvailabilityStatus, SpaceTypeID)
SpaceTypes (SpaceType, SpaceTypeID)
Permits (PermitID, StartDate, ExpirationDate, ExpirationTime, PermitTypeID, ParkingLotID, ZoneID, SpaceTypeID, UID_Phone, CarLicenseNumber)
PermitTypes (PermitType, PermitTypeID)
Citations (CitationNumber, CitationDate, CitationTime, PaymentStatus, CitationCategoryID, CarLicenseNumber, ParkingLotID, ZoneID, SpaceNumber)
CitationCategories (CitationCategoryID, CitationCategory, Fee)
Payments (PaymentID, PaymentDate, PaymentTime, Amount, CitationNumber)

9.3 Billing Staff View:

Citations (CitationNumber, CitationDate, CitationTime, PaymentStatus, CitationCategoryID, CarLicenseNumber, ParkingLotID, ZoneID, SpaceNumber)
CitationCategories (CitationCategoryID, CitationCategory, Fee)
Payments (PaymentID, PaymentDate, PaymentTime, Amount, CitationNumber)
Appeals (AppealID, AppealDescription, AppealStatus, CitationNumber)

9.4 Parking Enforcement Personnel View:

Vehicles (CarLicenceNumber, Model, Year, Manufacturer, Color)
Permits (PermitID, StartDate, ExpirationDate, ExpirationTime, PermitTypeID, ParkingLotID, ZoneID, SpaceTypeID, UID_Phone, CarLicenseNumber)
PermitTypes (PermitType, PermitTypeID)
Citations (CitationNumber, CitationDate, CitationTime, PaymentStatus, CitationCategoryID, CarLicenseNumber, ParkingLotID, ZoneID, SpaceNumber)
CitationCategories (CitationCategoryID, CitationCategory, Fee)

Report-2:

4) SQL Queries

4.1

Operation 1: Information Processing

Report 2 pg- 20

- Delete Driver (Updated Query)

SQL > DELETE FROM Drivers WHERE UID_Phone = 984985678
Query OK, 1 row affected (0.0028 sec)

Operation 3: Generating and Maintaining Citations (Earlier Missing)

Report 2: pg 24

- Detect parking violations by checking if a car has a valid permit in the lot.

SQL> SELECT CASE WHEN COUNT (*) > 0 THEN 'VALID PERMIT' ELSE 'INVALID PERMIT' END AS VIOLATION_STATUS FROM Permits p JOIN SpaceTypes st ON p.SpaceTypeID = st.SpaceTypeID WHERE ParkingLotID = 3 AND ZoneID = 'C' AND SpaceType = 'regular' AND CarLicenceNumber = 'ABC555' ;

2. Submitted via the submit board.

3. Transactions

Transaction:1 :Add Payment (Line :1828- In main code)

```
//Insert Payment
// When ever a payment is inserted the corresponding citation's payment
   status is updated to 'Paid'
//Thus these two operations are put in one transaction so that either
   bothe exceute or neither
 private static void insertPayment(Connection conn,Scanner scanner) {
          System.out.print("Enter Payment Date (YYYY-MM-DD): ");
          String pdt = scanner.nextLine();

          System.out.print("Enter Payment Time (HH:MM:SS): ");
          String ptm = scanner.nextLine();

          System.out.print("Enter Citation Number: ");
          String c = scanner.nextLine();

          Float amt= (float) 0.0;
    String SQL = "Select Fee from CitationCategories CC JOIN Citations C "
               + "ON C.CitationCategoryID=CC.CitationCategoryID where
  C.CitationNumber = ?;";

    System.out.println(SQL);

   try (PreparedStatement preparedStatement =
  connection.prepareStatement(SQL)) {
              preparedStatement.setString(1, c);
            try (ResultSet resultSet = preparedStatement.executeQuery())
  {
                while (resultSet.next()) {
                    amt = resultSet.getFloat("Fee");
```

```
                }
            }
    } catch (SQLException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }

        String insertSQL = "INSERT INTO Payments (PaymentID, PaymentDate,
  PaymentTime, Amount, CitationNumber) "
                + "VALUES (NULL, ?, ?, ?, ?);";

        try (PreparedStatement pstmt = conn.prepareStatement(insertSQL)) {
        //Set Auto commit to false to create a successful transaction
        connection.setAutoCommit(false);
            pstmt.setString(1, pdt);
            pstmt.setString(2, ptm);
            pstmt.setFloat(3, amt);
            pstmt.setString(4, c);

            int rowsAffected = pstmt.executeUpdate();

            if (rowsAffected > 0) {
                System.out.println("Payment inserted successfully!
  Updating Payment Status. \n");
//Call the updatepayment Status once the insert payement is successfully
  executed.
                updatePaymentStatus(conn, scanner, c, 0);
            } else {
                System.out.println("Failed to insert Payment.\n");
            }
//Close the connection and commit the transaction
        pstmt.close();
        connection.commit();

        } catch (SQLException e) {
            e.printStackTrace();

        if (connection!= null){
              try{
                    System.out.println("Transaction is being roll-
  backed");
                    conn.rollback();
              }catch(SQLException s){
                    s.printStackTrace();
              }
        }
        }
   finally{
        try{
             // Enable auto-commit after the transaction
             connection.setAutoCommit(true);
         }catch(SQLException s){
              s.printStackTrace();
              }
         }
    }
```

Transaction 2: Update Appeal Status (Line: 1959-in main code)

```
      //Update the Appeal Status
      // Once the appeal status is updated to 'Approved' then the citation's
      payment status must be updated to 'Waived-off'
      // Thus these operations are combined to one transaction such that
      either both excute or none.
      private static void updateAppealStatus(Connection conn,Scanner
scanner){

               System.out.print("Enter Appeal Status
(Approved/Pending/Rejected): ");
               String as = scanner.nextLine();

               System.out.print("Enter Appeal ID: ");
               String aid = scanner.nextLine();
               int ano = Integer.valueOf(aid);

               String SQL = "UPDATE Appeals "
                        + "SET AppealStatus = ? WHERE AppealID = ?;";


               try (PreparedStatement pstmt = conn.prepareStatement(SQL)) {
// Set the auto commit to false for the grouping the methods under one
successful transaction
                        conn.setAutoCommit(false);

                    Integer flag = 0;
                    if(as.equals("Approved"))
                    {
                          flag=1;
                    }
                    String a=aid;
                    pstmt.setString(1, as);
                    pstmt.setInt(2, ano);
                    System.out.println(flag+" "+as);
                    int rowsAffected = pstmt.executeUpdate();

              if (rowsAffected > 0) {
   System.out.println(as+" Appeal Status updated successfully!\n");

             if(flag==1){
             Integer cc=0;
             System.out.println("In here");
         String sql= "select CitationNumber from Appeals where AppealID = ?";
         System.out.println(sql);

 try (PreparedStatement preparedStatement = connection.prepareStatement(sql))
{     preparedStatement.setString(1, a);
      try (ResultSet resultSet = preparedStatement.executeQuery()) {
      while (resultSet.next()) {
              cc = resultSet.getInt("CitationNumber");
                                  }
                              }
```

```
      String cn = String.valueOf(cc);
//Once the appeal status is set to 'Approved' updatePaymentStatus is called
to update the citation's payment.
      updatePaymentStatus(conn, scanner, cn, 1);
                        }
            }
       }

 else {
       System.out.println("Failed to update Appeal Status.\n");
                  }
//Close the connection and commit the transaction
       pstmt.close();
       conn.commit();
 }catch (SQLException e) {
   e.printStackTrace();
//If the connection is not null then the transaction has to be roll-backed
      if (conn!= null){
            try{
              System.out.println("Transaction is being roll-backed");
               conn.rollback();
            }catch(SQLException s){
                  s.printStackTrace();
                              }
                        }
              }
            finally{
              try{
      // Enable auto-commit after the transaction
                conn.setAutoCommit(true);
              }catch(SQLException s){
                 s.printStackTrace();
                              }
                    }
            }
```

4. Documentation submitted via the submit board.

High-Level Design Decisions

The Wolf Parking System is designed with a modular structure, The initialize method encapsulates the main functionality of the program. A loop presents a menu to the user, allowing them to choose from different tasks related to the Wolf Parking Management System and divide tasks and operations into distinct categories. The user can choose from the following functions: Select (Display) Data, Insert Data, Update Data, Delete Data, Report, and Exit. The main menu allows users to navigate these options, providing a user-friendly interface.

For example, selecting "Select Data" leads to a sub-menu for displaying the data type (e.g., Vehicles, Drivers, etc.).

The main class (`Main.java`) acts as a controller, orchestrating the flow between different menu options and delegating tasks to specific types responsible for each category of operations. This design decision promotes code organization, readability, and maintainability by encapsulating related functionalities within separate classes.

Furthermore, the system integrates error handling to manage unexpected issues gracefully, preventing the application from crashing and providing users with informative error messages. Loops and switch statements enhance the user experience, allowing for repeated actions and easy navigation between different tasks. Overall, the system exhibits a well-structured and modular design, facilitating effective management of Wolf Parking operations.

Functional Roles:

Part 1:
Software Engineer: Boscosylvester (Prime), Janani (Backup)
Database Designer/Administrator: Aditi (Prime), Nahed (Backup)
Application Programmer: Janani (Prime), Boscosylvester (Backup)
Test Plan Engineer: Nahed (Prime), Aditi (Backup)

Part 2:
Software Engineer: Janani (Prime), Aditi (Backup)
Database Designer/Administrator: Boscosylvester (Prime), Nahed (Backup)
Application Programmer: Nahed (Prime), Janani (Backup)
Test Plan Engineer: Aditi (Prime), Boscosylvester (Backup)

Part 3:
Software Engineer: Nahed (Prime), Aditi (Backup)
Database Designer/Administrator: Boscosylvester (Prime), Janani (Backup)
Application Programmer: Aditi (Prime), Nahed (Backup)
Test Plan Engineer: Janani (Prime), Boscosylvester (Backup)