

PROJECT MILESTONE 1

PROJECT DETAIL:

- **Name of the project:** Analysis of US Accidents 2021 dataset
- **Team:** Level Up
- **Members:**
 - a. Prathamesh Kakade (50460522)
 - b. Shravani Soma (50477925)
 - c. Sriinitha Chinnapatlola (50478024)

PROBLEM STATEMENT:

DESCRIPTION OF THE DATA:

This is a countrywide car accident dataset, which covers 49 states of the USA. The accident data are collected from February 2016 to Dec 2021, using multiple APIs that provide streaming traffic incident (or event) data. These APIs broadcast traffic data captured by a variety of entities, such as the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road-networks. Currently, before any preprocessing is applied, there are about 2.8 million accident records in this dataset.

DESCRIPTION OF PROBLEM:

The objective is to design a relational database system that will use existing “accidents in US” data to provide insightful analytics and help existing Department of Transportation, as well as the future employees of the department to keep track and make important decisions regarding expansion, construction, and maintenance of roads in the United States.

An exponential amount of data is created per day on the internet. Storing huge amounts of data and retrieving knowledge from it is a challenging task. This data will surely help in gaining meaningful insights of what conditions and reasons could be the result of an accident and how it can be prevented for the safety of every individual. It provides an opportunity to the Department of Transportation to improve their services and for the Managers and employees working within to choose the best possible way out to help reduce any road accidents.

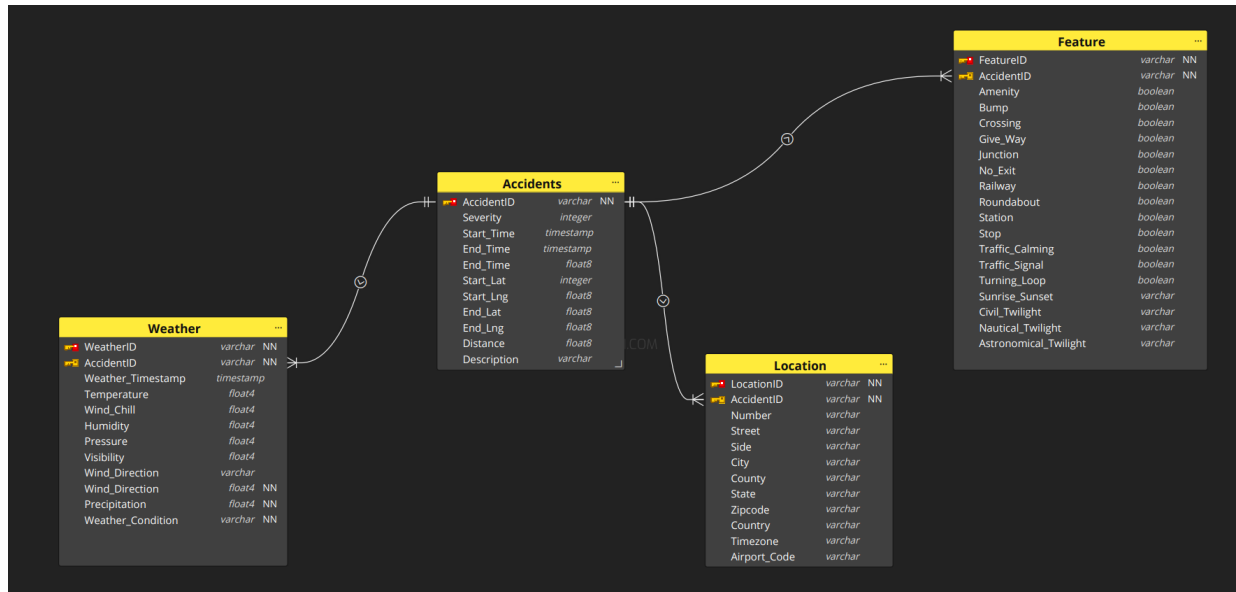
TARGET USER AND ADMIN:

Users of the system will be the managers who are willing to keep track of the mishaps that happen on the road and make important decisions based on the visual plots such as expanding the roads, making sure they are well maintained or building bridges and tunnels that will help in reducing the number of road accidents.

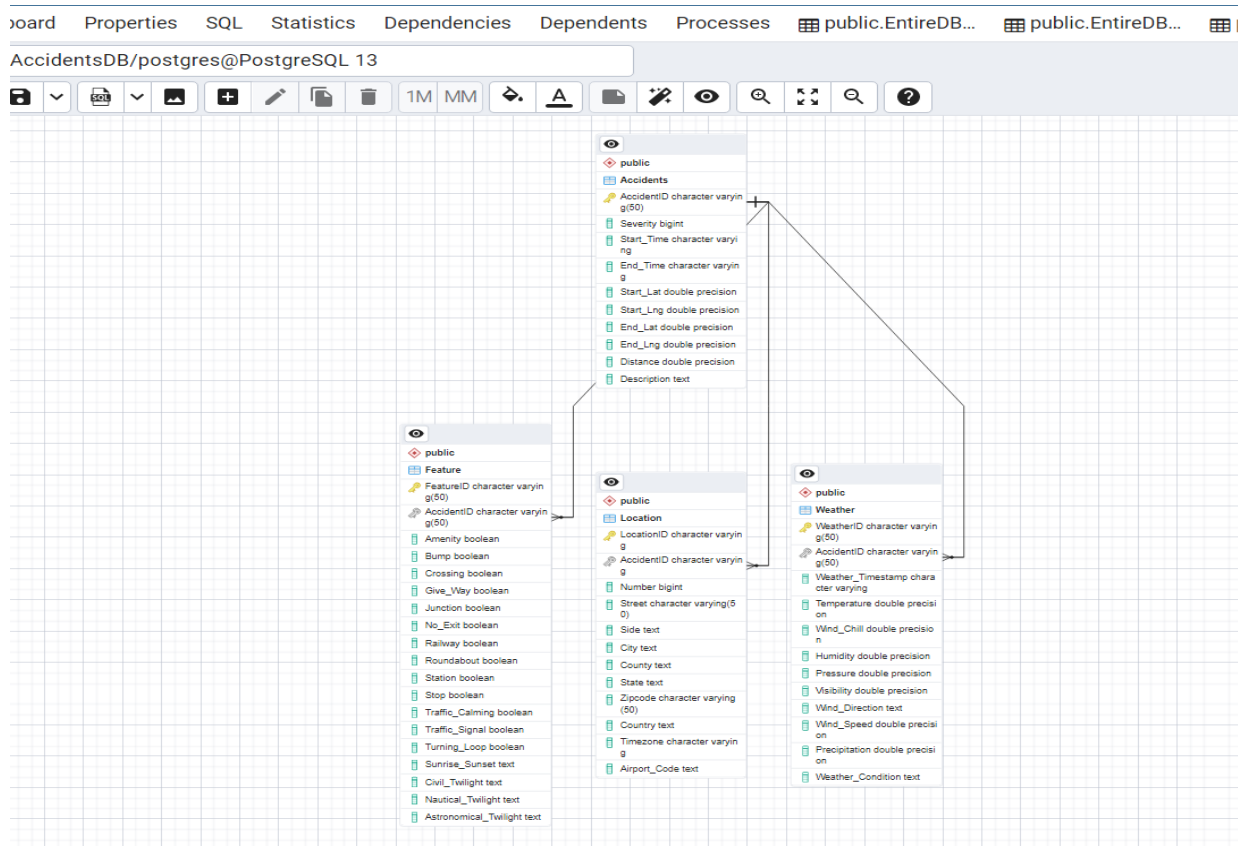
The Administrator of the system will be an employee of IT sector in the Department of Transportation. They will be responsible of maintaining this system and making sure all new data entries are added, modified, or required old entries are deleted from the database system. The admin will have the responsibility to check that the database is running (as we will be building UX for the database system as well).

E/R Diagram:

Created Manually:



Created Using PostgreSQL:



RELATIONSHIP BETWEEN TABLES:

- a. The Weather table is linked to Accidents table using AccidentID.
- b. The Location table is linked to Accidents table using AccidentID.
- c. The Feature table is linked to Accidents table using AccidentID.

DATA VISUALIZATIONS:

To justify the problem statement and to understand various relations in the data, we have plotted few graphs that reflect the information in the dataset pictorially.

1. Accidents with respect to the Latitudes and Longitudes in the dataset:

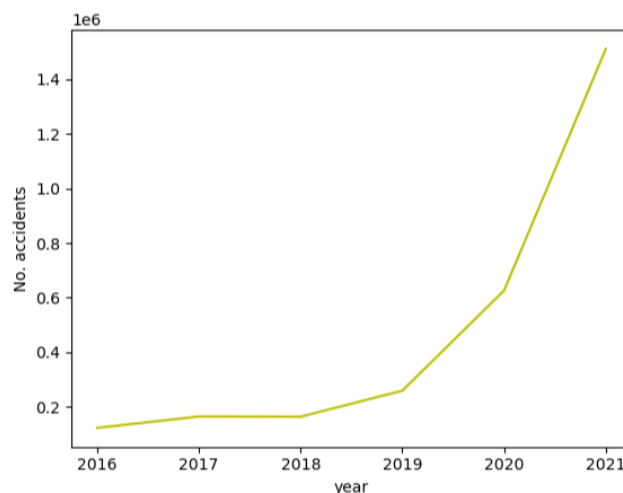
This scatterplot is plotted using the Latitudes and longitudes given in the dataset. It can be inferred that there are numerous accidents happening throughout the country (USA).

```
<AxesSubplot:xlabel='Start_Lng', ylabel='Start_Lat'>
```

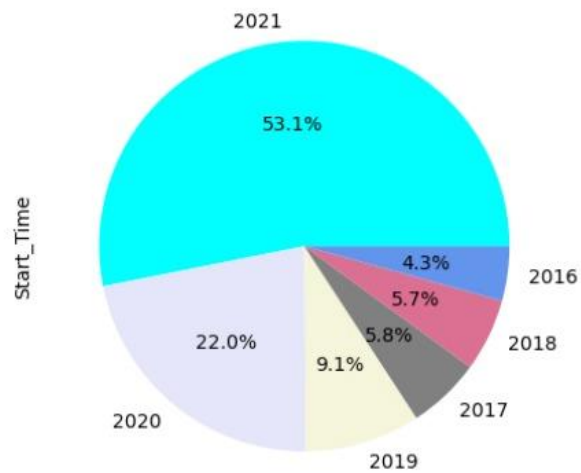


2. Accident Trends over the years (2016 – 2021):

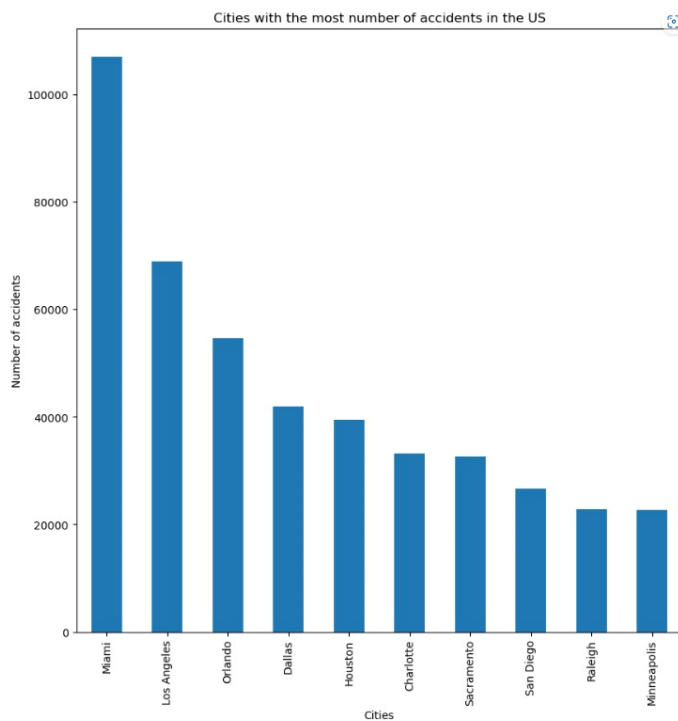
The dataset has accident-related data from the year 2016 until 2021. Here's a line graph representing the trend in the number of accidents over the years. It is observed that the number of accidents has been increasing over the years with a sudden increase particularly observed in between 2020-2021.



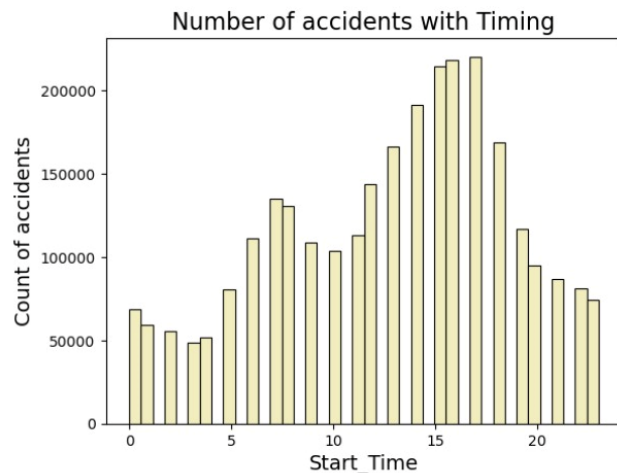
3. Pie Chart: Here's a pie chart describing the percentage of accidents in each year (in the dataset). It can be inferred that most of the accidents occurred in the year 2021 which agrees with the above line graph.



4. Cities with the highest number of accidents: There are many cities in the dataset, to know the cities with the highest number of accidents over the years we plotted a bar graph that shows the top 10 cities with highest accident rates. From the plot, it can be inferred that Miami has the highest number of accidents recorded over the years.



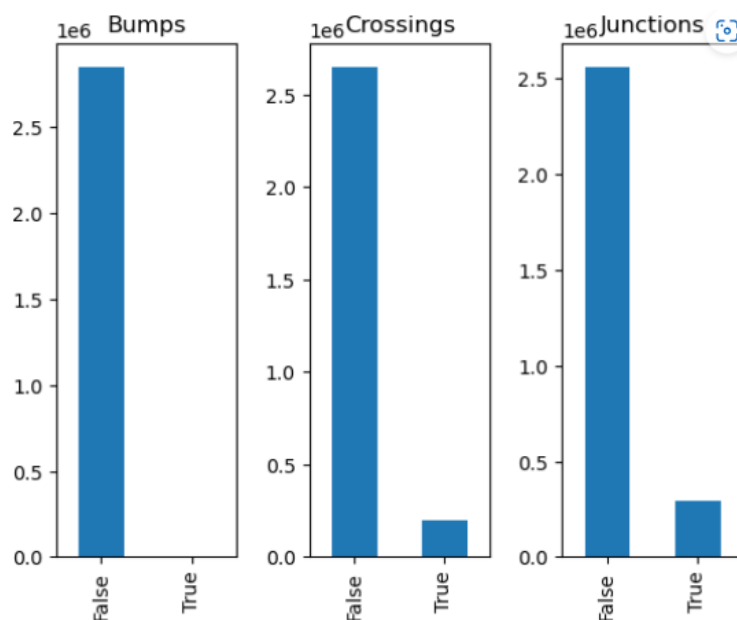
5. Accident rate vs time of the day: A histogram has been plotted to find out the time at which maximum accidents occurred. And from the below plot, we can infer that the greatest number of accidents are occurred around 3pm - 5pm which makes sense as most the people commute from workplace during that time.



6. Subplots to analyze the relationship between the number of accidents and road conditions like bumps, crossings, and junctions:

From the subplot below, the following inferences can be made:

- It can be inferred that all the three parameters played a very minimal role in the occurrence of accidents. The value “True” means that there was a bump/crossing/junction near to the place where an accident occurred. The value “False” means that there was no bump/crossing/junction near to the place where an accident occurred.
- It seems like there’s minimal to no effect of bumps on the number of accidents.



DATABASE IMPLEMENTATION:

DATA SCHEMAS

#	Attribute	Description
1	ID	This is a unique identifier of the accident record.
2	Severity	Shows the severity of the accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay).
3	Start_Time	Shows start time of the accident in local time zone.
4	End_Time	Shows end time of the accident in local time zone. End time here refers to when the impact of accident on traffic flow was dismissed.
5	Start_Lat	Shows latitude in GPS coordinate of the start point.
6	Start_Lng	Shows longitude in GPS coordinate of the start point.
7	End_Lat	Shows latitude in GPS coordinate of the end point.
8	End_Lng	Shows longitude in GPS coordinate of the end point.
9	Distance(mi)	The length of the road extent affected by the accident.
10	Description	Shows natural language description of the accident.
11	Number	Shows the street number in address field.
12	Street	Shows the street name in address field.
13	Side	Shows the relative side of the street (Right/Left) in address field.

#	Attribute	Description
14	City	Shows the city in address field.
15	County	Shows the county in address field.
16	State	Shows the state in address field.
17	Zipcode	Shows the zipcode in address field.
18	Country	Shows the country in address field.
19	Timezone	Shows timezone based on the location of the accident (eastern, central, etc.).
20	Airport_Code	Denotes an airport-based weather station which is the closest one to location of the accident.
21	Weather_Timestamp	Shows the timestamp of weather observation record (in local time).
22	Temperature(F)	Shows the temperature (in Fahrenheit).
23	Wind_Chill(F)	Shows the wind chill (in Fahrenheit).
24	Humidity(%)	Shows the humidity (in percentage).
25	Pressure(in)	Shows the air pressure (in inches).
26	Visibility(mi)	Shows visibility (in miles).
27	Wind_Direction	Shows wind direction.
28	Wind_Speed(mph)	Shows wind speed (in miles per hour).
29	Precipitation(in)	Shows precipitation amount in inches, if there is any.

#	Attribute	Description
30	Weather_Condition	Shows the weather condition (rain, snow, thunderstorm, fog, etc.)
31	Amenity	A POI annotation which indicates presence of amenity in a nearby location.
32	Bump	A POI annotation which indicates presence of speed bump or hump in a nearby location.
33	Crossing	A POI annotation which indicates presence of crossing in a nearby location.
34	Give_Way	A POI annotation which indicates presence of give_way in a nearby location.
35	Junction	A POI annotation which indicates presence of junction in a nearby location.
36	No_Exit	A POI annotation which indicates presence of no_exit in a nearby location.
37	Railway	A POI annotation which indicates presence of railway in a nearby location.
38	Roundabout	A POI annotation which indicates presence of roundabout in a nearby location.
39	Station	A POI annotation which indicates presence of station in a nearby location.
40	Stop	A POI annotation which indicates presence of stop in a nearby location.
41	Traffic_Calming	A POI annotation which indicates presence of traffic_calming in a nearby location.
42	Traffic_Signal	A POI annotation which indicates presence of traffic_signal in a nearby location.

#	Attribute	Description
43	Turning_Loop	A POI annotation which indicates presence of turning_loop in a nearby location.
44	Sunrise_Sunset	Shows the period of day (i.e. day or night) based on sunrise/sunset.
45	Civil_Twilight	Shows the period of day (i.e. day or night) based on civil twilight.
46	Nautical_Twilight	Shows the period of day (i.e. day or night) based on nautical twilight.
47	Astronomical_Twilight	Shows the period of day (i.e. day or night) based on astronomical twilight.

RELATIONSHIP BETWEEN TABLES:

- The Weather table is linked to Accidents table using AccidentID.
- The Location table is linked to Accidents table using AccidentID.
- The Feature table is linked to Accidents table using AccidentID.

ATTRIBUTES

Accident Relation:

Name	Data type
AccidentID (PK)	varchar (50)
Severity	int
Start_Time	text
End_Time	text
Start_Lat	decimal (9,6)
Start_Lng	decimal (9,6)
End_Lat	decimal (9,6)
End_Lng	decimal (9,6)
Distance	decimal (10,2)

Description	text
-------------	------

Weather Relation:

Name	Data type
WeatherID (PK)	varchar (50)
AccidentID (FK)	varchar (50)
Weather_Timestamp	text
Temperature	decimal (9,6)
Wind_Chill	decimal (9,6)
Humidity	decimal (9,6)
Pressure	decimal (9,6)
Visibility	decimal (9,6)
Wind_Direction	Text
Precipitation	decimal (9,6)
Weather_Condition	Text

Location Relation:

Name	Data type
LocationID	varchar (50)
AccidentID	int
Number	varchar (50)
Street	text
Side	text
City	text
County	text
State	text
Zipcode	varchar
Country	text
Timezone	varchar

Feature Relation:

Name	Data type
FeatureID	varchar (50)
AccidentID	varchar (50)
Amenity	boolean
Bump	boolean
Crossing	boolean
Give_Way	boolean
Junction	boolean
No_Exit	boolean
Railway	boolean
Roundabout	boolean
Station	boolean
Stop	boolean
Traffic_Calming	boolean
Traffic_Signal	boolean
Traffic_Loop	boolean
Sunrise_Sunset	text
Civil_Twilight	text
Nautical_Twilight	text
Astronomical_Twilight	text

INSERTION OF DATA IN POSTGRESQL:

Entire Database:

Data Output										Messages	Notifications	
	ID text	Severity bigint	Start_Time text	End_Time text	Start_Lat double precision	Start_Lng double precision	End_Lat double precision	End_Lng double precision	Distance double precision			
1	A-43	4	2016-02-09 18:20:58	2016-02-10 00:20:58	40.45112	-85.15048	40.35429	-85.14993	6.0			
2	A-44	4	2016-02-09 18:20:58	2016-02-10 00:20:58	40.35429	-85.14993	40.45112	-85.15048	6.0			
3	A-48	4	2016-02-10 06:18:49	2016-02-10 12:18:49	40.72813	-84.78965	40.74559	-84.78962	1.20			
4	A-51	2	2016-02-10 08:35:27	2016-02-10 14:35:27	41.83193	-80.10143000000002	41.84149	-80.11099	0.8240000000000000			
5	A-67	2	2016-02-10 12:54:39	2016-02-10 18:54:39	41.48339	-81.66297	41.47692	-81.66075	0.40			
6	A-90	2	2016-02-11 07:20:03	2016-02-11 13:20:03	38.33667	-81.65623000000002	38.33614	-81.65623000000002	0.03700000000000000			
7	A-91	2	2016-02-11 07:20:03	2016-02-11 13:20:03	38.33614	-81.65623000000002	38.33667	-81.65623000000002	0.03700000000000000			
8	A-113	2	2016-02-11 13:30:58	2016-02-11 19:30:58	40.58919	-80.09885	40.58919	-80.09885				
9	A-119	2	2016-02-11 16:56:28	2016-02-11 22:56:28	40.58919	-80.09885	40.58919	-80.09885				
10	A-149	3	2016-02-13 07:14:41	2016-02-13 13:14:41	40.484222	-80.13755400000002	40.503456	-80.139196	1.30			
11	A-192	4	2016-02-15 20:46:40	2016-02-16 02:46:40	38.824929	-85.47449499999998	38.82415	-85.63794	8.70			
12	A-204	2	2016-02-16 06:08:42	2016-02-16 12:08:42	41.06347	-81.50372	41.06472	-81.50414	0.08900000000000000			
13	A-264	2	2016-02-17 17:04:42	2016-02-17 23:04:42	41.47395	-81.69931	41.47865	-81.6931	0.40			
14	A-438	4	2016-02-24 12:27:56	2016-02-24 18:27:56	41.427584	-85.8495	41.471378	-85.839527	3.00			
15	A-439	4	2016-02-24 12:27:57	2016-02-24 18:27:57	41.471378	-85.839527	41.427584	-85.8495	3.00			
16	A-462	4	2016-02-25 06:14:20	2016-02-25 12:14:20	39.97527	-85.14018	39.98511	-85.14406	0.00			
17	A-463	4	2016-02-25 06:14:20	2016-02-25 12:14:20	39.98511	-85.14406	39.97527	-85.14018	0.00			
18	A-476	4	2016-02-25 13:13:48	2016-02-25 19:13:48	39.9672	-81.28699999999998	39.96392	-81.27197	0.80			
19	A-477	4	2016-02-25 13:13:48	2016-02-25 19:13:48	39.96392	-81.27197	39.9672	-81.28699999999998	0.80			
20	A-12532	2	2016-12-07 07:47:35	2016-12-07 13:47:35	38.814617	-104.75764	38.81371	-104.75765	0.00			
Total rows: 1000 of 943318 Query complete 00:00:23.124												
Ln 1, Col 1												

4 Tables were created and Data was inserted into those tables:
Queries for Table creation:

```

19: 1 cur.execute("""CREATE TABLE Accident (
2 AccidentID varchar(50) PRIMARY KEY,
3 Severity int,
4 Start_Time timestamp,
5 End_Time timestamp,
6 Start_Lat decimal(9,6),
7 Start_Lng decimal(9,6),
8 End_Lat decimal(9,6),
9 End_Lng decimal(9,6),
10 Distance decimal(10,2),
11 Description varchar(255));""")
12 print("Table created successfully")
13 conn.commit()
14
15 cur.execute("""CREATE TABLE Feature (
16 FeatureID varchar(50) PRIMARY KEY,
17 AccidentID varchar(50),
18 Amenity boolean,
19 Bump boolean,
20 Crossing boolean,
21 Give_Way boolean,
22 Junction boolean,
23 No_Exit boolean,
24 Railway boolean,
25 Roundabout boolean,
26 Station boolean,
27 Stop boolean,
28 Traffic_Calming boolean,
29 Traffic_Signal boolean,
30 Turning_Loop boolean,
31 Sunrise_Sunset varchar(10),
32 Civil_Twilight varchar(10),
33 Nautical_Twilight varchar(10),
34 Astronomical_Twilight varchar(10),
35 FOREIGN KEY (AccidentID) REFERENCES Accidents(AccidentID)
36 );""")
37 print("Table created successfully")
38 conn.commit()
39
40 cur.execute("""CREATE TABLE Weather (
41 WeatherID varchar(50) PRIMARY KEY,
42 AccidentID varchar(50),
43 Weather_TimeStamp datetime,
44 Temperature_F decimal(5,2),
45 Wind_Chill_F decimal(5,2),
46 Humidity decimal(5,2),
47 Pressure decimal(6,2),
48 Visibility_mi decimal(5,2),
49 Wind_Direction varchar(10),
50 Wind_Speed_mph decimal(5,2),
51 Precipitation_in decimal(5,2),
52 Weather_Condition varchar(50),
53 FOREIGN KEY (AccidentID) REFERENCES Accidents(AccidentID)
54 );""")
55 print("Table created successfully")
56 conn.commit()
57
58 cur.execute("""CREATE TABLE Feature (
59 FeatureID int PRIMARY KEY,
60 AccidentID int,
61 Amenity boolean,
62 Bump boolean,
63 Crossing boolean,
64 Give_Way boolean,
65 Junction boolean,
66 No_Exit boolean,
67 Railway boolean,
68 Roundabout boolean,
69 Station boolean,
70 Stop boolean,
71 Traffic_Calming boolean,
72 FOREIGN KEY (AccidentID) REFERENCES Accidents(AccidentID)
73 );""")
74
75 conn.commit()
76
77
78 conn.close()

```

Table created successfully

Table created successfully

Table created successfully

Table created successfully

Accident Table:

Data Output

Messages

Notifications

	AccidentID [PK] character varying (50)	Severity bigint	Start_Time character varying	End_Time character varying	Start_Lat double precision	Start_Lng double precision	End_Lat double precision	End_Lng double precision	Dislat double
1	A-1		4 2016-02-09 18:20:58	2016-02-10 00:20:58	40.45112	-85.15048	40.35429	-85.14993	
2	A-10		3 2016-02-13 07:14:41	2016-02-13 13:14:41	40.484222	-80.137554000000002	40.503456	-80.139196	
3	A-100		2 2017-01-25 06:58:11	2017-01-25 12:58:11	44.776453	-93.657095	44.77629	-93.68094	
4	A-1000		2 2017-01-30 09:21:22	2017-01-30 15:21:22	44.9649	-93.24474	44.96586	-93.24726	
5	A-10000		2 2021-11-06 12:12:00	2021-11-06 12:34:22	40.652174	-73.961416	40.653471	-73.95943299999998	0.13
6	A-100000		2 2021-12-12 14:40:00	2021-12-12 16:42:25	25.927378	-80.15355	25.925974	-80.153147	
7	A-100001		2 2021-11-21 15:03:55	2021-11-21 16:21:39	45.597552	-111.110433	45.598358000000001	-111.104324	
8	A-100002		2 2021-12-05 17:48:52	2021-12-05 19:08:20	40.220145	-75.300302	40.221088	-75.301873	
9	A-100003		2 2021-10-18 17:28:05	2021-10-18 18:10:12	43.148563	-77.597188	43.141527	-77.584878	0.13
10	A-100004		2 2021-11-16 18:53:00	2021-11-16 20:09:21	43.00783	-83.698431	43.007275	-83.699655	
11	A-100005		2 2021-07-21 12:21:00	2021-07-21 14:03:09	42.35964	-87.844752	42.359628	-87.843506	
12	A-100006		2 2021-05-24 14:56:42	2021-05-24 16:14:13	30.33093	-81.66834	30.33576	-81.667007	
13	A-100007		2 2021-09-17 18:25:00	2021-09-17 19:44:22	39.788259	-84.200829	39.788696	-84.200682	
14	A-100008		2 2021-12-31 07:12:00	2021-12-31 09:57:00	36.51775	-79.228903	36.53238	-79.213763	
15	A-100009		2 2021-09-23 08:04:33	2021-09-23 09:49:33	43.302783000000001	-124.211613	43.302334	-124.211617	
16	A-10001		2 2021-11-30 21:43:00	2021-11-30 22:43:00	43.68592	-124.145879	43.682866	-124.14954	
17	A-100010		2 2021-10-04 18:14:30	2021-10-04 20:21:37	28.657037	-81.224985	28.654379	-81.227171	
18	A-100011		2 2021-04-02 18:15:26	2021-04-02 22:18:26	45.802376	-108.438362	45.802923	-108.43708	0.07
19	A-100012		2 2021-07-22 15:24:10	2021-07-22 15:44:10	30.417886	-91.142289	30.42223	-91.139799	
20	A-100013		2 2021-10-07 01:46:00	2021-10-07 03:58:50	34.134981	-118.361132	34.134936	-118.361042	

Total rows: 1000 of 943318

Query complete 00:00:10.308

✓

Successfully run. Total query runtime: 10 secs 308 msec. 943318 rows affected.

✗

Location Table:

Data OutputMessagesNotifications

	LocationID [PK] character varying	AccidentID character varying	Number bigint	Street character varying (50)	Side text	City text	County text	State text	Zipcode character varying (50)
1	L-1	A-1	9001	W State Road 26	R	Dunkirk	Jay	IN	47336
2	L-10	A-10	5	Forest Grove Rd	L	Coraopolis	Allegheny	PA	15108-3485
3	L-100	A-100	4836	Highway 212 E	L	Chaska	Carver	MN	55318-9249
4	L-1000	A-1000	804	20th Ave S	R	Minneapolis	Hennepin	MN	55454
5	L-10000	A-10000	2014	Caton Ave	R	Brooklyn	Kings	NY	11226-2804
6	L-100000	A-100000	16390	Biscayne Blvd	L	North Miami Beach	Miami-Dade	FL	33160
7	L-100001	A-100001	10072	Cottonwood Rd	R	Bozeman	Gallatin	MT	59718-8969
8	L-100002	A-100002	781	Summeytown Pike	R	Lansdale	Montgomery	PA	19446-5301
9	L-100003	A-100003	344	Monroe Ave	L	Rochester	Monroe	NY	14607-3662
10	L-100004	A-100004	901	Cedar St	L	Flint	Genesee	MI	48503-3657
11	L-100005	A-100005	1098	Washington St	R	Waukegan	Lake	IL	60085-5429
12	L-100006	A-100006	300	N Davis St	L	Jacksonville	Duval	FL	32202-4816
13	L-100007	A-100007	2633	Ridge Ave	L	Dayton	Montgomery	OH	45414-5431
14	L-100008	A-100008	9990	NC Highway 62 N	R	Milton	Caswell	NC	27305-9346
15	L-100009	A-100009	60792	Highway 101	L	Coos Bay	Coos	OR	97420
16	L-10001	A-10001	75906	US Highway 101	L	Reedsport	Douglas	OR	97467
17	L-100010	A-100010	1803	W Broadway St	R	Oviedo	Seminole	FL	32765
18	L-100011	A-100011	1831	Old Hardin Rd	L	Billings	Yellowstone	MT	59101-6562
19	L-100012	A-100012	3180	College Dr	R	Baton Rouge	East Baton Rouge	LA	70808-3119
20	L-100013	A-100013	3701	Cahuenga Blvd	R	Studio City	Los Angeles	CA	91604-3504

Total rows: 1000 of 943318Query complete 00:00:21.212

Successfully run. Total query runtime: 7 secs 212 msec. 943318 rows affected.

Feature Table:

Data OutputMessagesNotifications

	FeatureID [PK] character varying (50)	AccidentID character varying (50)	Amenity boolean	Bump boolean	Crossing boolean	Give_Way boolean	Junction boolean	No_Exit boolean	Railway boolean	Roundabout boolean	Station boolean	Stop boolean
1	F-1	A-1	false	true	true	true	false	false	false	true	false	true
2	F-10	A-10	false	false	false	true	true	true	false	false	true	false
3	F-100	A-100	false	false	true	false	false	true	true	false	true	true
4	F-1000	A-1000	false	true	true	false	false	true	true	true	true	true
5	F-10000	A-10000	true	true	false	false	true	true	true	false	true	false
6	F-100000	A-100000	false	false	false	true	true	true	false	true	false	true
7	F-100001	A-100001	true	true	false	true	true	false	true	true	true	true
8	F-100002	A-100002	false	false	true	true	true	false	false	true	false	false
9	F-100003	A-100003	true	true	true	true	true	false	false	false	true	false
10	F-100004	A-100004	false	true	true	false	true	true	true	true	false	false
11	F-100005	A-100005	true	true	false	true	false	false	true	false	false	false
12	F-100006	A-100006	true	false	false	false	true	true	false	false	true	true
13	F-100007	A-100007	false	true	true	false	true	true	true	false	false	true
14	F-100008	A-100008	false	true	false	true	true	false	false	true	true	false
15	F-100009	A-100009	false	true	true	true	true	true	true	false	true	false
16	F-10001	A-10001	false	false	false	true	true	true	true	true	true	true
17	F-100010	A-100010	false	false	false	false	false	true	true	true	false	false
18	F-100011	A-100011	false	true	true	false	false	false	false	true	false	true
19	F-100012	A-100012	false	false	true	true	true	false	true	true	true	true
20	F-100013	A-100013	false	false	true	false	true	true	true	true	true	false

Total rows: 1000 of 943318Query complete 00:00:08.588

Successfully run. Total query runtime: 8 secs 588 msec. 943318 rows affected.

Weather Table:

Data Output Messages Notifications										
	WeatherID [PK] character varying (50)	AccidentID character varying (50)	Weather_Timestamp character varying	Temperature double precision	Wind_Chill double precision	Humidity double precision	Pressure double precision	Visibility double precision	Wind_Direc text	
1	W-1	A-1	2016-02-09 18:20:00	19.9	7.3	81	29.85	2	WNW	
2	W-10	A-10	2016-02-13 06:51:00	6.1	-12.2	63	30.32	10	WNW	
3	W-100	A-100	2017-01-25 06:53:00	30	20.5	88	29.7	1.2	North	
4	W-1000	A-1000	2017-01-30 09:07:00	25	13.1	85	29.67	10	South	
5	W-10000	A-10000	2021-11-06 11:51:00	49	49	44	30.33	10	CALM	
6	W-100000	A-100000	2021-12-12 14:53:00	82	82	58	30.12	10	ENE	
7	W-100001	A-100001	2021-11-21 14:56:00	46	46	30	25.75	10	WNW	
8	W-100002	A-100002	2021-12-05 17:55:00	47	43	58	29.95	10	SE	
9	W-100003	A-100003	2021-10-18 17:54:00	48	46	80	29.39	10	W	
10	W-100004	A-100004	2021-11-16 18:53:00	39	33	60	29.23	10	SE	
11	W-100005	A-100005	2021-07-21 11:55:00	69	69	75	29.39	10	NNE	
12	W-100006	A-100006	2021-05-24 14:53:00	87	87	43	30.14	10	SE	
13	W-100007	A-100007	2021-09-17 17:56:00	86	86	43	29.03	10	SW	
14	W-100008	A-100008	2021-12-31 06:53:00	54	54	97	29.35	0.5	CALM	
15	W-100009	A-100009	2021-09-23 07:56:00	53	53	100	30.22	10	N	
16	W-10001	A-10001	2021-11-30 21:56:00	51	51	96	30.34	5	CALM	
17	W-100010	A-100010	2021-10-04 17:53:00	83	83	63	29.89	10	E	
18	W-100011	A-100011	2021-04-02 17:53:00	71	71	12	26.23	10	SW	
19	W-100012	A-100012	2021-07-22 15:20:00	87	87	67	30.04	10	S	
20	W-100013	A-100013	2021-10-07 01:51:00	63	63	82	29.14	10	S	

Total rows: 1000 of 943318 Query complete 00:00:06.034

✓ Successfully run. Total query runtime: 6 secs 34 msec. 943318 rows affected. ✕

PRIMARY AND FOREIGN KEYS

- Accidents Relation
 - Primary key - AccidentID
 - Foreign key - No Foreign keys
- Weather Relation
 - Primary key - WeatherID
 - Foreignkey - id referenced from Accidents relation on Accidents.AccidentID
 - Foreignkey - id referenced from Location relation on Location.LocationID
- Location Relation
 - Primary key - LocationID
 - Foreignkey - id referenced from Accidents relation on Accidents.AccidentID
- Feature Relation
 - Primary key - FeatureID
 - Foreignkey - id referenced from Accidents relation on Accidents.AccidentID

SAMPLE QUERIES:


- **Display Severity of all Accidents in Erie County:**
Select "Accidents"."Severity"
from "Accidents"
join "Location" on "Accidents"."AccidentID" = "Location"."AccidentID"
where "Location"."County"='Erie';

Data Output Messages Notifications		
	Severity bigint	
1	4	
2	4	
3	2	
4	2	
5	2	
6	2	
7	2	
8	4	
Total rows: 17 of 17 Query complete 00:00:00.700		

- **Display Description of all Accidents with AccidentID between A-100 and A-120:**
Select "Description"
from "Accidents"
where "AccidentID"
between 'A-100' and 'A-120';




Data Output Messages Notifications		
	Description text	
1	Closed between US-421 and IN-3 - Road closed due to accident.	
2	Ramp to Exit 125B - Accident.	
3	At CR-43 - Accident.	
4	At Cologne - Accident.	
5	At Broadway St - Accident.	
6	At Galpin Blvd - Accident.	
7	At 31st St - Accident. Right lane closed.	
8	At 31st St - Accident.	
9	At CR-21/Trout Lake Rd - Accident.	
10	Closed at CO-12/Highway of Legends - Road closed due to accident.	
11	Closed at US-87/US-85/I-25-BL/Main St/7th St - Road closed due to accident.	
12	At 208th Ln - Accident.	
13	Closed at Frank Dr - Road closed due to accident.	
14	At Calhoun St - Accident.	
15	At Iron Horse Trl - Accident.	
16	At NE-98/854th Rd - Accident.	

- **Display Count of all Accidents that had a Junction nearby:**
 Select COUNT("Junction")
 from "Feature"
 where "Junction"='True';

Data Output		Messages	Notifications
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> </div>			
	count bigint 		
1	472348		

Total rows: 1 of 1 Query complete 00:00:00.633

- **Display AccidentID, Wind Speed and Humidity for AccidentID=A-1000:**
 Select "Accidents"."AccidentID", "Weather"."Wind_Speed", "Weather"."Humidity"
 from "Weather"
 join "Accidents"
 on "Weather"."AccidentID"="Accidents"."AccidentID"
 where "Accidents"."AccidentID"='A-1000';

Data Output				Messages	Notifications
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> </div>					
	AccidentID character varying (50) 	Wind_Speed double precision 	Humidity double precision 		
1	A-1000	13.8	85		

Total rows: 1 of 1 Query complete 00:00:00.224

REMOVING DEPENDENCIES:

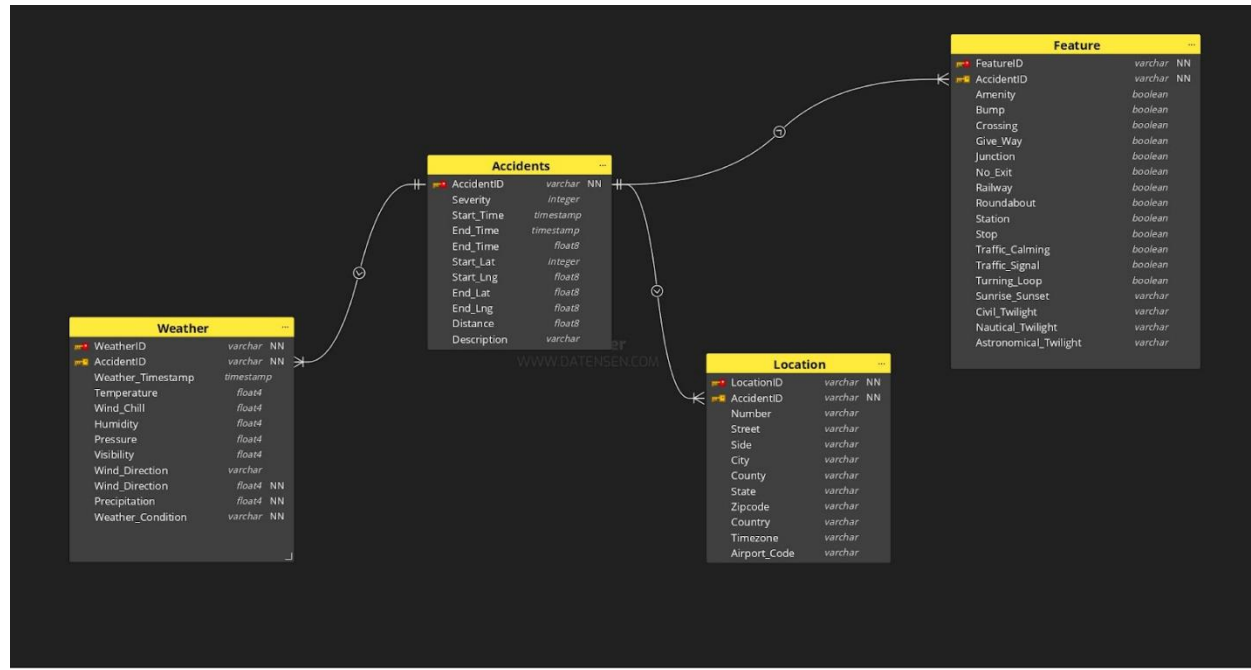


Fig: Original ER diagram for the database

In the initial phase, we have created the four tables “Accident”, “Location”, “Weather”, and “Feature”. All these four tables satisfied the BCNF criteria.

The following is a closer look at each table to see how it satisfies BCNF.

Table 1: Accident

The primary key attribute is AccidentID, which uniquely identifies each record. There are no non-trivial functional dependencies between attributes, as each attribute is independent of the others.

Table 2: Weather

The primary key attribute is WeatherID, which uniquely identifies each record. The only functional dependency between attributes is the foreign key relationship with AccidentID. However, this is a trivial functional dependency, as the Weather table cannot exist without the corresponding Accident record. Therefore, the table satisfies BCNF.

Table 3: Feature

The primary key attribute is FeatureID, which uniquely identifies each record. The only functional dependency between attributes is the foreign key relationship with AccidentID. However, this is a trivial functional dependency, as the Feature table cannot exist without the corresponding Accident record. Therefore, the table satisfies BCNF.

Table 4: Location

The primary key attribute is LocationID, which uniquely identifies each record. The only functional dependency between attributes is the foreign key relationship with AccidentID. However, this is a trivial functional dependency, as the Location table cannot exist without the corresponding Accident record. Therefore, the table satisfies BCNF.

The original database was already in BCNF. However, the new database introduces a few changes to improve the design and avoid redundancies. The first change is the splitting of the location information into two separate tables: Start Location and End Location. This change was made to avoid redundant data and to improve the scalability and performance of the database. In the original database, both the start and end location data were stored in the same table, which could result in redundant data if an accident occurred at the same location for both the start and end points. For example, let's say there is an accident that occurs at the same location for both the start and end points. In the original database, this would result in duplicate location data being stored in the Accident table. However, in the new database, this data is split into two separate tables, Start Location and End Location, where each table only stores the location information specific to the start or end of the accident.

The second change is the addition of a Twilight Phase table to store information related to the time of day when the accident occurred. This table avoids redundant data and provides more specific information about the accident, which can be useful for data analysis and modeling. For example, let's say an accident occurs during the Civil Twilight phase. In the original database, this information would be stored in the Accident table as a string, which would be less specific and more difficult to work with. However, in the new database, this information is stored in a separate table that contains specific data about the different twilight phases, making it easier to analyze and model.

Overall, the changes made in the new database were done to improve the design, scalability, and performance of the database, while also avoiding redundancies and providing more specific information about the accidents.

ER DIAGRAM:

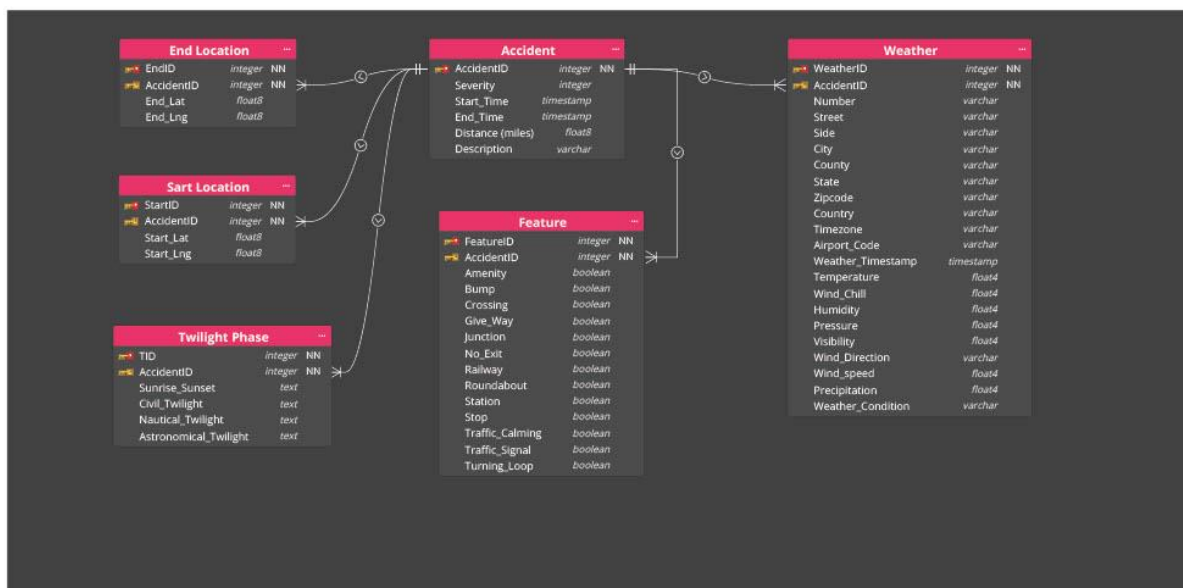


Fig: ER diagram for the new database

The new database has 6 tables namely “End Location”, “Start Location”, “Twilight Phase”, “Accident”, “Feature” and “Weather”. We have split the Accident table from the original database into End Location, Start Location tables with primary keys “EndID” and “StartID” respectively. These tables have “AccidentID” as the foreign key to refer to the Accident table. Some columns from the Feature table have been removed and added to the new table named “Twilight Phase” which has “TID” as its primary key and “AccidentID” as the foreign key.

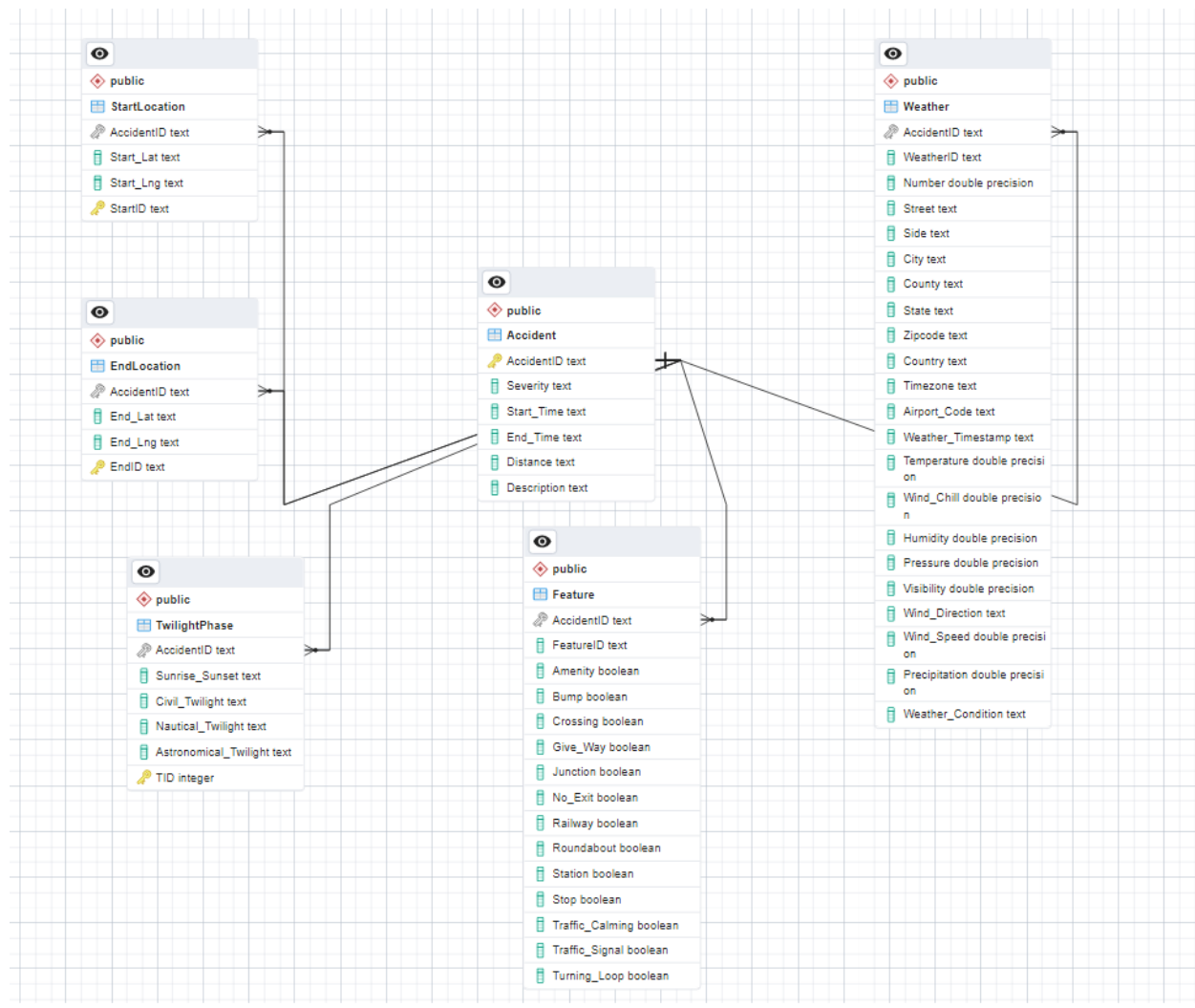


Fig: ER diagram for the new database (PostgreSQL)

Some constraints that could be applicable to this use case are:

1. Unique AccidentID: Each accident should have a unique ID assigned to it.
2. Non-null attributes: Certain attributes such as Start_Time, End_Time, and Start_Lat/Start_Lng should not be null, as they are crucial to the accident record.
3. Referential integrity: Foreign key constraints should be applied to ensure that records in the Weather, Feature, End Location, and Start Location tables match up to existing records in the Accident table.
4. Check constraints: For attributes such as Severity, Distance(mi), and Weather_Condition, check constraints can be applied to ensure that the values entered fall within a certain range or set of values.

INDEXING:

1. As the dataset we chose consists of many rows, it is inevitable that we face issues while performing queries after splitting the data into different tables.

Initially, our database consisted of 4 tables namely: Accidents, Weather, Feature and Location. Though the tables were already in BCNF, we learned that the execution time required to implement a query involving the above tables took comparatively more time than the queries performed after splitting the tables into 6 (which are also in BCNF).

SQL Query (On the initial tables):

Query execution time: 16315 msec or 16.315 sec.

Query Query History Scratch Pad x

```
1 SELECT "LocationID", "AccidentID", "Number", "Street", "Side", "City", "County", "State", "Zipcode"
2 FROM public."Location"
3 ORDER BY "AccidentID" DESC
4
```

Data Output Messages Notifications

	LocationID text	AccidentID text	Number double precision	Street text	Side text	City text	County text	State text	Zipcode text	Country text	T te
1	L-99999	A-99999	13499	NE 9th Ave	R	North Miami	Miami-Dade	FL	33161-4115	US	L
2	L-99998	A-99998	6783	De Moss Dr	L	Houston	Harris	TX	77074-4909	US	L
3	L-99997	A-99997	1304	W Valley Pkwy	R	Escondido	San Diego	CA	92029-2132	US	L
4	L-99996	A-99996	4715	S Tryon St	R	Charlotte	Mecklenburg	NC	28217	US	L
5	L-99995	A-99995	5009	Canal St	R	New Orleans	Orleans	LA	70119-5834	US	L
6	L-99994	A-99994	1199	Rockv Grove Ave	L	Franklin	Venanoo	PA	16323-6243	US	L

Total rows: 1000 of 943318 Query complete 00:00:16.315 Ln 2, Col 23

SQL Query (On the tables after the split):

Query execution time: 12727 msec or 12.727 sec.

Query Query History Scratch Pad x

```
1 SELECT * FROM public."Weather"
2 ORDER BY "AccidentID" DESC
3
```

Data Output Messages Notifications

	AccidentID text	WeatherID text	Number double precision	Street text	Side text	City text	County text	State text	Zipcode text	Country text	Ti te
1	A-99999	W-99999	13499	NE 9th Ave	R	North Miami	Miami-Dade	FL	33161-4115	US	U
2	A-99998	W-99998	6783	De Moss Dr	L	Houston	Harris	TX	77074-4909	US	U
3	A-99997	W-99997	1304	W Valley Pkwy	R	Escondido	San Diego	CA	92029-2132	US	U
4	A-99996	W-99996	4715	S Tryon St	R	Charlotte	Mecklenburg	NC	28217	US	U
5	A-99995	W-99995	5009	Canal St	R	New Orleans	Orleans	LA	70119-5834	US	U
6	A-99994	W-99994	1199	Rockv Grove Ave	L	Franklin	Venanoo	PA	16323-6243	US	U

Total rows: 1000 of 943318 Query complete 00:00:12.727 Ln 1, Col 27

We observe that the execution time of the above query performed on the tables before splitting is 16.315 seconds while the execution time of the query performed after splitting the tables is 12.727 seconds. This is one of the problems we faced while handling large dataset and splitting the tables further resolved the same.

2. As part of the project, we were asked to implement various insertion/deletion and complex select queries. It was observed that, due to the complexity of the SELECT query and the also the amount of data it must sequentially search to provide a result consumes considerable amount of time. Hence, to reduce the data retrieval time and reduce the execution time, we have implemented indexing on the tables in our database.

Below are a few of the queries that demonstrated significant change in the execution time before and after implementing indexing on the relevant tables.

(a) The below query groups the data by the state in the Accident table and calculates the average temperature and number of accidents for each state.

Query execution time (Before Indexing): 2990 msec.

The screenshot displays a database query editor interface. The top section shows a SQL query in a text area, with a 'Query History' tab and a 'Scratch Pad' tab. The query is as follows:

```
1 SELECT
2   w."State",
3   AVG(w."Temperature") AS "AvgTemperature",
4   COUNT(*) AS NumAccidents
5 FROM
6   public."Accident" a
7   JOIN public."Weather" w ON a."AccidentID" = w."AccidentID"
8 GROUP BY w."State";
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with the following columns: State, AvgTemperature, and numaccidents. The table contains 8 rows of data, corresponding to the states AL, AR, AZ, CA, CO, CT, DC, and DE. The bottom of the interface shows the total number of rows (49 of 49) and the query completion time (00:00:02.990).

	State	AvgTemperature	numaccidents
1	AL	64.79888081680738	5093
2	AR	60.87812391831084	2889
3	AZ	74.68556956996723	20138
4	CA	62.83944692040495	215629
5	CO	53.18078431372548	3570
6	CT	53.06176584986386	2571
7	DC	63.26280649926145	3385
8	DE	63.75506784326893	1474

Total rows: 49 of 49 Query complete 00:00:02.990 Ln 3, Col 43

Query execution time (After indexing): 673 msec.

Query

Query History

Scratch Pad

1

SELECT

2

w."State",

3

AVG(w."Temperature") AS "AvgTemperature",

4

COUNT(*) AS NumAccidents

5

FROM

6

public."Accident" a

7

JOIN public."Weather" w ON a."AccidentID" = w."AccidentID"

8

GROUP BY w."State";

Data Output

Messages

Notifications

	State	AvgTemperature	numaccidents
	text	double precision	bigint
1	AL	64.79888081680738	5093
2	AR	60.87812391831084	2889
3	AZ	74.68556956996723	20138
4	CA	62.83944692040495	215629
5	CO	53.18078431372549	3570
6	CT	53.06176584986386	2571
7	DC	63.26280649926145	3385
8	DE	63.70580670432659	1474

Total rows: 49 of 49 Query complete 00:00:00.673

Ln 3, Col 43

Explanation: The above query has been implemented using indexing on the “AccidentID” column of the Accident table, “AccidentID” and “State” columns of the weather table. These indexes help in improving query performance while joining or filtering on the columns that are indexed.

(b) This below query selects the AccidentID, Severity, Temperature, and Amenity columns for all accidents that occurred in the United States during the month of January 2017.

Query execution time (Before Indexing): 990 msec.

Query

Query History

Scratch Pad

```
1 SELECT
2   A."AccidentID",
3   A."Severity",
4   W."Temperature",
5   F."Amenity"
6 FROM
7   public."Accident" A
8   INNER JOIN
9     public."Weather" W ON A."AccidentID" = W."AccidentID"
10  LEFT JOIN
11    public."Feature" F ON A."AccidentID" = F."AccidentID"
12 WHERE
13   A."Start_Time" BETWEEN '2017-01-01 00:00:00' AND '2017-01-31 23:59:59'
14   AND W."Country" = 'US';
```

Data Output

Messages

Notifications

	AccidentID text	Severity text	Temperature double precision	Amenity boolean
1	A-366	4	39	true
2	A-367	4	39	false
3	A-368	3	39	false
4	A-369	3	39.9	false
5	A-370	4	42.8	true
6	A-371	4	41	false
7	A-372	4	41.9	true
8	A-373	4	41.0	true

Total rows: 366 of 366

Query complete 00:00:00.990

Ln 1, Col 1

Query execution time (After Indexing): 61 msec.

Query

Query History

Scratch Pad

```
1 SELECT
2   A."AccidentID",
3   A."Severity",
4   W."Temperature",
5   F."Amenity"
6 FROM
7   public."Accident" A
8 INNER JOIN
9   public."Weather" W ON A."AccidentID" = W."AccidentID"
10 LEFT JOIN
11   public."Feature" F ON A."AccidentID" = F."AccidentID"
12 WHERE
13   A."Start_Time" BETWEEN '2017-01-01 00:00:00' AND '2017-01-31 23:59:59'
14   AND W."Country" = 'US';
```

Data Output

Messages

Notifications

	AccidentID text	Severity text	Temperature double precision	Amenity boolean
1	A-366	4	39	true
2	A-367	4	39	false
3	A-368	3	39	false
4	A-369	3	39.9	false
5	A-370	4	42.8	true
6	A-371	4	41	false
7	A-372	4	41.9	true

Total rows: 366 of 366

Query complete 00:00:00.061

Ln 1, Col 1

Explanation: The above query has been implemented using indexing on the “AccidentID” and “Start_Time” columns of the Accident table, “Accident_ID” and “Country” columns of the Weather table and “AccidentID” column of the feature table.

Below we can observe the Indexes that were created for all the 6 tables which helped with the optimization of our queries. How these indexes really helped in optimization process has been explained later with the help of snippets of execution flow and cost after indexes was applied.

Tables (8)

Accident

Columns

Constraints

Indexes (3)

idx_accident_accidentid

idx_accident_severity

idx_accident_starttime

RLS Policies

Rules

Triggers

Feature

Columns

Constraints

Indexes (1)

idx_feature_accidentid

RLS Policies

Rules

Triggers

Location

StartLocation

TwilightPhase

Weather

Columns

Constraints

Indexes (3)

idx_weather_accidentid

idx_weather_city

idx_weather_country

SQL QUERIES:

Insertion Queries:

(i) INSERT INTO Accident (“Severity”, “Start_Time”, “End_Time”, “Distance”, “Description”) VALUES (3, '2023-05-03 14:30:00', '2023-05-03 15:00:00', 2.5, 'Car accident on highway');

Explanation: Insert a new accident record into the Accident table with Severity level 3, Start_Time as '2023-05-03 14:30:00', End_Time as '2023-05-03 15:00:00', Distance as 2.5, and Description as 'Car accident on highway'.

```
Query    Query History
1  INSERT INTO Accident ("Severity", "Start_Time", "End_Time", "Distance", "Description")
2  VALUES (3, '2023-05-03 14:30:00', '2023-05-03 15:00:00', 2.5, 'Car accident on highway');|

Data Output  Messages  Notifications
INSERT 0 1

Query returned successfully in 61 msec.
```

(ii) INSERT INTO Feature (“AccidentID”, “Amenity”, “Bump”, “Crossing”, “Give_Way”, “Junction”, “No_Exit”, “Railway”, “Roundabout”, “Station”, “Stop”, “Traffic_Calming”, “Traffic_Signal”, “Turning_Loop”) VALUES (1001, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0);

Explanation: Insert a new feature record into the Feature table for the accident with AccidentID 1001, with values of 0 for all attributes except for Railway, which is 1.

```
Query    Query History
1  INSERT INTO Feature ("AccidentID", "Amenity", "Bump", "Crossing", "Give_Way", "Junction", "No_Exit", "Railway",
2  "Roundabout", "Station", "Stop", "Traffic_Calming", "Traffic_Signal", "Turning_Loop")
3  VALUES (1001, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0);

Data Output  Messages  Notifications
INSERT 0 1

Query returned successfully in 61 msec.
```

Deletion Queries:

(i) DELETE FROM Accident WHERE "Severity" = 2;

Explanation: Delete all accident records in the Accident table with Severity level 2.

Query	Query History
1	DELETE FROM Accident WHERE "Severity" = 2;
Data Output	Messages
DELETE 889904	
Query returned successfully in 3 secs 898 msec.	

(ii) DELETE FROM Feature WHERE "FeatureID" = 'F-1';

Explanation: Delete the feature record in the Feature table with FeatureID F-1.

Query	Query History
1	DELETE FROM Feature WHERE "FeatureID" = 'F-1';
2	
Data Output	Messages
DELETE 1	
Query returned successfully in 204 msec.	

Updating Queries:

(i) UPDATE Accident SET 'Severity' = 4 WHERE "AccidentID" = 'A-1';

Explanation: Update the Severity level of the accident with AccidentID A-1 to 4.

Query	Query History
1	UPDATE Accident SET "Severity" = 4 WHERE "AccidentID" = 'A-1';

Data Output	Messages	Notifications
UPDATE 1	Query returned successfully in 215 msec.	

(ii) UPDATE EndLocation SET "End_Lat" = 37.789, "End_Lng" = -122.434 WHERE "EndID" = 201 AND "AccidentID" = 'A-1001';

Explanation: Update the End_Lat and End_Lng values of the End Location record with EndID 201 for the accident with AccidentID A-1001.

Query	Query History
1	UPDATE EndLocation SET "End_Lat" = 37.789, "End_Lng" = -122.434 WHERE "EndID" = 201 AND "AccidentID" = 'A-1001';
2	

Data Output	Messages	Notifications
UPDATE 0	Query returned successfully in 43 msec.	

Select Queries:

(i) Here's an example query that demonstrates the use of **joins** to retrieve information from multiple tables:

```
SELECT A. "AccidentID", A. "Severity", W. "Temperature", F. "Amenity"  
FROM Accident A  
INNER JOIN Weather W ON A. "AccidentID" = W. "AccidentID"  
LEFT JOIN Feature F ON A. "AccidentID" = F. "AccidentID"  
WHERE A. "Start_Time" BETWEEN '2017-01-01 00:00:00' AND '2017-01-31 23:59:59'  
AND W. "Country" = 'US'
```

Explanation: This query joins the Accident, Weather, and Feature tables. It selects the AccidentID, Severity, Temperature, and Amenity columns for all accidents that occurred in the United States during the month of January 2017. The INNER JOIN clause joins the Accident and Weather tables on the AccidentID column, while the LEFT JOIN clause joins the Accident and Feature tables on the same column. This allows us to retrieve information about the weather conditions and features present at the location of each accident.

Query

Query History

```
1 SELECT A."AccidentID", A."Severity", W."Temperature", F."Amenity"
2 FROM Accident A
3 INNER JOIN Weather W ON A."AccidentID" = W."AccidentID"
4 LEFT JOIN Feature F ON A."AccidentID" = F."AccidentID"
5 WHERE A."Start_Time" BETWEEN '2017-01-01 00:00:00' AND '2017-01-31 23:59:59'
6 AND W."Country" = 'US';
```

Data Output

Messages

Notifications

	AccidentID text	Severity bigint	Temperature double precision	Amenity boolean
1	A-382	4	42.1	false
2	A-384	4	46	true
3	A-410	4	30	true
4	A-419	4	14	true
5	A-426	4	21.9	false
6	A-432	4	10	true

Total rows: 236 of 236

Query complete 00:00:00.380

(ii) Here's an example of a SELECT query using the **GROUP BY** clause:

```
SELECT  
w. "State",  
AVG(w. "Temperature") AS "AvgTemperature",  
COUNT(*) AS NumAccidents  
FROM  
Accident a  
JOIN Weather w ON a. "AccidentID" = w. "AccidentID"  
GROUP BY  
w. "State";
```

Explanation: This query groups the data by the state in the Accident table and calculates the average temperature and number of accidents for each state using the AVG() and COUNT() aggregate functions, respectively. The JOIN operation is used to combine the Accident and Weather tables using the AccidentID foreign key.

Query		Query History	
1	SELECT		
2	w."State",		
3	AVG(w."Temperature") AS "AvgTemperature",		
4	COUNT(*) AS NumAccidents		
5	FROM		
6	Accident a		
7	JOIN Weather w ON a."AccidentID" = w."AccidentID"		
8	GROUP BY		
9	w."State";		
10			

Data Output		Messages	Notifications
	State text	AvgTemperature double precision	numaccidents bigint
1	AL	65.87892271662763	427
2	AR	63.06936090225564	532
3	AZ	81.76778004998864	4401
4	CA	63.8958476321208	2914
5	CO	53.043959731543616	1788
6	CT	51.187587006960555	862
Total rows: 49 of 49		Query complete 00:00:00.261	

(iii) Here's an example query using just two tables and demonstrating **ORDER BY**:

```
SELECT Accident."Start_Time", Accident."End_Time", Weather."Temperature"
FROM Accident
INNER JOIN Weather ON Accident."AccidentID" = Weather."AccidentID"
WHERE Accident."Severity" >= 2 AND Weather."Precipitation" > 0.5
ORDER BY Accident."Start_Time" DESC;
```

This query selects the start and end times of accidents with a severity level of at least 2, and the temperature data from the corresponding weather reports where precipitation is greater than 0.5 inches. The results are then ordered by the start time of the accidents in descending order.

Query		Query History	
1	SELECT Accident."Start_Time", Accident."End_Time", Weather."Temperature"		
2	FROM Accident		
3	INNER JOIN Weather ON Accident."AccidentID" = Weather."AccidentID"		
4	WHERE Accident."Severity" >= 2 AND Weather."Precipitation" > 0.5		
5	ORDER BY Accident."Start_Time" DESC;		
6			

Data Output		Messages	Notifications
	Start_Time text	End_Time text	Temperature double precision
1	2021-10-29 09:59:00	2021-10-29 12:04:09	59
2	2021-09-17 18:27:00	2021-09-17 19:58:29	74
3	2021-08-26 14:03:00	2021-08-26 22:44:44	71
4	2021-06-21 17:02:54	2021-06-21 19:46:42	68
5	2021-06-10 18:06:32	2021-06-10 20:47:44	76
6	2021-03-31 16:32:30	2021-03-31 18:38:23	57
Total rows: 31 of 31		Query complete 00:00:00.203	

(iv) Here's an example of a SELECT query that uses a **subquery** to find the maximum severity level for each city:

```
SELECT
    "City",
    MAX("Severity") as "Max_Severity"
FROM
    Accident
    JOIN Weather ON Accident."AccidentID" = Weather."AccidentID"
WHERE
    "City" IN (
        SELECT DISTINCT
            "City"
        FROM
            Weather
    )
GROUP BY "City";
```

Explanation: In this query, the subquery is used to find a distinct list of cities that have weather data. The main query then joins the Accident and Weather tables and filters by these cities. It then uses a GROUP BY clause to group the results by city and find the maximum severity level for each city.

Query		Query History	
1	SELECT		
2	"City",		
3	MAX("Severity") as "Max_Severity"		
4	FROM		
5	Accident		
6	JOIN Weather ON Accident."AccidentID" = Weather."AccidentID"		
7	WHERE		
8	"City" IN (
9	SELECT DISTINCT		
10	"City"		
11	FROM		
12	Weather		
13)		
14	GROUP BY		
15	"City";		
16			
Data Output		Messages	Notifications
	City text		Max_Severity bigint
1	Aaronsburg		3
2	Abbeville		4
Total rows: 1000 of 5616		Query complete 00:00:00.619	

EXECUTION PLAN FOR COMPLEX QUERIES:

The following are the three problematic queries executed. We applied indexing on these complex queries and the results were optimized. In PostgreSQL, EXPLAIN is a command used to obtain a query plan for a specified SQL statement. The output of this EXPLAIN command provides details about how the PostgreSQL planner intends to execute the query. This includes the type of join being used, the order in which tables are accessed, the estimated number of rows to be scanned, and other relevant statistics.

The output of the Explain is the total estimated cost for all the operations in a query. The cost is an estimate of the amount of resources that would be required to perform each step in the execution plan. The lower the cost, the faster the query will execute and better the results. We have used indexing to reduce this estimated time and for quicker results.

First complex query:

```
SELECT A. "AccidentID", A. "Severity", W. "Temperature", F. "Amenity"  
FROM Accident A  
INNER JOIN Weather W ON A. "AccidentID" = W. "AccidentID"  
LEFT JOIN Feature F ON A. "AccidentID" = F. "AccidentID"  
WHERE A. "Start_Time" BETWEEN '2017-01-01 00:00:00' AND '2017-01-31 23:59:59'  
AND W. "Country" = 'US'
```

Cost:

Graphical

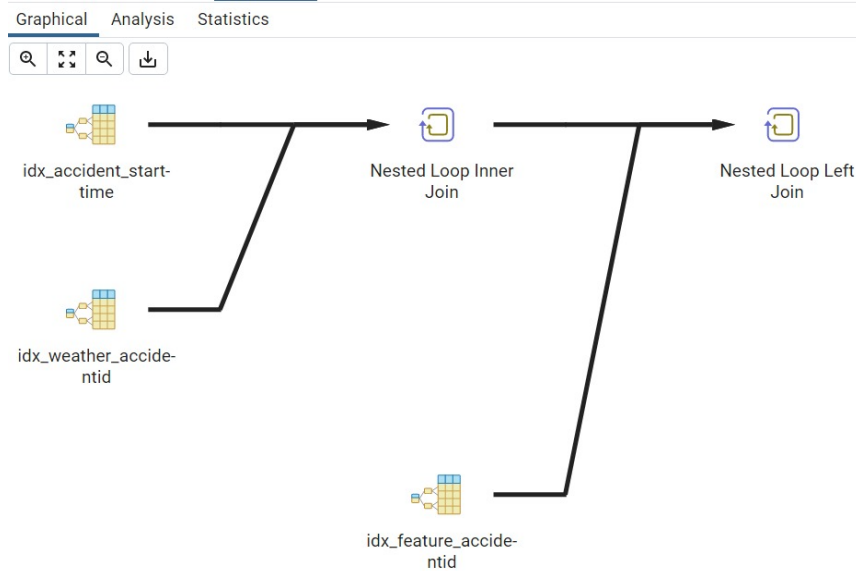
Analysis

Statistics

Statistics per Node Type	
Node type	Count
Index Scan	3
Nested Loop Inner Join	1
Nested Loop Left Join	1

Statistics per Relation	
Relation name	Scan count
Node type	Count
Accident	1
Index Scan	1
Feature	1
Index Scan	1
Weather	1
Index Scan	1

Execution plan:



Second complex query:

```
SELECT
  w. "State",
  AVG(w. "Temperature") AS "AvgTemperature",
  COUNT(*) AS NumAccidents
FROM
  Accident a
  JOIN Weather w ON a. "AccidentID" = w. "AccidentID"
GROUP BY
  w. "State";
```

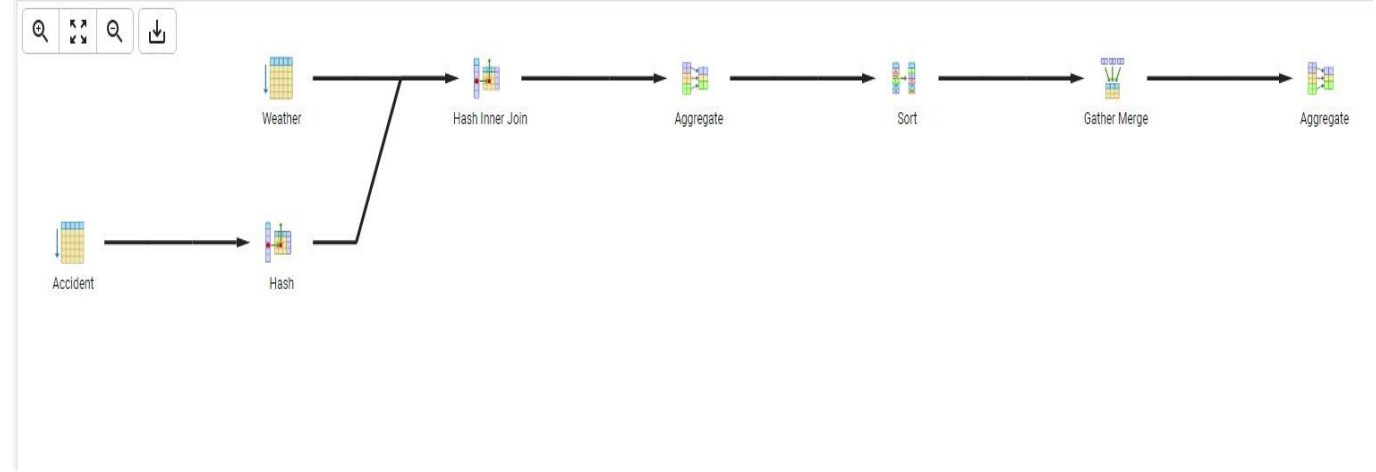
Cost:

Statistics per Node Type	
Node type	Count
Aggregate	2
Gather Merge	1
Hash	1
Hash Inner Join	1
Seq Scan	2
Sort	1

Statistics per Relation	
Relation name	Scan count
Node type	Count
Accident	1
Seq Scan	1
Weather	1
Seq Scan	1

Total rows: 1 of 1 Query complete 00:00:00.050 Ln 3, Col 43

Execution plan:



Third complex query:

```
SELECT Accident.“Start_Time”, Accident.“End_Time”, Weather.“Temperature”
FROM Accident
INNER JOIN Weather ON Accident.“AccidentID” = Weather.“AccidentID”
WHERE Accident.“Severity” >= 2 AND Weather.“Precipitation” > 0.5
ORDER BY Accident.“Start_Time” DESC;
```

Cost:

Graphical Analysis Statistics

Statistics per Node Type	
Node type	Count
Gather Merge	1
Index Scan	1
Nested Loop Inner Join	1
Seq Scan	1
Sort	1

Statistics per Relation	
Relation name	Scan count
Node type	Count
Accident	1
Index Scan	1
Weather	1
Seq Scan	1

Execution plan:

Query Query History Scratch Pad

```
1 SELECT accidenttable."Start_Time", accidenttable."End_Time", weathertable."Temperature"
2 FROM accidenttable
3 INNER JOIN weathertable ON accidenttable."AccidentID" = weathertable."AccidentID"
4 WHERE accidenttable."Severity" >= '2' AND weathertable."Precipitation" > '0.5'
5 ORDER BY accidenttable."Start_Time" DESC
```

Data Output Messages Explain x Notifications

Graphical Analysis Statistics

weather table
idx_accident_accidentid

Nested Loop Inner Join

Sort

Gather Merge

Total rows: 1 of 1 Query complete 00:00:00.043 Ln 5, Col 41

USER INTERFACE:

PostgreSQL UI

AccidentID:

Severity: <input type="text"/>	Number: <input type="text"/>	Amenity: <input type="text"/>	Start_Lat: <input type="text"/>	End_Lat: <input type="text"/>	Sunrise_Sunset: <input type="text"/>
Start_Time: <input type="text"/>	Street: <input type="text"/>	Bump: <input type="text"/>	Start_Lng: <input type="text"/>	End_Lng: <input type="text"/>	Civil_Twilight: <input type="text"/>
End_Time: <input type="text"/>	Side: <input type="text"/>	Crossing: <input type="text"/>	StartID: <input type="text"/>	EndID: <input type="text"/>	Astronomical_Twilight: <input type="text"/>
Distance: <input type="text"/>	City: <input type="text"/>	Give_Way: <input type="text"/>			TID: <input type="text"/>
Description: <input type="text"/>	County: <input type="text"/>	Junction: <input type="text"/>			
	State: <input type="text"/>	No_Exit: <input type="text"/>			
	Zipcode: <input type="text"/>	Railway: <input type="text"/>			
	Country: <input type="text"/>	Roundabout: <input type="text"/>			
	Timezone: <input type="text"/>	Station: <input type="text"/>			
	Airport_Code: <input type="text"/>	Stop: <input type="text"/>			
	Weather_Timestamp: <input type="text"/>	Traffic_Calming: <input type="text"/>			
	Temperature(F): <input type="text"/>	Traffic_Signal: <input type="text"/>			
	Wind_Chill(F): <input type="text"/>	Turning_Loop: <input type="text"/>			
	Humidity(%): <input type="text"/>	FeatureID: <input type="text"/>			
	Pressure(in): <input type="text"/>				
	Visibility(mi): <input type="text"/>				
	Wind_Direction: <input type="text"/>				
	Wind_Speed(mph): <input type="text"/>				
	Precipitation(in): <input type="text"/>				
	Weather_Condition: <input type="text"/>				
	WeatherID: <input type="text"/>				

This User Interface is developed in python with the help of psycopg2 library. Since all the tables are linked to AccidentID, this is written in the top at the center of the screen. Every coloumn represents the attributes of different tables. Since we have a total of 6 tables, there are 6 columns along with their text boxes.

This UI is very user friendly as it produces output for every operation.

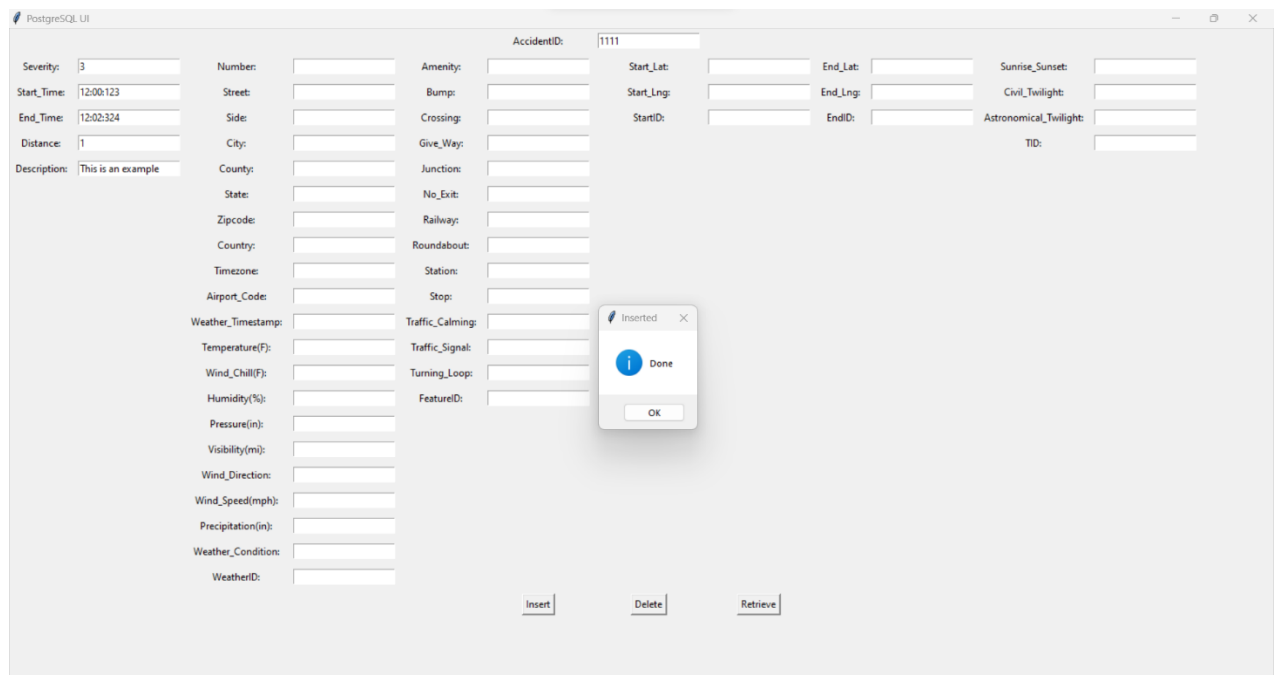
The user can perform the following 3 operations using the UI, namely,

- **Insertion**
- **Deletion**
- **Retrieval**

INSERTION:

The user can insert values into the database using the UI. For every successful insertion operation performed, the UI produces a pop up. Whereas, for every failed insertion operation, there is a pop up which states why the insertion operation failed.

Successful Insertion:



The screenshot displays the PostgreSQL UI interface. At the top, the 'AccidentID' field is populated with '1111'. Below this, there are several input fields for various attributes, including Severity, Start_Time, End_Time, Distance, Description, Number, Streets, Side, City, County, State, Zipcode, Country, Timezone, Airport_Code, Weather_Timestamp, Temperature(F), Wind_Chill(F), Humidity(%), Pressure(in), Visibility(mi), Wind_Direction, Wind_Speed(mph), Precipitation(in), Weather_Condition, and WeatherID. A small pop-up window titled 'Inserted' with a blue information icon and the text 'Done' is visible in the center, indicating a successful operation. At the bottom of the interface, there are three buttons: 'Insert', 'Delete', and 'Retrieve'.

Unsuccessful Insertion:

The screenshot shows the PostgreSQL UI interface. The 'AccidentID' field is set to 'A-1'. An error dialog box is displayed in the center, stating: 'duplicate key value violates unique constraint "accidentable_pkey" DETAIL: Key ("AccidentID")=(A-1) already exists.' The dialog has an 'OK' button. The background form has various input fields for accident details, including Severity, Start/End Times, Distance, Description, Location (Street, City, County, State, Zipcode, Country, Timezone, Airport_Code), Weather (Timestamp, Temperature, Wind, Humidity, Pressure, Visibility, Wind Direction, Wind Speed, Precipitation, Condition), and Features (Amenity, Bump, Crossing, Give_Way, Junction, No_Exit, Railway, Roundabout, Station, Stop, Traffic_Calming, Traffic_Signal, Turning_Loop, FeatureID). At the bottom, there are 'Insert', 'Delete', and 'Retrieve' buttons.

DELETION:

The user can delete values from the database using the UI. For every successful deletion operation performed, the UI produces a pop up. Whereas, for every failed deletion operation, there is a pop up which states why the deletion operation failed.

Successful Deletion:

The screenshot shows the PostgreSQL UI interface. The 'AccidentID' field is set to 'AA-1'. The 'Description' field is set to 'Delete'. A 'Deleted' dialog box is displayed in the center, stating: 'Done'. The dialog has an 'OK' button. The background form has various input fields for accident details, including Severity, Start/End Times, Distance, Description, Location (Street, City, County, State, Zipcode, Country, Timezone, Airport_Code), Weather (Timestamp, Temperature, Wind, Humidity, Pressure, Visibility, Wind Direction, Wind Speed, Precipitation, Condition), and Features (Amenity, Bump, Crossing, Give_Way, Junction, No_Exit, Railway, Roundabout, Station, Stop, Traffic_Calming, Traffic_Signal, Turning_Loop, FeatureID). At the bottom, there are 'Insert', 'Delete', and 'Retrieve' buttons.

Unsuccessful Deletion:

The screenshot shows the PostgreSQL UI interface. The 'AccidentID' field is set to 'AAA-1'. The 'Delete' button is highlighted. An 'Error' dialog box is displayed in the center, indicating 'Not Found'. The dialog box has a blue information icon and an 'OK' button.

PostgreSQL UI

AccidentID: AAA-1

Severity: 2 Start_Time: 12:12:43 End_Time: 01:22:41 Distance: 1 Description: Delete

Number: Street: Side: City: County: State: Zipcode: Country: Timezone: Airport_Code: Weather_Timestamp: Temperature(F): Wind_Chill(F): Humidity(%): Pressure(in): Visibility(mi): Wind_Direction: Wind_Speed(mph): Precipitation(in): Weather_Condition: WeatherID:

Amenity: Bump: Crossing: Give_Way: Junction: No_Exit: Railway: Roundabout: Station: Stop: Traffic_Calming: Traffic_Signal: Turning_Loop: FeatureID:

Start_Lat: Start_Lng: StartID: End_Lat: End_Lng: EndID: Sunrise_Sunset: Civil_Twilight: Astronomical_Twilight: TID:

Insert Delete Retrieve

Error

Not Found

OK

RETRIEVAL:

The user can retrieve values from the database using the UI. For every successful retrieval operation performed, the UI produces a pop up. Whereas, for every failed retrieval operation, there is a pop up which states why the retrieval operation failed.

Successful retrieval:

The screenshot shows the PostgreSQL UI interface. The 'AccidentID' field is set to 'AA-1'. The 'Retrieve' button is highlighted. A 'Data' dialog box is displayed in the center, showing the retrieved data for 'AA-1'. The dialog box has a blue information icon and an 'OK' button.

PostgreSQL UI

AccidentID: AA-1

Severity: 2 Start_Time: 12:12:43 End_Time: 01:22:41 Distance: 1 Description: Delete

Number: Street: Side: City: County: State: Zipcode: Country: Timezone: Airport_Code: Weather_Timestamp: Temperature(F): Wind_Chill(F): Humidity(%): Pressure(in): Visibility(mi): Wind_Direction: Wind_Speed(mph): Precipitation(in): Weather_Condition: WeatherID:

Amenity: Bump: Crossing: Give_Way: Junction: No_Exit: Railway: Roundabout: Station: Stop: Traffic_Calming: Traffic_Signal: Turning_Loop: FeatureID:

Start_Lat: Start_Lng: StartID: End_Lat: End_Lng: EndID: Sunrise_Sunset: Civil_Twilight: Astronomical_Twilight: TID:

Insert Delete Retrieve

Data

ID: AA-1
Severity: 2
Start Time: 12:12:43
End Time: 01:22:41
Distance: 1
Description: Delete

OK

Unsuccessful retrieval:

The screenshot shows the PostgreSQL UI interface. On the left, there are input fields for various accident details: Severity (2), Start_Time (12:12:43), End_Time (01:22:41), Distance (1), and Description (Delete). The main area contains a grid of input fields for accident details, including AccidentID (AAA-1), Start_Lat, End_Lat, Start_Lng, End_Lng, StartID, EndID, Sunrise_Sunset, Civil_Twilight, Astronomical_Twilight, TID, and many others. An error dialog box is centered on the screen, displaying a blue information icon and the text 'Not Found' with an 'OK' button.

All the insertion and deletion operations are reflected in the backend i.e. postgresql. This python code is connected to the POSTGRESQL server with the help of connection.

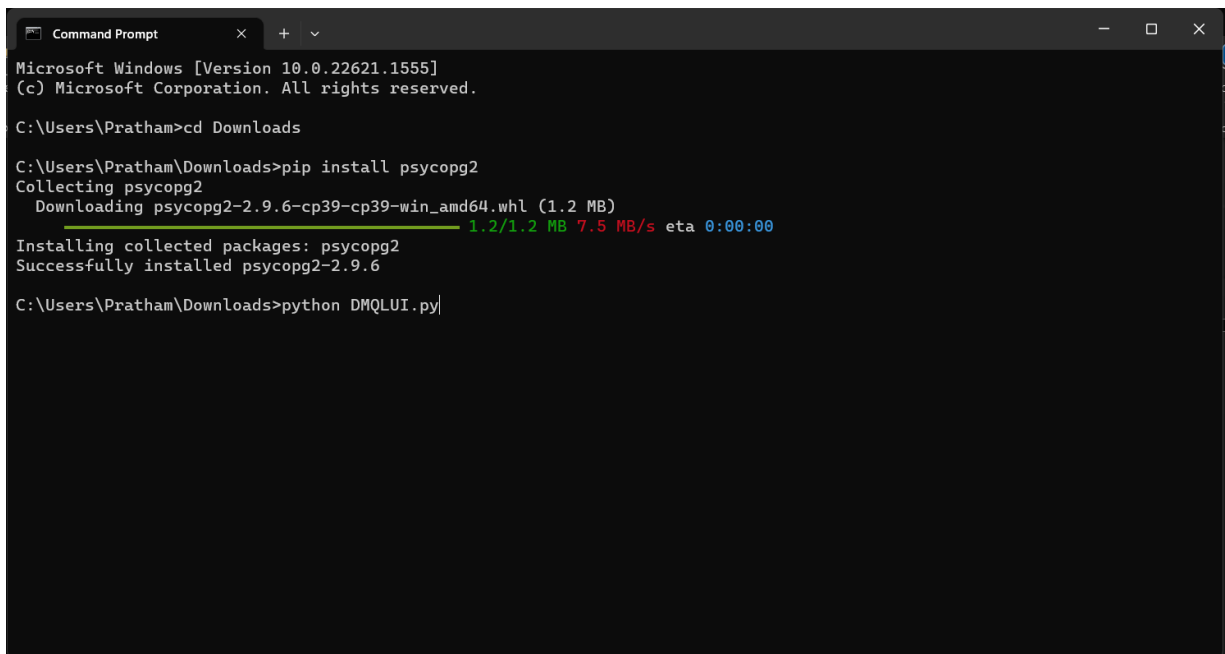
The Code snipped is shown below for the same:

```
2 from tkinter import *
3
4 # Connect to the PostgreSQL database
5 conn = psycopg2.connect(
6     host="localhost",
7     database="AccidentsDB",
8     user="postgres",
9     password="3016"
10 )
11
12 In [4]:
13 # Open a cursor to perform database operations
14 cur = conn.cursor()
15
16 # Execute the SQL command to retrieve a List of tables
17 cur.execute("SELECT table_name FROM information_schema.tables WHERE table_schema='public' AND table_type = 'BASE TABLE'")
18 # Fetch all the rows in a list of lists
19 table_list = cur.fetchall()
20
21 # Print the List of tables
22 for table in table_list:
23     print(table[0])
24
25 Accident
26 Weather
27 Location
28 Feature
29 EndLocation
30 EntireDB
31 TwilightPhase
32 StartLocation
33
34 In [5]:
35 1 print(cur)
```

This UI can be executed locally with the help of command prompt. This is the easiest way of locally running a UI.

To execute this UI, it should be noted that the POSTGRESQL is connected successfully.

We then open the command prompt traverse to the directory where the DMQLUI.py file is saved and then we type “python DMQLUI.py” and press enter. This way the UI executes. Brief steps for the same are mentioned in the read.txt file.



```
Command Prompt
Microsoft Windows [Version 10.0.22621.1555]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Pratham>cd Downloads

C:\Users\Pratham\Downloads>pip install psycopg2
Collecting psycopg2
  Downloading psycopg2-2.9.6-cp39-cp39-win_amd64.whl (1.2 MB)
    1.2/1.2 MB 7.5 MB/s eta 0:00:00
Installing collected packages: psycopg2
Successfully installed psycopg2-2.9.6

C:\Users\Pratham\Downloads>python DMQLUI.py|
```

[*Link to find all the necessary files for the project:*](#)

[*https://github.com/Prathamkakade/DMQL-project-Accident*](https://github.com/Prathamkakade/DMQL-project-Accident)

REFERENCE

Dataset: <https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents>

Papers:

- Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, and Rajiv Ramnath. "A Countrywide Traffic Accident Dataset.", 2019.
- Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, Radu Teodorescu, and Rajiv Ramnath. "Accident Risk Prediction based on Heterogeneous Sparse Data: New Dataset and Insights." , 2019.

Other References:

- <https://learn.g2.com/data-preprocessing#:~:text=The%20four%20stages%20of%20data%20preprocessing%201%201.,3.%20Data%20reduction%20...%204%204.%20Data%20transformation>
- <https://www.postgresqltutorial.com/postgresql-python/connect/>
- <https://www.geeksforgeeks.org/sql-using-python/>