DATABASE APPLICATION PROJECT ON PARKING LOT SYSTEM

E-R DIAGRAM OF OUR IMPLEMENTATION OF THE PROJECT:

The E-R diagram has been represented in two pictures as shown below.

We need 2 pictures because even though the parking lots and spaces are the most important aspects, they are actually isolated from the rest of the core connectivity in the ER diagram.

Spaces is a weak entity. So, sid is uniquely identified within a unique parking lot. Spaces is linked to Parking lots using a relation 'has'.

Now, Permits can be of either visitor or non-visitor category. Hence, we have modelled the permits using a IS-A Hierarchy. The common attributes between visitor and non-visitor permits are the ones which are inherited (the attributes directly connected to the Permits entity).

The Vehicle information is connected to the Permits entity using a relation 'PV'.

Now, the citations are connected to the permits using a 'PC' relation.
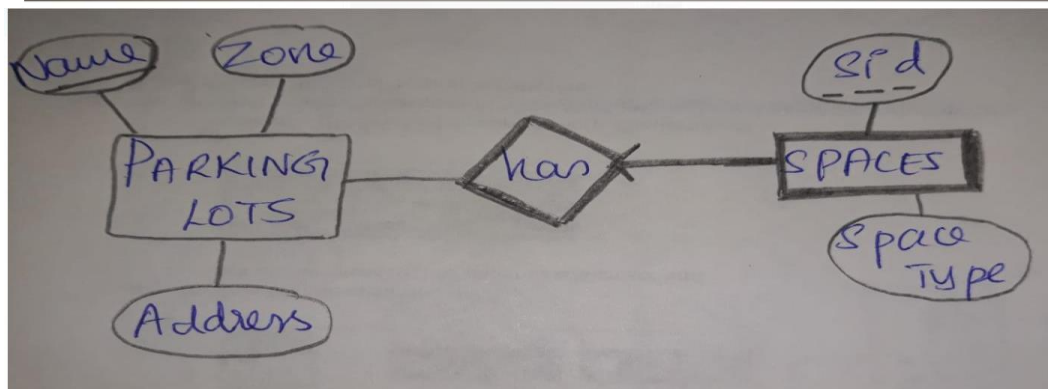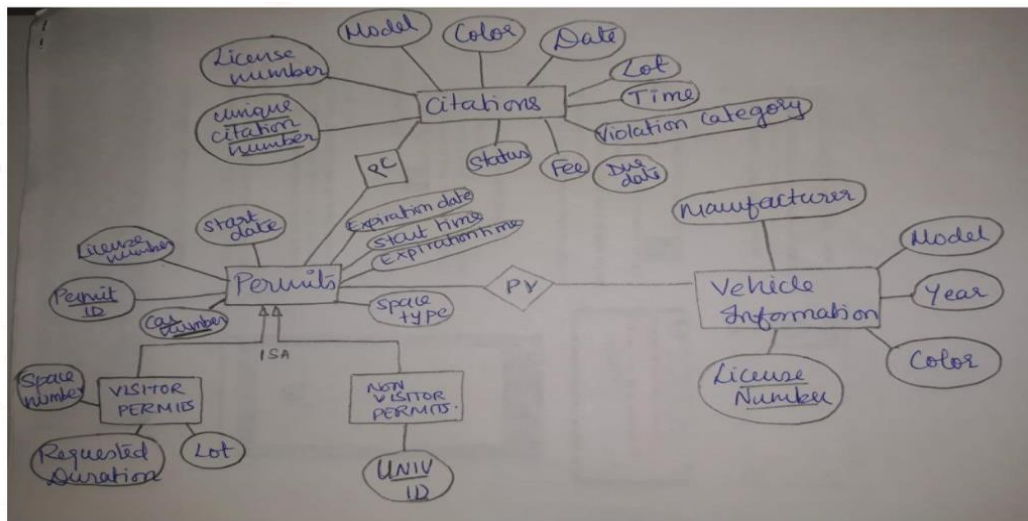
CONSTRAINTS USED IN THE APPLICATION CODE VS SQL DATABASE CODE:

Now, ideally there would be cases where some constraints observed couldn't be put into the SQL code, but can be implemented in the application code.

But in our design, we have primarily focussed on application programming for the database, and hence have implemented all the possible constraints considered in the Application code as well as the SQL code.

The following are the constraints coded into our database table wise :

1. TABLE p_lots:

Key constraint: name- is the primary key.

2. TABLE Spaces :

Key constraint: (name,sid) are the primary key.

(Name) is the foreign key to reference the parking lots (p_lots)- cause there needs to be a link between the spaces and the parking lots.

3. TABLE NON_VISITOR_PERMITS:

Key Constraint: Primary key – (Unique permit Id, car_count).

The number of cars allowed for the permits and the space type requested for the permit are checked using the CHECK constraint.

4. TABLE VISITOR_PERMITS:

Key Constraints: Primary key –(Unique permit id).

The lot attribute is used as a foreign key to reference the lot in the p_lots table.

The visitor space ('R', 'V' or 'H' or 'E') is checked using the CHECK statement.

5. TABLE VEHICLE_INFO:

Key constraints: Primary key is License number

6. TABLE CITATIONS:

Key constraints: (UNIQUE_CITATION_NUMBER) is the primary key.

The violation categories and the check_fee use the CHECK constraint.

FUNCTIONAL DEPENDENCIES IN THE DATABASE:

So, there are quite a few functional dependencies which dictate how much influence one or more rows have on a set of rows.

The following are some of the dependencies observed in our database, table wise:

1. TABLE P_lots :

Name-> Address　　　is an F.D cause the same address is there for one particular lot name.

Name-> (Address, zone)　(name

is the primary key here)

2. TABLE SPACES:

(Name,sid)-> (Space_type)

Name,sid is the primary key

3. TABLE NON_VISITOR_PERMITS:

(Start date, UNIV_ID) -> Expiration_date

This is because if we know if a person is a student/employee , then just knowing the start date we can determine the end date properly.

4. TABLE VISITOR_PERMITS:

(Start_time, Duration)-> Expiration time

Start_date-> Expiration_date

5. TABLE VEHICLE_INFO:

License plate number -> Color, Year, Model, Manufacturer

(its obvious since license plate no. is the key here)

6. TABLE CITATIONS: **Violation Category-> Fee** and

UNIQUE_CITATION_NUMBER -> (LICENSE_NO., MODEL, COLOR, ISSUE_DATE, LOT, TIME, VIOLATION_CATEGORY, FEE, DUE_DATE, STATUS)

Here UNIQUE_CITATION_NUMBER is the primary key.

FUNCTIONS USED IN THE PROJECT, THEIR DESCRIPTION AND DESIGN REASONING

1. **AddLot :**

Inputs: Name of the lot, Parking Lot Address, number of spaces in the lot, starting regular space ('R') number, space, zone.

Functionality: We need the AddLot table to add a new parking lot of regular type. The administrator uses this function to add a new tuple to the parking lots (p_lots here) table.

Design reasoning: Our design of the AddLot has 2 uses- to insert the name,address and zone

into the p_lots table; and to insert into the spaces table the newly assigned spaces- from start_n till all

the n_spaces have been allocated.

2. **AssignZoneToLot :**

Inputs: name of Lot where we have to append the zones, Zones corrresponding to the Lot

Functionality: The AssignZoneToLot function is used to assign zones to the newly assigned lots, after the AddLot has been called by the admin.

Design Reasoning: We use a dictionary to keep track of all the kinds of parking zones {'A':1,'B':1,'C':1,'D':1,'AS':2,'BS':2,'CS':2,'DS':2,'V':3 } where all the employee zones have a value of 1, student zones have a value of 2, and the visitor zones have a value of 3.

Now, when we get the input zones, we separate them with the separator '/'. We then check of all the zones mentoioned are valid ones by seeing if they are present in the dictionary we defined before, and if yes, we Update the zones in the p_lots.

3. **AssignTypeToSpace :**

Inputs: Name of parking lot, starting regular space ('R') number ('start_n'), Ending Regular Space Number (end_n), Parking lot type (p_type)

Functionality: AssignTypeToSpace is used assign a specific type of parking type to an individual parking space.

Design: We use a dictionary to assign a number value to each of the Regular, Visitor, ElectricVehicle and Handicapped categories.

We check if the entered type exists in the dictionary above, if yes, we update the spaces and it's name.

4. **AddVehicle:**

Inputs: License number of the car, car manufacturer, car model, year manufactured, car color.

Functionality: AddVehicle is used to just add the new vehicle information into the database, and store it for the database's records. It is usually called after the IssuePermit (to get the non-visitor permit) and the GetVisitorPermit t add the vehicle to the parking lot's database.

Design: We just insert the commands we get from the user directly into the database and make sure they are valid entries. Nothing too intricate in this design.

5. **IssuePermit :**

Inputs: unique permit id, zone identifier, license plate number of the vehicle, permit start date, permit space type, university id, name of car manufacturer, car model, year manufactured, car color

Functionality: IssuePermit is used to issue permits to a new vehicle only for students and Employees- after collecting some information about the vehicle and the person obtaining the permit.

Design: We use a dictionary- which assigns a number to each of the parking zone. {'A':1,'B':1,'C':1,'D':1,'AS':2,'BS':2,'CS':2,'DS':2,'V':3 }

If the zone requested for the permit is present in the dictionary, we go ahead and mark the start date for the permit first. Now, if a student requests a permit, the valid duratoion of the permit is by default =4 months, and the expiration date on the permit is set accordingly.

Similarly, for an employee, the expiration duration is 1 year after the start date. Since, after issuing a permit we need to add a Vehicle to the Vehicles present in the parking lot, the AddVehicle() function is called after the permit has been issued.

6. **GetVisitorPermit :**

Inputs: Unique Permit id, license number of vehicle, start date of the permit, start time for permit, duration needed for the permit, Lot requested, Space type requested, space number needed, car manufacturer, car model, year manufactured, car color.

Functionality: GetVisitorPermit is a function used to issue permits solely for the visitors-after collecting information about the car.

Design: This is similar to how the Non-Visitor permit works, but here the time in terms of hours for ther permit need to be taken as input from the user and then processsed, unlike how the duration is a default of 4 months or 1 year for the non visitors.

After the required input has been processed, we can then approve the permit following which the AddVehicle command can be called to add the vehicle to the Vehicle list.

7. **IssueCitation :**

Inputs: License number of car, car model, car color, citation issuing date, citation issuing time, Parking lot, Parking violation category, outstanding fee incurred, due date to pay the citation fee

Functionality : Issuecitation just takes the inputs observed and fed in by the administartors, and generates the citations.

Design: We just insert the required values into the citations table- adding a new tuple each time a citation is generated.

ASSUMPTION : A SPACE IN A PARKING LOT IS ASSUMED TO BE FREE IF IT'S NOT PRESENT IN THE PERMITS TABLE.

8. **ExitLot :**

Inputs: Unique Permit ID

Functionality: ExitLot Removes a visitor permit and de-allocates the space in the corresponding lot in the parking lot making it available.

Also, it checks for the EXPIRED PERMIT case.

Design: The expiration date and the time corresponding to the permit ID is obtained. This date and time are compared with the date and time when the user actually exits the lot. So, basically if the user exits the lot after the expiration date or time, a citation can be issued [ONLY FOR VISITORS] owing to the time overage. Else, we can just delete the record of the vehicle in the permits table - freeing up space.

9. **ChangeVehicleList**:

Inputs: Unique Permit ID, Car License number, Univ ID of student/employee , Functionality sought- (Add or DELETE)

Functionality: ChangeVehicleList is used to basically help the Non-visitors using the parking lot to update or add or remove their vehicles in the parking lot.

Design : The design is three fold :

i)   ADD : So, when a user wants to add a vehicle, a student can add a maximum of 1 car, an employee can add a maximum of 2 cars. The current count of the cars in the parking lot correspondig UNIV ID

is checked ,and accordingly the vehicles can be added. If the limit of 1 or 2 cars is exceeded, then an error message is output saying maximum limit reached.

ii)   DELETE : Now, the delete functionality is used to remove a car from the parking list and also to check if there is any violation or citation incurred for the Non_visitor. If any citation has been observed, it will be raised and the citation table is populated before the car is taken out of the permits table(ie., before freeing up space in the parking lot.)

iii)   REPLACE : Basically if any person wishes to replace the vehicle they have in the permits table, then the DELETE operation followed by the ADD operation can be done.

10. **PayCitation:**

INPUT: LICENSE number of the car as input is prompted on running this function.

Functionality: Used to update the status of the citation from 'unpaid' to 'paid'.

Design: Basically, the visitor or the non-visitor calls this function to pay any of his outstanding citation fees. On calling this function and paying the fee, the status of the citation payment status is uppdated from 'UNPAID' to 'PAID'.

11. **CheckNVValidParking:**

Inputs: Current date and time at the time of calling the function, Permit number or ID to be checked.

Functionality: To check if a NonVisitor permit is still valid at that moment in time or not. Also, it gives one of 3 outputs regarding the status of the permit:

NO-PERMIT, EXPIRED, VALID permit.

Design: We basically input the various parameters from the admin on demand, and then check to see if the permit ID is in the permits table. If not it's a NO PERMIT condition.

Similarly, if the current instant of time if greater than the expiration time and date of the corresponding permit number, then it is an EXPIRED PERMIT.

Else, it's a VALID permit.

12. **CheckVValidParking:**

Inputs: Current time and date, Space number requested, Lot number, Car License number

Functionality : To check if a Visitor permit is still valid at that moment in time or not. Also, it gives one of 4 outputs regarding the status of the permit:

NO-PERMIT, EXPIRED, VALID and INVALID permit.

Design: We basically input the various parameters from the admin on demand, and then check to see if the permit ID is in the permits table. If not it's a NO PERMIT condition.

Similarly, if the current instant of time if greater than the expiration time and date of the corresponding permit number, then it is an EXPIRED PERMIT.

If license number, space number and LOT aren't matching , i.e., if the car is parked in some other place, we assign the INVALID PERMIT.
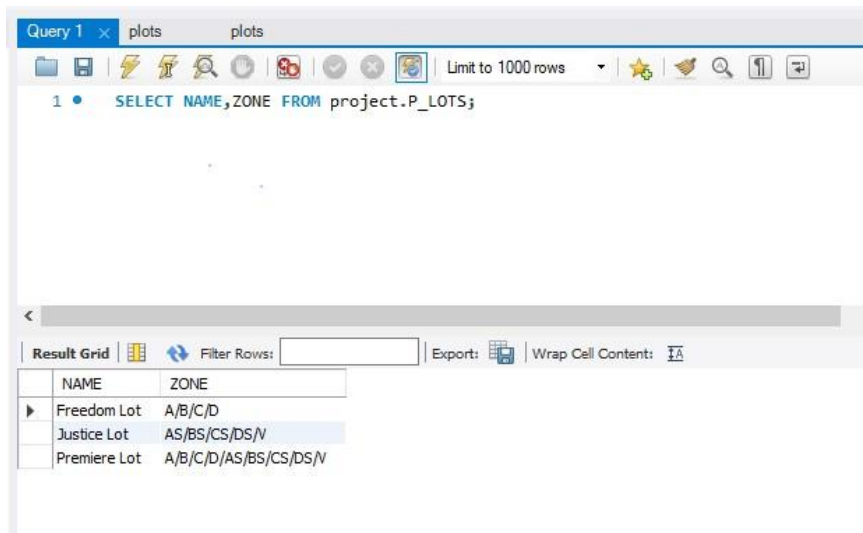
Else, it's a VALID permit.

SAMPLE QUERIES AND THEIR OUTPUTS AS SHOWN IN SQL WORKBENCH:

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

Query :

SELECT NAME,ZONE FROM project.P_LOTS; Output:
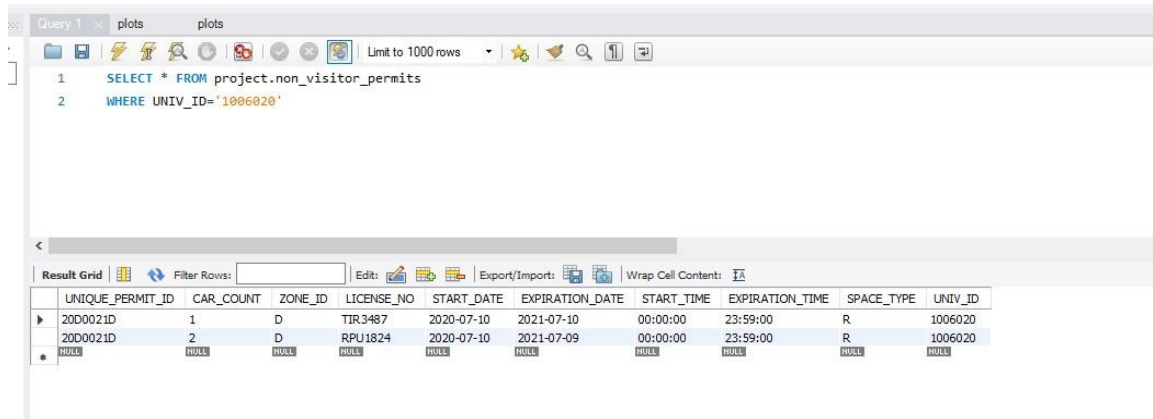
2. Get permit information for a given employee with UnivID: 1006020

Query :

SELECT * FROM project.non_visitor_permits

WHERE UNIV_ID='1006020' **Output:**

3. Get vehicle information for a particular UnivID: 1006003
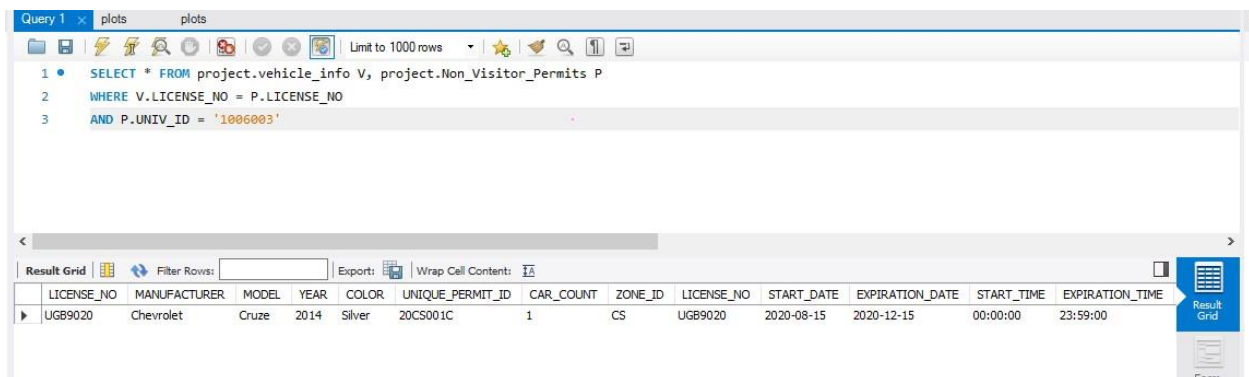
Query:

SELECT * FROM project.vehicle_info V, project.Non_Visitor_Permits P

WHERE V.LICENSE_NO = P.LICENSE_NO

AND P.UNIV_ID = '1006003'
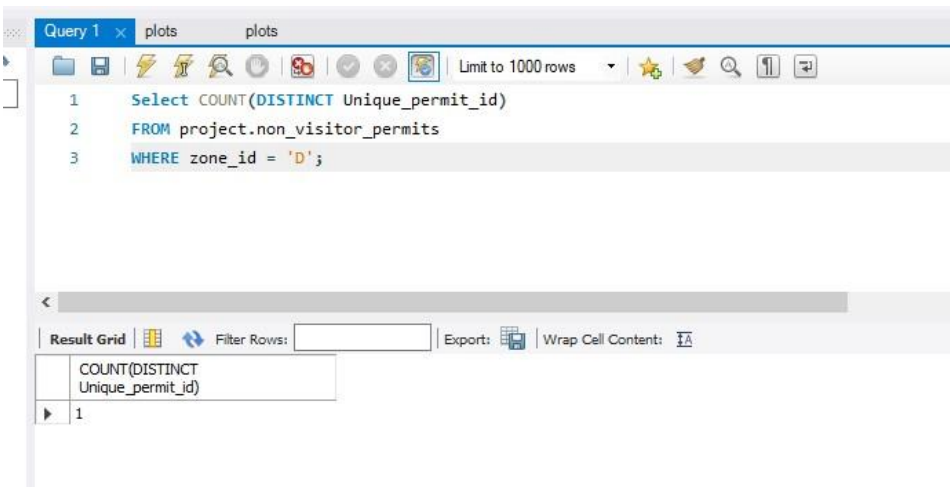
Output :

4. How many employees have permits for parking zone D .

Query :

Select COUNT(DISTINCT Unique_permit_id)

FROM non_visitor_permits

WHERE zone_id = 'D';

Output: