Database Management Concepts and Systems

Team P Members	Unity ID
Saurabh Patil	srpatil4
Aishwarya Karad	amkarad
Priyanka Chawla	pchawla
Rahul Saxena	rsaxena2

UPS: University Parking System

The design and development processes of DBMS for UPS are completed in the Project Report

Description of the Project:

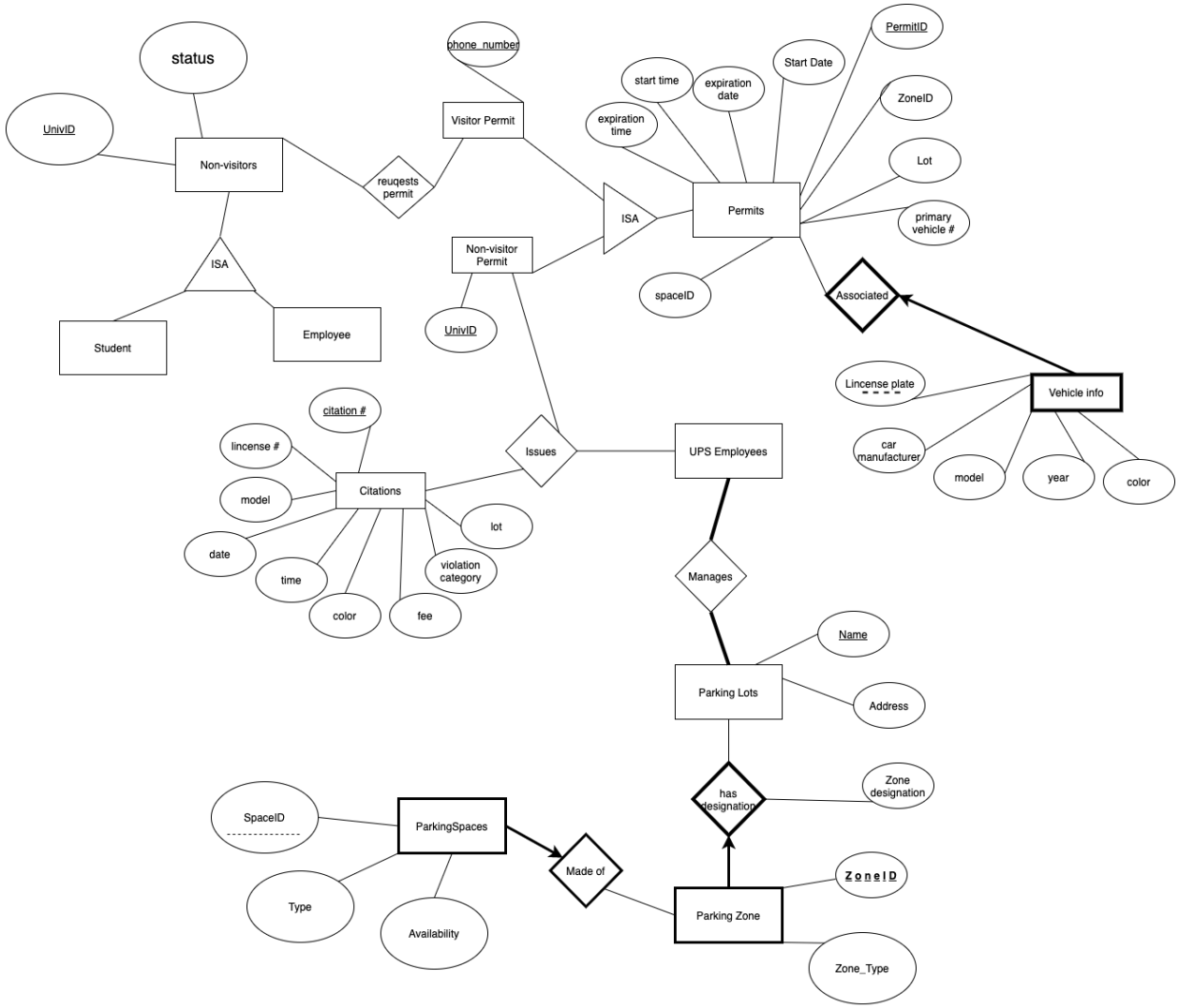
The UPS is designed and built to manage and maintain information of Lots, spaces, Parking Permits for Visitors and Non-Visitors, Citations in case of violation. It is not only used to store data but also update entries in case of any change in the data.

For each Parking entry, this tool will maintain records such as phone/university id, permit lot-zone-space information, type of spaces, parking time entry, duration requested by visitors and much more.

The database system is developed for UPS and is used by admin, students and employees. The UPS database system addresses constraints like

1. Availability of parking slots depending on the requirement of user
2. Maintaining details of Visitor, Non-Visitor (Employee and Students), Vehicles, Citations, start time and end time.

E/R Model and Design:



Key for the E/R model:

Bold line: Total participation
Bold Arrow: EXACTLY 1 Cardinality
Ovals: Attributes of entity
Regular Line: 0 to many Cardinality
Bold Rectangle: Weak Entity

We have a Non Visitor entity set which captures all the non visitors and their necessary information such as University ID. Students and Employees have a “generalized” relationship with the Non-Visitor entity set.

ParkingLots have designated **ParkingZones** made of **ParkingSpaces** and since any parking lot can have multiple zone “designations” we have created two separate entity sets for both Zones and Lots in our E/R Diagram. Also, since parking zones physically contain spaces where each space is uniquely identified within that parking lot we have created a separate entity for ParkingSpaces which is **weakly related to** ParkingLots in a **many to one relationship.**

Non-Visitor and Visitor Permits have a “generalized” relationship with the Permits entity which is captured via E/R approach.

The Vehicle info entity set is weakly related to the Permits entity set and contains all the information related to the vehicle for which the permit was generated.

We have created the table (entity set) **Citations** which is in a many to many relationship with UPS employees via the relationship set **Issues.**

Entity UPS employees have a **many to many** relationship with Parking lots indicated in the E/R model using the “manages” relationship. UPS employees also have a **many to many** relationship with the Non Visitor Permit entity set.

We aim to create a table based on the entity Vehicle info, with **primary key** license plate and is **weakly related** using the “associated” relationship with Permit Entity

Relational Design:

The following tables can be created using the design_tables.sql file and then filled with appropriate data tuples using the fill_table.sql file. Both these files contain the relation design part of the project.

- *NON_VISITOR*
  This table has two attributes: **uid** (contains the university ID) and **nv_status** (defines the type of non visitor, employee or student). The primary key for this table is uid as the university ID is unique for all. For any new non visitor entity, an entry is created in this table.

- *ParkingLots*
  This table has four attributes: **lot_name** (contains the name of the lot), **lot_addr** (contains the address of the lot), **zone_designation** (designation of the zones like A, B, C), and **total_num_spaces** . The primary key for this table is the lot_name, as each lot has a unique name associated with it.

- *ParkingZone*
  This table has three attributes: **zone_id**(id of the zone like A,B,C etc), **num_spaces**(contains the number of spaces present in that zone) and **lot_name**(contains the name of the lot). The primary key for this table is a combination of lot_name and zone_id, as different lots can have the same zone ids, xone_id is unique only to a particular lot. The lot_name attribute is referenced from the ParkingLots table as foreign key.

- *ParkingSpaces*
  This table has five attributes: **space_id** (uniquely identifies each space in a zone), **space_type** (contains the type of space like Handicapped or Regular or Electric), **availability** (which defines if the space is available or not), and **lot_name** and **zone_id** which are referred from the ParkingZone table as foreign keys.

- *Permits*
  This table has nine attributes: **zone_id**(id of the zone like A,B,C etc), **permit_id**(a unique identifier for each permit),**start_date**(start date of the permit),**expiration_date**(contains expiration date of the permit),**start_time**(contains start time of the permit), **expiration_time** (contains the expiration time of the permit), **duration**(duration allotted to Visitor for parking in that particular space),**space_id**(uniquely identifies each space in a zone),**lot_name**(contains the name of the lot). The primary key for this table is permit_id. It is one of the crucial tables in our design as it maintains the crucial details about the permit. The attributes lot_name,zone_id, and space_id are referenced from the ParkingSpaces table as foreign keys.

- *Non_Visitor_Permit*
  This table has only two attributes: **univid** which is referenced from the NON_VISITOR table and **permit_id** which is referenced from the Permits table. Both are foreign key constraints.

- *Visitor_Permit*
  This table has two attributes: **phone** (which contains the phone number of visitors for unique identification) and **permit_id** which is referenced as foreign key form the Permits table.

- *Vehicle_Info*
  This table has eight attributes: **car_year(**tells about purchase year of the car), **univid** referenced from the NON_VISITOR table, **phone**(an identifier for visitors),**permit_id** (a unique identifier for each permit) , **license_plate** (unique identifier for the car), **model** (tells about the model of car), **color**(tells about color of the car), **car_manufacturer**(tells about the manufacturer of car). The primary key of this table is a combination of permit_id and license_plate. Also, the attributes like permit_id, univid, and phone are referenced from the Permits, Non_Visitor_Permit, and Visitor_Permit tables respectively as foreign keys. This

table is referred to by other tables and queries to get information about cars, using their license plate which is the primary key for this table.

- *Violations*
  Now, this table is created separately from the CITATIONS table to capture the functional dependency present in the CITATIONS table. This is further explained in the functional dependency section below. It has two attributes: **VIOLATION** which captures the category of violation like No permit, Expired permit, etc. Next, it has a **fee** which contains the amount needed to be paid for that particular citation category.

- *CITATIONS*
  This table has eight attributes: citation_number, cit_date, cit_time, ispaid, due_date. The other attribute is **lot_name** referenced from the ParkingLots table as foreign key. **VIOLATION** which is referenced from the Violations table as foreign key. Then, **license_plate** which is referenced from the Vehicle_Info table as foreign key. This particular table also has a CHECK constraint which verifies whether the due date is one month after the citation date.

- *Notification*
  This table just captures whether the notification has been sent to the visitor or non visitor after the citation has been paid. It has two attributes: **citation_number** which is referenced from the CITATIONS table as foreign key and **contact_info** which can be either a phone number for visitors or an university ID for non visitors.

Functional Dependencies:

We have identified in the Citations table that “Violation” and “Fee” attributes are interdependent on each other. For any particular violation, a fixed fee has been defined. In case if we do not maintain a separate table for Violation and Fees then the problem of loss/insufficient data may arise in future. For instance, if we have a single entry for an “Invalid permit” in the citation table and later if that entry is removed, then we won’t be having any information on the fee related to the “Invalid Permit”. Also, if we want to add a new violation category in the table, for which an already present entry is not in the table, we won;t be able to add it. This can be surely handled easily in the application code, but we have handled it using DBMS. To overcome this issue, we have created a separate table of “Violation and Fees”, and removed the “fee” attribute from the citation table. “Violations” acts as a primary key in the table of “Violation and Fees” and it is also present in the citation table for the reference. This is how we have used functional dependencies in our database design.

Constraints handled through Application Code:

Some constraints handled through the application code rather than DBMS are:

1. In the project description it is mentioned that each employee can have at most two vehicles associated with a non-visitor permit and each student can have only a single vehicle associated with a non-visitor permit. This constraint is handled in the application code rather than using Triggers in DBMS. So, the non-visitors can add/remove the vehicles in a permit using only the ChangeVehicleLost() function. To apply the above constraints we have restricted the number of vehicles inside the application code. Before adding a new vehicle to the permit, the code first checks if a max number of vehicles are associated, and if yes, we do not allow that action.
2. Also, in the description it is mentioned that the expiration date and time of each non-visitor permit is 12 months and 11:59 pm from the start date for employees and 4 months and 11:59 pm from the start date for the students. Again, rather than using triggers before INSERT, we have implemented this constraint in the application code. The permits are generated using the AssignPermit() function. In this code, we do not take the expiration date and time input from the user, but just simply calculate the values based on the start date.

Application Code Design:

This program code allows retrieving, storing, manipulating and deleting any data from the DBMS through a UI built in python.
Initially, the sample data has been loaded into the database using the script “demo_data.py”.

Later, it follows below architecture

Interaction between user and programs takes place via a UI called Menu.

Menu:

At first place, it will ask for what action program is needed and provide below options

1. *UPS Admin*
2. *Employee*
3. *Student*
4. *Sample_query*
5. *Reportig_query*

Opted for 1> ups_admin

Then, below are the actions can be performed by selecting corresponding input number(Purpose of each action has been defined in Functions table)

1. *ExitLot*
2. *IssueCitation*
3. *PayCitation*
4. *AssignPermit*
5. *addLot*
6. *AssignZoneToLot*
7. *AssignTypetoSpace*
8. *CheckVValidParking*
9. *CheckNVValidParking*

Opted for 2> Employee

Then, below are the actions can be performed by selecting corresponding input number(Purpose of each action has been defined in Functions table)

1. *ChangeVehicleList*
2. *GetVisitorPermit*

Opted for 3> Student

Then, below are the actions can be performed by selecting corresponding input number(Purpose of each action has been defined in Functions table)

1. *GetVisitorPermit*

Opted for 4> Sample_query

Then, below are the actions can be performed by selecting corresponding input number(Purpose of each action has been defined in Functions table)

1. *Show the list of zones for each lot as tuple pairs (lot, zone).*
2. *Get permit information for a given employee with UnivID: 1006020*
3. *Get vehicle information for a particular UnivID: 1006003*
4. *Find an available space# for Visitor for an electric vehicle in a specific parking lot: Justice Lot*
5. *Find any cars that are currently in violation*
6. *How many employees have permits for parking zone D*

Opted for 5> Reporting_query

Then, below are the actions can be performed by selecting corresponding input number(Purpose of each action has been defined in Functions table)

1. *For each lot, generate the total number of citations given in all zones in the lot for a three month period (07/01/2020 - 09/30/2020)*
2. *For Justice Lot , generate the number of visitor permits in a date range: 08/12/2020 -08/20/2020 , grouped by permit type e.g. regular, electric, handicapped.*
3. *For each visitor's parking zone, show the total amount of revenue generated (including pending citation fines) for each day in August 2020*

All of the above-mentioned queries are performed in the backend on the MySQL, using

- ***SELECT***
- ***INSERT***
- ***UPDATE***
- ***DELETE***

Functions Details:

`ExitLot(db, permit_id)`

This function checks the validity of the visitor's permit and accordingly creates a citation if necessary. The conditions for issuing citation are:

- If the permit is not present, create No permit citation,
- If the time overage has occurred, create permit expired citation.

The input to this function is permit_id.

Eg: For python 2 → '10020'

For python 3 → 10020

(Enter the permit_id as '0' if no permit present)

`ChangeVehicleList(db, permit_id, univid)`

This function makes changes in the Vehicle Info table. It is used to delete or add a new entry into the Vehicle Info table for a particular permit ID.

Eg: For python 2:

Permit id → '10020

Univid → '100603'

For python 3:

Permit id → 10020

Univid → 100603

`IssueCitation(db,permit_id,citation_time,permit_type)`

This function creates a citation for a particular permit, that is adds an entry in the CITATIONS table. Once the citation is created, we also create a new entry in the notification table for the same. There are 3 types of Citations possible: No permit, Expired permit, and Invalid Permit.

For No permit; we check if the permits table has that particular permit_id. If not, create a citation.

For Expired permit; we calculate the time overage and if it is greater than expiration time, create a citation.
For Invalid permit; we run the CheckVValidParking and CheckNVValidParking functions. These functions are explained below. If these fail, we create a citation.
The input for this function are:
Eg: For python 2:

permit_id → '10020'
Citation_time → '03:00:00' (for 3pm)
Permit_type → 1 (for No permit)
2 (for Expired)
3 (for Invalid)

For python 3:

permit_id → 10020
Citation_time → 03:00:00 (for 3pm)
Permit_type → 1 (for No permit)
2 (for Expired)
3 (for Invalid)

PayCitation(db, cin)

This function is used to update the "ispaid" attribute entry in the citation table corresponding to a particular citation number.
Eg: For python 2:

Cin → 10001

For python 3:

Cin → 10001

GetVisitorPermit(db, phone, duration, R_H_E, lot_name)

This function is used to create an entry for visitors in the Permits table as well as in the Visitor_Permit table. Firstly, it will check the availability of spaces on the basis of input given by the user and then create an entry in both the tables.
Eg: For python 2:

Phone → '7777177777'
Duration → 3
R_H_E → 'Regular'

Lot_name → 'Justice lot'

For python 3:

Phone → 7777177777
Duration → 3
R_H_E → Regular
Lot_name → Justice lot

AssignPermit(db, univid, lot_name, S_E, R_H_E)

This function is used to create an entry for non-visitors (Students or Employees) in the Permits table as well as in the Non_Visitor_Permit table. Firstly, it will check the availability of spaces on the basis of input given by the user and then create an entry in both the tables

Eg: For python 2:

univid → '100603'
Lot_name → 'Justice lot'
S_E → 'S'
R_H_E → 'Regular'

For python 3:

univid → 100603
Lot_name → Justice lot
S_E → S
R_H_E → Regular

addLot(db, Name, Address, Zones, Total_Spaces)

NOTE:(You will have to add the range of spaces for each zone in order by running the menu_ui code file one after the other. For example, In the parking lot 'NCSU Lot' you have zones A,B,V and a number of spaces 500. Then, you will have to run the menu_ui.py file 3 times. One for each zone: AssignZoneToLot() and AssignTypetoSpace().

1.AssignZoneToLot()--> zoneId A, # of spaces is 100
  AssignTypetoSpace()--> zoneId A, range of spaces 0:100
2. AssignZoneToLot()--> zoneId B, # of spaces is 100
  AssignTypetoSpace()--> zoneId A, range of spaces 101:201
And handicapped is 180:189 and electric 190:201**)**

This function adds a new tuple in the ParkingLots table, i.e creates a new lot in the DB. Input to this function are:

Eg: For python 2:

Name → 'NCSU Lot'
Address → 'Parkway Lane, 27600'
Zones → 'A, B, C, V'
Total_Spaces → 250

For python 3:

Name → NCSU Lot
Address → Parkway Lane, 27600
Zones → A, B, C, V
Total_Spaces → 250

AssignZoneToLot(db,Name,Zones,Total_num_spaces_per_zone)

This function is used to assign zones to existing lots. It will take input from the user and accordingly will create zones in the lot.

Eg: For python 2:

Name → 'NCSU Lot'
Zones → 'A'
Total_num_spaces_per_zone → '500'

For python 3:

Name → NCSU Lot
Zones → A
Total_num_spaces_per_zone → 500

AssignTypetoSpace(db, Name, Zones,num_spaces_range, handicapped_range, electric_range)

This function basically assigns new spaces to the zones. That is, creates new tuples in the ParkingSpaces table for a particular parking lot and its zones.

Eg: For python 2:

Name → 'NCSU Lot'
zone_id → 'A'
num_spaces_range → '0:50'. These ranges are the number of spaces per zone.

Handicapped → '31:40' . These ranges must be specified in order of their zone id. If there is no handicapped spaces for this zone mention '0:0' (if no Handicapped spaces)

Electric → '41:50' These ranges must be specified in order of their zone id. If there is no handicapped spaces for this zone mention '0:0' (if no Electric spaces)

For python 3:

Name → NCSU Lot

zone_id → A

num_spaces_range → 0:50. These ranges are the number of spaces per zone.

Handicapped → 31:40. If there is no handicapped spaces for this zone mention '0:0' (if no Handicapped spaces)

Electric → 41:50 If there is no handicapped spaces for this zone mention '0:0' (if no Electric spaces)

CheckVValidParking(db, lot_name, space_id, zone_id)

This function checks if the parking is valid or not based upon the three arguments: lot_name, space_id, zone_id. It will verify if that particular entry is present in the ParkingSpaces table or not. Here, our assumption is that we are trying to simulate a real life situation. Say, the UPS admin sees a car parked in some lot, zone, space. According he/she checks the permit and verifies using this function whether it is correctly parked.

Eg: For python 2:

lot_name → 'NCSU Lot'

Space_id -> 10

zone_id → 'A'

For python 3:

lot_name → NCSU Lot

Space_id -> 10

zone_id → A

CheckNVValidParking(db,permit_id)

This function is used whether the student has parked in the employee parking space during their work hours, if so, then it will show invalid parking else valid. Here, again we are trying to simulate a real life situation. We check the zone of parked car and verify if the present (real-life) zone on the student's permit is 'S' or not before 5pm. Assuming, the database entry of the permits table will have other zone_id than S for the student's permit before 5pm, we will treat it as Invalid.

Eg: For python 2:

permit_id → '10020'

For python 3:
permit_id → 10020

File Contents and Descriptions:

The following is the list of existing files required by the program and their content.

main_ui.py

This is the front-end file from which the user interacts with the program. It provides connection to the database, and also prompts options in front of the user, from which user decides which action he/she needs to perform. For Sample queries and Reporting queries, there SQL queries have been called out in this file and it provides results based on the data stored in the database.

admin_actions.py

In this file all the functions and their logics have been defined. This is where the actual brain of the code resides. Once the user provides any input after being prompted by main_ui.py, it will refer to this file and the corresponding function will perform the desired action. Also, for each function there are few input parameters required, which will also be prompted on the UI screen and the user has to provide those details for further processing. It performs all the actions on the data stored in the database, so it is the file that takes requests from the user and queries the database, performs the action on top of it and generates results for the user. All the functions described in Functions tables, are defined in this file

demo_data.py

It is an independent python program created to load the Demo Data given by the TAs. It performs following sequence of events:
> Drops all of the existing tables to clean up DBMS of any existing data
> Creates all tables used in this project in the required order.
>Loads the Demo Data into the database.

SQL Query files:

Sample_queries.sql and ***reporting_queries.sql*** are the files where we have defined the queries for the below the requirements

>sample_queries.sql

1. *Show the list of zones for each lot as tuple pairs (lot, zone).*
2. *Get permit information for a given employee with UnivID: 1006020*
3. *Get vehicle information for a particular UnivID: 1006003*

4. Find an available space# for Visitor for an electric vehicle in a specific parking lot: Justice Lot

5. Find any cars that are currently in violation

6. How many employees have permits for parking zone D

> reporting_queries.sql

1. For each lot, generate the total number of citations given in all zones in the lot for a three month period (07/01/2020 - 09/30/2020)

2. For Justice Lot , generate the number of visitor permits in a date range: 08/12/2020 -08/20/2020 , grouped by permit type e.g. regular, electric, handicapped.

3. For each visitor's parking zone, show the total amount of revenue generated (including pending citation fines) for each day in August 2020

Run the UPS Application:

Install the below required software:

- **MySQL Community Server version 8.0.21.**
- **Python version 2.7 (Prefered) or version 3.0. (Steps for running the functions in each case have been specified above)**
- **MySQL Workbench version 8.0.21 (Optional)**
- **For connecting python with MySql we need to install mysql python connector which can be installed using the below :**

  pip install mysql-connector-python

Key Steps while Installation:

You need to specify a **root password** using strong password encryption while installing MySql Community Server. This root password must be used to connect to the database created using SQL Scripts and the Python scripts would use the same credentials to connect to the database. The username in each case would be **root.**

Load the Project Demo Data:

There are two ways to load demo data into the database,

1. SQL:
   a) Create tables into the database using commands in design_tables.sql file.
   b) Fill in the data into the tables using commands in fill_tables.sql

2. Python: In order to load the Demo Data, execute the below command:
   python demo_data.py

In order to run using Python you need to edit the lines 862-866 as shown below:

```
if __name__ == '__main__':
db = mysql.connector.connect(host='localhost',
                             database='TEST_PROJECT',
                             user='root',
                             password='Rajson251710@')
```

The database name can be kept accordingly keeping in mind the same database has been created in SQL. The username and password must be kept the same that is kept during installation of MySQL Community Server.

Starting the program:

Execute *"python main_ui.py"* and then give the inputs as per the requirements(Detail on each option is provided in the **Menu** Section above)