CSC 540 Database Management Systems

Wolf Parking Management System

Project Report 3

16th November 2023

Team Members:

Aravinda Raman Jatavallabha

Prathyusha Kodali

Suraj Raghu Kumar

Yuktasree Muppala

Assumptions:

1. Vehicles are limited to one parking lot per visit and this will not change throughout their visit.
2. One driver can be associated with one or two vehicles.
3. A driver can hold multiple permits.
4. Drivers can have multiple citations associated with them.
5. One citation is associated with only violation at a time.
6. A vehicle can be issued only one permit at a time; with exactly one permit type ("residential", "commuter", "peak hours", "special event", and "Park & Ride").
7. A handicapped person should only be assigned to a handicap space in the parking space, regardless of the vehicle type (eg. "electric", "handicap", "compact car", or "regular).
8. A driver can be assigned to only one status at a time (eg. 'S', 'E', or 'V' depending on whether a student, or an employee or a visitor). For instance, a student working on campus will still be considered as a student and not an employee.
9. In the case of a student or an employee, UnivID is the value for the ID attribute for a Driver. Whereas, in the case of a Visitor, their phone number will be considered as a value for the ID.
10. Zones are included in the Parking Lot and within each zone there are dedicated parking spaces for vehicles.
11. A driver can submit at most one appeal per citation.
12. Each zone has a particular name associated with it.

<u>Corrections made in Project Reports 1 and 2:</u>

1. Included the following APIs (which were missing in Project Report 1):
    a. Update Driver Information (Page 21 of Project Report 2)
    b. Update Parking Lot Information (Page 22 of Project Report 2)
    c. Update Zone Information (Page 23 of Project Report 2)
    d. Update Space Information (Page 24 of Project Report 2)
    e. Add/Delete Permit (Page 25 of Project Report 2)

2. Included the Foreign keys (missing in Project Report 2): Page 20-23 of

Project Report 3

3. Corrected the following queries (from Project Report 2):
    a. Return the license number of the cars that are currently in violation: Page 44 of Project Report 3
    b. Return the count of number of cars that are currently in violation: Page 45 of Project Report 3

Part of Report 1

1. Problem Statement:

This problem involves designing the Wolf Parking Management System, a comprehensive database solution for efficiently managing driver information, parking lot information, zone information, space information, vehicle information, and citation information on a university campus.

The system performs four significant types of tasks, each comprising various operations: *Information processing*, which involves updating driver, parking lot, zone, space, and permit information, along with citation management; *maintaining permits and vehicle details for each drive*r, involving permit assignments based on driver status, and managing vehicle ownership information. It also handles *generating and maintaining citations*, checking permit validity, and allowing drivers to pay or appeal citations. Additionally, the system *generates reports*, including citation and zone-related reports, car violation counts, employee permit statistics, and permit and space lookup functions.

A database system is a wiser choice than depending on a simple file structure because many people will view and change the data simultaneously. Employing a database and running data queries gives a more practical option for searching through files containing obsolete details, given the regular input and updates of significant data volumes. The necessity of re-entering complete data files or requiring system users to reload files before utilization could be more efficient. Thus, employing a simple set of files is impractical for the Wolf Parking Management System. Instead, opting for a database system eradicates the risk of users inadvertently overwriting each other's updates and prevents drivers from submitting incomplete information. This safeguard ensures data integrity and

accuracy within the Wolf Parking System, eliminating the possibility of data loss or discrepancies.

2. Intended Users:

Administrators of the Parking System:

- Add Parking Lots: Administrators can add new parking lots to the system.
- Assign Zones and Spaces: They can assign specific zones and spaces within the parking lots.
- Assign Parking Permits: Administrators can assign parking permits to drivers based on their status (Student, Employee, or Visitor).
- Change Space Availability: Administrators can change the availability status of parking spaces.
- Verify Permit Validity: Administrators can check if a car has a valid permit to park in a particular lot

Drivers (Students, Employees, Visitors):

- Obtain and Display Permits: Drivers must obtain the necessary permits to park in specific zones and spaces designated for their status.
- Ensure Correct Permit Type: Drivers must ensure their permits match the appropriate zone and space types (e.g., electric car spaces).

Security Personnel:

- Citation Management: Security personnel create, update, and delete citations for vehicles that violate parking regulations.
- Payment Processing: They handle the processing of citation payments, updating payment status from unpaid to paid through a payment procedure.

These users collectively contribute to the efficient management of the parking system. Additionally, it is essential to note that students and visitors have certain limitations on the number of vehicles they can have on a permit, while employees

have different allowances. Furthermore, special event permits and 'Park & Ride' permits are available to students and employees as additional options.

3. Five Main Entities:

1) Driver - driverID, name, status
2) Vehicle - licenseNum, color, model, year, manufacturer
3) Security - securityID
4) Permit - permitID, permitType, spaceType, startDate, expirationDate, expirationTime
5) Parking Lot - lotID, name, address, category

4. Tasks and Operations- Realistic Situations:

- **Situation 1:** John, an employee, enters the WolfParking lot in his electric car, and his university ID and name are entered into the database. He is then assigned a permit, which has all the information about the parking lot, vehicle license number, and his permit type, which is residential. Finally, he parks his car in the assigned parking lot in Zone 'A' in the 'electric' space.

- **Situation 2:** A driver named Steffen parks his vehicle, a red 2023 Honda Civic, at the WolfParking lot. Upon returning, he noticed a $30 citation for an expired permit on his windshield. John appeals for security verification of the citation, and after it is confirmed to be valid, he promptly settles the payment. The security officer updates the payment status, effectively managing the citation process and ensuring accurate record-keeping.

5. Application Programming Interfaces:

Information Processing

- addDriverInfo(driverID, name, status):
    - Description: Adds or updates basic information about drivers.
    - Returns True if added/updated successfully, False otherwise.

- deleteDriveInfo(DriverID):
    - Description: Deletes driver information.
    - Returns True if deleted successfully, False otherwise.

- addParkingLotInfo(lotID, category, address, name):
    - Description: Adds parking lot information.
    - Returns True if added/updated successfully, False otherwise.

- deleteParkingLotInfo(lotID):
    - Description: Deletes parking lot information.
    - Returns True if deleted successfully, False otherwise.

- addZoneInfo(zoneID, lotID):
    - Description :Adds specified zones to lots.
    - Returns True if added successfully, False otherwise.

- deleteZoneInfo(zoneID):
    - Deletes zone information.
    - Returns True if deleted successfully, False otherwise.

- addSpaceInfo(spaceNum, zoneID, lotID, availabilityStatus)
    - Adds or updates space information to zones.
    - Returns True if added successfully, False otherwise.

- deleteSpaceInfo(spaceNum, zoneID, lotID):
    - Deletes space information from zones.
    - Returns True if deleted successfully, False otherwise

- checkPermitValidity(licenseNum, lotID):
    - Checks if a car has a valid permit for the specified lot.
    - Returns True if valid, False otherwise.

- addVehicleOwnership(licenseNum, driverID):
    - Checks if a vehicle is owned by the specified driver.
    - Returns True if owned, False otherwise.

Maintaining Permits and Vehicle Information

- assignPermitToDriver(driverID, permitID):
    - Assigns a permit to a driver.
    - Returns True if assigned successfully, False otherwise.

- updatePermitInfo(PermitID, PermitType, ExpirationTime, ExpirationDate, StartDate, SpaceType, Lot):
    - Updates permit information.
    - Returns True if added/updated successfully, False otherwise.

- updateVehicleInfo(licenseNum, color, manufacturer, model, year):
    - Updates vehicle information.
    - Returns True if added/updated successfully, False otherwise.

- addVehicleInfo(licenseNum):
    - Removes vehicles from the database.
    - Returns True if updated successfully, False otherwise.

- removeVehicleInfo(licenseNum):
    - Removes vehicle information from the database.
    - Returns True if updated successfully, False otherwise.

Generating and Maintaining Citations

- generateCitation(driverID, lotID, zoneID, spaceNum, category):
    - Generates a citation.
    - Returns True if generated successfully, False otherwise.

- settleCitation(driverID, citationNum):
    - Pays a citation.
    - Returns True if paid successfully, False otherwise.

- appealCitation(driverID, citationNum):
    - Appeals a citation.
    - Returns True if appealed successfully, False otherwise.

- updateCitationInfo(citationNum, paymentStatus, fee, lot, category):
    - Updates citation information.
    - Returns True if updated successfully, False otherwise.

- deleteCitation(citationNum):
    - Deletes a citation.
    - Returns True if deleted successfully, False otherwise.

- getCitationsForDriver(driverID):
    - Returns a list of citations associated with the specified driver.

Reports

- generateCitationReport(timeRange, lotID):
    - Returns a detailed report of all citations within the specified time range for the specified lot.

- fetchZoneList(zoneID, lotID)
    - Return the list of zones for each lot as tuple pairs (lot, zone).

- getAvailableSpacesInLot(lotID):
    - Returns a list of available parking spaces in the specified lot.

6. Description of Views:

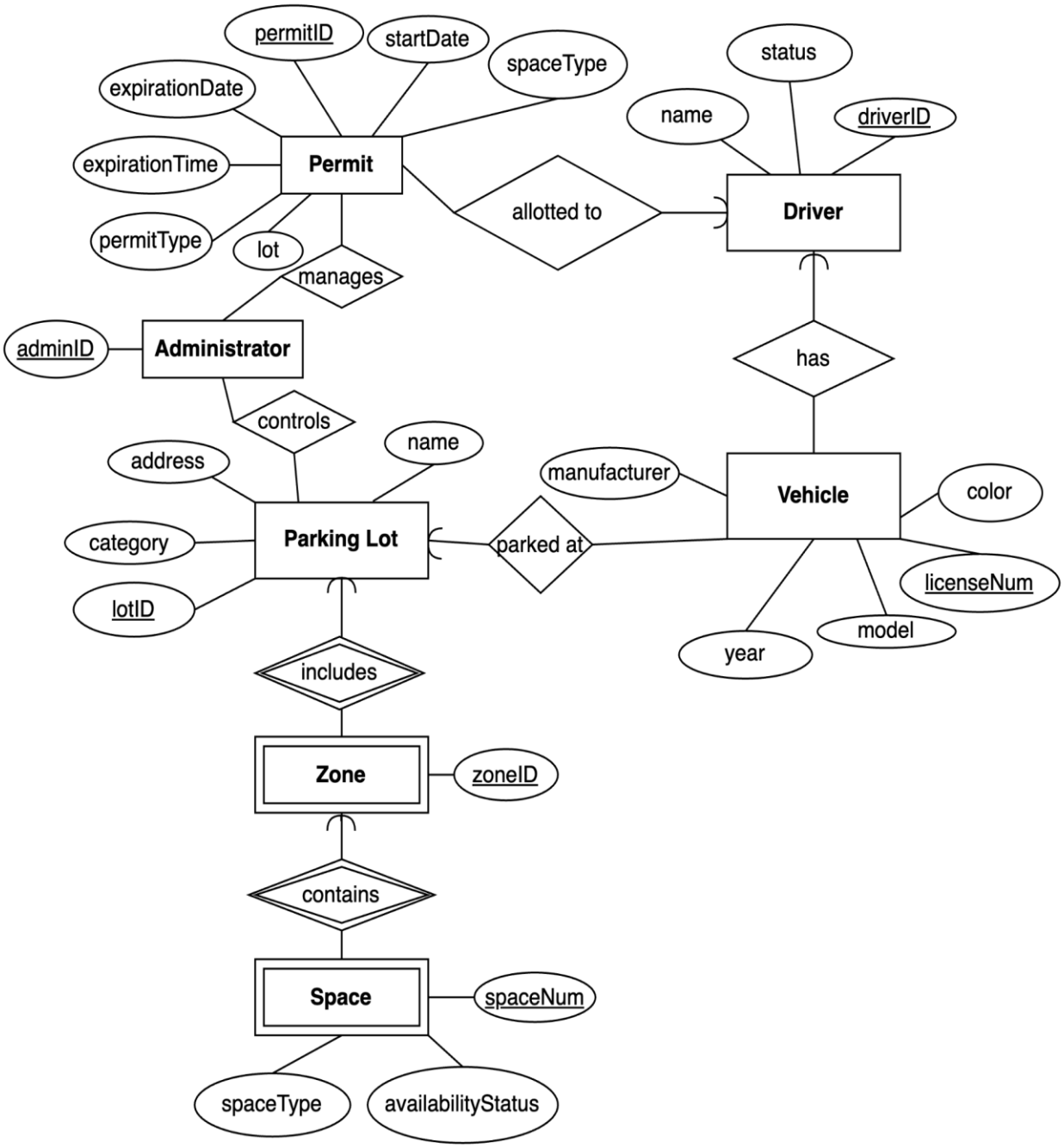
Driver's View: Drivers can have vehicles parked in the parking lot. They are allowed to get the permits and can appeal for the citations.

Security's View: Security can view and handle citation information, approve or reject the citation appeals, can generate monthly or annual citation reports.

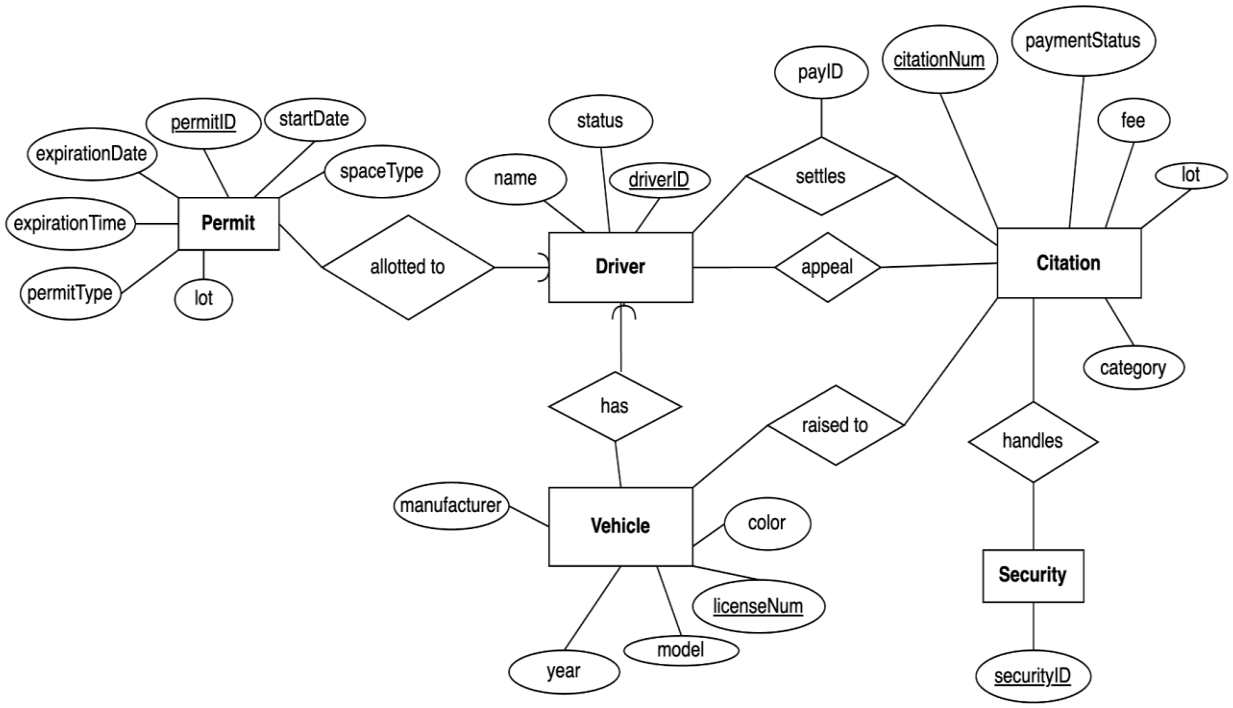
Administrator's View: Administrators can add parking lots to the system, assign zones and spaces to the lots, assign a parking permit to a driver, change the space availability, and verifies if a car has a valid permit.

7. Local E/R Diagrams:

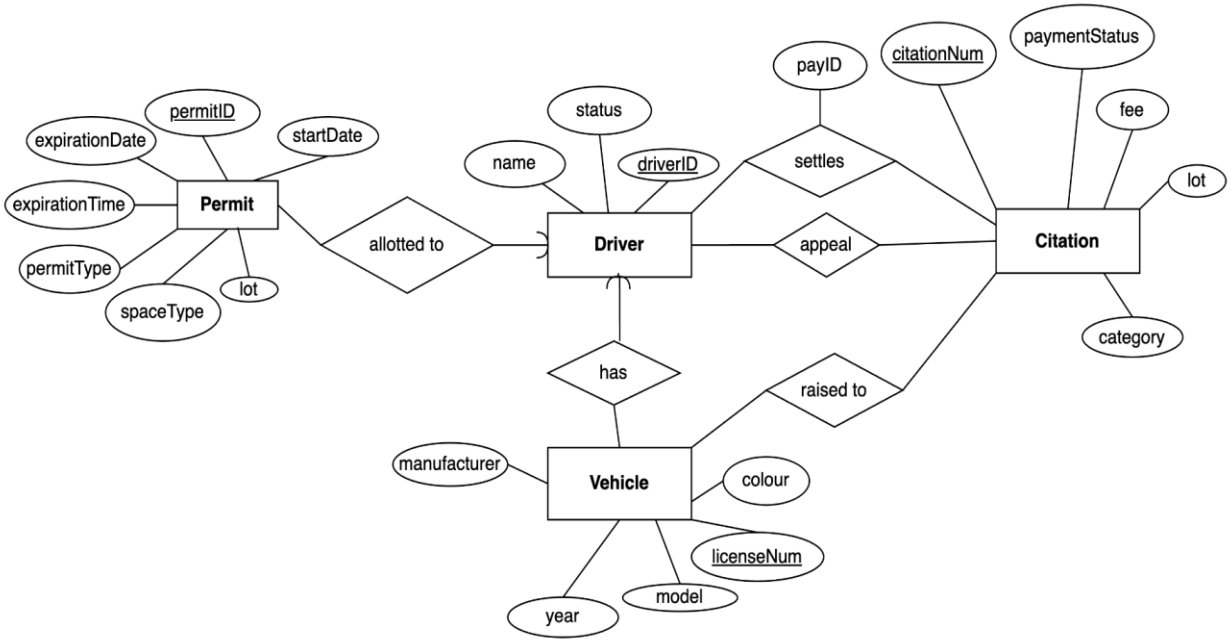
Administrator's View:



Security's View:



Driver's View:



8. Description of Local E/R diagrams:

- Every driver who arrives at the parking lot has a status (student, employee, or visitor), a unique driverID, and a name.
- One driver can have multiple vehicles, as per assumption #2**,** which a unique licenseNum can identify**.**
- A driver can also be associated with multiple permits as per assumption #3**,** where a unique permitID can identify the permit, as one particular permitID cannot be related to numerous drivers.
- Citations are identified by a unique citationNum associated with a particular driver raised on their vehicle.
- Drivers and citations have a many-many relationship. That means that many drivers can have multiple citations associated with them for their vehicles they own, according to assumption #4.
- Administrators identified by a unique adminID manage the permit, assign a parking permit to a driver, control the parking lot by assigning spaces to drivers based on availability, and check if a car has a valid permit in their lot.
- As per assumption #1, a vehicle can be parked only in a single parking lot, identified by a lotID, with an assigned zone and a space.
- Zones are also identified by a zoneID, which is unique as no two zones can have the same zoneID within a parking lot.
- Spaces are also identified by a spaceNum which is unique, as no two spaces in a zone inside a parking lot can have the same spaceNum.
- Parking lot is a strong entity associated with Zones, a weak entity further connected to another weak entity, Spaces. This is as per assumption #10**.**
- Security, who can be identified by a unique securityID, handles the citation information that violates parking regulations.
- Security also handles changing the status of payment made on the citation by the driver.

9. Local Relational Schemas:

Driver's View:

Driver (driverID, name, status)
Vehicle (licenseNum, manufacturer, color, model, year)
Permit (permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot)
Citation (citationNum, lot, fee, paymentStatus, category)

AllotedTo (permitID, driverID)
Appeals (driverID, citationNum)
Settles (driverID, citationNum, payID)
RaisedTo (citationNum, licenseNum)
Has(driverID, licenseNum)

Security's View:

Security (securityID)
Citation (citationNum, lot, fee, paymentStatus, category)
Driver (driverID, name, status)
Permit (permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot)
Vehicle (licenseNum, manufacturer, color, model, year)

Appeals (driverID, citationNum)
RaisedTo (citationNum, licenseNum)
Settles (driverID, citationNum, payID)
Handles (securityID, citationNum)
AllotedTo (permitID, driverID)
Has(driverID, licenseNum)

Administrator's View:

Driver (driverID, name, status)
Vehicle (licenseNum, manufacturer, color, model, year)

Permit (permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot)
Parking Lot (lotID, category, address, name)
Zone (zoneID, lotID)
Space (spaceNum, zoneID, lotID, spaceType, availabilityStatus)
Administrator (adminID)

Manages (permitID, adminID)
ParkedAt (lotID, licenseNum)
Controls (adminID, lotID)
AllotedTo (permitID, driverID)
Has(driverID, licenseNum)

10. Local Schema Documentation:

Entity Sets to Relations:
The entity sets Permit, Driver,Citation, Parking Lot, Security, were converted into relations with attributes users, drivers, vehicles, permits, citations, and parking lots.

Combining Many - One Relationships:

Weak entity set- "Zone" was made into a relation with all its attributes plus the lotID attribute, which is a foreign key from the Parking Lot entity set. Comprising lotID represents the "includes" relationship in the diagram between Zone and Parking lot; therefore, we do not have a separate relationship in the schema for "includes". Zone was made into a weak entity set because it has a many-one relationship where we need to know the lotID in combination with the zoneID to look it up.

Weak entity set- "Space" was made into a relation with all its attributes plus the ZoneID and lotID attributes, which are foreign keys from the Zone and Parking Lot entity sets. Additional attributes zoneID and lotID represent the "contains" relationship in the diagram between the Zone and Space Entity set; therefore, we do not have a separate relationship in the schema for "Contains". Space was made into

a weak entity set because it has a many-one relationship with Zone where we need to know the lotID, ZoneID and combination with the SpaceID to look it up.

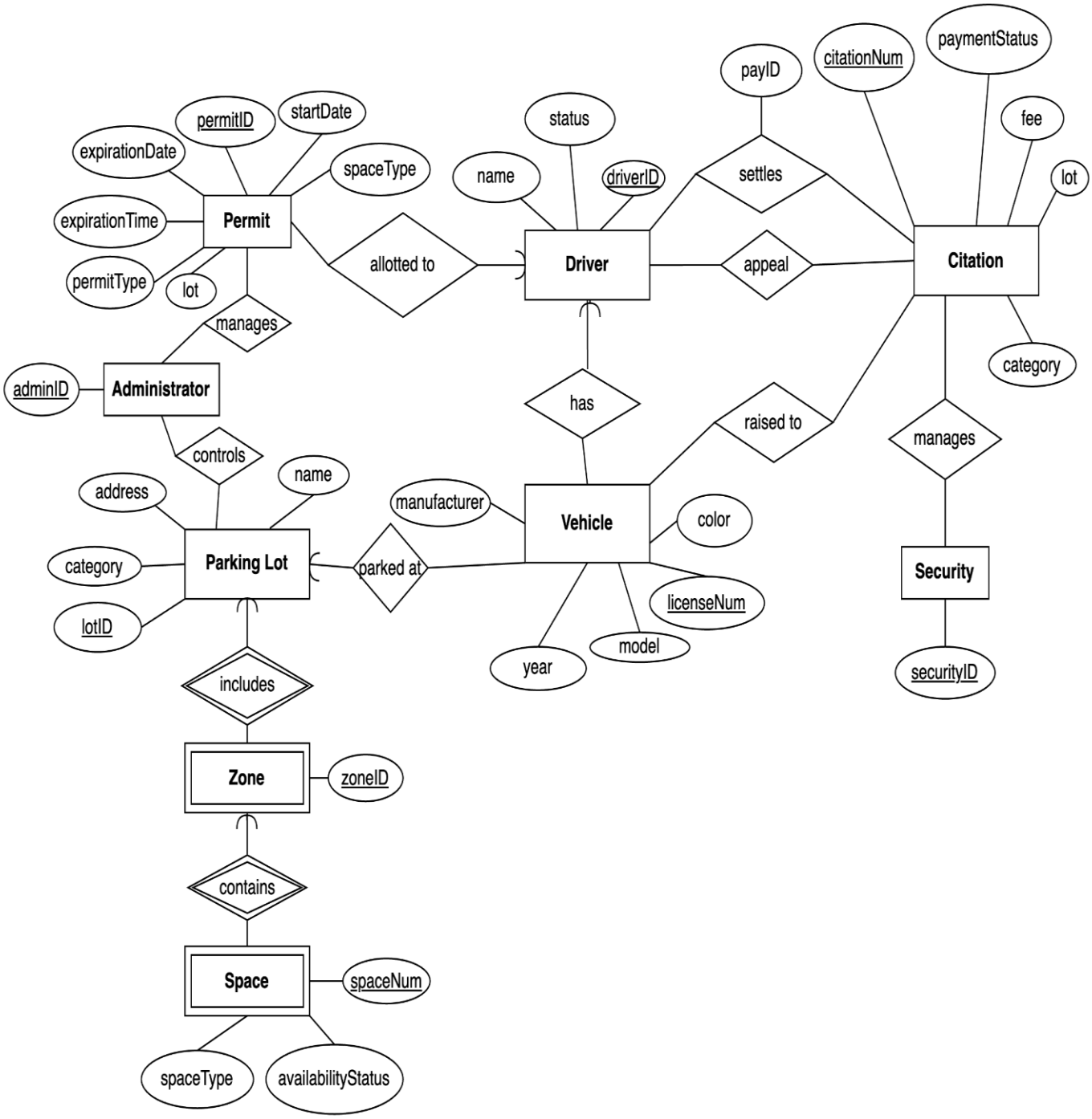
Relationships to Relations:
Relationships parked at, allotted to, settles, appeal, raised to, controls, manages, handles has from ER diagram have all been turned into relations. The attributes in each relation are the primary keys of the entities they are connecting plus the attributes of that relationship itself. Settles relation has payID attribute as it is common to both the entities it is connecting, Driver and Citation.

Foreign Keys:
In Permit entity, lot attribute is a foreign key referencing the Parking Lot entity and spaceType attribute references the Space entity. Similarly, lot attribute in Citation entity is a foreign key that references Parking Lot entity. Through these foreign keys, we are ensuring that referential integrity is maintained and query performance is improved.

11. Inconsistency identified in the given problem prompt:

A handicapped person visiting the parking lot in an electric car or any other car type which has a dedicated space for parking in the lot must be allocated the handicapped space only for parking their car and not any other space type. If this is not followed, we will be violating the 1NF condition. Therefore, we are following assumption #7, thereby resolving the inconsistency.

12. An Overview of the Complete ER Diagram



Part of Report 2

1. Global Relational Database Schema:

Driver(driverID, name, status, isHandicapped)

driverID → driverID, name, status, isHandicapped holds because each driverID is unique, and identifies an individual name and status. Any other combination of the attributes will limit the possibilities of our database. For example, many drivers can have the same name and status. Since, the left hand side is a superkey of the functional dependency (FD) mentioned above for the driverID attribute, the FD is in BCNF (thereby in 3NF).

Vehicle(licenseNum, manufacturer, color, model, year)

licenseNum → licenseNum, manufacturer, color, model, year holds because each licenseNum is unique and identifies the model, year, manufacturer and color of a vehicle. No other attributes can be a key because there could be multiple instances of them. For instance, multiple vehicles could have the same manufacturer, color, model and year. Since licenseNum is able to determine all of the other attributes in the above functional dependency, it is in BCNF and therefore 3NF.

Permit(permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot, zoneID)

permitID → permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot, zoneID holds because permitID for each permit is unique and hence able to determine all of the other attributes like spaceType, startDate, expirationDate, permitType, expirationTime, lot and zoneID. No other attribute can be a key as multiple permits could have the same spaceType, startDate, expirationDate, permitType, expirationTime, lot and zoneID. Since permitID alone is able to determine all of the other attributes, it is in BCNF and hence 3NF.

Citation(citationNum, citationTime, citationDate, lot, fee, paymentStatus, category)

citationNum → (citationNum, citationTime, citationDate, lot, paymentStatus, category) holds and **category → fee** also holds. citationNum is unique for each citation issued and hence able to determine all of the other attributes like lot, fee, paymentStatus, and category. Fees are charged based on the violation

category ($25 for category “Invalid Permit,” $30 for category “Expired Permit,” $40 for category “No Permit”). Since the FD category → fee with respect to the above relation schema is not in BCNF (as the attribute category is not the superkey of the relation) we decompose the above table into two tables **Citation(citationNum, citationTime, lot, paymentStatus, category)** and **CitationFee(category , fee)** which are in BCNF and 3NF.

AllottedTo(permitID, driverID)

permitID, driverID → permitID, driverID holds because there are only 2 attributes and both are in the superkey. Hence the above FD is in BCNF and therefore, 3NF.

Appeals(driverID, citationNum)

driverID, citationNum → driverID, **citationNum** holds because there are only 2 attributes and both are in the superkey. Hence the above FD is in BCNF and therefore, 3NF.

Settles(driverID, citationNum)

driverID, citationNum → driverID, **citationNum** holds because there are only 2 attributes and both are in the superkey. Hence the above FD is in BCNF and therefore, 3NF.

RaisedTo(citationNum, licenseNum)

citationNum, licenseNum → citationNum, licenseNum holds because there are only 2 attributes and both are in the superkey. Hence the above FD is in BCNF and therefore, 3NF.

Has(driverID, licenseNum)

driverID, licenseNum → driverID, licenseNum holds because there are only 2 attributes and both are in the superkey. Hence the above FD is in BCNF and therefore, 3NF.

Security(securityID)

securityID → securityID is in 3NF because it is in BCNF, as there is only one attribute in this functional dependency, and the one attribute is the key.

Administrator(<u>adminID</u>)

adminID → adminID is in 3NF because it is in BCNF, as there is only one attribute in this functional dependency, and the one attribute is the key.

Handles(<u>securityID</u>, <u>citationNum</u>)

securityID, citationNum → securityID, citationNum holds because there are only 2 attributes and both are in the superkey. Hence the above FD is in BCNF and therefore, 3NF.

Parking Lot(<u>lotID</u>, address, name)

lotID → lotID, address, name holds because lotID is unique for each citation issued and hence able to determine all of the other attributes like lotID, category, address, name. No other attribute can be a key since there could be the same category, address and name for multiple parking lots allotted. Also since lotID is able to determine all of the other attributes, it is in BCNF and hence 3NF.

Zone(<u>zoneID</u>, <u>lotID,</u> zone_name)

zoneID, lotID → zoneID, lotID, zone_name holds because zoneID and lotID are able to determine all the other attributes in the functional dependency. Hence the above FD is in BCNF and therefore, 3NF.

Space(<u>spaceNum</u>, <u>zoneID</u>, <u>lotID</u>, spaceType, availabilityStatus)

spaceNum, zoneID, lotID → spaceNum, zoneID, lotID, spaceType, availabilityStatus holds because spaceNum, zoneID and lotID are able to uniquely determine spaceType and availabilityStatus. No other attribute can be a key because there could be Spaces with the same availabilityStatus and spaceType. Since spaceNum, zoneID, and lotID are able to determine all the other attributes in the FDs, it is therefore in BCNF and hence 3NF.

Manages(<u>permitID</u>, <u>adminID</u>)

permitID, adminID → permitID, adminID holds because there are only 2 attributes and both are in the superkey. Hence the above FD is in BCNF and therefore, 3NF.

ParkedAt(<u>lotID</u>, <u>licenseNum</u>, zoneID, spaceNum)
lotID,licenseNum → lotID, licenseNum, zoneID, spaceNum
lotID and licenseNum are primary keys and zoneID and spaceNum are relationship

attributes. Since lotID, licenseNum are able to determine all the other attributes in the FD, it is therefore in BCNF and hence 3NF.

Controls(<u>adminID</u>, <u>lotID</u>)

adminID, lotID → adminID, lotID holds because there are only 2 attributes and both are in the superkey. Hence the above FD is in BCNF and therefore, 3NF.

2. **Design for Global Schema:**

   The entity sets in our diagram were made into relations with their respective attributes for Permit, Driver, Citation, Administrator, Parking Lot, Vehicle, and Security.

   Other relationships have each been tuned into relations in our schema. Their attributes in the schema are the keys of the connecting entities.

   Weak entity set- "Zone" was made into a relation with all its attributes plus the lotID attribute, a foreign key from the Parking Lot entity set. Comprising lotID represents the "includes" relationship in the diagram between the Zone and Parking lot; therefore, we do not have a separate relationship in the schema for "includes." Zone was made into a weak entity set because it has a many-one relationship where we need to know the lotID in combination with the zoneID to look it up.

   Weak entity set- "Space" was made into a relation with all its attributes plus the ZoneID and lotID attributes, which are foreign keys from the Zone and Parking Lot entity sets. Additional attributes zoneID and lotID represent the "contains" relationship in the diagram between the Zone and Space Entity set; therefore, we do not have a separate relationship in the schema for "Contains." Space was made into a weak entity set because it has a many-one relationship with Zone where we need to know the lotID, ZoneID, and combination with the SpaceID to look it up.

Driver(<u>driverID</u>, name, status, isHandicapped)

driverID is the primary key.
name, status and isHandicapped are NOT NULL.

Vehicle(licenseNum, manufacturer, color, model, year)

lisceneNum is the primary key;
manufacturer, color, model and year are NOT NULL.

Permit(permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot, zoneID)

permitID is the primary key.
lot is the foreign key which references ParkingLot(lotID)
spaceType, startDate, expirationDate, permitType, expirationTime, lot and zoneID are NOT NULL.

Citation(citationNum, citationTime, citationDate, lot, paymentStatus, category)

citationNum is the primary key.
lot is the foreign key which references ParkingLot(lotID)
category is the foreign key which references CitationFee(category)
citationTime, citationDate, lot, paymentStatus and category are NOT NULL.

CitationFee(category, fee)

category is the primary key.
fee is NOT NULL

AllottedTo(permitID, driverID)

permitID is one of the two primary keys.
driverID is the second among the two primary keys.
driverID is the foreign key which references Driver(driverID)

Appeals(driverID, citationNum)

driverID is one of the two primary keys.
citationNum is the second among the two primary keys.
driverID is the foreign key which references Driver(driverID)
citationNum is the foreign key which references Citation(citationNum)

Settles(driverID, citationNum, payID)

driverID is one of the two primary keys.
citationNum is the second among the two primary keys.
driverID is the foreign key which references Driver(driverID)
citationNum is the foreign key which references Citation(citationNum)

payID is allowed to be NULL in the scenario where the driver has not yet paid for the citation issued or has proposed for an appeal. This can be updated later when the driver makes the payment.

RaisedTo(citationNum, licenseNum)

citationNum is one of the two primary keys.
licenseNum is the second among the two primary keys.
citationNum is the foreign key which references Citation(citationNum)
licenseNum is the foreign key which references Vehicle(licenseNum)

Has(driverID, licenseNum)

driverID is one of the two primary keys.
licenseNum is the second among the two primary keys.
driverID is the foreign key which references Driver(driverID)
licenseNum is the foreign key which references Vehicle(licenseNum)

Security(securityID)

securityID is the primary key.

Administrator(adminID)

adminID is the primary key.

Handles(securityID, citationNum)

securityID is one of the two primary keys.
citationNum is the second among the two primary keys.
citationNum is the foreign key which references Citation(citationNum)
securityID is the foreign key which references Security(securityID)

Parking Lot(lotID, address, name)

lotID is the primary key.
address and name are NOT NULL.

Zone(zoneID, lotID, zone_name)

zoneID is one of the two primary keys.
lotID is the second among the two primary keys.
lot is the foreign key which references ParkingLot(lotID)

zone_name is NOT NULL

Space(spaceNum, zoneID, lotID, spaceType, availabilityStatus)

spaceNum is one of the three primary keys.
zoneID is the second among the three primary keys.
lotID is the third among the three primary keys.
lot is the foreign key which references ParkingLot(lotID)
zoneID is the foreign key which references Zone(zoneID)
spaceType and availabilityStatus are NOT NULL.

Manages(permitID, adminID)

permitID is one of the two primary keys.
adminID is the second among the two primary keys.
permitID is the foreign key which references Permit(permitID)
adminID is the foreign key which references Administrator(adminID)

ParkedAt(lotID, licenseNum, zoneID, spaceNum)

lotID is one of the two primary keys.
licenseNum is the second among the two primary keys.
lot is the foreign key which references ParkingLot(lotID)
licenseNum is the foreign key which references Vehicle(licenseNum)
spaceNum and zoneID are NOT NULL.

Controls(adminID, lotID)

adminID is one of the two primary keys.
lotID is the second among the two primary keys.
adminID is the foreign key which references Administrator(adminID)
lot is the foreign key which references ParkingLot(lotID)

3. **Base Relations:**

```
CREATE TABLE Driver (
   driverID INT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   status VARCHAR(50) NOT NULL,
   isHandicapped VARCHAR(5) NOT NULL
);
```

```
CREATE TABLE Vehicle (
   licenseNum VARCHAR(15) PRIMARY KEY,
   manufacturer VARCHAR(100)NOT NULL,
   color VARCHAR(50) NOT NULL,
   model VARCHAR(100) NOT NULL,
   year INT NOT NULL
);

CREATE TABLE Permit (
   permitID INT PRIMARY KEY,
   spaceType VARCHAR(50) NOT NULL,
   startDate DATE NOT NULL,
   expirationDate DATE NOT NULL,
   permitType VARCHAR(50) NOT NULL,
   expirationTime TIME NOT NULL,
   lot INT NOT NULL,
   zoneID VARCHAR(2) NOT NULL,
   FOREIGN KEY (lot) REFERENCES ParkingLot(lotID)
   ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Citation (
   citationNum INT PRIMARY KEY,
   citationTime TIME NOT NULL,
```

```
    citationDate DATE NOT NULL,
    lot INT NOT NULL,
    paymentStatus VARCHAR(20) NOT NULL,
    category VARCHAR(50) NOT NULL,
    FOREIGN KEY (lot) REFERENCES ParkingLot(lotID),
    FOREIGN KEY (category) REFERENCES CitationFee(category)
    ON UPDATE CASCADE
);

CREATE TABLE CitationFee(
     category VARCHAR(50) PRIMARY KEY,
     fee DECIMAL(10, 2) NOT NULL
  );

CREATE TABLE AllottedTo (
    permitID INT,
    driverID INT,
    PRIMARY KEY (permitID, driverID),
    FOREIGN KEY (permitID) REFERENCES Permit(permitID)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (driverID) REFERENCES Driver(driverID)
    ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Appeals (
    driverID INT,
    citationNum INT,
    PRIMARY KEY (driverID, citationNum),
    FOREIGN KEY (driverID) REFERENCES Driver(driverID)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (citationNum) REFERENCES Citation(citationNum)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

```
CREATE TABLE Settles (
   driverID INT,
   citationNum INT,
   payID INT,
   PRIMARY KEY (driverID, citationNum),
   FOREIGN KEY (driverID) REFERENCES Driver(driverID)
   ON UPDATE CASCADE,
   FOREIGN KEY (citationNum) REFERENCES Citation(citationNum)
   ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE RaisedTo (
   citationNum INT,
   licenseNum VARCHAR(15),
   PRIMARY KEY (citationNum, licenseNum),
   FOREIGN KEY (citationNum) REFERENCES Citation(citationNum)
   ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (licenseNum) REFERENCES Vehicle(licenseNum)
   ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Has (
   driverID INT,
   licenseNum VARCHAR(15),
   PRIMARY KEY (driverID, licenseNum),
   FOREIGN KEY (driverID) REFERENCES Driver(driverID)
   ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (licenseNum) REFERENCES Vehicle(licenseNum)
   ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Administrator(
   adminID INT PRIMARY KEY
);
```

```
CREATE TABLE Security (
   securityID INT PRIMARY KEY
);

CREATE TABLE Handles (
 securityID INT,
 citationNum INT,
 PRIMARY KEY (securityID, citationNum),
 FOREIGN KEY (securityID) REFERENCES Security(securityID)
 ON UPDATE CASCADE ON DELETE CASCADE,
 FOREIGN KEY (citationNum) REFERENCES Citation(citationNum)
 ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE ParkingLot (
  lotID INT PRIMARY KEY,
  address VARCHAR(255) NOT NULL,
  name VARCHAR(100) NOT NULL
);

CREATE TABLE Zone (
   zoneID VARCHAR(2),
   zone_name VARCHAR(25),
   lotID INT,
   PRIMARY KEY (zoneID, lotID),
   FOREIGN KEY (lotID) REFERENCES ParkingLot(lotID)
   ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Space (
   spaceNum INT,
   zoneID VARCHAR(2),
   lotID INT,
   spaceType VARCHAR(50) NOT NULL,
   availabilityStatus VARCHAR(20)NOT NULL,
   PRIMARY KEY (spaceNum, zoneID, lotID),
```

```
    FOREIGN KEY (zoneID) REFERENCES Zone(zoneID)
    ON UPDATE CASCADE ON DELETE CASCADE,
   FOREIGN KEY (lotID) REFERENCES ParkingLot(lotID)
    ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Manages (
    permitID INT,
    adminID INT,
    PRIMARY KEY (permitID, adminID),
    FOREIGN KEY (permitID) REFERENCES Permit(permitID)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (adminID) REFERENCES Administrator(adminID)
    ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE ParkedAt (
    lotID INT,
    licenseNum VARCHAR(15),
    PRIMARY KEY (lotID, licenseNum),
    FOREIGN KEY (lotID) REFERENCES ParkingLot(lotID)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (licenseNum) REFERENCES Vehicle(licenseNum)
    ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Controls (
    adminID INT,
    lotID INT,
    PRIMARY KEY (adminID, lotID),
    FOREIGN KEY (adminID) REFERENCES Administrator(adminID)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (lotID) REFERENCES ParkingLot(lotID)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

SELECT * FROM Driver;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Driver;
+----------+----------------+--------+---------------+
| driverID | name           | status | isHandicapped |
+----------+----------------+--------+---------------+
|        1 | John Doe       | E      | No            |
|        2 | Jane Smith     | S      | No            |
|        3 | Bob Johnson    | E      | No            |
|        4 | Alice Brown    | S      | Yes           |
|        5 | Charlie Wilson | V      | No            |
|        6 | Eva Davis      | S      | No            |
|        7 | Frank Clark    | E      | No            |
|        8 | Grace White    | S      | No            |
|        9 | Henry Lee      | E      | No            |
|       10 | Isabel Hall    | S      | No            |
+----------+----------------+--------+---------------+
10 rows in set (0.0011 sec)
```

SELECT * FROM Vehicle;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Vehicle;
+------------+--------------+--------+---------+------+
| licenseNum | manufacturer | color  | model   | year |
+------------+--------------+--------+---------+------+
| ABC123     | Toyota       | Blue   | Camry   | 2018 |
| DEF456     | Ford         | Silver | Focus   | 2019 |
| GHI987     | Hyundai      | Green  | Elantra | 2016 |
| JKL987     | Chevrolet    | Black  | Malibu  | 2021 |
| LMN654     | Volkswagen   | Blue   | Jetta   | 2018 |
| MNO654     | Nissan       | White  | Altima  | 2017 |
| PQR321     | Subaru       | Gray   | Outback | 2022 |
| STU456     | Kia          | Silver | Forte   | 2019 |
| VWX123     | Mazda        | Red    | CX-5    | 2020 |
| XYZ789     | Honda        | Red    | Civic   | 2020 |
+------------+--------------+--------+---------+------+
10 rows in set (0.0030 sec)
```

SELECT * FROM ParkingLot;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM ParkingLot;
+-------+---------------+--------+
| lotID | address       | name   |
+-------+---------------+--------+
|     1 | 123 Main St   | Lot 1  |
|     2 | 456 Elm St    | Lot 2  |
|     3 | 789 Oak St    | Lot 3  |
|     4 | 101 Pine St   | Lot 4  |
|     5 | 222 Maple St  | Lot 5  |
|     6 | 333 Birch St  | Lot 6  |
|     7 | 444 Cedar St  | Lot 7  |
|     8 | 555 Walnut St | Lot 8  |
|     9 | 666 Oak St    | Lot 9  |
|    10 | 777 Cherry St | Lot 10 |
+-------+---------------+--------+
10 rows in set (0.0012 sec)
```

SELECT * FROM Zone;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Zone;
+--------+----------------------+-------+
| zoneID | zone_name            | lotID |
+--------+----------------------+-------+
| A      | Zone A               |     1 |
| AS     | Student Zone A       |     3 |
| B      | Zone B               |     1 |
| BS     | Student Zone B       |     3 |
| C      | Zone C               |     2 |
| CS     | Student Zone C       |     4 |
| D      | Zone D               |     2 |
| DS     | Student Zone D       |     4 |
| V      | Visitor Zone         |     5 |
| VS     | Student Visitor Zone |     5 |
+--------+----------------------+-------+
10 rows in set (0.0014 sec)
```

SELECT * FROM Space;

```
MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Space;
+----------+--------+-------+-----------+--------------------+
| spaceNum | zoneID | lotID | spaceType | availabilityStatus |
+----------+--------+-------+-----------+--------------------+
|        1 | A      |     1 | Regular   | Available          |
|        2 | A      |     1 | Compact   | Available          |
|        3 | B      |     1 | Regular   | Available          |
|        4 | B      |     1 | Compact   | Occupied           |
|        5 | AS     |     3 | Regular   | Available          |
|        6 | AS     |     3 | Electric  | Available          |
|        7 | V      |     5 | Regular   | Available          |
|        8 | V      |     5 | Compact   | Available          |
|        9 | VS     |     5 | Regular   | Available          |
|       10 | VS     |     5 | Compact   | Occupied           |
+----------+--------+-------+-----------+--------------------+
10 rows in set (0.0013 sec)
```

SELECT * FROM Permit;

```
MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Permit;
+----------+-----------+------------+----------------+--------------+----------------+-----+
| permitID | spaceType | startDate  | expirationDate | permitType   | expirationTime | lot |
+----------+-----------+------------+----------------+--------------+----------------+-----+
|        1 | Regular   | 2023-01-01 | 2023-12-31     | residential  | 23:59:59       |   1 |
|        2 | Compact   | 2023-01-01 | 2023-12-31     | peak hours   | 23:59:59       |   2 |
|        3 | Regular   | 2023-03-01 | 2023-08-31     | residential  | 23:59:59       |   3 |
|        4 | Handicap  | 2023-02-15 | 2023-12-15     | peak hours   | 23:59:59       |   4 |
|        5 | Regular   | 2023-04-01 | 2023-09-30     | special event| 23:59:59       |   5 |
|        6 | Electric  | 2023-05-01 | 2023-10-31     | residential  | 23:59:59       |   6 |
|        7 | Regular   | 2023-06-01 | 2023-11-30     | commuter     | 23:59:59       |   7 |
|        8 | Compact   | 2023-01-01 | 2023-12-31     | commuter     | 23:59:59       |   8 |
|        9 | Regular   | 2023-07-01 | 2023-12-31     | residential  | 23:59:59       |   9 |
|       10 | Compact   | 2023-01-01 | 2023-12-31     | park & ride  | 23:59:59       |  10 |
+----------+-----------+------------+----------------+--------------+----------------+-----+
10 rows in set (0.0012 sec)
```

SELECT * FROM CitationFee;

```
MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM CitationFee;
+----------------+-------+
| category       | fee   |
+----------------+-------+
| Expired Permit | 30.00 |
| Invalid Permit | 25.00 |
| No Permit      | 40.00 |
+----------------+-------+
3 rows in set (0.0066 sec)
```

SELECT * FROM Citation;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Citation;
+-------------+--------------+--------------+-----+---------------+----------------+
| citationNum | citationTime | citationDate | lot | paymentStatus | category       |
+-------------+--------------+--------------+-----+---------------+----------------+
|           1 | 08:30:00     | 2023-10-15   |   1 | Unpaid        | Expired Permit |
|           2 | 10:45:00     | 2023-10-18   |   2 | Paid          | Invalid Permit |
|           3 | 11:15:00     | 2023-10-18   |   3 | Unpaid        | No Permit      |
|           4 | 12:30:00     | 2023-10-20   |   4 | Unpaid        | Expired Permit |
+-------------+--------------+--------------+-----+---------------+----------------+
4 rows in set (0.0013 sec)
```

SELECT * FROM AllottedTo;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM AllottedTo;
+----------+----------+
| permitID | driverID |
+----------+----------+
|        1 |        1 |
|        2 |        2 |
|        3 |        3 |
|        4 |        4 |
|        5 |        5 |
+----------+----------+
5 rows in set (0.0014 sec)
```

SELECT * FROM Appeals;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Appeals;
+----------+-------------+
| driverID | citationNum |
+----------+-------------+
|        1 |           1 |
|        2 |           2 |
|        3 |           3 |
|        4 |           4 |
+----------+-------------+
4 rows in set (0.0012 sec)
```

SELECT * FROM Settles;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Settles;
+----------+-------------+-------+
| driverID | citationNum | payID |
+----------+-------------+-------+
|        1 |           1 |     1 |
|        2 |           2 |     2 |
|        3 |           3 |  NULL |
|        4 |           4 |     4 |
+----------+-------------+-------+
4 rows in set (0.0013 sec)
```

SELECT * FROM RaisedTo;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM RaisedTo;
+-------------+------------+
| citationNum | licenseNum |
+-------------+------------+
|           1 | ABC123     |
|           2 | XYZ789     |
|           3 | DEF456     |
|           4 | JKL987     |
+-------------+------------+
4 rows in set (0.0017 sec)
```

SELECT * FROM Has;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Has;
+----------+------------+
| driverID | licenseNum |
+----------+------------+
|        1 | ABC123     |
|        2 | XYZ789     |
|        3 | DEF456     |
|        4 | JKL987     |
|        5 | MNO654     |
+----------+------------+
5 rows in set (0.0017 sec)
```

SELECT * FROM Security;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Security;
+------------+
| securityID |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
5 rows in set (0.0013 sec)
```

SELECT * FROM Administrator;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Administrator;
+---------+
| adminID |
+---------+
|       1 |
|       2 |
|       3 |
|       4 |
|       5 |
+---------+
5 rows in set (0.0013 sec)
```

SELECT * FROM Handles;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Handles;
+------------+-------------+
| securityID | citationNum |
+------------+-------------+
|          1 |           1 |
|          2 |           2 |
|          3 |           3 |
|          4 |           4 |
+------------+-------------+
4 rows in set (0.0013 sec)
```

SELECT * FROM Manages;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Manages;
+----------+---------+
| permitID | adminID |
+----------+---------+
|        1 |       1 |
|        2 |       2 |
|        3 |       1 |
|        4 |       3 |
|        5 |       2 |
+----------+---------+
5 rows in set (0.0013 sec)
```

SELECT * FROM ParkedAt;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM ParkedAt;
+-------+------------+
| lotID | licenseNum |
+-------+------------+
|     1 | ABC123     |
|     2 | XYZ789     |
|     3 | DEF456     |
|     4 | JKL987     |
|     5 | MNO654     |
+-------+------------+
5 rows in set (0.0013 sec)
```

SELECT * FROM Controls;

```
 MySQL  classdb2.csc.ncsu.edu:3306  arjatava  SQL > SELECT * FROM Controls;
+---------+-------+
| adminID | lotID |
+---------+-------+
|       1 |     1 |
|       2 |     2 |
|       3 |     3 |
|       4 |     4 |
|       5 |     5 |
+---------+-------+
5 rows in set (0.0015 sec)
```

4. SQL Queries:

4.1. Information Processing:

- **Adding Driver Information:**

INSERT INTO Driver (driverID, name, status, isHandicapped)

VALUES (16, 'Hopeton', 'E', 'No');

> Query OK, 1 row affected (0.0024 sec)

- **Delete Driver Information:**

DELETE FROM Driver

WHERE driverID = 16;

> Query OK, 1 row affected (0.0024 sec)

- **Update Driver Information:**

UPDATE Driver

SET name = 'John Pope'

WHERE driverID= 1;

> Query OK, 1 row affected (0.0022 sec)

Rows matched: 1 Changed: 1 Warnings: 0

- **Get Driver Information:**

SELECT * FROM Driver

WHERE driverID = 16;

> Query OK, 1 row affected (0.0122 sec)

- **Add Parking Lot Information:**

INSERT INTO ParkingLot (lotID, address, name)

VALUES (11, '123 Russell Rd', 'Avent Ferry Lot');

> Query OK, 1 row affected (0.0025 sec)

- **Delete Parking Lot Information:**

DELETE FROM ParkingLot

WHERE lotID = 11;

> Query OK, 1 row affected (0.0122 sec)

- **Update Parking Lot Information:**

UPDATE ParkingLot

SET address = '2510 Main St'

WHERE lotID= 10;

> Query OK, 1 row affected (0.0031 sec)

Rows matched: 1 Changed: 1 Warnings: 0

- **Get Parking Lot Information:**

SELECT * FROM ParkingLot

WHERE name = 'Lot 6';

> Query OK, 1 row affected (0.0122 sec)

● **Add Zone Information:**

INSERT INTO Zone (zoneID, lotID, zone_name)

VALUES (‘V’ , 3, ‘Wolf Zone’) ;

> Query OK, 1 row affected (0.0026 sec)

● **Get Zone**

SELECT * FROM Zone

WHERE zoneID = ‘V’ AND lotID = 13 AND zone_name = ‘Wolf zone’;

> Query OK, 0 rows affected (0.0025 sec)

● **Delete Zone Information:**

DELETE FROM Zone

WHERE zoneID = ‘V’ AND lotID = 13 AND zone_name = ‘Wolf zone’;

> Query OK, 0 rows affected (0.0014 sec)

● **Update Zone Information:**

UPDATE Zone

SET zone_name= ‘Cary Zone’

WHERE zoneID= ‘V’ AND lotID = 3;

> Query OK, 1 row affected (0.0023 sec)

Rows matched: 1 Changed: 1 Warnings: 0

- **Add Space Information:**

INSERT INTO Space(spaceNum, zoneID, lotID, spaceType, availabilityStatus)

VALUES (11, 'AS', 1, 'Electric', 'Available');

> Query OK, 1 row affected (0.0024 sec)

- **Get Space Information:**

SELECT * FROM Space

WHERE spaceNum= 11 AND zoneID = 'AS' AND lotID = 1;

> Query OK, 1 row affected (0.0024 sec)

- **Delete Space Information:**

DELETE FROM Space

WHERE spaceNum= 11 AND zoneID = 'AS' AND lotID = 1;

> Query OK, 1 row affected (0.0026 sec)

- **Update Space Information:**

UPDATE Space

SET spaceType= 'Regular'

WHERE spaceNum= 5 AND zoneID = 'AS' AND lotID = 3;

> Query OK, 0 rows affected (0.0025 sec)

Rows matched: 1 Changed: 0 Warnings: 0

● **Add Permit Information:**

INSERT INTO Permit (permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot)

VALUES (11, 'regular', '2023-10-20', '2024-10-20', 'special event', '12:00:00', 2);

> Query OK, 1 row affected (0.0026 sec)

● **Update Permit Information:**

UPDATE Permit

SET expirationDate = '2025-10-20'

WHERE permitID = 11;

> Query OK, 1 row affected (0.0023 sec)

Rows matched: 1 Changed: 1 Warnings: 0

● **Get Permit Information:**

SELECT * FROM Permit

WHERE permitID = 11;

> Query OK, 1 row affected (0.0026 sec)

● **Delete Permit Information:**

DELETE FROM Permit

WHERE permitID = 11;

> Query OK, 1 row affected (0.0024 sec)

● **Add Vehicle Owner Information:**

INSERT INTO Has (driverID, licenseNum)

VALUES (11, 'HDK173');

> Query OK, 1 row affected (0.0023 sec)

● **Get Vehicle Owner Information:**

SELECT * FROM Has

WHERE driverID = 10 AND licenseNum = 'PQR321';

> Query OK, 1 row affected (0.0026 sec)

● **Delete Vehicle Owner Information:**

DELETE FROM Has

WHERE driverID = 10 AND licenseNum = 'PQR321';

> Query OK, 1 row affected (0.0024 sec)

Maintaining Permits and Vehicle Information:

● **Assign Permit to Driver:**

INSERT INTO AllottedTo (permitID, driverID)

VALUES (6,6);

> Query OK, 1 row affected (0.0022 sec)

● **Update Permit Information:**

UPDATE Permit

SET permitType = 'residential',

expirationTime = '10:00:00',

expirationDate = '2023-12-31',

startDate = '2023-01-01',

spaceType = 'Regular',

lot = 10

WHERE permitID = 6;

> Query OK, 1 row affected (0.0027 sec)

Rows matched: 1 Changed: 1 Warnings: 0

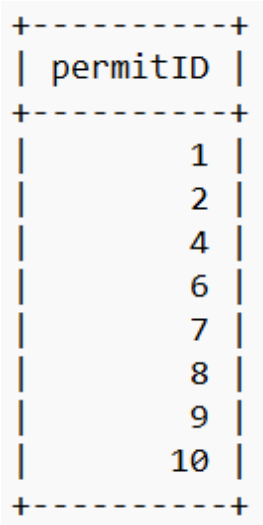
● **Check Permit Validity:**

SELECT permitID

FROM Permit

WHERE (expirationDate > CURDATE())

OR (expirationDate = CURDATE() AND expirationTime > CURTIME());



permitID
1
2
4
6
7
8
9
10

● **Update Vehicle Information:**

UPDATE Vehicle

SET color = 'Blue',

manufacturer = 'Toyota',

model = 'Camry',

year = 2020

WHERE licenseNum= 'HDK173';

> Query OK, 1 row affected (0.0027 sec)

Rows matched: 1 Changed: 1 Warnings: 0

● **Add Vehicle Information:**

INSERT INTO Vehicle (licenseNum, manufacturer, color, model, year)

VALUES ('BSY420', 'Honda', 'Red', 'Civic', 2019);

> Query OK, 1 row affected (0.0032 sec)

● **Get Vehicle Information:**

SELECT * FROM Vehicle

WHERE licenseNum = "PQR321"

```
+------------+--------------+-------+-------+------+
| licenseNum | manufacturer | color | model | year |
+------------+--------------+-------+-------+------+
| PQR321     | Toyota       | Blue  | Camry | 2020 |
+------------+--------------+-------+-------+------+
1 row in set (0.0013 sec)
```

● **Remove Vehicle Information:**

DELETE FROM Vehicle

WHERE licenseNum = ‘BSY420’;

> Query OK, 1 row affected (0.0023 sec)

Generating and Maintaining Citations

● **Generate Citation:**

INSERT INTO Citation (citationNum, citationTime, citationDate, lot, paymentStatus, category)

VALUES (105, ‘10:00:00’, ‘2023-12-31’, 2, 'Paid', 'Expired Permit');

> Query OK, 1 row affected (0.0025 sec)

● **Detect Parking Violation: Returns cars that are currently in parking violation before generating a citation**

```
SELECT DISTINCT pa.licenseNum
FROM ParkedAt pa
LEFT JOIN Permit pe ON pa.lotID = pe.lot
WHERE pa.lotID IS NULL
  OR pa.zoneID IS NULL
  OR pa.spaceNum IS NULL
  OR pe.permitID IS NULL
  OR pa.lotID != pe.lot
  OR pa.zoneID != pe.zoneID
  OR pa.spaceNum NOT IN (
    SELECT spaceNum
    FROM Permit
    WHERE zoneID = pa.zoneID AND lot = pa.lotID
  );
```

```
+-------------+
| licenseNum  |
+-------------+
| ABC123      |
| DEF456      |
| JKL987      |
+-------------+
3 rows in set (0.0043 sec)
```

- **Return the count of number of cars that are currently in violation:**

```
SELECT COUNT(*)
FROM ParkedAt pa
LEFT JOIN Permit pe ON pa.lotID = pe.lot
WHERE pa.lotID IS NULL
  OR pa.zoneID IS NULL
  OR pa.spaceNum IS NULL
  OR pe.permitID IS NULL
  OR pa.lotID != pe.lot
  OR pa.zoneID != pe.zoneID
  OR pa.spaceNum NOT IN (
    SELECT spaceNum
    FROM Permit
    WHERE zoneID = pa.zoneID AND lot = pa.lotID
  );
```

- **Settle Citation:**

```
INSERT INTO Settles (driverID, citationNum, payID)

VALUES (5, 105, NULL);
```

> Query OK, 1 row affected (0.0025 sec)

- **Appeal Citation:**

```
INSERT INTO Appeals (driverID, citationNum)
```

VALUES (5, 105);

> Query OK, 1 row affected (0.0026 sec)

- **Update Citation Information:**

UPDATE Citation SET paymentStatus = 'Paid' WHERE citationNum = 105;

> Query OK, 0 rows affected (0.0031 sec)

Rows matched: 1 Changed: 0 Warnings: 0

- **Delete Citation Information:**

DELETE FROM Citation WHERE citationNum = 106 ;

> Query OK, 1 row affected (0.0186 sec)

- **Get Citations for a Particular Driver:**

SELECT R.citationNum

FROM RaisedTo R

JOIN Has H on R.licenseNum = H.licenseNum AND H.driverID = 3;

```
+-------------+
| citationNum |
+-------------+
|           3 |
+-------------+
1 row in set (0.0015 sec)
```

- **Generate Citation Fee:**

```
SELECT c.citationNum,
       c.citationDate,
       c.citationTime,
       c.lot,
       c.paymentStatus,
```

```
        c.category,
        d.driverID, -- Include driverID in the result
        cf.fee AS actualFee,
          -- Apply discount if the driver is handicapped
  CASE WHEN d.isHandicapped = 'Yes' THEN cf.fee * 0.5
  ELSE cf.fee
  END AS discountedFee
  FROM
    Citation c
  JOIN
    CitationFee cf ON c.category = cf.category
 LEFT JOIN
    Appeals a ON c.citationNum = a.citationNum
LEFT JOIN
    Driver d ON a.driverID = d.driverID
WHERE
    c.paymentStatus = 'Unpaid';
```

citationNum	citationDate	citationTime	lot	paymentStatus	category	driverID	actualFee	discountedFee
1	2023-10-15	08:30:00	1	Unpaid	Expired Permit	1	30.00	30.000
4	2023-10-20	12:30:00	4	Unpaid	Expired Permit	4	30.00	15.000
151	2023-11-12	12:00:45	2	Unpaid	Expired Permit	NULL	30.00	30.000
3	2023-10-18	11:15:00	3	Unpaid	No Permit	3	40.00	40.000
3	2023-10-18	11:15:00	3	Unpaid	No Permit	4	40.00	20.000

5 rows in set (0.0045 sec)

- **Return the number of employees having permits for a given parking zone:**

```
SELECT COUNT(DISTINCT d.driverID) AS NumberOfEmployees
FROM Zone z
JOIN Permit p ON z.lotID = p.lot
JOIN AllottedTo a ON a.permitID = p.permitID
JOIN Driver d ON a.driverID = d.driverID
WHERE d.status = "E" and z.zoneID = ‘A’;
```

NumberOfEmployees
1

1 row in set (0.0019 sec)

- **Return an available space number given a space type in a given parking lot:**

```
SELECT s.spaceNum
FROM Space s
JOIN ParkingLot p ON s.lotID = p.lotID
WHERE s.spaceType = ‘Regular’ AND s.lotID = 1
AND s.availabilityStatus = 'Available';
```

```
+----------+
| spaceNum |
+----------+
|        1 |
|        3 |
+----------+
2 rows in set (0.0015 sec)
```

Reports:

- **For each lot, 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):**

```
SELECT l.lotID AS LotID,
COUNT(c.citationNum) AS TotalCitations
FROM ParkingLot l
JOIN Citation c ON l.lotID = c.lot
WHERE c.citationDate BETWEEN '2023-10-01' AND '2023-10-30'
GROUP BY l.lotID
ORDER BY l.lotID;
```

```
+-------+-----------------+
| LotID | TotalCitations |
+-------+-----------------+
|     1 |               1 |
|     2 |               1 |
|     3 |               1 |
|     4 |               1 |
+-------+-----------------+
4 rows in set (0.0015 sec)
```

- **Generate Citation Report:**

SELECT * FROM Citation WHERE lot = 2 AND citationDate BETWEEN '2023-10-01' AND '2023-12-31';

```
+-------------+--------------+--------------+-----+---------------+-----------------+
| citationNum | citationTime | citationDate | lot | paymentStatus | category        |
+-------------+--------------+--------------+-----+---------------+-----------------+
|           2 | 10:45:00     | 2023-10-18   |   2 | Paid          | Invalid Permit |
|         105 | 00:00:00     | 2023-12-31   |   2 | Paid          | Expired Permit |
+-------------+--------------+--------------+-----+---------------+-----------------+
```

- **Return permit information for a given ID:**

SELECT p.permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot, zoneID

FROM Permit p

JOIN AllottedTo ato ON p.permitID = ato.permitID

WHERE ato.driverID = 5;

```
+----------+-----------+------------+----------------+--------------+----------------+-----+
| permitID | spaceType | startDate  | expirationDate | permitType   | expirationTime | lot |
+----------+-----------+------------+----------------+--------------+----------------+-----+
|        5 | Regular   | 2023-04-01 | 2023-09-30     | special event | 23:59:59       |   5 |
+----------+-----------+------------+----------------+--------------+----------------+-----+
1 row in set (0.0015 sec)
```

- **Return the list of zones for each lot as tuple pairs (lot, zone).**

  SELECT lotID, zoneID FROM Zone GROUP BY lotID,zoneID;

lotID	zoneID
1	A
1	B
2	C
2	D
3	AS
3	BS
3	V
4	CS
4	DS
5	V
5	VS
13	V

4.2.

Get Driver Information:

SELECT * FROM Driver WHERE driverID = 16;

1. EXPLAIN SELECT * FROM Driver WHERE name = "John Pope";

2.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	Driver	ALL	NULL	NULL	NULL	NULL	10	Using where

1 row in set (0.0013 sec)

3. CREATE INDEX idxname ON Driver(name);

> Query OK, 0 rows affected (0.0235 sec)

4.

```
+----+-------------+--------+------+---------------+---------+---------+-------+------+-----------------------+
| id | select_type | table  | type | possible_keys | key     | key_len | ref   | rows | Extra                 |
+----+-------------+--------+------+---------------+---------+---------+-------+------+-----------------------+
|  1 | SIMPLE      | Driver | ref  | idxname       | idxname | 257     | const |    1 | Using index condition |
+----+-------------+--------+------+---------------+---------+---------+-------+------+-----------------------+
1 row in set (0.0014 sec)
```

Get Parking Lot Information:

SELECT * FROM ParkingLot WHERE name = 'Lot 6';

1. EXPLAIN SELECT * FROM ParkingLot WHERE name = 'Lot 6';

2.

```
+----+-------------+------------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table      | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+------------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | ParkingLot | ALL  | NULL          | NULL | NULL    | NULL |   11 | Using where |
+----+-------------+------------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.0047 sec)
```

3. CREATE INDEX idxlotname ON ParkingLot(name);

> Query OK, 0 rows affected (0.0706 sec)

4.

```
+----+-------------+------------+------+---------------+------------+---------+-------+------+-----------------------+
| id | select_type | table      | type | possible_keys | key        | key_len | ref   | rows | Extra                 |
+----+-------------+------------+------+---------------+------------+---------+-------+------+-----------------------+
|  1 | SIMPLE      | ParkingLot | ref  | idxlotname    | idxlotname | 102     | const |    1 | Using index condition |
+----+-------------+------------+------+---------------+------------+---------+-------+------+-----------------------+
1 row in set (0.0051 sec)
```

4.3

1) Return permit information for a given ID:

SELECT p.permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot, zoneID

FROM Permit p

JOIN AllottedTo ato ON p.permitID = ato.permitID

WHERE ato.driverID = 5;

$$\Pi_{\text{permitID, spaceType, startDate, expirationDate, permitType, expirationTime, lot}} (\sigma_{\text{driverID=5}} (\rho_{p}(\text{Permit}) \bowtie_{\text{p.permitID=ato.permitID}} \rho_{ato}(\text{AllottedTo}))$$

Suppose p is any tuple in the "Permit" relation, and t is any tuple in the "AllottedTo" relation, such that the value of p.permitID and t.permitID is the same. The combination of the tuples p and t will provide all permit information for a permit that is allotted to the driver with ID 5. This combination will return the permit's ID, space type, start date, expiration date, permit type, expiration time, and the parking lot associated with the permit. By looking at the combination of whether the permit IDs match between the "Permit" and "AllottedTo" tables, we obtain all the permit information for drivers assigned to driver ID 5. This query accomplishes the task of retrieving the specific permit details associated with a particular driver, which is exactly what was required.

2) **Return an available space number given a space type in a given parking lot:**

```
SELECT s.spaceNum
FROM Space s
JOIN ParkingLot p ON s.lotID = p.lotID
WHERE s.spaceType = ‘Regular’ AND s.lotID = 1
AND s.availabilityStatus = 'Available';
```

$$\pi_{\text{s.spaceNum}} (\sigma_{\text{s.spaceType = 'Regular' AND s.lotID = 1 AND s.availabilityStatus = 'Available'}} (\rho_{s}(\text{Space}) \bowtie_{\text{s.lotID = p.lotID}} \rho_{p}(\text{ParkingLot}))$$

Suppose s is any tuple in the "Space" relation, and p is any tuple in the "ParkingLot" relation, such that the value of s.lotID and p.lotID is the same. The combination of the tuples s and p will provide all information about parking spaces that are associated with parking lot ID 1 and have a space type of 'Regular' and an availability status of 'Available.' This combination will return the space numbers (s.spaceNum) for these parking spaces. By looking at the combination of whether

the lot IDs match between the "Space" and "ParkingLot " tables and whether the space type and availability status meet the specified criteria, we obtain the list of space numbers that meet these conditions. This query accomplishes the task of retrieving the specific space numbers associated for a given parking lot and meeting the defined criteria for space type (‘Regular’) and availability status (‘Available’), which is exactly what was required.

Transaction 1: Add Vehicle

```
   private static void addVehicle() throws SQLException {
         Connection connection = null;
       try {
         connection = DriverManager.getConnection(JDBC_URL, USER,
PASSWORD);
           Scanner scanner = new Scanner(System.in);

        // Disable autocommit to start a transaction
           connection.setAutoCommit(false);
           System.out.print("Enter License Number: ");
           String licenseNum = scanner.nextLine();

           System.out.print("Enter Manufacturer: ");
           String manufacturer = scanner.nextLine();

           System.out.print("Enter Color: ");
           String color = scanner.nextLine();

           System.out.print("Enter Model: ");
           String model = scanner.nextLine();

           System.out.print("Enter Year: ");
           int year = scanner.nextInt();
```

```
        // Insert new vehicle into the Vehicle table
        String insertQuery = "INSERT INTO Vehicle (licenseNum, manufacturer, color, model, year) VALUES (?, ?, ?, ?, ?)";
        PreparedStatement preparedStatement = connection.prepareStatement(insertQuery);
            preparedStatement.setString(1, licenseNum);
            preparedStatement.setString(2, manufacturer);
            preparedStatement.setString(3, color);
            preparedStatement.setString(4, model);
            preparedStatement.setInt(5, year);

            int rowsAffected = preparedStatement.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Vehicle added successfully!");
            } else {
                System.out.println("Failed to add the vehicle.");
            }
            // Commit the transaction
            connection.commit();
        }
        catch (SQLException e) {
                // Rollback the transaction in case of exception
                try {
                    if (connection != null) {
                        connection.rollback();
                    }
                } catch (SQLException rollbackException) {
                    rollbackException.printStackTrace();
                }

                e.printStackTrace();
```

```
    } finally {

        if (connection != null) {
            try {
                // Restore autocommit to true and close the connection
                connection.setAutoCommit(true);
                connection.close();
                System.out.println("Connection closed");
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Transaction 2: Add Parking Lot to the System

```
    private static void addParkingLot() throws SQLException {
        Connection connection = null;
        Scanner scanner = new Scanner(System.in);

        // Disable autocommit to start a transaction
    connection.setAutoCommit(false);

        System.out.print("Enter Lot ID: ");
        int lotID = scanner.nextInt();

        scanner.nextLine();

        System.out.print("Enter Lot Name: ");
        String lotName = scanner.nextLine();
```

```
        System.out.print("Enter Lot Address: ");
        String lotAddress = scanner.nextLine();

        // SQL query to add a parking lot to the system
        String sqlQuery = "INSERT INTO ParkingLot (lotID, name, address) VALUES (?, ?, ?)";

        Connection connection = null;
    try {
            connection = DriverManager.getConnection(JDBC_URL, USER, PASSWORD);
            PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery);

            preparedStatement.setInt(1, lotID);
            preparedStatement.setString(2, lotName);
            preparedStatement.setString(3, lotAddress);

            int rowsAffected = preparedStatement.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Parking Lot added successfully!");
            } else {
                System.out.println("Failed to add the parking lot.");
            }
            connection.commit();
        } catch (SQLException e) {
                    // Rollback the transaction in case of exception
                    try {
                        if (connection != null) {
                            connection.rollback();
                        }
```

```
                } catch (SQLException rollbackException) {
                    rollbackException.printStackTrace();
                }

                e.printStackTrace();
    } finally {

        if (connection != null) {
            try {
                // Restore autocommit to true and close the connection
                connection.setAutoCommit(true);
                connection.close();
                System.out.println("Connection closed");
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Design Decisions:

The system has a main menu that gives the user the following options: “Driver”, “Parking Lot, Space and Zone”, “Vehicle”, “Permit”, “Citation”, “Security”, “Administrator” and “Exit”. When prompted, the user enters a number 0-8 corresponding with what kind of action they would like to take. For each menu option there is a submenu displayed that has specific operations listed. Our system has a main class (DBMS_Demo.java) that facilitates switching between the main menu options and separate classes for each related group of operations. This was done to break the application into more manageable and maintainable pieces of code. Our program also contains two helper classes. One helper class is DriverManager, which, through the use of a shared database connection, provides the ability to issue SQL queries and perform transactions through various utility

methods. The other class is Scanner, which is a class that attempts to abstract the details of getting different types of input from the user. “Statement” and “PreparedStatement” were used to develop and execute SQL queries.

Functional Roles:

Part 1:

Software Engineer: Suraj (Prime), Aravinda (Backup)

Database Designer/Administrator: Aravinda (Prime), Prathyusha (Backup)

Application Programmer: Prathyusha (Prime), Yuktasree (Backup)

Test Plan Engineer: Yuktasree (Prime), Suraj (Backup)

Part 2:

Software Engineer: Yuktasree (Prime), Prathyusha (Backup)

Database Designer/Administrator: Prathyusha (Prime), Suraj (Backup)

Application Programmer: Suraj (Prime), Aravinda (Backup)

Test Plan Engineer: Aravinda (Prime), Yuktasree (Backup)

Part 3:

Software Engineer: Prathyusha (Prime), Suraj (Backup)

Database Designer/Administrator: Yuktasree (Prime), Prathyusha (Backup)

Application Programmer: Aravinda (Prime), Yuktasree (Backup)

Test Plan Engineer: Suraj (Prime), Aravinda (Backup)