Team NULL

Team Members -

- Arunesh Singh (2018279)
- Paarmita Jhalani (2018353)
- Paritosh Shukla (2018063)
- Shivang Saigal (2018310)
- Shubham Mittal (2018101)

Application/Domain: Travel, Tourism, Safety

Designing a comprehensive database which allows travellers to visit multiple places as per their interest, by suggesting a custom made itinerary just for them. It would have data about all the places of interest, including the best time to view them and its reviews among other things. Thereby, allowing a traveller to achieve the most out of the limited budget and time they have. It also allows them to find out where their friends are. Moreover, it also allows users to add their own places of interest to the pre existing categories.

The travellers are assisted by a team of drivers who also access the database to find the best route to take, the nearest gas station etc.

The police also use this database extensively to determine where patrolling is needed by checking the database to find out where the other police officers are, and which areas are congested or deserted at that moment so that reinforcements can be sent to that road.

This database is also pivotal for the government in taking out new policy decisions. The government uses the data of road congestion over time and changes in the population over time to decide where roads need to be improved. It can also utilize the database to find out how much revenue it's earning from the entry tickets, challans, taxes etc.

Innovation/Bonus Feature: This project goes one step further with the following bonus features:

- A beautiful and usable cross platform app for the stakeholders. It makes it very easy to use the database as the user doesn't have to write queries to get their answer. They just enter their choices and we do all the heavy work for them.
- Ensuring the safety of its users. It not only provides the fastest route but also allows people to get there more safely.
- It also has a unique feature of displaying trivia of various landmarks along the route.
- Allows drivers to find other passengers to share the ride with, thus saving time, money and being ecological.

Stakeholders:

- Travellers
- Drivers
- Police
- Policy Makers

Stakeholder Roles and Questions:

Travellers (Use our application to better plan their travel and add newly found places) -

- I have limited time/money/interest. Where can I go in the city?
- Is it safe to travel to XYZ place right now?
- How is XYZ Place?
- What is the best time to go to XYZ place?
- Where are my friends right now?
- Can I get a ride to XYZ place?

Drivers (Plan the navigation) -

- What is the route from A to B?
- How safe is the route from A to B?
- What is the route for the gas station?
- Is there another passenger also going somewhere between A and B?
- What is the route for parking location?

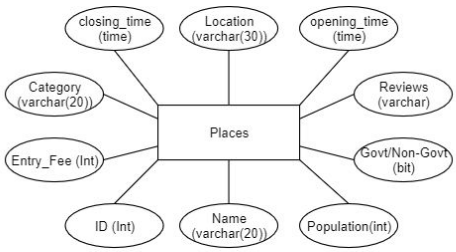
Police (Increase efficiency of policing and improving vigilance) -

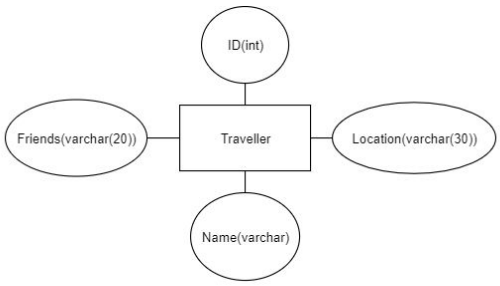
- Where have most of the accidents occurred?
- Which driver has committed most traffic violations?
- Which roads are the most deserted at XYZ time?
- Which is the most congested road/place at XYZ time?
- Where are the other police officers at this moment?

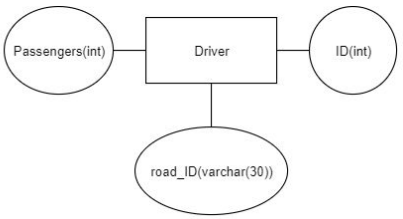
Policy Makers (To better plan and develop policies for the region)

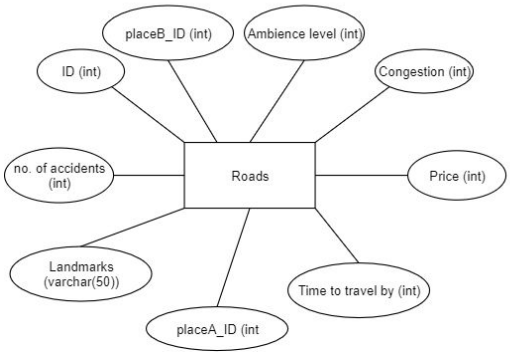
- What is the most common reason for challans?
- How is the traffic congestion on XYZ road over time?
- Which road needs better surveillance?
- What is the best road to improve street lighting?
- What is the best road to improve road quality?
- Which places are visited most frequently?
- How much revenue has been generated for the government?

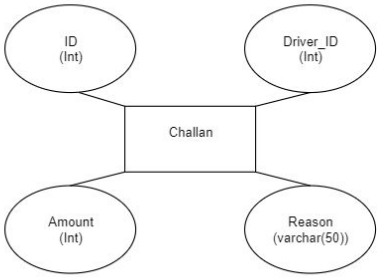
DB Entities -

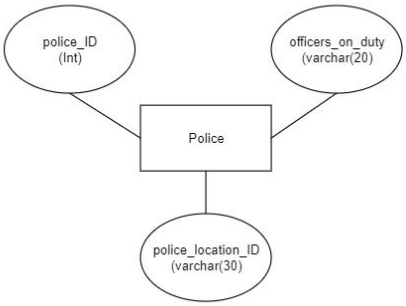












Relationships-

- Travellers visit Places
- Places are connected through Roads
- Police gives Challan
- Challan is given to Drivers
- Drivers drive Travellers
- Drivers drive on Roads
- Police patrols on Roads

Queries-

- **What is the population at Tirupati?**
  Select p.population from place where place_Name = ‘Tirupati’

  $\pi_{population}(\sigma_{place\_Name='Tirupati'}(place))$

- **Can I get a ride to XYZ place?**
  Input-place_id(int p)
  SQL-Select d.driver_ID from drivers d where p=d.road_ID
  Relational- $\pi_{driverid}(\sigma_{placeid=p}(drivers))$

- **Which places are visited most frequently?**
  Select p.place_Name from place p where p.population=(Select MAX(p.population) from place p

- **How is XYZ Place?**
  Input-place_id(int p)
  SQL-Select category,reason from place p,review r where p=p.place_id and p=r.place_ID
  Relational- $\pi_{category,reason}(\sigma_{placeid=p}(place, review))$

- **Find all the reasons for challan?**
  Select r.reason from road r

  $\sigma_{reason}(road)$

- **What is the best road to improve street lighting?**
  Select r.road_ID from road r where r.ambience_level=(Select MIN(r.ambience_level) from road r)

- **Where are my fellow officers?**
  Input - officer_id
  SQL - Select F.fellow_officer_ID, T.place_Name, T.location from fellow_police as F, police as P, place as T where " + str(officer_id) + "= F.officer_ID and T.place_ID= P.location_ID and F.fellow_officer_ID=P.officer_ID
  Relational -
  $\pi_{fellowofficerID,placeName,location}(\sigma_{officerid=F.officerID\ and\ T.placeID=P.locationID\ and\ F.fellowofficerID=officerID}(fellowpolice, police,$
- **Where are my friends (travellers?**
  Input - person_id

SQL - Select T.name,T.location from friend as F,traveller as T where " +str(person_id)+ "= F.person_ID and T.traveller_ID= F.friend_ID
Relational - $\pi_{name,location}(\sigma_{personid = F.personID\ and\ F.friendID=T.travellerID}(friend, traveller))$

- **Which road needs better surveillance?**
  Select r.road_ID from r.road where r.no_of_accidents = (Select MAX(r.no_of_accidents) from r.road)
- **A new challan has been done by a policeman (Insert query).**
  Input - driver_id,amount,reason
  SQL - INSERT INTO challan (challan_ID,driver_ID,amount,reason) VALUES ('"+str(id)+"','"+str(driver_id)+"','"+str(amount)+"','"+reason+"')

- **How much revenue has been generated for the government?**
  Select SUM(p.entry_fee*p.population) from place p

Embedded Queries

- **Maximum accidents have occurred at which road?**

```
def query(HOST,USER,PASSWORD,DATABASE):
  db =
pymysql.connect(host=HOST,user=USER,passwd=PASSWORD,database=DATABASE
)
  project=db.cursor()
  project.execute ("Select road_ID from road where no_of_accidents = (Select
MAX(no_of_accidents) from road)")
  k=0
  k=int(k)
  for i in project:
    k+=i[0];
  print(k)
  return 0
```

- **Most congested place at XYZ time?**

```
def query(HOST,USER,PASSWORD,DATABASE,time):
        db =
pymysql.connect(host=HOST,user=USER,passwd=PASSWORD,database=DATABASE
)
        project=db.cursor()
        if(time<7):
                project.execute ("Select place_ID from population where people1 =
(Select MAX(people1) from population)")
        elif(time<13):
```

```
            project.execute ("Select place_ID from population where people2 =
(Select MAX(people2) from population)")
        elif(time<19):
            project.execute ("Select place_ID from population where people3 =
(Select MAX(people3) from population)")
        elif(time<25):
            project.execute ("Select place_ID from population where people4 =
(Select MAX(people4) from population)")
        else:
            print("Invalid time")
        cnt=0
        cnt=int(cnt)
        print("Place_id of the most congested place during this time is :",end="")
        for i in project:
            cnt+=1
            if(cnt>1):
                print(" and ",end="")
            print(i[0], end="")
        return 0
```

- **Least congested place at XYZ time?**

```
        db =
pymysql.connect(host=HOST,user=USER,passwd=PASSWORD,database=DATABASE
)
        project=db.cursor()
        if(time<7):
            project.execute ("Select place_ID from population where people1 =
(Select MIN(people1) from population)")
        elif(time<13):
            project.execute ("Select place_ID from population where people2 =
(Select MIN(people2) from population)")
        elif(time<19):
            project.execute ("Select place_ID from population where people3 =
(Select MIN(people3) from population)")
        elif(time<25):
            project.execute ("Select place_ID from population where people4 =
(Select MIN(people4) from population)")
        else:
            print("Invalid time")
        cnt=0
        cnt=int(cnt)
        print("Place_id of the least congested place during this time is :",end="")
        for i in project:
```

```
            cnt+=1
            if(cnt>1):
                print(" and ",end="")
            print(i[0], end="")
        return 0
```

- **Which driver has most challans?**

```
        db =
pymysql.connect(host=HOST,user=USER,passwd=PASSWORD,database=DATABASE
)
        project=db.cursor()
        #project.execute("GRANT ALL")
        project.execute ("select driver_ID, count(*) as challans from challan group by
driver_ID order by count(*) desc")
        cnt=0
        cnt=int(cnt)
        print("The driver(s) with most challans is driver number: ")
        for i in project:
            cnt+=1
            if(cnt==1):
                print(i[0])
        return 0
```

- **What is the most common reason for challans?**

```
def query(HOST,USER,PASSWORD,DATABASE):
  db =
pymysql.connect(host=HOST,user=USER,passwd=PASSWORD,database=DATABASE
)
  project=db.cursor()
  project.execute ("Select reason from challan Group By reason having
count(reason)=(select count(reason) as great from challan Group By reason Order By
Great desc limit 1)")
  k="
  for i in project:
    k+=i[0];
  print(k)
  return 0
```

- **What is the best road to improve road quality?**

```
def query(HOST,USER,PASSWORD,DATABASE):
```

```
  db =
pymysql.connect(host=HOST,user=USER,passwd=PASSWORD,database=DATABASE
)
  project=db.cursor()
  project.execute ("Select road_id from road where congestion*ambience_level=(Select
MAX(congestion*ambience_level) from road)")
  k=0
  k=int(k)
  for i in project:
    k+=i[0];
  print(k)
  return 0
```

- **What is the route from A to B?**

```
project.execute ("Select placeA_ID,placeB_ID from road")
  number_of_places = 150
  adjacency_list = [[] for i in range(number_of_places + 1)]

  for i in project:
        adjacency_list[i[0]].append(i[1])
        adjacency_list[i[1]].append(i[0])

  path = bfs(adjacency_list,source,destination,number_of_places)
  return insert_landmarks(path,project,number_of_places)

def insert_landmarks(path,project,number_of_places):
  project.execute("Select R.placeA_ID,R.placeB_ID,L.name from landmarks L,road R
where L.road_ID=R.road_ID")
  dictionary ={}
  for i in range(1,number_of_places+1):
        dictionary[i] = {}

  for i in project:
        (dictionary[i[0]])[i[1]] = list(i[2].split(';'))
        (dictionary[i[1]])[i[0]] = (dictionary[i[0]])[i[1]][::-1]
  ans = []
  ambience ={}
  for i in range(1,number_of_places+1):
        ambience[i]={}
  project.execute("Select placeA_ID,placeB_ID,ambience_level from road")
  for i in project:
        ambience[i[0]][i[1]] = i[2]
        ambience[i[1]][i[0]] = i[2]
```

```
    ambience_level = 0
    for i in range(0,len(path)-1):
            ambience_level += ambience[path[i]][path[i+1]]
            ans.append("You are at "+str(path[i]))
            ans.extend(dictionary[path[i]][path[i+1]])
    if(len(path)!=1):
            ambience_level = ambience_level/(len(path)-1)
    ans.append("You reached the destination "+str(path[-1]))
    return [ambience_level, ans]

def bfs(adjacency_list,source,destination,number_of_places):

    q = queue.Queue()
    par = [0] * (number_of_places + 1)
    root = par[source] = source
    q.put(root)

    while(not q.empty()):
            root = q.get()
            for i in adjacency_list[root]:
            if( par[i] == 0):
            par[i] = root
            q.put(i)

    path = [destination]
    while (destination != source):
            destination = par[destination]
            path.append(destination)
    return path[::-1]
```

- **What is the route for the gas station from the given station?**

```
project.execute ("Select P.place_ID from place P where P.place_Name='Gas Station'")
    available_gas_stations = []
    for i in project:
            available_gas_stations.append(i[0])

    project.execute ("Select placeA_ID,placeB_ID from road")
    number_of_places = 150
    adjacency_list = [[] for i in range(number_of_places + 1)]

    for i in project:
            adjacency_list[i[0]].append(i[1])
            adjacency_list[i[1]].append(i[0])
```

```
    place_name = {}
    project.execute("Select place_Id,place_name from place")
    for i in project:
            place_name[i[0]]=i[1]
    path = bfs(adjacency_list,source,available_gas_stations ,number_of_places)
    return [place_name[i] for i in path]

  def bfs(adjacency_list,source,available_destinations,number_of_places):

    q = queue.Queue()
    par = [0] * (number_of_places + 1)
    root = par[source] = source
    q.put(root)
    destination = -1
    while(not q.empty()):
            root = q.get()
            if(root in available_destinations):
            destination = root
            break
            for i in adjacency_list[root]:
            if( par[i] == 0):
            par[i] = root
            q.put(i)

    path = [destination]
    while (destination != source):
            destination = par[destination]
            path.append(destination)
    return path[::-1]
```

- **What is the route from the given station to the nearest Parking?**

```
  project.execute ("Select P.place_ID from place P where P.place_Name='Parking'")
    available_parking_stations = []
    for i in project:
            available_parking_stations.append(i[0])

    project.execute ("Select placeA_ID,placeB_ID from road")
    number_of_places = 150
    adjacency_list = [[] for i in range(number_of_places + 1)]

    for i in project:
            adjacency_list[i[0]].append(i[1])
```

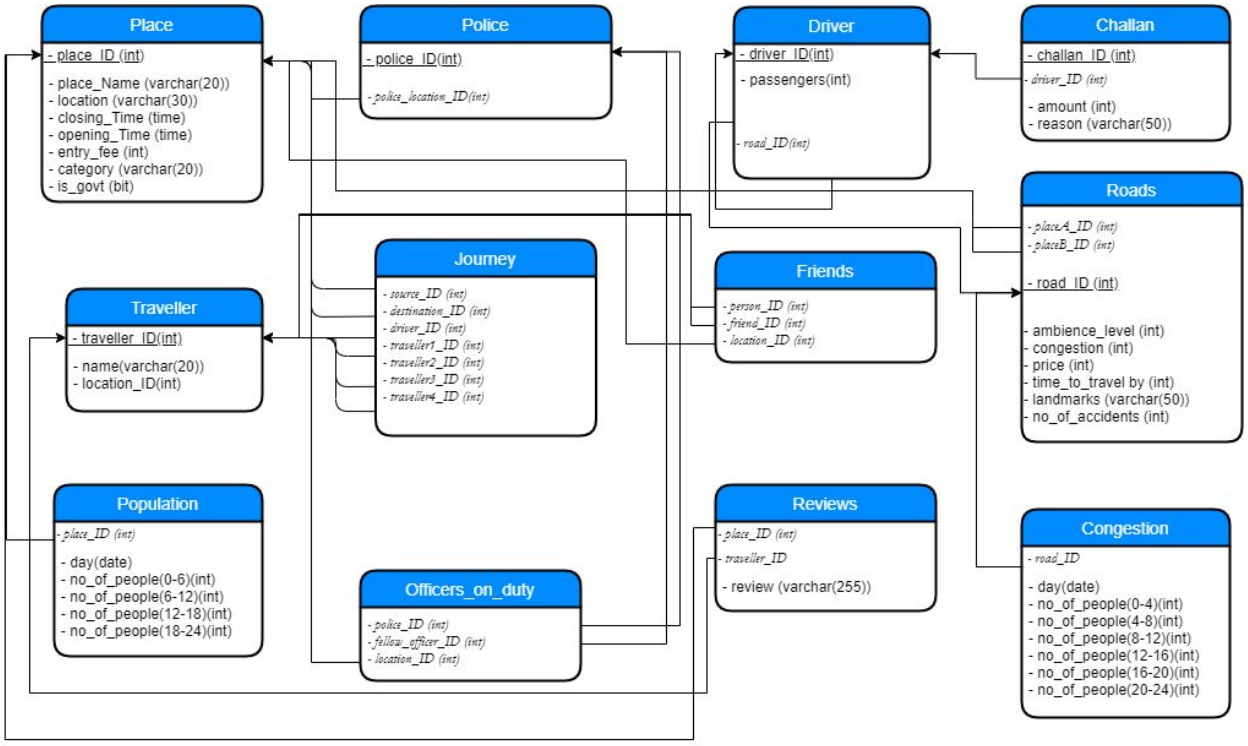
```
        adjacency_list[i[1]].append(i[0])

  place_name = {}
  project.execute("Select place_Id,place_name from place")
  for i in project:
        place_name[i[0]]=i[1]
  path = bfs(adjacency_list,source,available_parking_stations ,number_of_places)
  return [place_name[i] for i in path]

def bfs(adjacency_list,source,available_destinations,number_of_places):

  q = queue.Queue()
  par = [0] * (number_of_places + 1)
  root = par[source] = source
  q.put(root)
  destination = -1
  while(not q.empty()):
        root = q.get()
        if(root in available_destinations):
        destination = root
        break
        for i in adjacency_list[root]:
        if( par[i] == 0):
        par[i] = root
        q.put(i)

  path = [destination]
  while (destination != source):
        destination = par[destination]
        path.append(destination)
  return path[::-1]
```

- **I have limited time/money/interest. What can I go in the city?**

```
project.execute("Select P.place_ID from place P where P.category='"+interest+"'")
  available_places = []
  for i in project:
        available_places.append(i[0])

  project.execute ("Select placeA_ID,placeB_ID,time_to_travel_by,price from road")
  number_of_places = 150
  adjacency_list = [{} for i in range(number_of_places + 1)]

#time,money for road
```

```
  for i in project:
        adjacency_list[i[0]][i[1]]=[i[2],i[3]]
        adjacency_list[i[1]][i[0]]=[i[2],i[3]]

  return
bfs(adjacency_list,position,available_places,number_of_places,time,money,project)

def bfs(adjacency_list,source,available_places,number_of_places,time,money,project):
  #time taken,money req, number of category of interset
  q = queue.Queue()
  par = [[] for i in range(number_of_places+1)]
  parents =[0]*(number_of_places +  1)
  root = parents[root] = source
  par[root] = [0,0,0]
  q.put(root)

  while(not q.empty()):
        root = q.get()
        for i in adjacency_list[root]:
        if( par[i] == []):
        par[i] = [par[root][0] + 2 * adjacency_list[root][i][0], par[root][1] + 2 *
adjacency_list[root][i][1], par[root][2]]
        if i in available_places:
        par[i][2] += 1
        parents[i] = root
        q.put(i)

  possible_to_go = []

  for i in available_places:
        if(par[i][0] <= time and par[i][1] <= money):
        par[i].append(i)
        possible_to_go.append(par[i])

  possible_to_go = sorted(possible_to_go, key = lambda x:(x[2],-x[0],-x[1]),reverse=True)

  if possible_to_go==[]:
        return "Not Possible To Go"

  else:
        project.execute("Select place_Name from place where
place_ID='"+str(possible_to_go[0][3])+"'")
        for i in project:
```

```
destination = i[0]
return "Visit "+i[0]+" with place_ID "+str(possible_to_go[0][3])
```



Database Schema -

Integrity constraints and Ranges-

- Place -
    - place_ID (Primary Key)
    - place_Name (not Null)
    - Location (not Null)
    - opening_Time
    - closing_Time
    - Entry_fee (not Null, default - 0)
    - Category (in ‘Religious, Sports, Lifestyle, Parking, Gas Station, Entertainment, Comedy, Dine Out, Adventure, Sightseeing, Bar’)
- Police -
    - police_ID (Primary key)
    - police_location_ID (not Null, foreign key referenced to Place)

- Driver -
    - driver_ID (Primary Key)
    - Passengers
    - road_ID(Foreign Key referenced to Roads)
- Challan -
    - challan_id (Primary Key)
    - driver_id (Foriegn Key referenced to Driver)
    - amount (not Null, default-0)
    - reason (not Null, in 'Overspeeding, No Helmet, No Seatbelt, Drunk Driving, Traffic Light Jump, No License, Pollution')
- Traveller -
    - traveller_ID(Primary Key)
    - name(not Null)
    - Location (not Null)
- Journey -
    - source_ID (Foriegn Key referenced to Place)
    - destination_ID (Foriegn Key referenced to Place)
    - driver_ID (Foriegn Key referenced to Driver)
    - traveller1_ID (Foriegn Key referenced to Traveller)
    - traveller2_ID (Foriegn Key referenced to Traveller)
    - traveller3_ID (Foriegn Key referenced to Traveller)
    - traveller4_ID (Foriegn Key referenced to Traveller)
- Population -
    - place_ID(not Null, Foreign Key referenced to Place)
    - day(date) (not Null)
    - no_of_people(0-6) (default -0, not Null)
    - no_of_people(6-12) (default -0, not Null)
    - no_of_people(12-18) (default -0, not Null)
    - no_of_people(18-24) (default -0, not Null)
- Officers_on_duty -
    - police_ID(not Null, foreign key referenced to Police)
    - friend_officer_ID(not Null, foreign key referenced to Police)
    - location_ID(not Null, foreign key referenced to Place)
- Reviews -
    - place_ID (not Null, foreign key referenced to Place)
    - Traveller (not Null, foreign key referenced to Traveller)
- Congestion -
    - road_ID (foreign key reference to Roads)
    - Day (not Null)
    - no_of_people(0-4) (default zero, not Null)
    - no_of_people(4-8) (default zero, not Null)
    - no_of_people(8-12) (default zero, not Null)
    - no_of_people(12-16) (default zero, not Null)

    - no_of_people(16-20) (default zero, not Null)
    - no_of_people(20-24) (default zero, not Null)
- Roads -
    - placeA_id(Foriegn Key referenced to Place)
    - placeB_id(Foriegn Key referenced to Place)
    - road_id
    - ambience_level (int)(not Null, default 0)
    - congestion(not Null, default 0)
    - price(not Null, default 0)
    - time_to_travel_by(not Null, default 0)
    - landmarks(not Null, default 0)
    - no_of_accidents(not Null, default 0)
- Friends-
    - person_ID (Foriegn Key reference to Traveller)
    - friend_ID (Foriegn Key reference to Traveller)

Assumptions -

- Every person only spends 30 min at a location
- There are at max 4 passengers in a cab
- Time data for population is taken every 6 hours
- Every place is a point of interest, and every point of interest is connected by road/s
- Assuming constant speed for travel

Functional Dependencies-

- **Place -**
  place_ID -> (place_Name, Location, opening_Time, closing_Time, Entry_fee,Category)
- **Police -**
  police_ID -> police_location_ID
- **Driver -**
  driver_ID -> (Passengers, road_ID)
- **Challan -**
  challan_id -> (driver_id, amount, reason)
- **Traveller -**
  traveller_ID -> (name, Location)

- **Journey -**
  (source_ID ,destination_ID) -> (driver_ID , traveller1_ID, traveller2_ID, traveller3_ID, traveller4_ID)
- **Population -**
  (place_ID, day) -> (no_of_people(0-6), no_of_people(6-12), no_of_people(12-18), no_of_people(18-24))
- **Officers_on_duty -**
  police_ID -> (friend_officer_ID, location_ID)
- **Reviews -**
  (place_ID,Traveller_ID) -> Review
- **Congestion -**
  (road_ID,Day) -> (no_of_people(0-4), no_of_people(4-8), no_of_people(8-12), no_of_people(12-16), no_of_people(16-20), no_of_people(20-24))
- **Roads -**
  (placeA_id, placeB_id) -> (road_id, ambience_level, congestion, price, time_to_travel_by, landmarks, no_of_accidents)
- **Friends-**
  person_ID -> friend_ID

Normalisation

Tables-

1) Place -
   - place_ID (Primary Key)
   - place_Name (not Null)
   - Location (not Null)
   - opening_Time
   - closing_Time
   - Entry_fee (not Null, default - 0)
   - Category (in ‘Religious, Sports, Lifestyle, Parking, Gas Station, Entertainment, Comedy, Dine Out, Adventure, Sightseeing, Bar’)
2) Police -
   - police_ID (Primary key)
   - police_location_ID (not Null, foreign key referenced to Place)
3) Driver -
   - driver_ID (Primary Key)
   - Passengers
   - road_ID(Foreign Key referenced to Roads)
4) Challan -
   - challan_id (Primary Key)

- driver_id (Foriegn Key referenced to Driver)
- amount (not Null, default-0)
- reason (not Null, in ‘Overspeeding, No Helmet, No Seatbelt, Drunk Driving, Traffic Light Jump, No License, Pollution’)

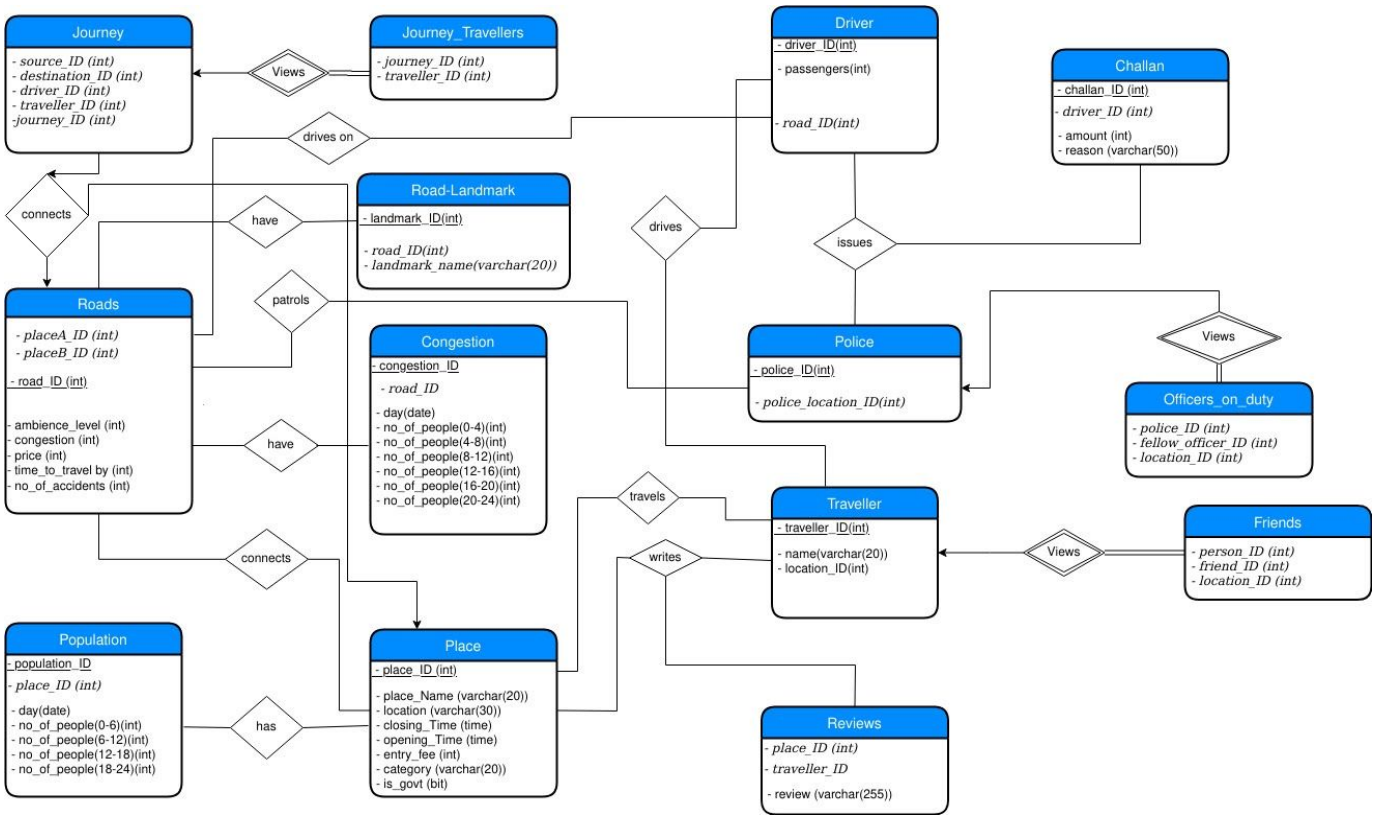
5) Traveller -
   - traveller_ID(Primary Key)
   - name(not Null)
   - Location (not Null)
6) Journey -
   - source_ID (Foriegn Key referenced to Place)
   - destination_ID (Foriegn Key referenced to Place)
   - driver_ID (Foriegn Key referenced to Driver)
   - journey_ID (Primary key)
7) Journey_Travellers
   - journey_ID (foriegn key referenced to journey)
   - travellers_ID (Primary Key)
8) Population -
   - population_ID (Primary key)
   - place_ID(not Null, Foreign Key referenced to Place)
   - day(date) (not Null)
   - no_of_people(0-6) (default -0, not Null)
   - no_of_people(6-12) (default -0, not Null)
   - no_of_people(12-18) (default -0, not Null)
   - no_of_people(18-24) (default -0, not Null)
9) Officers_on_duty -
   - police_ID(not Null, foreign key referenced to Police)
   - friend_officer_ID(not Null, foreign key referenced to Police)
   - location_ID(not Null, foreign key referenced to Place)
10) Reviews -
   - place_ID (not Null, foreign key referenced to Place)
   - Traveller (not Null, foreign key referenced to Traveller)
11) Congestion -
   - congestion_ID (Primary Key)
   - road_ID (foreign key reference to Roads)
   - Day (not Null)
   - no_of_people(0-4) (default zero, not Null)
   - no_of_people(4-8) (default zero, not Null)
   - no_of_people(8-12) (default zero, not Null)
   - no_of_people(12-16) (default zero, not Null)
   - no_of_people(16-20) (default zero, not Null)
   - no_of_people(20-24) (default zero, not Null)
12) Roads -
   - placeA_id(Foriegn Key referenced to Place)

- placeB_id(Foriegn Key referenced to Place)
- Road_id (Primary key)
- ambience_level (int)(not Null, default 0)
- congestion(not Null, default 0)
- price(not Null, default 0)
- time_to_travel_by(not Null, default 0)
- no_of_accidents(not Null, default 0)

13) Friends-
- person_ID (Foriegn Key reference to Traveller)
- friend_ID (Foriegn Key reference to Traveller)

14) Road-Landmark-
- Landmark_ID (primary key, default 0)
- Road_ID (foreign key ref to Roads)

ER Diagram (Revised)



Indexing

- CREATE UNIQUE INDEX roadx
  ON road(road_ID);

- CREATE UNIQUE INDEX placex
  ON place(place_ID);

- CREATE UNIQUE INDEX driverx
  ON driver(driver_ID);

- CREATE UNIQUE INDEX challanx
  ON challan(challan_ID);

- CREATE UNIQUE INDEX travellerx
  ON traveller(traveller_ID);

- CREATE UNIQUE INDEX popx
  ON population(place_ID);

- CREATE UNIQUE INDEX offx
  ON officers_on_duty(police_ID);

- CREATE UNIQUE INDEX congestionx
  ON congestion(road_ID);

- CREATE UNIQUE INDEX reviewx
  ON review(place_ID);

- CREATE UNIQUE INDEX friendx
  ON friends(traveller_ID);

Final Application (Frontend and Backend) -

We have made a cross platform app using **Flutter**. The database is hosted on www.remotemysql.com . We access the database through a locally hosted server made with **flask** and **pymysql**. We then use the locally hosted server as the backend of our application to run queries on the database and give results to the flutter app through http requests.

Roles -

- **Ideation -** Paritosh, Arunesh, Paarmita,Shivang, Shubham
- **Analysing stakeholders and their roles -** Shubham, Shivang, Paarmita, Arunesh, Paritosh
- **Creating entities and their respective relationship-** Shivang, Arunesh, Paritosh, Paarmita, Shubham
- **Formalising database schema and their constraints-** Paarmita, Shubham, Arunesh, Shivang, Paritosh
- **Functional Dependencies-** Paritosh
- **Normalisation-** Arunesh, Paarmita,Shivang, Shubham, Paritosh
- **Database Creation-** Shubham, Shivang, Paarmita, Arunesh, Paritosh
- **Indexing-** Paarmita
- **Queries-** Shubham, Shivang, Paarmita
- **Frontend App-** Arunesh
- **Linking Frontend and Backend-** Paritosh