Wolf Parking Management System

CSC 540 Database Management Systems

Project Report 1

Team Members:

Naga Jahnavi Kommareddy (nkommar)

Shail Shah (sshah38)

Shrishty Singh (ssingh67)

Shyamal Tarpanbhai Gandhi (sgandhi6)

Assumptions

- Parking lots consist of Zones. Zone Id is unique within a parking lot. Zones consist of spaces. Each space has a space number which is unique within a Zone.
- UnivID and phone number never clash. Single ID attribute can be used to store UnivID or phone number based on the status of the driver. UnivID is stored in the ID attribute in case of a student or employee and phone number is stored in case of a visitor.
- A vehicle can have at most one permit at any time.
- A permit is given for a particular space number in a particular zone in a particular parking lot.
- All appeals for parking citations are handled in person. Drivers must visit the security office, where they can submit their appeal and have it resolved on the spot.
- It is mandatory for the permit card to be visibly displayed on the vehicle whenever it is parked. Security personnel will identify parking violations by inspecting the displayed permit cards periodically.
- When a security guard records a citation in the database, they will also affix a physical ticket to the vehicle. This ticket contains details about the violation and provides instructions for payment.
- Drivers have the option to initiate the payment process online by entering the citation number, which is found on the ticket. They can then proceed to pay the required fine based on the violation.
- Only those vehicle information are stored that are currently on a permit or have had permits previously. Vehicle ownership information is not explicitly stored. Driver holding a permit currently is considered the owner of the vehicle.
- Handicap users getting 50% discount on all citations will be handled at the operations level.
- Restrictions on number of permits and number of vehicles on each permit based on the status of the driver will be handled at operations level.

1. (Change in entire ques 1)
Description:

The objective of the Wolf Parking Management System is to create a database for efficiently managing parking lots and their users on a university campus. Each parking area is subdivided into parking lots, zones, and spaces. Each space has a unique space number, a space type, and an availability status. Drivers are categorized as students, employees, or visitors, identified by their ID or phone number. Drivers can obtain permits specifying the zone, space type, license number, and permit type. Citations are issued for parking violations, varying based on different categories. Administrators have the authority to manage parking lots, assign zones and spaces, issue permits, check permit validity, and handle citation-related tasks. Users must possess a permit matching the lot's designated zones and space types to park. Users are limited to one permit for students and visitors and up to two permits for employees. The system's core tasks include information processing, permit and vehicle maintenance, citation generation, and report generation for various parking-related data, ensuring effective parking management on campus.

Reasons for DBMS

- **Consistency and Integrity:**
  - Prevents conflicts, and reduces the risk of erroneous data
- **Redundancy Reduction:**
  - Minimizes data redundancy through normalization
- **Concurrent Access:**
  - Efficiently handles concurrent access
- **Security:**
  - Robust access control and authentication mechanisms secure data
- **Scalability:**
  - Can handle growing datasets and maintain performance
- **Durability:**
  - Automated data backup, recovery, and fault tolerance mechanisms
- **Data Maintenance and Updates:**
  - SQL operations reduce the risk of data inconsistencies
- **Report Generation:**
  - Powerful querying and reporting tools

2. **Intended Users**:

Administrator: Admin can use parking services to maintain information related to the wolf parking management system. They have access to altering, deleting, updating the

information of any user. They can add parking lots and change availability of parking lots, check vehicle credentials, and have complete control of the wolf parking management system.

Students and Employees: Students and Employees with a unique univID and a valid permit can use the system to park in a parking space. They can take additional parking permit for special events or Park and Ride.

Visitors: They can take permit for a parking space in visitor's zone using their ID(phone number).

Security: They can create, update, delete citations information for those vehicles that don't follow the regulations. They can also process payment from drivers and update payment status.

3. **Five Main entities:**

- **DriverInformation:** name, status(student, employee, visitor), ID
- **Zone Information:** zone ID
- **Space Information:** space number, space type, availability status.
- **Citation Records:** citation number, car license number, citation date, citation time, lot, category, fee, payment status.
- **Permit Information:** permit ID, lot, zone ID, space number, start date, expiration date, expiration time, driverId, and permit type, carLicenseNumber.

4.

Situation 1: Security sees that a student's parking permit has expired so they generate a citation. The student with his expired permit then initiates a payment procedure to pay $30 as fees because he falls under the "Expired Permit" category of parking violations. Then the payment status gets updated from unpaid to paid. (creating the citation and also initiating a payment procedure to update the payment status are both operations).

Situation 2: A student requests admin for a new permit. Admin checks for an available space number based on the required space type, and driver status. Based on the availability, the admin creates permits for the student. (checking for the available space number and assigning permits are both operations).

5.

Application Programming Interface

Information Processing:

- addDriver(name, ID, status)
  return confirmation

- updateDriver(name, ID, status)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated

- deleteDriver(ID)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated

- addParkingLot(name, address)
  return confirmation

- updateParkingLot(name, address)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated

- deleteParkingLot(name, address)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated

- assignZoneToLot(zoneId, lotName)
  return confirmation

- updateZone(zoneId, lotName, newZoneId, newLotName)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated

- deleteZone(zoneId, lotName)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated

- assignTypeToSpace(spaceNumber, zoneId, lotName, type)
  return confirmation

- addspace(spaceNumber, zoneId, lotName, type, availabilityStatus)
  return confirmation

- updateSpace(spaceNumber, zoneId, lotName, type, availabilityStatus)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated.

- deleteSpace(spaceNumber, zoneId, lotName)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated.

- createCitation(citationNumber, date, time, category, fee, paymentStatus, lotName, permitId, licenseNumber)
  return confirmation
  *NULL value allowed for permitId(Incase of no permit citation).

- updateCitation(citationNumber, date, time, category, fee, paymentStatus, lotName, permitId, licenseNumber)
  return confirmation
  *NULL value allowed for permitId(Incase of no permit citation).

- deleteCitation(citationNumber)
  return confirmation

Maintaining permits and vehicle information for each driver:

- assignPermitToDriver(permitId, permitType, startDate, expirationTime, expirationDate, driverId, spaceNumber, zoneId, lotName, vehiclesList)
  return confirmation

- updatePermit(permitId, permitType, startDate. expirationTime, expirationDate, driverId, spaceNumber, zoneId, lotName, vehiclesList)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated.

- deletePermit(permitId)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated.

- addVehicleToAPermit(permitId, licenseNumber, model, manufacturer, year)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated.

- deleteVehicleFromAPermit(permitId, licenseNumber)
  return confirmation
  * If NULL value for any of the fields, then they will not be updated.

- addVehicle(licenseNumber, model, color, manufacturer, year, driverId)
  return confirmation

- updateVehicle(licenseNumber, model, color, manufacturer, year, driverId)
  return confirmation

- deleteVehicle(licenseNumber, model, color, manufacturer, year, driverId)
  return confirmation

Generating and maintaining citations:

- checkIfValidPermit(licenseNumber,lotName,zoneId, spaceNumber)
  returns True if the vehicle has a valid permit for the space number, zone and lot provided, False otherwise.

- createCitation(citationNumber, date, time, category, fee, paymentStatus, lotName, permitId, licenseNumber)
  return confirmation
  *NULL value allowed for permitId(Incase of no permit citation).

- updateCitation(citationNumber, date, time, category, fee, paymentStatus, lotName, permitId, licenseNumber)
  return confirmation
  *NULL value allowed for permitId(Incase of no permit citation).

- deleteCitation(citationNumber)
  return confirmation

- payCitationFee(citationNumber, feePaid)
  returns confirmation

- addAppeals(driverId,citationNumber,description,status)
  returns confirmation

- updateAppeals(driverId,citationNumber,description,status)
  returns confirmation
- deleteAppeals(citationNumber)
  returns confirmation

Reports

- getCitations(startTime, endTime)
  returns a list of citations from the startTime to endTime given.

- getCitationsInLot(lotName, startTime, endTime)
  returns a list of citations from startTime to endTime in a given parking lot.

- getZonesInLots()
  returns a list of all zones with their respective parking lots in tuple pairs.

- getVehiclesInViolation()

returns a list of all the vehicles having citations and for which payment has not been done.

- getPermitsInZone(zoneId)

returns a list of Permits in a zone.

- getPermit(driverId)

returns a list of permits currently held by a driver.

- getAvailableSpaceInParkingLot(spaceType, zoneId, lotName)

returns an available space number given the zoneId and lotName.

6

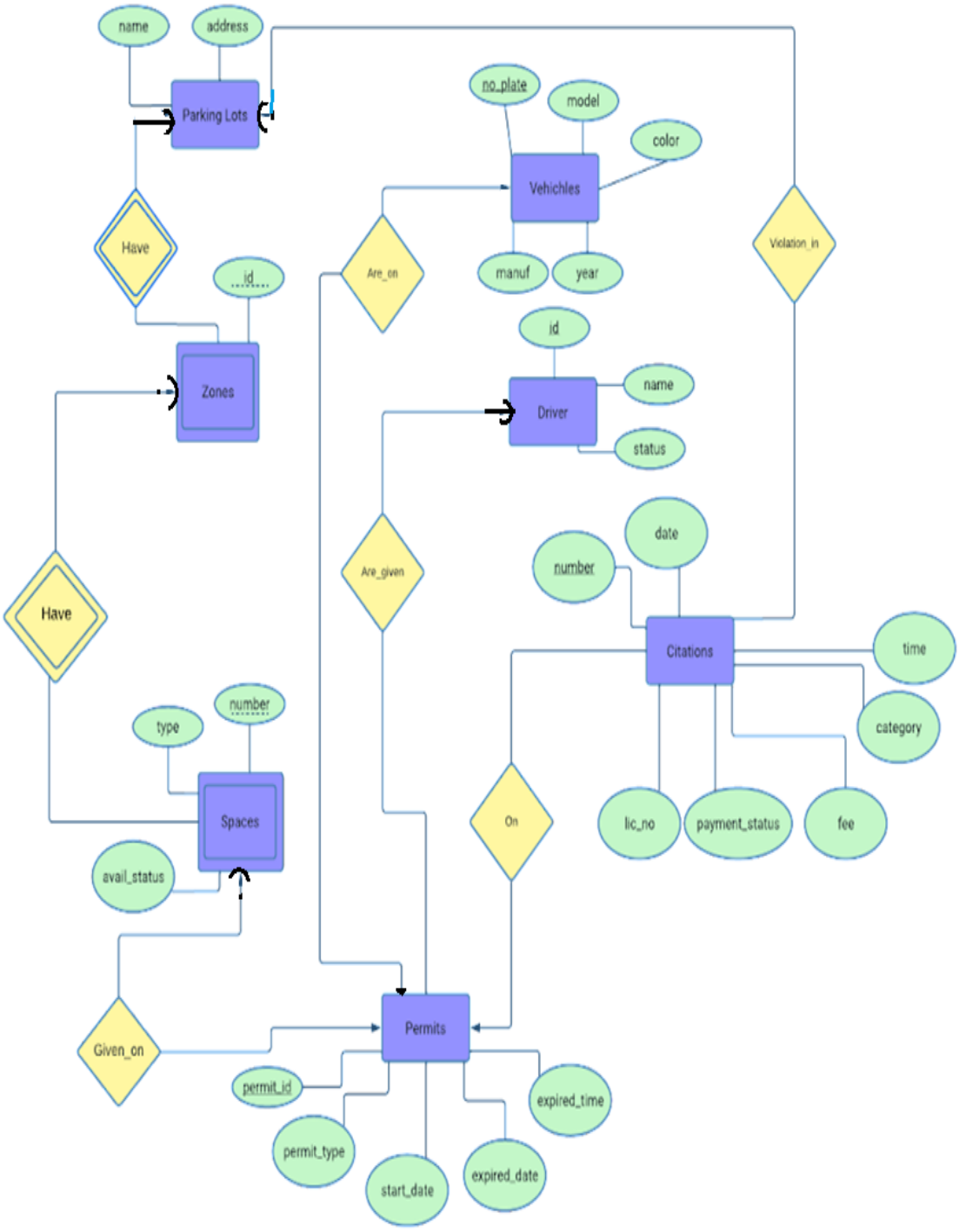
Admin: can add/update the vehicle information, the information of all drivers (students, employees and visitors), vehicles, parking lot, zone and space information,citations and the permit information in the database. They have a general view and accessibility to add/update/delete everything in the database.

Security: can create/update/delete citations to vehicles that violate parking rules. Security will hence have a general view of all the citations and permits because citations are depending on them.

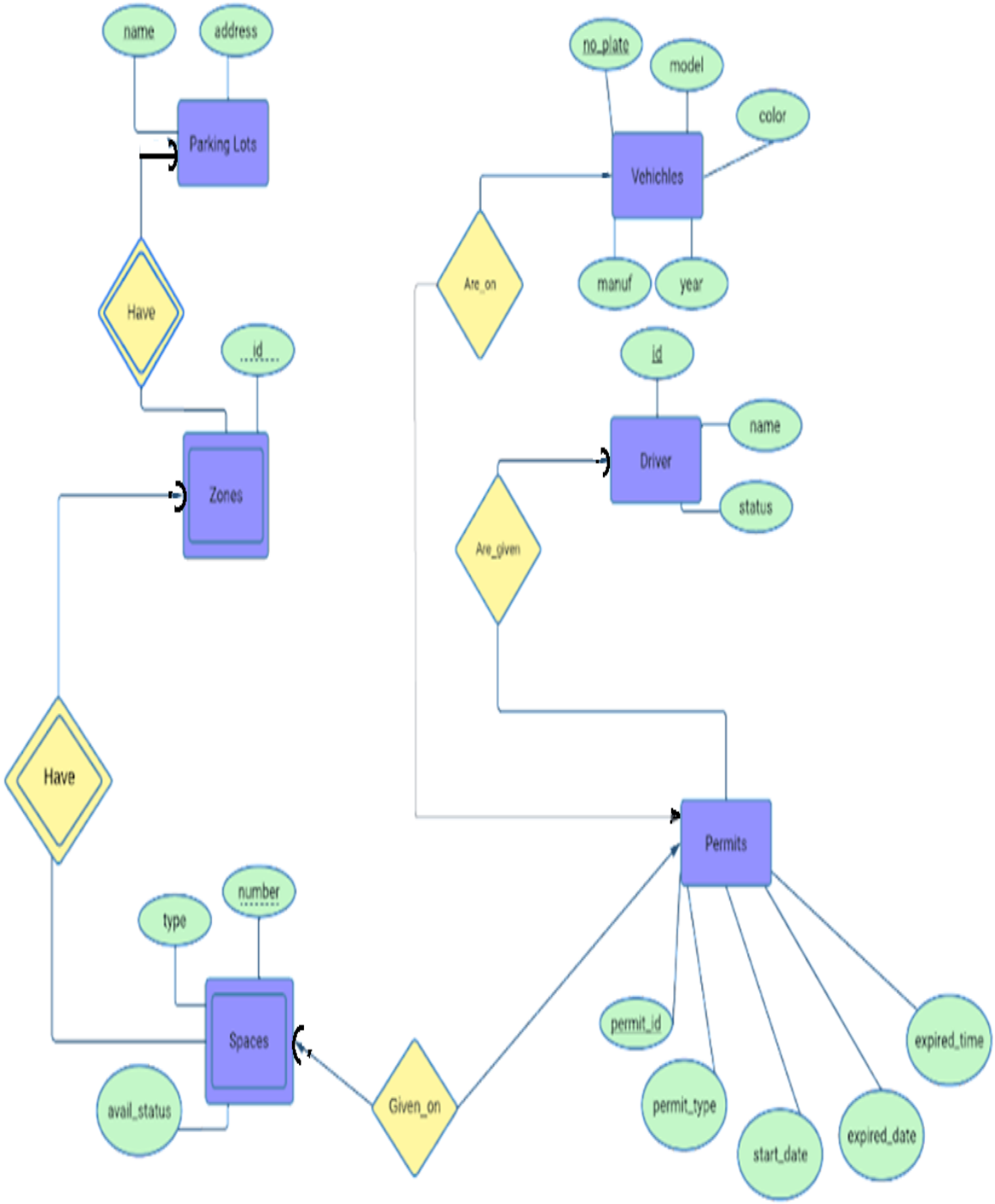
Drivers: there are three types of drivers: students, employees and visitors on the basis of permits allowed to them. Drivers can see their own permits, vehicles and parking lot information. Since citations are issued to drivers by placing a ticket on the vehicle, drivers do not need the view of citations.

7.

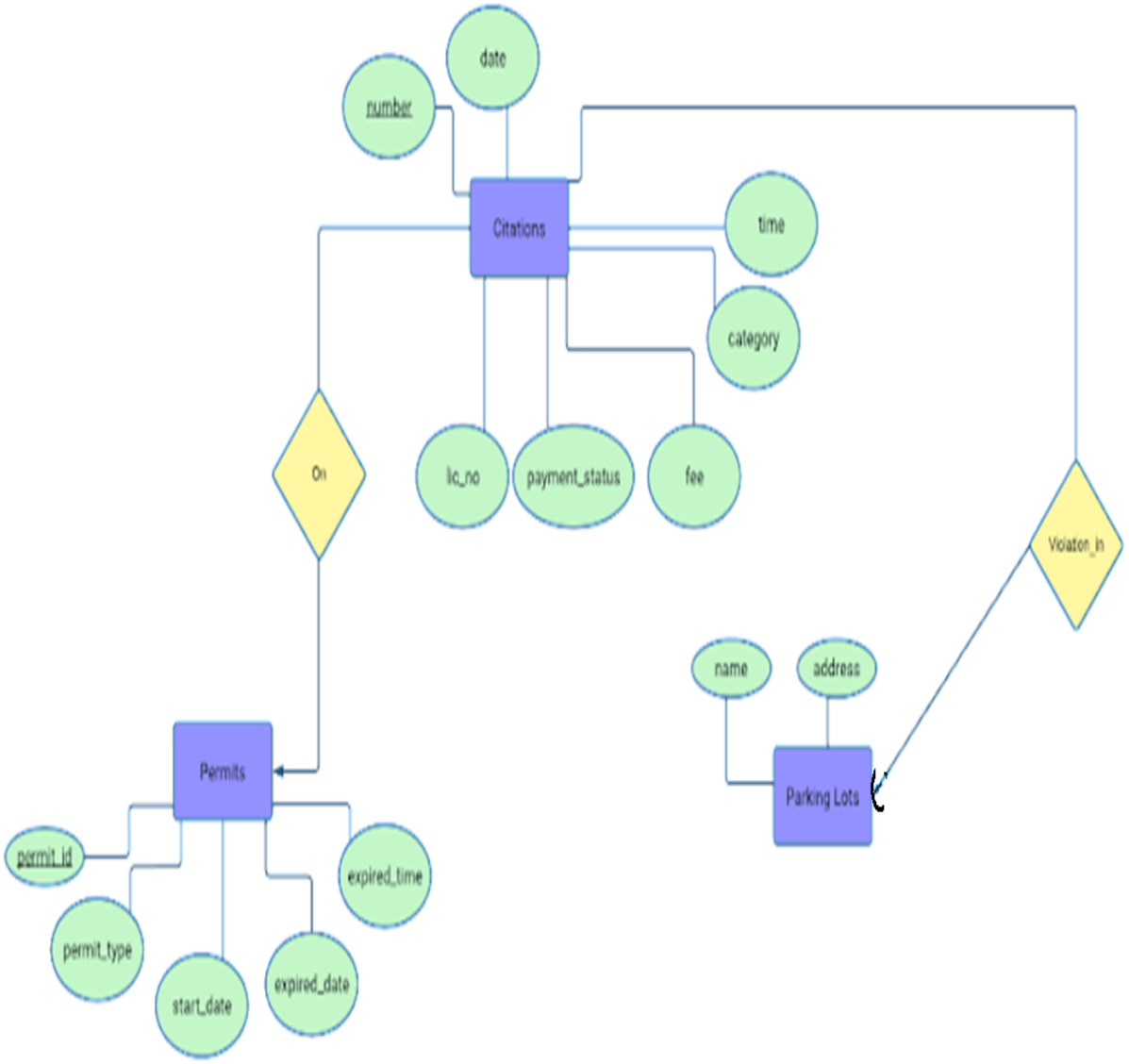
Admin


name
address
Parking Lots
no_plate
model
color
Vehichles
Have
Are_on
Violation_in
manuf
year
id
Zones
id
Driver
name
status
Have
Are_given
date
number
Citations
time
category
type
number
Spaces
On
lic_no
payment_status
fee
avail_status
Given_on
Permits
permit_id
expired_time
permit_type
start_date
expired_date

Driver


name
address
Parking Lots
Have
id
Zones
Have
number
type
Spaces
avail_status
Given_on
no_plate
model
color
Vehichles
Are_on
manuf
year
Id
Driver
name
status
Are_given
Permits
permit_id
permit_type
start_date
expired_date
expired_time

Security


date
number
Citations
time
category
On
lic_no
payment_status
fee
Violaton_In
name
address
Parking Lots
Permits
permit_id
permit_type
start_date
expired_date
expired_time

8. Description of Local E/R diagrams:

- The drivers can be Students, Employees of University or visitors. They are identified by DriverID and have name , status(Student, Employee or Visitor).
- Each vehicle is uniquely identified by the license number, also they have different attributes: model, color, manufacturer and the year which represents the vehicle. A vehicle can have utmost one permit at any point of time.
- A permit is identified by permitId and has attributes permit type, start date, expiration date, expiration time. There can be more than one vehicle on a permit(depending on the status of the driver). A permit can belong to exactly one driver.
- Any driver can ask for one or two parking permits depending on the occasion and their identification(Student/Employee/Visitor).
- Each parking lot contains different Zones and each Zone contains different Spaces. Here, Spaces is a weak entity which can be defined using Zones and Zones is also a weak entity which relies on Parking Lot.
- Each space only belongs to exactly one zone and each zone only belongs to exactly one parking lot.
- Spaces records the availability status of a particular parking space in a specific zone and parking lot. Also there can be different types of parking spaces for Electric Vehicle, Compact vehicles, Handicap and regular vehicles.
- Each Citation is uniquely identified by Citation number and has attributes date, licenseNumber, payment, status, category. Also there is a many-one relationship(violation_in) from Citation to parking lots indicating the lot in which the violation occurred.
- There is a many-one relationship between Citation and Permit to capture the permit associated with violation(invalid permit, expired permit). This helps in providing context of the violation. There can be at most one permit associated with a violation as there is no permit violation also.

9.

Admin's View

ParkingLots(<u>name</u>, address)

Zones(<u>zoneId</u>, <u>lotName</u>)

Spaces(<u>spaceNumber</u>, <u>zoneID</u>, <u>lotName</u>, type, availabilityStatus)

Citations(number, date, time, category, fee, paymentStatus, permitID, lotNumber, licenseNumber)

Permits(permitID, permitType, startDate, expirationTime, expirationDate, driverID, spaceNumber, zoneID, lotName)

Vehicles(carLicenseNumber, model, color, manufacturer, year, permitID)

Drivers(name, ID, status)

Security's view

Citations(number, date, time, category, fee, paymentStatus, permitID, lotNumber, licenseNumber)

Permits(permitID, permitType, startDate, expirationTime, expirationDate)

ParkingLots(name, address)

Drivers View

ParkingLots(name, address)

Zones(zoneId, lotName)

Spaces(spaceNumber, zoneID, lotName, type, availabilityStatus)

Permits(permitID, permitType, startDate, expirationTime, expirationDate, driverID, spaceNumber, zoneID, lotName)

Vehicles(carLicenseNumber, model, color, manufacturer, year, permitID)

Drivers(name, ID, status)

10. Description of Local Schemas

Entity sets to relations

· Entity sets were translated into relations with the same attributes. This was followed for translating ParkingLots,Vehicles, Drivers, Permits and Citations into relations.

· Weak Entity sets were translated into relations with its attributes and also the key attributes of the supporting entity sets. This was followed for translating Zones and Spaces entities into relations.

Combining Many-One and One-One relationships

· If there are two entities E1 and E2. In case if there is a many-one or one-one relationship from E1 to E2, then we have not created a separate relation schema for relationship between those entities. We have added the key attribute of E2 into relation schema of E1.This reduces redundancy. This was followed for the following:

    * Citations entity translated into Citations relation with its attributes number, date, time category. Fee, payment status, licenseNumber. Also the many-one relationships from Citations to Permits and Citations to Parking Lots were combined into the citations relation. Hence, lotNumber and permitId were added into Citations relation.

* Permits entity translated into Permits relation with its attributes permitId, permitType, startDate, expirationTime, expirationDate. Also the many-one relationships from Permits to Drivers and one –one relation from Permits to Spaces were combined into the Permits relation. Hence, driverID and spaceNumber, zoneID, lotName were added into Permits relation.

* Vehicles entity translated into Vehicles relation with its attributes plateNo, model, color, manufacturer and year. Also the many-one relationship from Vehicles to Permits is combined into Vehicles relation. Hence permitId is added into Vehicles relation.

Wolf Parking Management System

CSC 540 Database Management Systems

Project Report 2

Team Members:

Naga Jahnavi Kommareddy (nkommar)

Shail Shah (sshah38)

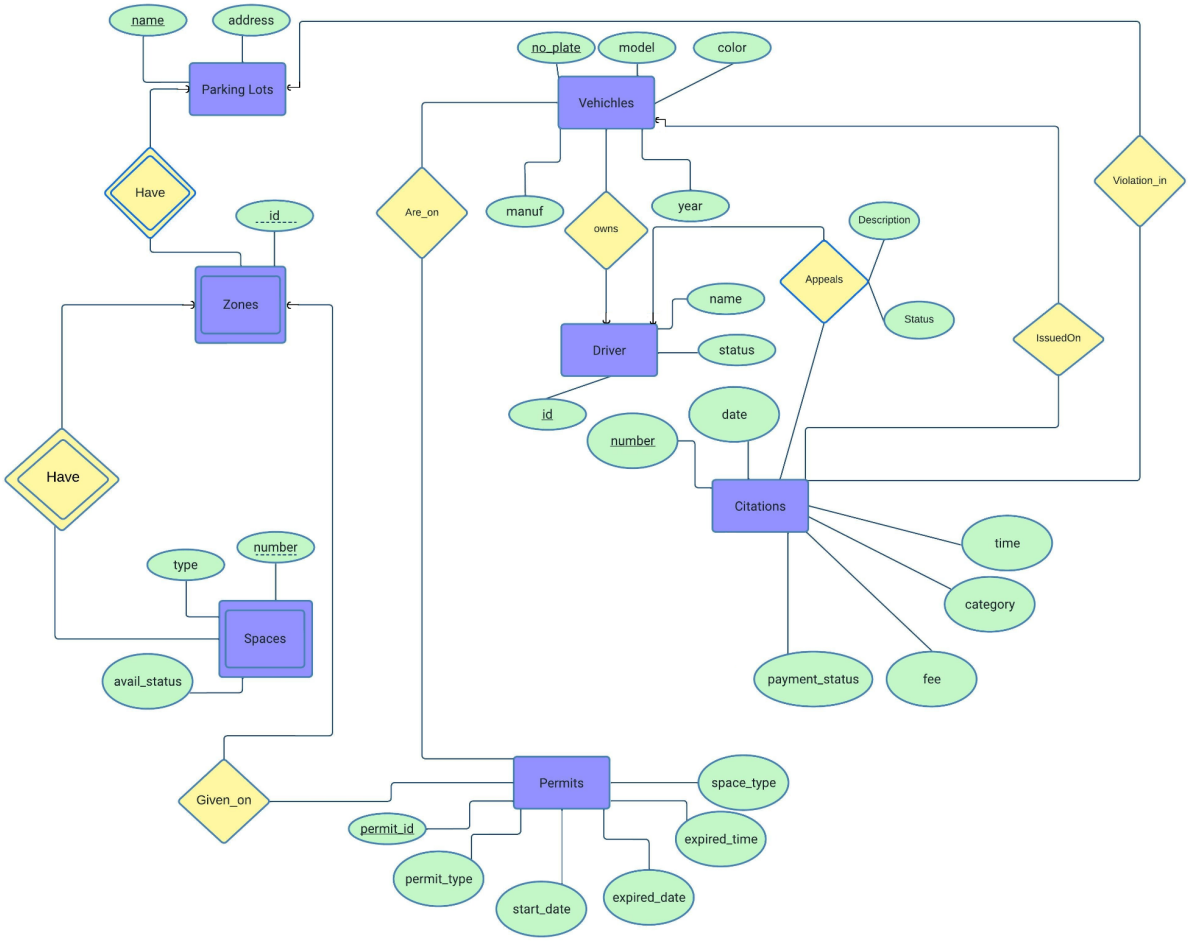
Shrishty Singh (ssingh67)

Shyamal Tarpanbhai Gandhi (sgandhi6)

Assumptions

- Parking lots consist of Zones. Zone Id is unique within a parking lot. Zones consist of spaces. Each space has a space number which is unique within a Zone.
- UnivID and phone number never clash. Single ID attribute can be used to store UnivID or phone number based on the status of the driver. UnivID is stored in the ID attribute in case of a student or employee and phone number is stored in case of a visitor. UnivID is a 9 digit number and phone Number consists of 10 digits.
- It is mandatory for the permit card to be visibly displayed on the vehicle whenever it is parked. Security personnel will identify parking violations by inspecting the displayed permit cards periodically.
- When a security guard records a citation in the database, they will also affix a physical ticket to the vehicle. This ticket contains details about the violation and provides instructions for payment.
- Drivers have the option to initiate the payment process online by entering the citation number, which is found on the ticket. They can then proceed to pay the required fine based on the violation.
- Handicap users getting 50% discount on all citations will be handled at the operations level.
- Restrictions on number of permits and number of vehicles on each permit based on the status of the driver will be handled at operations level.
- One driver can make only one appeal per citation.
- A Vehicle cannot have multiple permits at a time.
- In the case of ‘No permit’, the security guard will manually check whether the vehicle has permit or not, if not, then the security guard will manually add a citation for that particular vehicle by asking for all the information regarding vehicle and driver on the spot. Because in the case of no permit it might happen that neither the driver or vehicle table will contain the data.
- Admin updates spaces after availability status changes

Updated ER Diagram based on Report 1 Feedback:



1.

Global Relational Database Schema

ParkingLot(Name,Address)
Name->Name,Address holds because the name of a parking lot will be unique and it will identify the address because one parking lot cannot have two addresses. The address cannot be a key because it cannot uniquely determine the name.

Since this is a two-attribute relation and two-attribute relations are always in BCNF, this relation is in 3NF

Zones(Id,LotName)
Id,LotName->Id,LotName holds because the relation Zones is made as it is a weak entity set and will therefore require the primary key attributes of its own as well as the supporting strong entity set to describe the relation.

Since both attributes form a superkey and it is a two-attribute relation, it is in BCNF and hence in 3NF.

Spaces(SpaceNumber,SpaceType,Avail_Status,ZoneId,LotName)
SpaceNumber,ZoneId,LotName->SpaceNumber,ZoneId,LotName,SpaceType,Avail_Status holds because a space number, its zone id, and the parking lot name is required to uniquely identify a space and all of its parameters or attributes like the type of space or the availability status of the space. No other combination of attributes can uniquely determine the other attributes for example two similar space types ‘Electric’ having availability can be present in two different space number,zone id, and lot name.

Since the LHS of FD is a superkey the FD is in BCNF and hence it is 3NF.

Permits(PermitId,PermitType,StartDate,ExpireDate,SpaceType,ExpireTime,ZoneId,LotName)
PermitId->PermitId,StartDate,ExpireDate,SpaceType,ExpiredTime,ZoneId,LotName holds because permit id of a permit is a unique identifier as one permit will have one id and it will be able to identify all the other parameters of a permit like start date, expired date, expired time, space type, zone id and lot name. No other combination of attributes can determine other attributes for example two permits can have same start date and expired date but can be for different zones and parking lots.

Since the LHS of FD is a superkey the relation is in BCNF and hence it is 3NF.

Vehicles(NumberPlate,Model,Color,Manufacturer,Year,DriverId)
NumberPlate->Model,Color,Manufacturer,Year,DriverId holds because the number plate of a vehicle is globally unique for all vehicles and is able to determine the vehicle's properties like color,manufacturer,year and the driver id of the person who's vehicle it is. Other attributes like color, model can be same but can have different manufacturer or year. Also one driver (same driver id) cannot have two vehicles with the same number plate.

Since the left hand side of the functional dependency is the super key, the relation is in BCNF and hence in 3NF.

Drivers(Id,Name,Status)
Id->Id,Name,Status holds because a driver's id will be unique for a driver and will be able to uniquely identify his name and status (student, employee, or visitor). Two drivers can have the same name but different statuses and IDs for example a student and an employee can have the same name so no other functional dependency will hold.

The left-hand side of the functional dependency is a super key hence the relation is in BCNF and therefore in 3NF.

Citations(<u>CitationNumber</u>,CitationDate,CitationTime,Fee,PaymentStatus,Category,NumberPlate,LotName)
CitationNumber->CitationNumber,CitationDate,CitationTime,Fee,PaymentStatus,Category,NumberPlate,LotName holds because a citation number is the unique identifier for citations. It also is sufficient to find out the other parameters of a citation like the time, date,fee etc. Also, no other combinations can determine other attributes for example two same cars (same no_plate) can have 2 different citations (2 citation_number) and hence different date, time etc for the citations.

The left-hand side of the functional dependency is a superkey hence the relation is in BCNF and therefore in 3NF.

CitationDate,CitationTime,NumberPlate->CitationNumber,Fee,PaymentStatus,Category,LotName holds because one vehicle (same number plate) cannot have two different citations at the same time and date. If the number plate, citation date and citation time is same then it is the same citation hence the citation number, fee, payment status, category and lot name will be same.

We take the closure of LHS of this functional dependency to determine if its a superkey or not. $(\text{CitationDate, CitationTime, NumberPlate})^{+}$ =CitationNumber, CitationDate, CitationTime, NumberPlate, Fee, PaymentStatus, Category, LotName. This gives the entire relation, hence LHS of this functional dependency is a superkey.

The left-hand side of the functional dependency is a superkey hence the relation is in BCNF and 3NF.

PermitsToVehichles(<u>PermitId,NumberPlate</u>)
PermitId,NumberPlate->PermitId,NumberPlate holds because this relation is made from the relationship between permits relation and vehicle relation and hence has both of its primary keys as its primary key. No other functional dependencies can be formed.

A two-attribute relation is always in BCNF and hence this is in 3NF.

Appeals(<u>CitationNumber</u>,DriverId, Description,Status)
DriverId,CitationNumber->DriverId,CitationNumber, Description, Status holds because the driver id and citation number will be unique for an appeal, one driver appealing on one citation will have the same description and status of his appeal. No other combinations can determine

anything because an appeal having the same description might be made by different drivers or for different citations.

CitationNumber->CitationNumber,DriverId, Description,Status holds because the citation number for an appeal will always be unique as per our assumption that one driver can make only one appeal per citation. Citation number can determine driver ID because one citation belongs to only one driver but driver ID cannot determine citation number because one driver can have many citations to his name on different vehicles.

The second functional dependency has the left-hand side as the primary key; hence, it is in BCNF and by extension in 3NF. In the first functional dependency, the left-hand side is a superkey (a superset of the primary key) and hence this functional dependency also does not violate BCNF and by extension 3NF.

2.
Design decision for global schema:

Entity sets to relations

- Entity sets were translated into relations with the same attributes. This was followed for translating ParkingLots,Vehicles, Drivers, Permits and Citations into relations.
- Weak Entity sets were translated into relations with its attributes and also the key attributes of the supporting entity sets. This was followed for translating Zones and Spaces entities into relations.

Combining Many-One relationships

- If there are two entities E1 and E2. In case if there is a many-one or one-one relationship from E1 to E2, then we have not created a separate relation schema for relationship between those entities. We have added the key attribute of E2 into relation schema of E1.This reduces redundancy. This was followed for the following:
- Citations entity translated into Citations relation with its attributes number,date, time category. Fee, payment status. Also the many-one relationships from Citations to Vehicles and Citations to Parking Lots were combined into the citations relation. Hence, numberPlate and lotName were added into Citations relation.
- Permits entity translated into Permits relation with its attributes permitId, spaceType, permitType, startDate, expirationTime, expirationDate. Also the many-one relationships from Permits to Zones is combined into the Permits relation. Hence, zoneId, lotName were added into Permits relation.

- Vehicles entity translated into Vehicles relation with its attributes numberPlate , model, color, manufacturer and year. Also the many-one relationship from Vehicles to Drivers is combined into Vehicles relation. Hence driverId is added into Vehicles relation.

Remaining Relationships:

- Combining Many-One relationship was not followed for Appeals relation between Driver and Citation because Appeals relationship has its own attributes description and status. Also not all citations would have appeals. Hence, appeals relation was translated into a relation with key attributes from Drivers and Citations entity sets and its own attributes description and status.
- The Many-Many relationship between Vehicles and Permits was translated into a relation with key attributes from two connected entity sets: Permits and Vehicles.

ParkingLot(Name, Address)
Name is the Primary Key.
Address cannot be NULL

Zones(Id, LotName)
Id ,LotName together form a Primary Key.
LotName is the ForeignKey to the Name in the ParkingLot table.

Spaces(SpaceNumber, ZoneId, LotName, SpaceType, Avail_Status)
SpaceNumber, ZoneId, LotName together form a PrimaryKey.
{ZoneId,LotName} form a Foreign key which references {Id, LotName} in the Zones table.
SpaceType cannot be NULL.
Avail_Status cannot be NULL.

Permits(PermitId, ZoneId, LotName, SpaceType, PermitType, StartDate, ExpireDate, ExpireTime)
PermitId is the Primary Key.
{ZoneId,LotName} form a Foreign key which references {Id, LotName} in the Zones table.
SpaceType cannot be NULL.
PermitType cannot be NULL.
StartDate cannot be NULL.
ExpireDate cannot be NULL.
ExpireTime cannot be NULL.

Vehicles(NumberPlate, Model, Color, Manufacturer, Year, DriverId)
NumberPlate is the PrimaryKey.

DriverId is the ForeignKey to Id in the Drivers table.
Model can be NULL as model value is not a mandatory field to perform any operation. It is extra information that is stored which could be useful in certain situations for better vehicle identification.
Color can be NULL as color value is not a mandatory field to perform any operation. It is extra information that is stored which could be useful in certain situations for better vehicle identification.
Manufacturer can be NULL as manufacturer value is not a mandatory field to perform any operation. It is extra information that is stored which could be useful in certain situations for better vehicle identification.
Year can be NULL as year value is not a mandatory field to perform any operation. It is extra information that is stored which could be useful in certain situations for better vehicle identification.

Drivers(Id, Name, Status)
Id is the PrimaryKey.
Name cannot be NULL.
Status cannot be NULL.

Citations(CitationNumber, CitationDate, CitationTime, Fee, PaymentStatus, Category, NumberPlate, LotName)
CitationNumber is the PrimaryKey.
CitationDate cannot be NULL.
CitationTime cannot be NULL.
Fee cannot be NULL.
PaymentStatus cannot be NULL.
Category cannot be NULL.
NumberPlate is the ForeignKey to NumberPlate in the Vehicles table.
LotName is the ForeignKey to Name in the ParkingLot table.

PermitsToVehicles(PermitId, NumberPlate)
{PermitId, NumberPlate} together form a PrimaryKey.
PermitId is the ForeignKey to PermitId in the Permits table.
NumberPlate is the ForeignKey to NumberPlate in the Vehicles table

Appeals(DriverId, CitationNumber, Description, Status)
CitationNumber is the PrimaryKey (This is based on our assumption that a particular citation can have only one appeal).
DriverId is the ForeignKey to the Id in the Drivers table.
CitationNumber is the ForeignKey to CitationNumber in the Citations table.
Description cannot be NULL.
Status cannot be NULL.

3. Base Relations:

```
CREATE TABLE ParkingLot(
        Name VARCHAR(32) PRIMARY KEY,
        Address VARCHAR(128) NOT NULL
);

CREATE TABLE Zones(
        Id VARCHAR(2) NOT NULL,
        LotName VARCHAR(32) NOT NULL,
        PRIMARY KEY(Id, LotName),
        FOREIGN KEY(LotName) REFERENCES ParkingLot(Name) ON UPDATE
CASCADE ON DELETE CASCADE
);

DELIMITER $$
CREATE TRIGGER checkZoneId BEFORE INSERT ON Zones
FOR EACH ROW
BEGIN
   IF NEW.Id not in ("A","B","C","D","AS","BS","CS","DS","V") THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong insert for ZoneId. Enter
"A","B","C","D","AS","BS","CS","DS" or "V"';
   END IF;
END$$
DELIMITER ;

CREATE TABLE Spaces(
        SpaceNumber INT NOT NULL,
        ZoneId VARCHAR(2) NOT NULL,
        LotName VARCHAR(32) NOT NULL,
        SpaceType VARCHAR(32) DEFAULT "Regular" NOT NULL,
        Avail_Status VARCHAR(5) DEFAULT "YES" NOT NULL,
        PRIMARY KEY(SpaceNumber,ZoneId,LotName),
        FOREIGN KEY(ZoneId,LotName) REFERENCES Zones(Id,LotName) ON
UPDATE CASCADE ON DELETE CASCADE
);



DELIMITER $$
CREATE TRIGGER checkSpaceType BEFORE INSERT ON Spaces
```

```
FOR EACH ROW
BEGIN
   IF NEW.SpaceType not in ("Electric", "Handicap", "Compact Car", "Regular") THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong insert for SpaceType. Enter
Electric, Handicap, Compact Car or Regular';
   END IF;
END$$
DELIMITER ;
```

CREATE TABLE Permits(
PermitId VARCHAR(10) PRIMARY KEY,
ZoneId VARCHAR(2) NOT NULL,
LotName VARCHAR(32) NOT NULL,
SpaceType VARCHAR(32) NOT NULL,
PermitType VARCHAR(32) NOT NULL,
StartDate Date NOT NULL,
ExpireDate Date NOT NULL,
ExpireTime Time NOT NULL,
FOREIGN KEY(ZoneId,LotName) REFERENCES Zones(Id,LotName) ON UPDATE CASCADE ON DELETE CASCADE
);

```
DELIMITER $$
CREATE TRIGGER checkAvail BEFORE INSERT ON Permits
FOR EACH ROW
BEGIN
   IF (select count(*) from Spaces where ZoneId=NEW.ZoneID and LotName=NEW.LotName and
SpaceType=NEW.SpaceType)=0 THEN
   SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Space does not exists';
   END IF;
   IF (select count(distinct Avail_Status),Avail_Status  from Spaces where ZoneId=NEW.ZoneID and
LotName=NEW.LotName and SpaceType=NEW.SpaceType) = (1,"NO") THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Space not available or does not exists';
   END IF;
END$$
```

CREATE TABLE Drivers(
Id BigINT(10) PRIMARY KEY,
Name VARCHAR(32) NOT NULL,
Status VARCHAR(1) NOT NULL

```
);

CREATE TABLE Vehicles (
NumberPlate VARCHAR(10) PRIMARY KEY,
Model VARCHAR(32),
Color VARCHAR(50),
Manufacturer VARCHAR(32),
Year INT,
DriverId bigINT(10) NOT NULL,
FOREIGN KEY(DriverId) REFERENCES Drivers(Id) ON UPDATE CASCADE ON
DELETE CASCADE
);

CREATE TABLE Citations(
CitationNumber VARCHAR(10) PRIMARY KEY,
CitationDate date NOT NULL,
CitationTime time NOT NULL,
Fee INT NOT NULL,
PaymentStatus VARCHAR(50) NOT NULL,
Category VARCHAR(32) NOT NULL,
NumberPlate VARCHAR(10) NOT NULL,
LotName VARCHAR(32) NOT NULL,
FOREIGN KEY(NumberPlate) REFERENCES Vehicles(NumberPlate) ON UPDATE
CASCADE ON DELETE CASCADE,
FOREIGN KEY(LotName) REFERENCES ParkingLot(Name) ON UPDATE CASCADE
ON DELETE CASCADE
);

CREATE TABLE PermitsToVehicles(
PermitId VARCHAR(10) NOT NULL,
NumberPlate VARCHAR(10) NOT NULL,
PRIMARY KEY(PermitId,NumberPlate),
```

FOREIGN KEY(PermitId) REFERENCES Permits(PermitId) ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY(NumberPlate) REFERENCES Vehicles(NumberPlate) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Appeals(
DriverId bigINT(10) NOT NULL,
CitationNumber VARCHAR(10) PRIMARY KEY,
Description VARCHAR(128),
Status VARCHAR(10),
FOREIGN KEY(DriverId) REFERENCES Drivers(Id) ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY(CitationNumber) REFERENCES Citations(CitationNumber) ON UPDATE CASCADE ON DELETE CASCADE
);

SELECT * FROM ParkingLot;

```
+------------+--------------------+
| Name       | Address            |
+------------+--------------------+
| Centennial | AventFerry Road    |
| Main       | Hillsborough Road  |
| Mckimmon   | Gorman Street      |
| Talley     | Cates Ave          |
+------------+--------------------+
4 rows in set (0.0014 sec)
```

SELECT * FROM Zones;

```
+----+------------+
| Id | LotName    |
+----+------------+
| AS | Centennial |
| AS | Mckimmon   |
| AS | Talley     |
| B  | Mckimmon   |
| BS | Centennial |
| BS | Talley     |
| C  | Centennial |
| C  | Talley     |
| CS | Main       |
| CS | Mckimmon   |
| D  | Main       |
| DS | Main       |
| V  | Centennial |
| V  | Main       |
| V  | Mckimmon   |
| V  | Talley     |
+----+------------+
16 rows in set (0.0018 sec)
```

SELECT * FROM Spaces;

SpaceNumber	ZoneId	LotName	SpaceType	Avail_Status
1	AS	Centennial	regular	NO
1	AS	Mckimmon	regular	YES
1	AS	Talley	regular	NO
1	B	Mckimmon	regular	NO
1	BS	Centennial	regular	YES
1	BS	Talley	regular	YES
1	C	Centennial	regular	NO
1	C	Talley	regular	NO
1	CS	Main	regular	YES
1	CS	Mckimmon	regular	YES
1	D	Main	regular	NO
1	DS	Main	regular	YES
1	V	Centennial	regular	YES
1	V	Main	electric	NO
1	V	Mckimmon	electric	NO
1	V	Talley	regular	YES
2	AS	Centennial	handicap	YES
2	AS	Mckimmon	handicap	YES
2	AS	Talley	handicap	YES
2	B	Mckimmon	regular	YES
2	BS	Centennial	handicap	YES
2	BS	Talley	handicap	YES
2	C	Centennial	regular	NO
2	C	Talley	regular	NO
2	CS	Main	handicap	YES
2	CS	Mckimmon	handicap	YES
2	D	Main	regular	YES
2	DS	Main	handicap	YES
2	V	Centennial	regular	YES
2	V	Main	regular	YES
2	V	Mckimmon	regular	YES
2	V	Talley	regular	YES
3	AS	Centennial	regular	YES
3	AS	Mckimmon	regular	YES
3	AS	Talley	regular	YES
3	BS	Centennial	regular	YES
3	BS	Talley	regular	YES
3	CS	Main	regular	YES
3	CS	Mckimmon	regular	YES
3	DS	Main	regular	YES
4	AS	Centennial	regular	YES
4	AS	Mckimmon	regular	YES
4	AS	Talley	regular	YES
4	BS	Centennial	regular	YES
4	BS	Talley	regular	YES
4	CS	Main	regular	YES
4	CS	Mckimmon	regular	YES

```
|           4 | DS     | Main       | regular   | YES           |
|           5 | AS     | Centennial | electric  | YES           |
|           5 | AS     | Mckimmon   | electric  | YES           |
|           5 | AS     | Talley     | electric  | YES           |
|           5 | CS     | Main       | electric  | YES           |
|           6 | AS     | Centennial | electric  | YES           |
|           6 | AS     | Mckimmon   | electric  | YES           |
|           6 | AS     | Talley     | electric  | YES           |
|           6 | CS     | Main       | electric  | YES           |
+-------------+--------+------------+-----------+---------------+
56 rows in set (0.0022 sec)
```

SELECT * FROM Permits;

```
+----------+--------+------------+-----------+--------------+------------+------------+------------+
| PermitId | ZoneId | LotName    | SpaceType | PermitType   | StartDate  | ExpireDate | ExpireTime |
+----------+--------+------------+-----------+--------------+------------+------------+------------+
|        1 | AS     | Centennial | regular   | residential  | 2023-10-17 | 2023-10-17 | 15:00:00   |
|        2 | C      | Centennial | regular   | commuter     | 2023-10-07 | 2023-10-07 | 17:00:00   |
|        3 | C      | Centennial | regular   | residential  | 2023-10-17 | 2023-10-17 | 10:00:00   |
|        4 | D      | Main       | regular   | commuter     | 2023-10-17 | 2023-10-17 | 17:00:00   |
|        5 | V      | Main       | electric  | park and ride| 2023-10-17 | 2023-10-17 | 18:00:00   |
+----------+--------+------------+-----------+--------------+------------+------------+------------+
5 rows in set (0.0021 sec)
```

SELECT * FROM Drivers;

```
+------------+--------+--------+
| Id         | Name   | Status |
+------------+--------+--------+
|  200505648 | John   | S      |
|  200505649 | Stuart | S      |
|  200505650 | Roy    | E      |
|  200505651 | Rachel | E      |
| 9191234588 | Mike   | V      |
+------------+--------+--------+
5 rows in set (0.0014 sec)
```

SELECT * FROM Vehicles;

```
+-------------+-------+-------+--------------+------+------------+
| NumberPlate | Model | Color | Manufacturer | Year | DriverId   |
+-------------+-------+-------+--------------+------+------------+
| ABC123      | M1    | Black | Mf1          | 2021 |  200505648 |
| DEF456      | M2    | White | Mf2          | 2023 |  200505649 |
| GHI789      | M3    | Gray  | Mf3          | 2022 |  200505650 |
| JKL012      | M4    | White | Mf2          | 2022 |  200505651 |
| MNO345      | M5    | Red   | Mf4          | 2019 | 9191234588 |
+-------------+-------+-------+--------------+------+------------+
5 rows in set (0.0022 sec)
```

SELECT * FROM Citations;

```
+----------------+--------------+--------------+-----+---------------+---------------+-------------+------------+
| CitationNumber | CitationDate | CitationTime | Fee | PaymentStatus | Category      | NumberPlate | LotName    |
+----------------+--------------+--------------+-----+---------------+---------------+-------------+------------+
|              1 | 2023-10-17   | 16:00:00     |  30 | Complete      | Expired Permit | ABC123      | Centennial |
|              2 | 2023-10-17   | 17:00:00     |  25 | Incomplete    | Invalid Permit | GHI789      | Centennial |
|              3 | 2023-10-18   | 13:00:00     |  30 | Complete      | Expired Permit | MNO345      | Main       |
|              4 | 2023-10-17   | 09:00:00     |  25 | Incomplete    | Invalid Permit | JKL012      | Main       |
+----------------+--------------+--------------+-----+---------------+---------------+-------------+------------+
4 rows in set (0.0015 sec)
```

SELECT * FROM PermitsToVehicles;

```
+----------+-------------+
| PermitId | NumberPlate |
+----------+-------------+
|        1 | ABC123      |
|        2 | GHI789      |
|        3 | JKL012      |
|        4 | GHI789      |
|        5 | MNO345      |
+----------+-------------+
5 rows in set (0.0013 sec)
```

SELECT * FROM Appeals;

```
+------------+----------------+---------------------------------+------------+
| DriverId   | CitationNumber | Description                     | Status     |
+------------+----------------+---------------------------------+------------+
|  200505648 |              1 | The out time is not correct     | In process |
|  200505650 |              2 | Fee charged is wrong            | In process |
| 9191234588 |              3 | Citation filed on wrong vehicle | resolved   |
|  200505651 |              4 | The out time is not correct     | resolved   |
+------------+----------------+---------------------------------+------------+
4 rows in set (0.0038 sec)
```

4. SQL Queries:

4.1: Information Processing

Add Driver Information
Input : SQL > INSERT INTO Drivers VALUES(9194458355, "Veronica", "V");
Output : Query OK, 1 row affected (0.0038 sec)

Update Driver Information
Input : SQL >UPDATE Drivers SET name='James' where Id = 9194458355;
Output : Query OK, 1 row affected (0.0026 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Delete Driver Information
Input:SQL > DELETE FROM Drivers where Id= 9194458355;
Output : Query OK, 1 row affected (0.0028 sec)

Add ParkingLot Information
Input :SQL > INSERT INTO ParkingLot VALUES("Textiles","Crest Rd");
Output : Query OK, 1 row affected (0.0039 sec)

Update ParkingLot Information
Input :SQL > UPDATE ParkingLot SET Address = "Ivy Commons" where Name = "Textiles";
Output : Query OK, 1 row affected (0.0038 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Delete ParkingLot Information
Input: SQL > DELETE FROM ParkingLot WHERE Name = "Textiles";
Output : Query OK, 1 row affected (0.0043 sec)

Enter Zone or **Assign Zone To Lot**

Input: SQL > INSERT INTO Zones VALUES("A", "Main");

Output : Query OK, 1 row affected (0.0039 sec)

Update Zone Information

Input: SQL >UPDATE Zones SET LotName="Centennial" WHERE Id="A" AND LotName="Main";

Output : Query OK, 1 row affected (0.0041 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Delete Zone Information

Input: SQL > DELETE FROM Zones WHERE Id="A" AND LotName="Centennial";

Output : Query OK, 1 row affected (0.0146 sec)

Assign Type To Space

Case 1: If that space number is not there in the table, we insert it with its space type

Input: SQL > INSERT INTO Spaces VALUES(7,"CS","Main","electric","YES");

Output : Query OK, 1 row affected (0.0029 sec)

Case 2: Or if it is already there in the table, space type can be updated with the new value given. This can be done through the update space command.

Since (7,"CS","Main","electric","YES") was not there in the table, it is case 1 and hence the insert query was executed.

Add Space

Input: SQL > INSERT INTO Spaces VALUES(7,"AS","Centennial","regular","YES");

Output: Query OK, 1 row affected (0.0038 sec)

Update Space

Input: SQL > UPDATE Spaces SET SpaceType="electric" WHERE SpaceNumber=7 AND ZoneId="AS" AND lotName=”Centennial”;

Output: Query OK, 1 row affected (0.0030 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Delete Space

Input:SQL > DELETE FROM Spaces WHERE SpaceNumber=7 AND ZoneId="CS" AND lotName=”Main”

Output: Query OK, 1 row affected (0.0028 sec)

Input: SQL > DELETE FROM Spaces WHERE SpaceNumber=7 AND ZoneId="AS" AND lotName=”Centennial”

Output: Query OK, 1 row affected (0.0027 sec)

Create Citation

Input: INSERT INTO Citations VALUES(5, "2023-10-17", "16:00:00", 25, "Incomplete", "Expired Permit", "ABC123","Centennial");

Output : Query OK, 1 row affected (0.0027 sec)

Update Citation

Input: SQL > UPDATE Citations SET Fee=30 WHERE CitationNumber=5;

Output : Query OK, 1 row affected (0.0027 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Delete Citation

Input: SQL > DELETE FROM Citations WHERE CitationNumber=5;

Output : Query OK, 1 row affected (0.0039 sec)

Maintaining permits and vehicle information for each driver:

Enter Vehicle:

Input: SQL > INSERT INTO Vehicles
VALUES("QWE123","M2","Black","Mf4",2008,9191234588);
Output: Query OK, 1 row affected (0.0030 sec)

Update Vehicle:

Input: SQL > UPDATE Vehicles SET Year = 2022 WHERE NumberPlate= "QWE123";
Output : Query OK, 1 row affected (0.0021 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Assign Permits:

The following are the sequence of SQL queries that will be executed for assigning a Permit to a Driver:

First we check if the given zoneId and lotName have the given space type or not using the following query:
SELECT COUNT(*)>0 from Spaces WHERE lotName = "Main" AND ZoneId = "V" AND SpaceType = "regular";

```
+--------------+
| COUNT(*) |
+---------------+
|       1           |
+---------------+
1 row in set (0.0016 sec)
```

Then for all the permit requests, we also check if the given driver owns that the particular vehicle.

```
SELECT COUNT(*) FROM Vehicles WHERE DriverId = "9191234588" AND NumberPlate ="QWE123";
+--------------+
| COUNT(*) |
+--------------+
|       1          |
+---------------+
1 row in set (0.0068 sec)
```

Then we enter the info into Permits and PermitsToVehicles relations using the following queries:

```
SQL > INSERT INTO Permits VALUES(6, "V", "Main", "regular", "park and ride","2023-10-17",  "2023-10-17", "18:00:00" );
Query OK, 1 row affected (0.0028 sec)

INSERT INTO PermitsToVehicles(6,'QWE123')
Query OK, 1 row affected (0.0024 sec)
```

Update Permit:

Input: SQL > UPDATE Permits SET ExpireTime = '18:30:00' WHERE PermitId= 6;
Output : Query OK, 1 row affected (0.0026 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Delete Permit:

The following are the SQL queries that will be executed for deleting a permit:

Input : DELETE FROM Permits WHERE PermitId= 6;
Output : Query OK, 1 row affected (0.0024 sec)

Input: DELETE FROM PermitsToVehicles WHERE PermitId = 6;
Output : Query OK, 0 rows affected (0.0012 sec)

Update vehicle ownership information:
Input: UPDATE Vehicles SET DriverId= 200505651 WHERE NumberPlate= "QWE123";
Output : Query OK, 1 row affected (0.0026 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Delete Vehicle:
Input: SQL > DELETE FROM Vehicles WHERE NumberPlate = "QWE123";
Output : Query OK, 1 row affected (0.0026 sec)

Generating and maintaining citations:

Check If Permit Is Valid
Input: SQL >SELECT CASE WHEN COUNT(*) > 0 THEN 'True' ELSE 'False' END AS IsValidPermit
FROM PermitsToVehicles pv
INNER JOIN Permits p ON pv.PermitId = p.PermitId
INNER JOIN Spaces s ON p.ZoneId = s.ZoneId AND p.LotName = s.LotName
WHERE pv.NumberPlate = 'ABC123' AND s.SpaceType = 'regular' AND s.LotName = 'Centennial' AND s.ZoneId = 'AS' AND p.StartDate <= CURDATE() AND (p.ExpireDate < CURDATE() OR (p.ExpireDate = CURDATE() AND p.ExpireTime < CURTIME()));

```
+----------------+
| IsValidPermit |
+----------------+
| True             |
+----------------+
1 row in set (0.0026 sec)
```

Create Citation

Input: SQL > INSERT INTO Citations VALUES(5, "2023-10-18", "16:00:00", 25, "Incomplete", "Expired Permit", "GHI789","Centennial");

Output : Query OK, 1 row affected (0.0027 sec)

Add Appeals Information

Input: SQL > INSERT INTO Appeals VALUES(200505650, 5, "The fee charged is wrong", "In process");

Output : Query OK, 1 row affected (0.0024 sec)

Update Citation

Input: SQL > UPDATE Citations SET Fee=30 WHERE CitationNumber=5;

Output : Query OK, 1 row affected (0.0027 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Update Appeals Information

Input: SQL > UPDATE Appeals SET Status="resolved" WHERE CitationNumber=5;

Output: Query OK, 1 row affected (0.0022 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Pay Citation

Input: SQL >UPDATE Citations SET PaymentStatus="Complete" WHERE CitationNumber=5;

Output: Query OK, 1 row affected (0.0033 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Delete Appeals Information

Input: SQL > DELETE FROM Appeals WHERE CitationNumber=5;

Output :4 rows in set (0.0012 sec)

Delete Citation

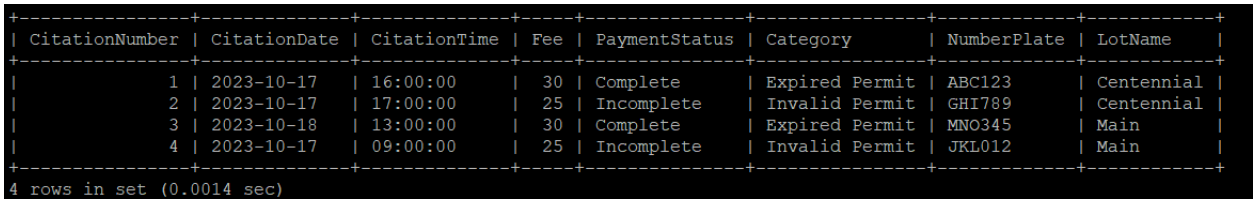
Input: SQL > DELETE FROM Citations WHERE CitationNumber=5;

Output: Query OK, 1 row affected (0.0023 sec)

REPORTS

Generate a report for citations.

SELECT CitationNumber, CitationDate,CitationTime,Fee,PaymentStatus,Category,NumberPlate, LotName FROM Citations;

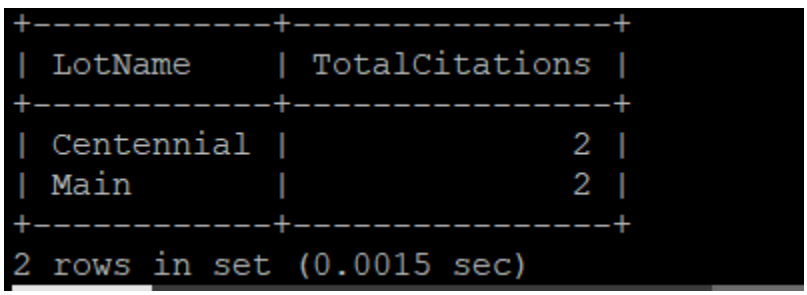


CitationNumber	CitationDate	CitationTime	Fee	PaymentStatus	Category	NumberPlate	LotName
1	2023-10-17	16:00:00	30	Complete	Expired Permit	ABC123	Centennial
2	2023-10-17	17:00:00	25	Incomplete	Invalid Permit	GHI789	Centennial
3	2023-10-18	13:00:00	30	Complete	Expired Permit	MNO345	Main
4	2023-10-17	09:00:00	25	Incomplete	Invalid Permit	JKL012	Main

4 rows in set (0.0014 sec)

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 LotName, COUNT(*) AS TotalCitations
FROM Citations
WHERE CitationDate BETWEEN '2023-10-17' AND '2023-10-18' GROUP BY LotName;



LotName	TotalCitations
Centennial	2
Main	2

2 rows in set (0.0015 sec)

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

SELECT LotName AS lot, Id AS Zone
FROM Zones
ORDER BY LotName, Id;

```
+------------+------+
| lot        | Zone |
+------------+------+
| Centennial | AS   |
| Centennial | BS   |
| Centennial | C    |
| Centennial | V    |
| Main       | CS   |
| Main       | D    |
| Main       | DS   |
| Main       | V    |
| Mckimmon   | AS   |
| Mckimmon   | B    |
| Mckimmon   | CS   |
| Mckimmon   | V    |
| Talley     | AS   |
| Talley     | BS   |
| Talley     | C    |
| Talley     | V    |
+------------+------+
16 rows in set (0.0015 sec)
```

Return the number of cars that are currently in violation.

We are assuming that all vehicles with payment status Incomplete are the ones currently in violation.

SELECT COUNT(DISTINCT NumberPlate) FROM Citations WHERE PaymentStatus = 'Incomplete';

```
+----------------------------+
| COUNT(DISTINCT NumberPlate) |
+----------------------------+
|                          2 |
+----------------------------+
1 row in set (0.0036 sec)
```

Return the number of employees having permits for a given parking zone.

SELECT COUNT(DISTINCT DriverId)

FROM Drivers

JOIN Vehicles ON Drivers.Id = Vehicles.DriverId

NATURAL JOIN PermitsToVehicles

NATURAL JOIN Permits

WHERE ZoneId = 'C' AND Status = 'E' AND lotName = 'Centennial';

```
+---------------------------+
| COUNT(DISTINCT DriverId)  |
+---------------------------+
|                         2 |
+---------------------------+
1 row in set (0.0022 sec)
```

Return permit information given an ID or phone number.

SELECT PermitId,ZoneId,LotName,SpaceType,PermitType,StartDate,ExpireDate,Expiretime,NumberPlate FROM Permits NATURAL JOIN PermitsToVehicles NATURAL JOIN Vehicles WHERE DriverId = ‘9191234588’;

```
+----------+--------+---------+-----------+--------------+------------+------------+------------+-------------+
| PermitId | ZoneId | LotName | SpaceType | PermitType   | StartDate  | ExpireDate | Expiretime | NumberPlate |
+----------+--------+---------+-----------+--------------+------------+------------+------------+-------------+
|        5 | V      | Main    | electric  | park and ride | 2023-10-17 | 2023-10-17 | 18:00:00   | MNO345      |
+----------+--------+---------+-----------+--------------+------------+------------+------------+-------------+
1 row in set (0.0016 sec)
```

Return an available space number given a space type in a given parking lot.

SELECT SpaceNumber FROM Spaces WHERE SpaceType = 'electric' AND LotName = 'Centennial' LIMIT 1;

```
+--------------+
| SpaceNumber  |
+--------------+
|            5 |
+--------------+
1 row in set (0.0017 sec)
```

4.2

Query 1:

EXPLAIN SELECT SpaceNumber FROM Spaces WHERE SpaceType = 'electric' AND LotName = 'Centennial' LIMIT 1;

```
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | Spaces | ALL  | NULL          | NULL | NULL    | NULL |   56 | Using where |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.0012 sec)
```

CREATE INDEX idx_SpaceType_LotName ON Spaces (SpaceType, LotName);

Query OK, 0 rows affected (0.0206 sec)

Records: 0 Duplicates: 0 Warnings: 0

EXPLAIN SELECT SpaceNumber FROM Spaces WHERE SpaceType = 'electric' AND LotName = 'Centennial' LIMIT 1;

```
+----+-------------+--------+------+----------------------+----------------------+---------+-------------+------+-----------------------+
| id | select_type | table  | type | possible_keys        | key                  | key_len | ref         | rows | Extra                 |
+----+-------------+--------+------+----------------------+----------------------+---------+-------------+------+-----------------------+
|  1 | SIMPLE      | Spaces | ref  | idx_SpaceType_LotName | idx_SpaceType_LotName | 68      | const,const |    2 | Using where; Using index |
+----+-------------+--------+------+----------------------+----------------------+---------+-------------+------+-----------------------+
1 row in set (0.0014 sec)
```

Query 2:

EXPLAIN SELECT COUNT(DISTINCT NumberPlate) FROM Citations WHERE PaymentStatus = 'Incomplete';

```
+----+-------------+-----------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table     | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-----------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | Citations | ALL  | NULL          | NULL | NULL    | NULL |    4 | Using where |
+----+-------------+-----------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.0013 sec)
```

CREATE INDEX PaymentStatusIndex ON Citations(PaymentStatus);

Query OK, 0 rows affected (0.0201 sec)

Records: 0 Duplicates: 0 Warnings: 0

EXPLAIN SELECT COUNT(DISTINCT NumberPlate) FROM Citations WHERE PaymentStatus = 'Incomplete';

```
+----+-------------+-----------+------+-------------------+-------------------+---------+-------+------+-----------------------+
| id | select_type | table     | type | possible_keys     | key               | key_len | ref   | rows | Extra                 |
+----+-------------+-----------+------+-------------------+-------------------+---------+-------+------+-----------------------+
|  1 | SIMPLE      | Citations | ref  | PaymentStatusIndex | PaymentStatusIndex | 12      | const |    2 | Using index condition |
+----+-------------+-----------+------+-------------------+-------------------+---------+-------+------+-----------------------+
1 row in set (0.0015 sec)
```

4.3

Query 1:

SELECT COUNT(DISTINCT DriverId)

FROM Drivers

JOIN Vehicles ON Drivers.Id = Vehicles.DriverId

NATURAL JOIN PermitsToVehicles

NATURAL JOIN Permits

WHERE ZoneId = 'C' AND Status = 'E' AND lotName = 'Centennial';

Relational Algebra query:

γ COUNT(DriverId)(δ (Π DriverId(σ ZoneId = ‘C’ AND Status = ‘E’ AND lotName =

$_{\text{'Centennial'}}$ (Drivers $\bowtie_{\text{Drivers.Id = Vehicles.DriverId}}$ Vehicles $\bowtie$ PermitsToVehicles $\bowtie$ Permits))))

Correctness Proof :

Suppose 'd' be a tuple in Drivers relation, 'v' be a tuple in Vehicles relation, 'ptov' be a tuple in PermitsTovehicles relation and p be a tuple in permits relation, such that d.Id = v.DriverId, v.NumberPlate = ptov.NumberPlate, ptov.PermitId = p.PermitId. The combination of all these tuples gives all information about Permits, Vehicle details corresponding to that permit and the corresponding driver details. Natural join gives all such tuples holding all these d.Id = v.DriverId, v.NumberPlate = ptov.NumberPlate, ptov.PermitId = p.PermitId conditions. Upon that we are selecting the rows having ZoneId = 'C' and Status = 'E' and lotName='Centennial' which gives us tuples which are belonging to Employees having permits in Zone C and Centennial ParkingLot. Then we are projecting DriverId, removing duplicates and printing the count which is exactly what we want.

Query 2:

```
SELECT
PermitId,ZoneId,LotName,SpaceType,PermitType,StartDate,ExpireDate,Expiretime,NumberPlate
FROM Permits NATURAL JOIN PermitsToVehicles NATURAL JOIN Vehicles WHERE DriverId =
'9191234588';
```

Relational Algebra Query:

$\Pi_{\text{PermitId,ZoneId,LotName.SpaceType,PermitType,StartDate,ExpireDate,ExpireTime,VehicleId}}(\sigma_{\text{DriverId= '9191234588'}}$ (Permits $\bowtie$ PermitsToVehicles $\bowtie$ Vehicles))

Correctness Proof :

Suppose v be a tuple in Vehicles relation, ptov be a tuple in PermitsTovehicles relation and p be a tuple in permits relation, such that v.NumberPlate = ptov.NumberPlate, ptov.PermitId = p.PermitId. The combination of all these tuples gives all information about Permits and Vehicle details corresponding to that permit. Natural join gives all such tuples holding all these v.NumberPlate = ptov.NumberPlate, ptov.PermitId = p.PermitId conditions. Upon that we are selecting the rows having DriverId = '9191234588' .Then we are projecting PermitId,ZoneId,LotName.SpaceType,PermitType,StartDate,ExpireDate,ExpireTime vehicles which are exactly what we want.

Wolf Parking Management System

CSC 540 Database Management Systems

Project Report 3

Team Members:

Naga Jahnavi Kommareddy (nkommar)

Shail Shah (sshah38)

Shrishty Singh (ssingh67)

Shyamal Tarpanbhai Gandhi (sgandhi6)

Assumptions

- Parking lots consist of Zones. Zone Id is unique within a parking lot. Zones consist of spaces. Each space has a space number which is unique within a Zone.
- UnivID and phone number never clash. Single ID attribute can be used to store UnivID or phone number based on the status of the driver. UnivID is stored in the ID attribute in case of a student or employee and phone number is stored in case of a visitor. UnivID is a 9 digit number and phone Number consists of 10 digits.
- It is mandatory for the permit card to be visibly displayed on the vehicle whenever it is parked. Security personnel will identify parking violations by inspecting the displayed permit cards periodically.
- When a security guard records a citation in the database, they will also affix a physical ticket to the vehicle. This ticket contains details about the violation and provides instructions for payment.
- Drivers have the option to initiate the payment process online by entering the citation number, which is found on the ticket. They can then proceed to pay the required fine based on the violation.
- Handicap users getting 50% discount on all citations will be handled at the operations level.
- Restrictions on number of permits and number of vehicles on each permit based on the status of the driver will be handled at operations level.
- One driver can make only one appeal per citation.
- A Vehicle cannot have multiple permits at a time.
- In the case of ‘No permit’, the security guard will manually check whether the vehicle has permit or not, if not, then the security guard will first request the administrator to add vehicle and driver details into the system and then security will manually add a citation for that particular vehicle by asking for all the information regarding vehicle and driver on the spot. (We are doing this because it might happen that in the case of no permit the drivers and vehicle’s details might not be there into the system and the citation table is linked with the vehicle's table so the foreign key constraint should be satisfied).
- When user raises a request for permit, we will check all the constraints based on user status and then if everything is ok then we will give parking permission to the user at particular location(parking lot, zone and space type), at this time hence manually checking every available space in the parking lot is not possible, we will randomly select any of the empty parking spaces and change its status to ‘NO’ instead of ‘YES’, which will help us to keep track of currently available parking spaces.
- A space is randomly selected and availability status is updated to NO in order to handle the permit overflow case in a space type in a particular zone and parking lot.
- Admin updates the Availability status of the space back to YES once that permit expires.
- The permit’s start date should only be the date on which the permit is being requested.

Corrections from Previous Reports

Report 1

Page 3 : In question 1 we made the description of the problem concise

Description:

The objective of the Wolf Parking Management System is to create a database for efficiently managing parking lots and their users on a university campus. Each parking area is subdivided into parking lots, into zones, and spaces. Each space has a unique space number, a space type, and an availability status. Drivers are categorized as students, employees, or visitors, identified by their ID or phone number. Drivers can obtain permits specifying the zone, space type, license number, and permit type. Citations are issued for parking violations, varying based on different categories. Administrators have the authority to manage parking lots, assign zones and spaces, issue permits, check permit validity, and handle citation-related tasks. Users must possess a permit matching the lot's designated zones and space types to park. Users are limited to one permit for students and visitors and up to two permits for employees. The system's core tasks include information processing, permit and vehicle maintenance, citation generation, and report generation for various parking-related data, ensuring effective parking management on campus.

Reasons for DBMS

- **Consistency and Integrity:**
    - Prevents conflicts, and reduces the risk of erroneous data
- **Redundancy Reduction:**
    - Minimizes data redundancy through normalization
- **Concurrent Access:**
    - Efficiently handles concurrent access
- **Security:**
    - Robust access control and authentication mechanisms secure data
- **Scalability:**
    - Can handle growing datasets and maintain performance
- **Durability:**
    - Automated data backup, recovery, and fault tolerance mechanisms
- **Data Maintenance and Updates:**
    - SQL operations reduce the risk of data inconsistencies
- **Report Generation:**

    - Powerful querying and reporting tools

Page7 : Added create/update/delete Vehicles API

- addVehicle(licenseNumber, model, color, manufacturer, year, driverId)
  return confirmation

- updateVehicle(licenseNumber, model, color, manufacturer, year, driverId)
  return confirmation

- deleteVehicle(licenseNumber, model, color, manufacturer, year, driverId)
  return confirmation

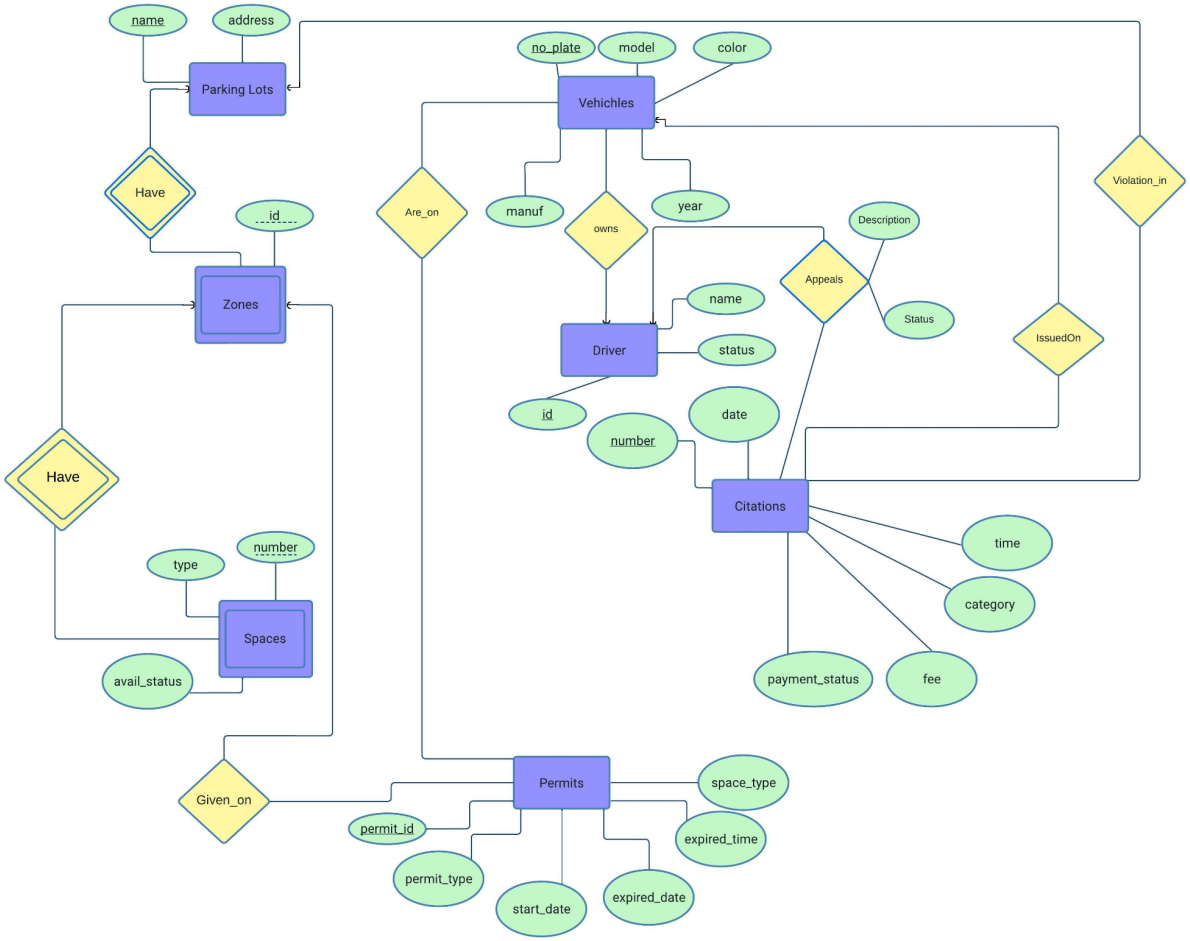
Page 8 : Added appeals API

- addAppeals(driverId,citationNumber,description,status)
  returns confirmation

- updateAppeals(driverId,citationNumber,description,status)
  returns confirmation
- deleteAppeals(citationNumber)
  returns confirmation

Report 2

(We made changes in er diagram to add “appeals”)



Page 4 : Fixed typo

ParkingLot(<u>Name</u>,Address)

Page 6: Added explanation for a non trivial FD in Citations Relation

CitationDate,CitationTime,NumberPlate->CitationNumber,Fee,PaymentStatus,Category,LotName holds because one vehicle (same number plate) cannot have two different citations at the same time and date. If the number plate, citation date and citation time is same then it is the same citation hence the citation number, fee, payment status, category and lot name will be same.

We take the closure of LHS of this functional dependency to determine if its a superkey or not. (CitationDate, CitationTime, NumberPlate)$^{+}$ =CitationNumber, CitationDate, CitationTime, NumberPlate, Fee, PaymentStatus, Category, LotName. This gives the entire relation, hence LHS of this functional dependency is a superkey.

The left-hand side of the functional dependency is a superkey hence the relation is in BCNF and 3NF.

Transaction 1 (Assign Permits)

```
//method to assign permit.
private void assignPermit() {
   try
   {
       dbManager.beginTransaction();// starting transaction
       //taking user input
       System.out.println("Enter permit id");
       String permitId = input.getString();
       System.out.println("Enter permit type");
       String permitType = input.getString();
       System.out.println("Enter start date");
       String startDate = input.getString();
       System.out.println("Enter expire date");
       String expireDate = input.getString();
       System.out.println("Enter expire time");
       String expireTime = input.getString();
       System.out.println("Enter space type");
       String spaceType = input.getString();
       System.out.println("Enter zone Id");
       String zoneId = input.getString();
```

```
        System.out.println("Enter lot name");

        String lotName = input.getString();

        System.out.println("Enter vehicle number plate");

        String numberPlate = input.getString();

        System.out.println("Enter driver id");

        BigInteger driverId = new BigInteger(input.getString());



        //Check is Permit exists.

        String checkPermitPresence = "SELECT * FROM Permits WHERE
PermitId = '"+permitId+"';";



        ResultSet resultSet =
dbManager.executeQuery(checkPermitPresence);

        if(resultSet.next()) {

            System.out.println("Permit already exists. Rolling back
the transaction");

            dbManager.rollbackTransaction(); // roll back the
transaction.

            return;

        }

        //Checking if driver id is present in drivers table.

        String driverPresence = "SELECT * FROM Drivers WHERE Id =
"+driverId+";";

        ResultSet resultSet1 = dbManager.executeQuery(driverPresence);

        if(!resultSet1.next()) {

            System.out.println("The given driver does not exist.
Please insert driver details using add driver operation first");

            System.out.println("Rolling back the transaction");

            dbManager.rollbackTransaction();//rollback the
transaction.
```

```
            return;

        } System.out.println();



        String checkVehiclePresence = "SELECT * FROM Vehicles WHERE
NumberPlate = '"+numberPlate+"';";

        //Checking if vehicle is present in vehicles table.

        ResultSet resultSet3 =
dbManager.executeQuery(checkVehiclePresence);

        if(!resultSet3.next()) {

            System.out.println("The given vehicle does not exist.
Please insert vehicle details using add vehcile operation first");

            System.out.println("Rolling back the transaction");

            dbManager.rollbackTransaction();//rollback the
transaction.

                return;

        } System.out.println();

        //Checking if driver is the owner of the vehicle.

        String checkVehicleDriverPresence = "SELECT * FROM Vehicles
WHERE NumberPlate = '"+numberPlate+"' AND DriverId = "+driverId+";";

        ResultSet resultSet2 =
dbManager.executeQuery(checkVehicleDriverPresence);

        if(!resultSet2.next()) {

            System.out.println("The given driver is not the owner of
the vehicle");

            System.out.println("Rolling back the transaction");

            dbManager.rollbackTransaction(); // roll back the
transaction.

            return;

        } System.out.println();

        //Check the number of current permits.
```

```
        String currentPermitsQuery = "SELECT COUNT(*) AS NumPermits "
+
                "FROM PermitsToVehicles AS PV " +
                "JOIN Vehicles AS V ON PV.NumberPlate = V.NumberPlate
" +
                "JOIN Permits AS P ON PV.PermitId = P.PermitId " +
                "WHERE V.DriverId = " + driverId + " " +
                "  AND (STR_TO_DATE(ExpireDate, '%Y-%m-%d') >
CURDATE() " +
                "OR (STR_TO_DATE(ExpireDate, '%Y-%m-%d') = CURDATE()
AND STR_TO_DATE(ExpireTime, '%H:%i:%s') > CURTIME()));";


        ResultSet resultSet4 =
dbManager.executeQuery(currentPermitsQuery);
        if(resultSet4 == null) {
            System.out.println("Error executing query");
            System.out.println("Rolling back the transaction");
            dbManager.rollbackTransaction();
            return;
        }
        int numCurPermits = 0;
        if(resultSet4.next()) {
            numCurPermits = resultSet4.getInt("NumPermits");
        }
        //Get driver status.
        String driverStatusQuery = "SELECT Status AS DriverStatus FROM
Drivers WHERE Id = "+driverId+";";
        ResultSet resultSet5 =
dbManager.executeQuery(driverStatusQuery);
```

```
        if(resultSet5 == null) {

            System.out.println("Error executing query");

            System.out.println("Rolling back the transaction");

            dbManager.rollbackTransaction(); //rollback the
transaction.

            return;

        }

        String driverStatus = null;

        if(resultSet5.next()) {

            driverStatus = resultSet5.getString("DriverStatus");

        }

        //Check max permit limit.

        if(("V".equals(driverStatus) && numCurPermits == 1)) {

            System.out.println("Max permits already exists. Cannot
give new Permit");

            System.out.println("Rolling back the transaction");

            dbManager.rollbackTransaction();

            return;

        }



        if(("S".equals(driverStatus) && numCurPermits == 1) ||
("E".equals(driverStatus) && numCurPermits == 2)) {

            if(!permitType.equals("special event")) {

                System.out.println("Max permits already exists. Cannot
give new Permit");

                System.out.println("Rolling back the transaction");

                dbManager.rollbackTransaction();

                return;
```

```
            }

        }


        //Check if given zone is valid for the driver.

        if(driverStatus.equals("V") && !zoneId.equals("V")) {

            System.out.println("Not allowed to take permit in this
zone");

            System.out.println("rolling back the transaction");

            dbManager.rollbackTransaction();

            return;

        }

        if(driverStatus.equals("S") && !(zoneId.equals("AS") ||
zoneId.equals("BS") || zoneId.equals("CS") || zoneId.equals("DS"))) {

            System.out.println("Not allowed to take permit in this
zone");

            System.out.println("rolling back the transaction");

            dbManager.rollbackTransaction();

            return;

        }



        if(driverStatus.equals("E") && !(zoneId.equals("A") ||
zoneId.equals("B") || zoneId.equals("C") || zoneId.equals("D"))) {

            System.out.println("Not allowed to take permit in this
zone");

            System.out.println("rolling back the transaction");

            dbManager.rollbackTransaction();

            return;

        }
```

```
        //Check if there is an available space.

        int space = -1;

        String query = "SELECT SpaceNumber AS SpaceNumber FROM Spaces
" +

                "WHERE SpaceType = '" + spaceType + "' AND ZoneId =
'"+zoneId+"' AND LotName = '" + lotName + "' " +

                "AND Avail_Status = 'YES' LIMIT 1";



        ResultSet resultSet6 = dbManager.executeQuery(query);



        if(resultSet6 == null) {

            System.out.println("Error executing query");

            System.out.println("Rolling back the transaction");

            dbManager.rollbackTransaction();

            return;

        }

        if(!resultSet6.next()) {

            System.out.println("No available space for the given space
type, zone and lot");

            System.out.println("Rolling back the transaction");

            dbManager.rollbackTransaction();

            return;

        } else {

            space = resultSet6.getInt("SpaceNumber");

        }



        //Query to uodate the availability status of random space to
NO to handle the permit overflow for a zone , space type and parking
lot.
```

```
        String query1 = "UPDATE Spaces SET Avail_Status = 'NO' WHERE
SpaceNumber = "+space+"  AND ZoneId = '"+zoneId+"' AND LotName = '" +
lotName + "';";



        //Queries to insert into PermitsTovehicles and permits tables.

        String query2 = "INSERT INTO Permits VALUES('%s', '%s', '%s',
'%s', '%s', '%s', '%s', '%s')";

        String query3 = "INSERT INTO PermitsToVehicles VALUES('%s' ,
'%s')";



        query2 = String.format(query2, permitId, zoneId, lotName,
spaceType, permitType, startDate, expireDate, expireTime);

        query3 = String.format(query3, permitId, numberPlate);



        //Executing the queries.

        int ra1 = dbManager.executeUpdate(query2);

        if(ra1 == -1) {

            System.out.println("Error inserting into Permits");

            System.out.println("Rolling back the transaction");

            dbManager.rollbackTransaction();// roll back the
transaction.

            return;

        }



        int ra2 = dbManager.executeUpdate(query3);

        if(ra2 == -1) {

            System.out.println("Error inserting into
PermitsToVehicles");

            System.out.println("Rolling back the transaction");
```

```
            dbManager.rollbackTransaction();//roll back the
transaction.

            return;

        }

        int ra3 = dbManager.executeUpdate(query1);

        if(ra3 == -1) {

            System.out.println("Error updating availability status");

            System.out.println("Rolling back the transaction");

            dbManager.rollbackTransaction();//roll back the
transaction.

            return;

        }



        System.out.println("Permit added successfully. Commiting the
transaction");



        dbManager.commitTransaction();//commit transaction.



    }

    catch(Exception e)

    {

        e.printStackTrace();

        System.out.println("Exception occured. Rolling back the
transaction.");

        dbManager.rollbackTransaction();//roll back the transaction.

    }

}
```

Transaction 2 (Delete Permit)

```
//Method to delete permit.
private void deletePermit() {
   try
   {
       dbManager.beginTransaction();//start transaction.
       System.out.println("Enter permitId: ");
       String permitId = input.getString();

       //Check Permit exists.
            String checkPresence = "SELECT * FROM Permits where
PermitId='"+permitId+"';" ;
       ResultSet resultSet = dbManager.executeQuery(checkPresence);
       if(!resultSet.next()) {
           System.out.println("No such permit found");
           dbManager.rollbackTransaction();
           return;
       } System.out.println();
       //Query to delete from PermitsTovehicles.
        String query1 = "DELETE FROM PermitsToVehicles WHERE PermitId
= '"+permitId+"'";

       //Query to delte from Permits.
           String query2 = "DELETE FROM Permits WHERE PermitId =
'"+permitId+"'";

       //Executing the delete queries.
       int rowsAffected1 = dbManager.executeUpdate(query1);
       if(rowsAffected1 == -1) {
             System.out.println("Error executing the query 1.. Hence
rolling back the transaction");
           dbManager.rollbackTransaction();
           return;
       }
```

```
       int rowsAffected2 = dbManager.executeUpdate(query2);
       if(rowsAffected2 == -1) {
             System.out.println("Error executing the query 2.. Hence
rolling back the transaction");
           dbManager.rollbackTransaction();
           return;
       }
                System.out.println("Number of rows affected in
PermitToVehicles are "+rowsAffected1);
         System.out.println("Number of rows affected in Permits are
"+rowsAffected2);
            System.out.println("Permit deleted successfully. Hence
committing the transaction");
       dbManager.commitTransaction();
       resultSet.close();
   }
   catch(SQLException e)
   {
       e.printStackTrace();
            System.out.println("Exception occured. Rolling back the
transaction.");
       dbManager.rollbackTransaction();//roll back the transaction.
   }

}
```

Design Decisions:

The system has a main menu that gives users the following options: “Information Processing”, “Maintaining permits and vehicle information for each driver”, “Generating and maintaining citations”, “Reports”, “Exit”. When prompted, the user enters a number 0-5 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 (Main.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 DBManager, which, through the use of a shared database connection, provides the ability to execute SQL queries and perform transactions through various utility methods. The other class is Input, which is a class that attempts to abstract the details of getting different types of input from the user, and includes a method for printing a ResultSet object.

Functional Roles:

Report 1:

Software Engineer: Shail(Prime), Shrishty(Backup)

Database designer/Administrator: Jahnavi(Prime), Shyamal(Backup)

Application Programmer: Shrishty(Prime), Shail(Backup)

Test Plan Engineer: Shyamal(Prime), Jahnavi(Backup)

Report 2:

Software Engineer: Shrishty(Prime), Shyamal(Backup)

Database designer/Administrator: Shail(Prime), Jahnavi(Backup)

Application Programmer: Jahnavi(Prime), Shrishty(Backup)

Test Plan Engineer: Shyamal(Prime), Shail(Backup)

Report 3:

Software Engineer: Jahnavi (Prime) , Shail (Backup)

Database designer/Administrator: Shrishty(Prime) , Shyamal(Backup)

Application Programmer: Shyamal(Prime) , Jahnavi(Backup)

Test Plan Engineer: Shail(Prime), Shrishty(Backup)