CSC540 Database Management Systems

University Parking Database Management System

Developed By Team “G”

Shreyas Muralidhara - schikkb

Niranjan Pandeshwar - nrpandes

Prathamesh Pandit - pppandi2

Sreemoyee ray - sray9

1. Problem Statement

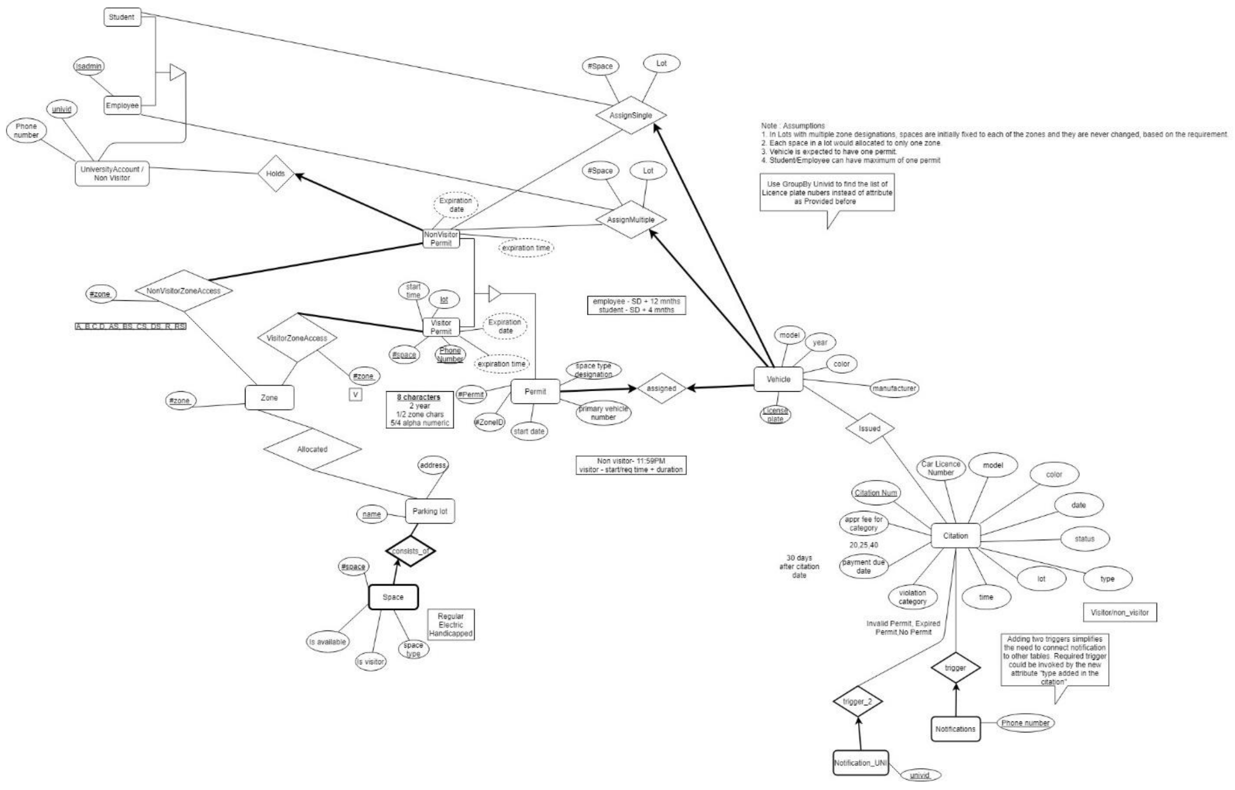
We have developed a university parking system (UPS) to manage campus parking lots and its user. The UPS issues parking permits to employees, students and visitors and there are different eligibility constraints for parking
permits in the different lots as well as time restrictions for eligibility. In addition to the permits, UPS issues tickets/citations for parking violations and collects fees for them. University students and employees all have a univid (integer) which is a unique identifier for identifying them and linking them to their vehicles as well as an attribute status that is either 'S' or 'E' or 'A' depending on whether a student or an employee or administrator (who is also an employee but works with UPS).

Assumptions:
1. Each space in a lot is allocated to only one zone.
2. A vehicle can have at most one valid permit at a time.
3. A student/employee can have one valid permit.
4. A visitor can have one valid permit at a time.
5. While creation of a lot, the number of visitor spaces are fixed. The remaining spaces can belong to any zone id.

The following are the most significant entities of the Parking Lot system

1. Parking Lot
2. Zone
3. Space
4. Non - Visitor – Employees and Students
5. Visitor
6. Permit – Visitor and Non- Visitor
7. Citations
8. Vehicles

ER diagram:



Link to ER diagram:
https://github.ncsu.edu/schikkb/DBMSWolf_UPS/blob/master/ER%20design.jpg

Users:

There are 4 types of users in this system who interact with the system using respective user interfaces.

1) **Visitor:** A visitor to the parking lot can perform one of the following actions:
    a. Enter the parking lot by getting a permit.
    b. Exit the lot
    c. Pay a citation

2) **Student:** A student can perform the following actions:
    a. Enter the lot
    b. Exit the lot
    c. View vehicle list assigned to their permit
    d. Change vehicle list assigned to their permit
    e. Pay citation

3) **Employee:** Employees can be of two types – regular employees and admins. Admins have special privileges and can perform some restricted functions.

   A regular employee can do the following:

   d. Enter the lot
   e. b. Exit the lot
   f. c. View vehicle list assigned to their permit
   g. d. Change vehicle list assigned to their permit
   h. e. Pay citation

4) **Admin:** Admin has special privileges like managing the parking lots, adding new lots, assigning zones to lots, issuing permits, etc. The list of actions admins can perform are given below:
   a. Add Lot
   b. Assign Zone to Lot
   c. Assign type to Zone
   d. Assign permit (to student and employees)
   e. Check visitor valid parking
   f. Check non visitor valid parking

User Interfaces and APIs:

There are 5 user interfaces in the system – the admin UI, employee UI, student UI, visitor UI and the demo UI. The demo UI allows us to run the preassigned queries.

Each UI offers certain functionality to the respective users thought a wide range of APIs.

The following section describes the different APIs of the UIs present in the system.

Admin APIs:

Addlot

This API is used to create a new parking lot. It takes from the admin the following inputs –
1. parking lot name
2. address
3. number of spaces for the lot
4. the beginning space number
5. initial zone designation.

It inserts 3 entries into the 3 tables – Parkinglot, Space, Rel_allocated according to the values given by the admin.

AssignZoneToLot

This API can be accessed by the admin employees to add a zone type to a particular lot by providing the following inputs –

1. Lot Name
2. Zone type
3. Number of spaces to be allotted (in case of zone type – V (visitor)

If the zone type is of type visitor, the isvisitor flag of the space table is set to 1 for as many records as the number of spaces to be allotted for zone type V for the lot.

If a zone is not of type visitor, but a visitor zone already exists in the parking lot, input the number of spaces to be allocated for the visitor zone and update space table accordingly. If visitor zone does not exist, no change needed in space table.

AssignTypetoSpace

This API assigns a type to a particular space within a parking lot. Spaces are by default regular. They can also be of a special type for e.g. handicapped or electric.

The API takes as input the following:

1. Lot Name
2. Space Number
3. Space type

AssignPermit

Admins can assign new permits for student and employees. Admin takes as input the univid of the user, preferred zone, space type, vehicle number and creates a new permit. Before permit creation, the system checks if a permit already exists for the user in the vehicle table. If it does, a new permit cannot be created.

CheckVValidParking

Admins can check whether a valid permit exists for a visitor given a Car license number, lot name and space id.

CheckNVValidParking

Admins can check whether a valid permit exists for a nonvisitor given a Car license number, lot name and space id.

Employee APIs:

emp_EnterLot:

An employee can enter a parking lot with a valid permit and park in a lot of their choice, out of the lots with available space.
This API checks the following:

1. Whether employee has a valid permit
2. Whether the vehicle number is correct for the permit

If a violation is found, a citation is issued for the employee.

emp_ExitLot:

An employee can exit the parking lot with this API. While exiting, the expire time of the permit is checked. It the permit is found to be invalid at that time, a citation is issued for the vehicle.

emp_ViewVehicleInfo:

An employee can view the vehicle details associated with their permit using this API.

emp_ChangeVehicleInfo:

An employee can change the vehicle details associated with their permit using this API.

Student APIs:

StudentEnterLot:

A student can enter a parking lot with a valid permit and park in a lot of their choice, out of the lots with available space.
This API checks the following:

1. Whether employee has a valid permit
2. Whether the vehicle number is correct for the permit

If a violation is found, a citation is issued for the student.

StudentExitLot:

A student can exit the parking lot with this API. While exiting, the expire time of the permit is checked. It the permit is found to be invalid at that time, a citation is issued for the vehicle.

StudentViewVehicleInfo:

A student can view the vehicle details associated with their permit using this API.

StudentChangeVehicleInfo:

A student can change the vehicle details associated with their permit using this API.

Visitor APIs:

GetVisitorPermit:

A visitor can use this API to get a new permit by giving their phone number.

Common APIs:

PayCitation:

Visitors and Non-visitors can use this API to pay for the citations received due to various reasons like parking without permit, parking with expired permit, etc.

Constraints:

There are certain constraints that could not be enforced at the database level and are controlled from the code. Below are the details:

1. The space id attribute is assigned through the API Addlot. It is assigned an integer value which is incremented by 1 for each space and lot. It is controlled by the program because it is not incremented according to a particular lot and not for the overall table.
2. In the AssignZoneToLot API, the API asks the admin to provide the number of visitor spaces to be allocated for the visitor zone type. It checks if the provided number of spaces are available in the parking lot. This constraint is controlled by the program and not present in table structure.
3. Constraint to check whether the zoneid in a permit follows in the allowable set of zones for the user while assigning permits. For e.g. zoneid for student cannot cannot be A or B; this constraint is implemented in the AssignPermit API.
4. Constraint to check if a permit already exists for a vehicle in the vehicle table while assigning new permits through the AssignPermit API.

5. In EnterLot API for student and employees, a constraint is in place to check if employee has a valid permit and if they are currently using only the vehicle for which the permit is issued.
6. While exiting lot, i.e. deleting a tuple from the respective table, a constraint checks if the permit is valid. If not, a citation is issued, and citation details are inserted into the citation table.
7. Only a student/employee with a valid permit can change the vehicle info associated with their permit.
8. While getting a new visitor permit, a constraint checks if the visitor already has a valid permit. If yes, a permit cannot be issued again.

Functional Dependencies

Given below are the attribute list, functional dependencies and normal forms for each relation in the schema.

PARKINGLOT (NAME, ADDRESS)

This relationship stores all the names of the parking lots along with their address.

NAME -> NAME, ADDRESS

Name attribute of Parkinglot uniquely identifies the address. This relationship is thus in BCNF and all the lower level functional dependencies also hold.

SPACE (SPACEID, LOTNAME, SPACETYPE)

This relationship stores information about spaces within parking lots. Each space has a spaceid which uniquely identifies the space within the given parking lot. Each space has a type, for e.g. regular or handicapped.

SPACEID, LOTNAME -> SPACEID, LOTNAME, SPACETYPE

The primary key for this relationship is (SPACEID, LOTNAME). All the attributes are uniquely identified by the primary key only. This relationship is thus in BCNF.

ZONE (ZONEID)

This relationship contains all the possible zones that can exist for parking lots. It can take values from the set ('A','B','C','D','AS','BS','CS','DS','R','RS','V').

ZONEID -> ZONEID

This relationship is in BCNF since there is only one attribute which is also the primary key.

REL_ALLOCATED (ZONEID, NAME)

This relationship identifies what all zones lies within a particular parking lot denoted by name attribute.

ZONEID, NAME -> ZONEID, NAME

This relationship is in BCNF and satisfies all lower level functional dependencies.

NONVISITOR (UNIVID, PHONENO)

This relationship stores the information for all the nonvisitors in the system. Nonvisitors are of two types -students and employees. Each nonvisitor can be uniquely identified using the UNIVID attribute.

UNIVID -> UNIVID, PHONENO

Univid attribute is the primary key and from the functional dependencies shown above, the relationship is in BCNF.

STUDENT (UNIVID)

This relationship stores all the student Univids.

UNIVID -> UNIVID

Unidiv is the primary key and this relationship is in BCNF.

EMPLOYEE (UNIVID, ISADMIN)

This relationship stores all employee univids and a flag to determine if they are admin or not.

UNIVID -> UNIVID, ISADMIN

This relationship is in BCNF.

PERMIT (PERMITNO, ZONEID, STARTDATE, PRIMARYVEHICLENO, SPACETYPE)

This relationship stores information of all the permits issued to visitors/nonvisitors for using parking lots.

PERMITNO -> PERMITNO, ZONEID, STARTDATE, PRIMARYVEHICLENO, SPACETYPE

Permitno is the primary attribute in this relationship and it determine all other attributes of this relationship. Hence, this relationship is in BCNF.

VISITORPERMIT (PERMITNO, LOTNAME, STARTTIME, EXPIRETIME, EXPIREDATE, SPACENO, PHONENO)

This relationship is used to store the additional attributes that should be present for a visitor permit.

PERMITNO -> PERMITNO, LOTNAME, STARTTIME, EXPIRETIME, EXPIREDATE, SPACENO, PHONENO

This relationship is in BCNF since the primary key determines all the attributes.

VISITORZONEACCESS (PERMITNO, ZONEID)

This relationship determines which zone is allowed for a particular permit no.

PERMITNO -> PERMITNO, ZONEID
PERMITNO is the primary attribute and hence, this relationship is in BCNF.

NONVISITORPERMIT (PERMITNO, UNIVID, EXPIRETIME, EXPIREDATE)

PERMITNO -> PERMITNO, UNIVID, EXPIRETIME, EXPIREDATE

This relationship is in BCNF since the primary attribute determines all the other attributes.

REL_NONVISITORZONEACCESS (PERMITNO, ZONEID)

This relationship is in BCNF since there are no other attributes apart from the primary key.

VEHICLE (LICENSEPLATE, MANUFACTURER, MODEL, YEAR, COLOR, PERMITNO)

This table stores the vehicle details that are currently

LICENSEPLATE -> LICENSEPLATE, MANUFACTURER, MODEL, YEAR, COLOR, PERMITNO

ASSIGNMULTIPLE (VEHICLENO, UNIVID, PERMITNO, SPACENO, LOTNAME, PARKDAT)

VEHICLENO -> VEHICLENO, UNIVID, PERMITNO, SPACENO, LOTNAME, PARKDAT
This relationship is in BCNF as can be observed from the FD above.

CITATION (<u>CITATIONNO</u>, CARLICENSENO, MODEL, COLOR, ISSUEDATE, STATUS, TYPE, LOT, ISSUETIME, VIOLATIONCATEGORY, PAYMENTDUE, VIOLATIONFEE)

CITATIONNO -> CITATIONNO, CARLICENSENO, MODEL, COLOR, ISSUEDATE, STATUS, TYPE, LOT, ISSUETIME, VIOLATIONCATEGORY, PAYMENTDUE, VIOLATIONFEE

CARLICENSENO -> CARLICENSENO, MODEL, COLOR

This relationship is in 2NF. The second functional dependency shown above violates the conditions to this to be in 3NF or BCNF.

NOTIFICATIONVISITOR (PHONENO, CITATIONNO)

PHONENO, CITATIONNO -> PHONENO, CITATIONNO

This relationship is in BCNF.

NOTIFICATIONNONVISITOR (UNIVID, CITATIONNO)

UNIVID, CITATIONNO -> UNIVID, CITATIONNO

This relationship is in BCNF.

ASSIGNSINGLE (UNIVID, PERMITNO, VEHICLENO, SPACENO, LOTNAME, PARKDAT)

VEHICLENO -> VEHICLENO, UNIVID, PERMITNO, SPACENO, LOTNAME, PARKDAT
This relationship is in BCNF as can be observed from the FD above.

Handling Transaction Management:

Each API makes changes to more than one table for an action. For e.g., when a new permit is being issued for a Visitor, an entry is made in the Permit table as well as Visitorpermit table. To ensure that the atomicity of this set of transactions is maintained, transaction management is implemented in the system. Without transaction management, one entry can be successful, and one can fail leaving the data in an erroneous state.

Transaction management uses transaction variables to track the status of each database transaction. At first, the auto - commit is disabled. Then multiple transactions occur one by one and the status of each one is captured. If all the transactions are successful, a commit is made of the database and auto – commit is set to true.