

CSE4/560 Project Milestone 2

Road Safety Analysis

Group Members:

| Name | UB ID |
|----------------------|--------------|
| Kaviyaa Vasudevan | 50443082 |
| Indu Raju | 50441491 |
| Rakshokini Umasankar | 50442672 |

Problem Statement

- By examining the characteristics of the accident information, the number of big accidents is to be decreased.
- There may be one or more specific causes, such as poor lighting or hazardous road conditions, that contribute to an increase in accidents.
- To pinpoint the precise circumstances that cause the majority of accidents, we will evaluate the dataset.
- The main difference in tackling these problems comes up due to the difference between a Database Management System and excel data files.

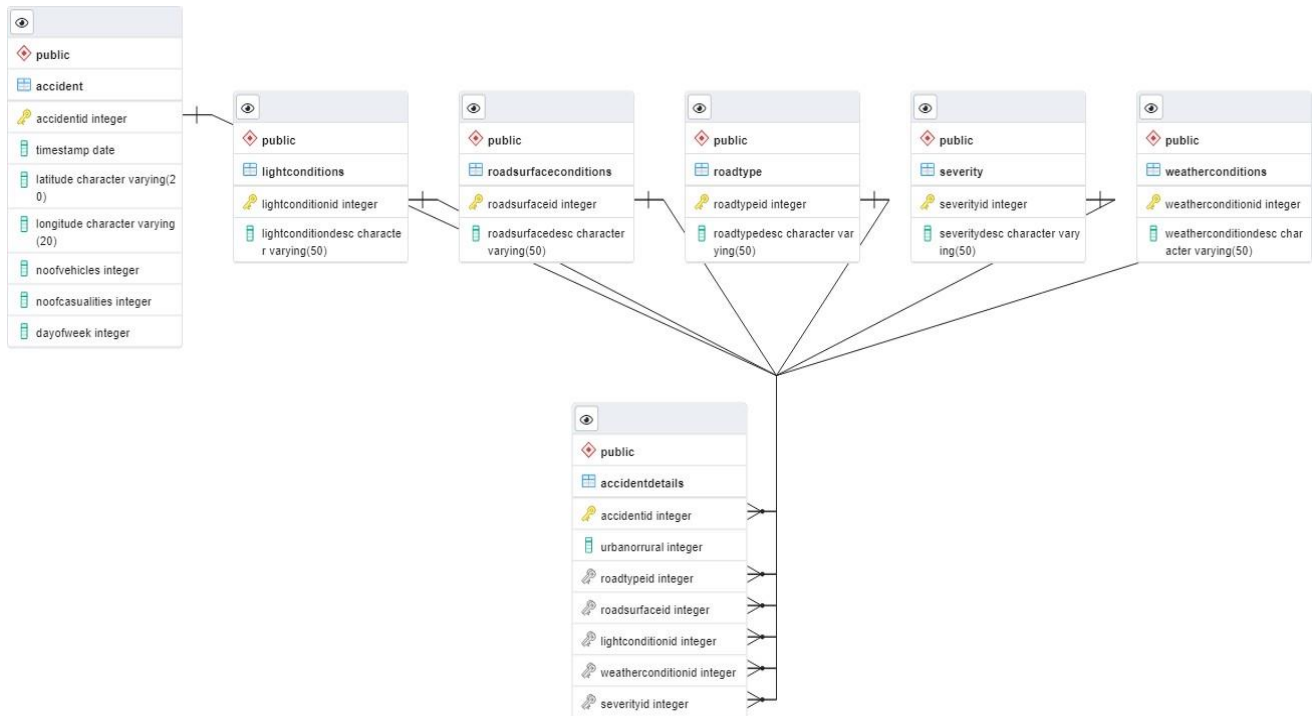
Advantages of database over excel

- In an excel file, it's very difficult to fetch the required data as there will be a lot of spreadsheets. We cannot
- Easily filter the data which is a time-consuming task.
- An excel spreadsheet is simply an electronic graph with rows and columns containing data, with which it is possible to organize and select data.
- Whereas a diverse Database Management System holds a collection of interrelated data and helps link and organize data into tables and relations, which can draw significant results.

Target User

- The target user for this project would typically be a road safety department monitoring specialist. A higher official such as the police department would typically monitor and authorize these databases to maintain data security and authenticity.

E/R Diagram



Database Implementation

1. Data Schemas:

We have created 7 tables called RoadType, Severity, RoadSurfaceConditions, LightConditions, Accident, WeatherConditions and AccidentDetails

```

CREATE TABLE RoadType
(RoadTypeID Integer NOT NULL PRIMARY KEY,
RoadTypeDesc varchar(50) NOT NULL)
  
```

Fig 1.1 Roadtype table

```

CREATE TABLE RoadSurfaceConditions
(RoadSurfaceID Integer NOT NULL PRIMARY KEY,
RoadSurfaceDesc varchar(50) NOT NULL)
  
```

Fig 1.2 RoadSurfaceConditions table

```

CREATE TABLE LightConditions
(LightConditionID Integer NOT NULL PRIMARY KEY,
LightConditionDesc varchar(50) NOT NULL)
  
```

Fig 1.3 LightConditions table

```
CREATE TABLE Severity
(SeverityID Integer NOT NULL PRIMARY KEY,
SeverityDesc varchar(50) NOT NULL)
```

Fig 1.4 Severity table

```
CREATE TABLE Accident
(AccidentID Integer NOT NULL PRIMARY KEY,
TimeStamp DATETIME, Latitude varchar(20),
Longitude varchar(20),
NoOfVehicles Integer,
NoOfCasualties Integer,
DayOfWeek Integer)
```

Fig 1.5 Accident table

```
CREATE TABLE WeatherConditions
(WeatherConditionID Integer NOT NULL PRIMARY KEY,
WeatherConditionDesc varchar(50) NOT NULL);
```

Fig 1.6 WeatherConditions table

```
CREATE TABLE AccidentDetails
(AccidentID Integer NOT NULL PRIMARY KEY,
UrbanOrRural Integer NOT NULL,
RoadTypeID Integer NOT NULL,
RoadSurfaceID Integer NOT NULL,
LightConditionID Integer NOT NULL,
WeatherConditionID Integer NOT NULL,
SeverityID Integer NOT NULL,
FOREIGN KEY(AccidentID) REFERENCES Accident(AccidentID) ON DELETE CASCADE,
FOREIGN KEY(RoadTypeID) REFERENCES RoadType(RoadTypeID) ON DELETE CASCADE,
FOREIGN KEY(RoadSurfaceID) REFERENCES RoadSurfaceConditions(RoadSurfaceID) ON DELETE CASCADE,
FOREIGN KEY(LightConditionID) REFERENCES LightConditions(LightConditionID) ON DELETE CASCADE,
FOREIGN KEY(WeatherConditionID) REFERENCES WeatherConditions(WeatherConditionID) ON DELETE CASCADE,
FOREIGN KEY(SeverityID) REFERENCES Severity(SeverityID) ON DELETE CASCADE);
```

Fig 1.7 AccidentDetails table

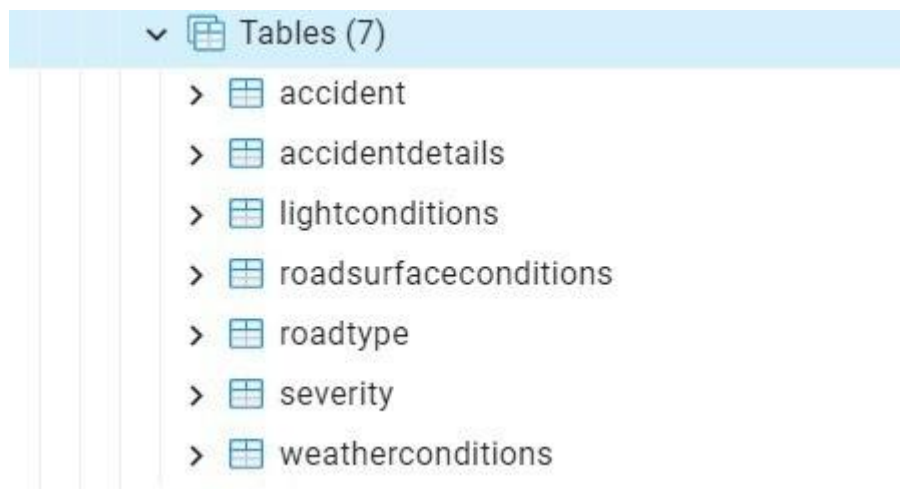


Fig 1.8 Database Schema

- In the AccidentDetails table AccidentID attribute is the foreign key which references the primary key of the Accident table
- RoadTypeID attribute is the foreign key which references the primary key of the RoadType table
- RoadSurfaceID attribute is the foreign key which references the primary key of the RoadSurfaceConditions table
- LightConditionID attribute is the foreign key which references the primary key of the LightConditions table
- WeatherConditionID attribute is the foreign key which references the primary key of the WeatherConditions table
- SeverityID attribute is the foreign key which references the primary key of the Severity table.

2. Attribute Description:

We have a diverse set of attributes in our dataset. The attributes are colored according to the tables they belong to, just to allow clear distinction. The primary key attributes are marked in bold to allow easy identification.

| Attribute | Datatype | Description |
|-------------------------|-------------------------|---|
| RoadTypeID | Integer | Number depicts the road type identification number which helps determine the most frequent types of roads which accidents occur on easily as recognized by the police specified road code |
| RoadTypeDesc | Variable Character (50) | Specifies the name of the type of road associated with the unique RoadTypeID |
| RoadSurfaceID | Integer | Unique ID for the description of the surface of the road which the accident occurred on. This would help generalize the roads according to seasonal and external factors which have impacted road surfaces. |
| RoadSurfaceDesc | Variable Character (50) | Specifies the name of the type of road surface condition associated with the unique RoadSurfaceID |
| LightConditionID | Integer | Describes the unique Identification number for the types of daylight/night light on the road on which the accident occurred, which would help categorize when accidents occur the most and how can those situations be avoided by changing the lighting situation on that road. |

| | | |
|--------------------|-------------------------|---|
| LightConditionDesc | Variable Character (50) | Specifies the name of the light condition associated with the unique LightConditionID |
| SeverityID | Integer | Consists of three numeric categories: Fatal, Serious and Slight; which helps categorize the severity of the intensity of the accident code-wise. |
| SeverityDesc | Variable Character (50) | Specifies the intensity of severity associated with the unique severity level given by SeverityID |
| AccidentID | Integer | When an accident occurs and is recorded by the police, it is uniquely identified in the records with a distinct AccidentID |
| Timestamp | Date | The date when the accident occurred |
| Longitude | Variable Character (20) | The longitude of the location where the accident took place |
| NoOfVehicles | Integer | The latitude of the location where the accident took place |
| NoOfCasualties | Integer | The number of deaths/injuries caused as a result of the particular accident ; the more the number of deaths, the area of focus is higher for the officials to analyze the other factors and take steps to prevent further casualties. |
| DayOfWeek | Integer | Numerical range from 1 to 7 to depict the seven days of the week; the day of the week when the accident occurred |
| UrbanOrRural | Integer | Binary number indicating the nature of the community in which the accident occurred; which would help the officials generalize where accident most frequently occur |
| WeatherConditionID | Integer | The number which indicates the type of weather persistent during the time of the accident; which would alert officials to take more precautionary measures while these weather conditions persist. This would be linked to the RoadType as both are correlated and have |

| | | |
|----------------------|-------------------------|---|
| | | more chances to change simultaneously. |
| WeatherConditionDesc | Variable Character (50) | Specifies the weather condition associated with the unique WeatherConditionID |

3. Records Insertion:

Around 91,199 records are inserted into the Accident and AccidentDetails tables, and roughly 8 – 10 records in the other descriptive tables created above and relations are drawn between the tables.

i. Accident Table:

Since we were attempting to insert a huge number of records(91,199) into our table, we use a python script to import this data from the csv dataset to the tables created in PostgreSQL.

We first connect the local host to the PostGRE SQL Server and then specify the fields we want to insert in. By looping through the various rows and converting our table into a dataframe, we insert all the values sequentially.

ii. RoadType Table:

| | roadtypeid [PK] integer | roadtypedesc character varying (50) |
|---|----------------------------|--|
| 1 | 2 | One way street |
| 2 | 3 | Dual carriageway |
| 3 | 6 | Single carriageway |
| 4 | 7 | Slip road |
| 5 | 9 | Unknown |
| 6 | 12 | One way street/Slip r... |
| 7 | 1 | Roundabout |
| 8 | -1 | Data missing or out o... |

iii. WeatherCondition Table:

| weatherconditionid [PK] integer | weatherconditiondesc character varying (50) |
|------------------------------------|--|
| 1 | Fine no high winds |
| 2 | Raining no high winds |
| 3 | Snowing no high winds |
| 4 | Fine + high winds |
| 5 | Raining + high winds |
| 6 | Snowing + high winds |
| 7 | Fog or mist |
| 8 | Other |
| 9 | Unknown |
| -1 | Data missing or out o... |

iv. Severity Table:

| severityid [PK] integer | severitydesc character varying (50) |
|----------------------------|--|
| 1 | Fatal |
| 2 | Serious |
| 3 | Slight |

v. LightCondition Table:

| lightconditionid [PK] integer | lightconditiondesc character varying (50) |
|----------------------------------|--|
| 1 | Daylight |
| 2 | Darkness - lights lit |
| 3 | Darkness - lights unlit |
| 4 | Darkness - no lighting |
| 5 | Darkness - lighting un... |
| 6 | Data missing or out o... |

vi. RoadSurfaceCondtion Table:

| | roadsurfaceid [PK] integer | roadsurfacedesc character varying (50) |
|---|-------------------------------|---|
| 1 | -1 | Data missing or out o... |
| 2 | 1 | Dry |
| 3 | 2 | Wet or Damp |
| 4 | 3 | Snow |
| 5 | 4 | Frost or Ice |
| 6 | 5 | Flood over 3cm. deep |
| 7 | 6 | Oil or Diesel |
| 8 | 7 | Mud |
| 9 | 9 | unknown |

vii. Accident Table:

Similar to the AccidentDetails table, since we were attempting to insert a huge number of records(91,199) into our table, we use a python script to import this data from the csv dataset to the tables created in PostgreSQL.

We first connect the local host to the PostGRE SQL Server and then specify the fields we want to insert in. By looping through the various rows and converting our table into a dataframe, we insert all the values sequentially.

4. Experimental Queries

- We execute a query to find the number of casualties that occur in urban settings. The total result is about 9 casualties for the data we have entered so far in our tables.

```
--select number of casulties in urban area
select sum(noofcasualities) from accident a where a.accidentid
in (select accidentid from accidentdetails b where b.urbanorrural=1 )
```



| | sum bigint |
|---|---------------|
| 1 | 9 |

- We then execute a query to find the number of casualties that occur based on another predictor concept, road type. This is done by joining the tables Accident and AccidentDetails. We sort the columns in ascending order of number of casualties to understand the seriousness of the accident and the fatal damage caused. It Is noted that most accidents occurred on Slip Roads.


```
--Number of Casualties based on RoadType, sorted by the Number of Casualties
select sum(NoOfCasualties), r.RoadTypeDesc from Accident a inner JOIN AccidentDetails ad on
a.AccidentID = ad.AccidentID inner join RoadType r on ad.RoadTypeID = r.RoadTypeID group by r.RoadTypeID
HAVING r.RoadTypeID NOT IN (-1,9) order by sum(NoOfCasualties) desc;
```

Data output Messages Notifications

| | sum | roadtypedesc |
|---|--------|------------------------|
| | bigint | character varying (50) |
| 1 | 7 | Slip road |
| 2 | 2 | Single carriageway |
| 3 | 2 | Roundabout |

- iii. We calculate a rough estimate of the number of casualties on the 2nd day of a week. This can correlate to the work stress, or mindset and concentration levels of people as the week progresses. Since it is just the second day of the week, we note a less number of casualties and hence less probable severe accidents.

```
--selecting number of casualties on 2nd day of week.
select sum(NoofCasualties) from accident where dayofweek=2
```

Data output Messages Notifications

| | sum |
|---|--------|
| | bigint |
| 1 | 2 |

5. Functional Dependencies

Armstrong's axiom refers to a complete set of rules that is used to test the logical implications of functional dependencies. If f is set of functional dependencies, then the closure of f is denoted as f^+ is the set of all functional dependencies implied by f . It generates the closure of functional dependencies.

- Accident:
Accidentid \rightarrow timestamp, latitude, longitude, noofvehicles, noofcasualties, dayofweek
- Lightconditions:
Lightconditionid \rightarrow lightconditiondesc
- Roadsurfaceconditions:
Roadsurfaceid \rightarrow roadsurfacedesc
- Roadtype:
Roadtypeid \rightarrow roadtypedesc
- Severity:
Severityid \rightarrow severitydesc
- Weatherconditions:
Weatherconditionid \rightarrow weatherconditiondesc
- Accidentdetails:
Accidentid \rightarrow urbanorrural, roadtypeid, roadsurfaceid, lightconditionid, weatherconditionid, severityid

6. Database Normalization

Boyce-Codd Normal Form (BCNF):

A relation is in BCNF if the below conditions are satisfied:

- Table must be in 3NF form: For a functional dependency $A \rightarrow B$, A must be the super key, B is the prime attribute and there are no transitive dependencies. (If $A \rightarrow B$ and $B \rightarrow C$, then the FD $A \rightarrow C$ should not exist)
- For a relation $A \rightarrow B$, if there is a non-trivial functional dependency, A must be the super key and B should not be the subset of A.

i. Accident:

Accidentid \rightarrow timestamp, latitude, longitude, noofvehicles, noofcasualties, dayofweek

The relation is in 3NF as accidentid is the super key and the primary key of the relation and there exists no dependencies with the other non-prime attributes. Hence, there is no transitive dependencies.

Since Accidentid is the only super key in this connection and can be used to deduce all other attributes, this observation serves as evidence that the relation is in BCNF. Due to its sole characteristic and the fact that no other suitable subset can be a super key, Accidentid can also be thought of as a candidate key.

ii. Lightconditions:

Lightconditionid \rightarrow lightconditiondesc

The relation is in 3NF as lightconditionid is the super key and the primary key of the relation and there exists no dependencies with the other non-prime attributes. Hence, there is no transitive dependencies.

Since Lightconditionid is the only super key in this connection and can be used to deduce all other attributes, this observation serves as evidence that the relation is in BCNF. Due to its sole characteristic and the fact that no other suitable subset can be a super key, Lightconditionid can also be thought of as a candidate key.

iii. Roadsurfaceconditions:

Roadsurfaceid \rightarrow roadsurfacedesc

The relation is in 3NF as roadsurfaceid is the super key and the primary key of the relation and there exists no dependencies with the other non-prime attributes. Hence, there is no transitive dependencies.

- a. Since Roadsurfaceid is the only super key in this connection and can be used to deduce all other attributes, this observation serves as evidence that the relation is in BCNF. Due to its sole characteristic and the fact that no other suitable subset can be a super key, Roadsurfaceid can also be thought of as a candidate key.

iv. Roadtype:

Roadtypeid \rightarrow roadtypedesc

The relation is in 3NF as accidentid is the super key and the primary key of the relation and there exists no dependencies with the other non-prime attributes. Hence, there is no transitive dependencies.

Since Roadtypeid is the only super key in this connection and can be used to deduce all other attributes, this observation serves as evidence that the relation is in BCNF. Due to its sole characteristic and the fact that no other suitable subset can be a super key, Roadtypeid can also be thought of as a candidate key.

v. Severity:

Severityid → severitydesc

The relation is in 3NF as accidentid is the super key and the primary key of the relation and there exists no dependencies with the other non-prime attributes. Hence, there is no transitive dependencies.

Since Severityid is the only super key in this connection and can be used to deduce all other attributes, this observation serves as evidence that the relation is in BCNF. Due to its sole characteristic and the fact that no other suitable subset can be a super key, Severityid can also be thought of as a candidate key.

vi. Weatherconditions:

Weatherconditionid → weatherconditiondesc

The relation is in 3NF as accidentid is the super key and the primary key of the relation and there exists no dependencies with the other non-prime attributes. Hence, there is no transitive dependencies.

Since Weatherconditionid is the only super key in this connection and can be used to deduce all other attributes, this observation serves as evidence that the relation is in BCNF. Due to its sole characteristic and the fact that no other suitable subset can be a super key, Weatherconditionid can also be thought of as a candidate key.

vii. Accidentdetails:

Accidentid → urbanorrural, roadtypeid, roadsurfaceid, lightconditionid, weatherconditionid, severityid

The relation is in 3NF as accidentid is the super key and the primary key of the relation and there exists no dependencies with the other non-prime attributes. Hence, there is no transitive dependencies.

Since Accidentid is the only super key in this connection and can be used to deduce all other attributes, this observation serves as evidence that the relation is in BCNF. Due to its sole characteristic and the fact that no other suitable subset can be a super key, Accidentid can also be thought of as a candidate key.

7. Queries

1. Number of Casualties based on RoadType, sorted by the Number of Casualties

This query helps in finding the number of casualties based on RoadType and it is sorted by the number of casualties. By this query we can find which road type is more prone to accidents.

```
select sum(NoOfCasualties), r.RoadTypeDesc from Accident a inner JOIN
AccidentDetails ad on a.AccidentID = ad.AccidentID inner join RoadType r
on ad.RoadTypeID = r.RoadTypeID group by r.RoadTypeID HAVING r.RoadTypeID
NOT IN (-1,9) order by sum(NoOfCasualties) desc;
```

| Data output | | Messages |
|-------------|----------------------|---|
| | | |
| | sum bigint | roadtypedesc character varying (50) |
| 1 | 84553 | Single carriageway |
| 2 | 18044 | Dual carriageway |
| 3 | 6558 | Roundabout |
| 4 | 2215 | One way street |
| 5 | 1987 | Slip road |

2. Number of Casualties based on RoadSurface, sorted by the Number of Casualties

This query helps in finding the number of casualties based on RoadSurface and it is sorted by the number of casualties. By this query we can find which road surface is more prone to accidents.

```
select sum(NoOfCasualties), r.RoadSurfaceDesc from Accident a
inner JOIN AccidentDetails ad on a.AccidentID = ad.AccidentID inner join
RoadSurfaceConditions r on ad.RoadSurfaceID = r.RoadSurfaceID group by r.RoadSurfaceID
HAVING r.RoadSurfaceID NOT IN (-1,9) order by sum(NoOfCasualties) desc;
```

| Data output | | Messages |
|-------------|----------------------|--|
| | | |
| | sum bigint | roadsurfacedesc character varying (50) |
| 1 | 78531 | Dry |
| 2 | 34311 | Wet or Damp |
| 3 | 981 | Frost or Ice |
| 4 | 252 | Flood over 3cm. deep |
| 5 | 203 | Snow |

3. Number of Casualties based on WeatherConditions, sorted by the Number of Casualties

This query helps in finding the number of casualties based on WeatherConditions and it is sorted by the number of casualties. By this query we can find which type of weather causes more number of accidents.

```
select sum(NoOfCasualties), w.WeatherConditionDesc from Accident a
inner JOIN AccidentDetails ad on a.AccidentID = ad.AccidentID inner join
WeatherConditions w on ad.WeatherConditionID = W.WeatherConditionID group by
w.WeatherConditionID
HAVING w.WeatherConditionID NOT IN (-1,8,9) order by sum(NoOfCasualties) desc;
```

| | sum bigint | weatherconditiondesc character varying (50) |
|---|---------------|--|
| 1 | 89518 | Fine no high winds |
| 2 | 14965 | Raining no high winds |
| 3 | 2244 | Raining + high winds |
| 4 | 1824 | Fine + high winds |
| 5 | 687 | Fog or mist |
| 6 | 249 | Snowing no high winds |
| 7 | 96 | Snowing + high winds |

4. Number of Casualties based on LightConditions, sorted by the Number of Casualties

This query helps in finding the number of casualties based on LightConditions and it is sorted by the number of casualties. By this query we can find the lightconditions which causes more number of accidents.

```
select sum(NoOfCasualties), l.LightConditionDesc from Accident a
inner JOIN AccidentDetails ad on a.AccidentID = ad.AccidentID inner join
LightConditions l on ad.LightConditionID = l.LightConditionID group by l.LightConditionID
HAVING l.LightConditionID NOT IN (-1,7) order by sum(NoOfCasualties) desc;
```

| Data output | | Messages |
|-------------|---------------|--|
| | sum bigint | lightconditiondesc character varying (50) |
| 1 | 80986 | Daylight |
| 2 | 24226 | Darkness - lights lit |
| 3 | 6970 | Darkness - no lighting |
| 4 | 843 | Darkness - lights unlit |

5. Number of Fatal Accidents based on Number of Vehicles, sorted by the Number of Accidents

This query helps in finding the number of fatal accidents based on number of vehicles and it is sorted by the number of accidents. By this query we can find the number of vehicles involded when the accidents were severe.

```
select a.NoOfVehicles, count(a.AccidentID) as NoOfAccidents from Accident a
inner join AccidentDetails ad on a.AccidentID = ad.AccidentID inner join
Severity s on ad.SeverityID = s.SeverityID and s.SeverityDesc = 'Fatal'
group by a.NoOfVehicles order by count(a.AccidentID) desc;
```

| Data output | | Messages |
|-------------|-------------------------|-------------------------|
| | noofvehicles integer | noofaccidents bigint |
| 1 | 1 | 662 |
| 2 | 2 | 525 |
| 3 | 3 | 142 |
| 4 | 4 | 41 |
| 5 | 5 | 7 |
| 6 | 6 | 7 |
| 7 | 8 | 4 |
| 8 | 7 | 3 |

6. Number of Accidents based on each Hour, sorted by the Number of Accidents

This query helps in finding the number of accidents based on each Hour, sorted by the number of accidents. By this query we can find the number of accidents that has occurred in each hour of the day.

```
SELECT strftime('%H',TimeStamp) as Hour, COUNT(*) as NoOfAccidents FROM Accident
GROUP BY strftime('%H',TimeStamp) order by COUNT(*) DESC;
```

7. Number of Accidents based on each DayOfWeek, sorted by the Number of Accidents

This query helps in finding the number of accidents based on each DayOfWeek, sorted by the Number of Accidents. By this query we can find the number of accidents that has occurred in each day of the week.

```
SELECT CASE DayOfWeek when '2' THEN 'Monday' when '3' THEN 'Tuesday'
when '4' THEN 'Wednesday' when '5' THEN 'Thursday' when '6' THEN 'Friday'
when '7' THEN 'Saturday' when '1' THEN 'Sunday' END as DayOfWeek, COUNT(*) as
NoOfAccidents
FROM Accident GROUP BY DayOfWeek order by COUNT(*) DESC;
```

| Data output | | Messages |
|-------------|-------------------|-------------------------|
| | dayofweek text | noofaccidents bigint |
| 1 | Friday | 14889 |
| 2 | Thursday | 14056 |
| 3 | Wednesday | 13563 |
| 4 | Tuesday | 13267 |
| 5 | Monday | 12771 |
| 6 | Saturday | 12335 |
| 7 | Sunday | 10315 |

8. Number of Accidents based on each UrbanOrRural, sorted by the Number of Accidents

This query helps in finding the number of accidents based on each UrbanOrRural, sorted by the Number of Accidents. By this query we can find the type of area which is more prone to accidents.

```
select count(*) as AccidentCount, CASE UrbanOrRural when '1' THEN 'Urban' ELSE 'Rural'
END as UrbanOrRural from AccidentDetails where UrbanOrRural != 3 group by
UrbanOrRural order by COUNT(*) DESC;
```

| | accidentcount bigint | urbanorrural text |
|---|-------------------------|----------------------|
| 1 | 61734 | Urban |
| 2 | 29448 | Rural |

9. Timestamp of the most severe accidents

This query helps in finding the Timestamp of the most severe accidents. By this query we can find the exact time when accidents occur frequently.

```
SELECT accident."timestamp", severitydesc FROM Accident
NATURAL JOIN accidentdetails NATURAL JOIN severity
WHERE severitydesc LIKE 'Fatal'
```

| | time1 timestamp without time zone | severitydesc character |
|----|--------------------------------------|---------------------------|
| 1 | 2020-01-01 04:30:00 | Fatal |
| 2 | 2020-01-02 21:00:00 | Fatal |
| 3 | 2020-01-03 06:15:00 | Fatal |
| 4 | 2020-01-08 12:55:00 | Fatal |
| 5 | 2020-01-11 20:56:00 | Fatal |
| 6 | 2020-01-19 15:31:00 | Fatal |
| 7 | 2020-01-21 05:41:00 | Fatal |
| 8 | 2020-01-21 18:24:00 | Fatal |
| 9 | 2020-01-23 21:30:00 | Fatal |
| 10 | 2020-01-26 00:42:00 | Fatal |
| 11 | 2020-01-27 17:45:00 | Fatal |
| 12 | 2020-01-30 06:22:00 | Fatal |
| 13 | 2020-02-19 21:41:00 | Fatal |
| 14 | 2020-02-20 12:35:00 | Fatal |
| 15 | 2020-02-20 13:17:00 | Fatal |
| 16 | 2020-02-22 18:20:00 | Fatal |
| 17 | 2020-02-29 05:13:00 | Fatal |
| 18 | 2020-03-01 13:37:00 | Fatal |
| 19 | 2020-03-09 12:02:00 | Fatal |
| 20 | 2020-03-17 04:20:00 | Fatal |

8. Analysis of SQL Queries (Indexing and Query Optimization)

1. Number of Casualties based on WeatherConditions, sorted by the Number of Casualties

```
explain ANALYZE select sum(NoOfCasualties), w.WeatherConditionDesc from Accident a
inner JOIN AccidentDetails ad on a.AccidentID = ad.AccidentID inner join WeatherConditions w
on ad.WeatherConditionID = W.WeatherConditionID
group by w.WeatherConditionID HAVING w.WeatherConditionID NOT IN (-1,8,9)
```

order by sum(NoOfCasualties) desc;

This query gives a high execution time, and hence we deploy indexing methods to reduce the execution time and optimize this query.

create index id2 on accident(noofcasualties)
create index id3 on accidentdetails(WeatherConditionID)

| | | | |
|------------------------|----------------------------|-----------------------|----------------------------|
| 20 | Planning Time: 3.162 ms | 20 | Planning Time: 3.309 ms |
| 21 | Execution Time: 215.354 ms | 21 | Execution Time: 159.311 ms |
| <i>Before indexing</i> | | <i>After Indexing</i> | |

2. Number of Casualties based on LightConditions, sorted by the Number of Casualties

EXPLAIN analyze select sum(NoOfCasualties), l.LightConditionDesc from Accident a inner JOIN AccidentDetails ad on a.AccidentID = ad.AccidentID inner join LightConditions l on ad.LightConditionID = l.LightConditionID group by l.LightConditionID
HAVING l.LightConditionID NOT IN (-1,7) order by sum(NoOfCasualties) desc;

This query gives a high execution time, and hence we deploy indexing methods to reduce the execution time and optimize query.

create index id5 on accidentdetails(LightConditionID)

| | | | |
|------------------------|----------------------------|-----------------------|----------------------------|
| 20 | Planning Time: 0.450 ms | 20 | Planning Time: 3.585 ms |
| 21 | Execution Time: 228.797 ms | 21 | Execution Time: 181.952 ms |
| <i>Before indexing</i> | | <i>After Indexing</i> | |

3. Number of Fatal Accidents based on Number of Vehicles, sorted by the Number of Accidents

EXPLAIN analyze select a.NoOfVehicles, count(a.AccidentID) as NoOfAccidents from Accident a inner join AccidentDetails ad on a.AccidentID = ad.AccidentID inner join Severity s on ad.SeverityID = s.SeverityID and s.SeverityDesc = 'Fatal'
group by a.NoOfVehicles order by count(a.AccidentID) desc;

This query gives a high execution time, and hence we deploy indexing methods to reduce the execution time and optimize this query.

create index id1 on accident(noofvehicles)
create index id4 on accidentdetails(SeverityID)

| | | | |
|------------------------|---------------------------|-----------------------|---------------------------|
| 18 | Planning Time: 0.345 ms | 18 | Planning Time: 4.540 ms |
| 19 | Execution Time: 17.295 ms | 19 | Execution Time: 15.025 ms |
| <i>Before indexing</i> | | <i>After Indexing</i> | |

9. CRUD Operations

1. CREATE TABLE WeatherConditions(WeatherConditionID Integer NOT NULL PRIMARY KEY, WeatherConditionDesc varchar(50) NOT NULL);
2. CREATE TABLE Severity(SeverityID Integer NOT NULL PRIMARY KEY, SeverityDesc varchar(50) NOT NULL);
3. Deleting a random record, to test trigger
DELETE FROM accident where accidentid=1050
4. Deleting the non-severe accident information
DELETE FROM AccidentDetails WHERE SeverityID = 3;
5. UPDATE weatherconditions SET WeatherConditiondesc = 'safe without high winds' WHERE WeatherConditionID = 1;
6. UPDATE weatherconditions SET WeatherConditiondesc = 'safe with high winds' WHERE WeatherConditionID = 4;

10. ADVANCED SQL Queries

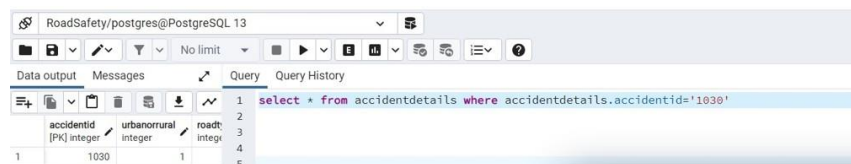
DELETE trigger

Due to the way our tables are linked, the need arises to create a trigger which deletes all records from the Accident table whenever a record(using the AccidentID) is deleted from the AccidentDetails table.

```
CREATE OR REPLACE FUNCTION public.clienteddelete()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    DELETE FROM accidentdetails WHERE accidentdetails.accidentid = OLD.accidentid;
    RETURN OLD;
END
$BODY$;
```

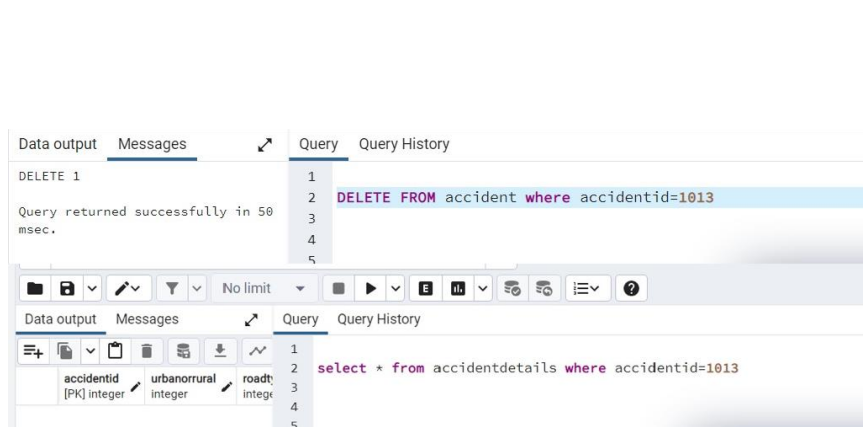
```
ALTER FUNCTION public.clienteddelete()
    OWNER TO postgres;
```

```
CREATE TRIGGER delete_contacto
    BEFORE DELETE
    ON public.accident
    FOR EACH ROW
    EXECUTE FUNCTION public.clienteddelete();
```



The screenshot shows a PostgreSQL query editor interface. The query being executed is: `select * from accidentdetails where accidentdetails.accidentid='1030'`. The results are displayed in a table with 5 columns: `accidentid` (integer), `urbanorural` (integer), `roadt` (integer), `severityid` (integer), and `weatherconditionid` (integer). The results table shows one row with the values: 1, 1030, 1, 3, 4.

| accidentid | urbanorural | roadt | severityid | weatherconditionid |
|------------|-------------|-------|------------|--------------------|
| 1 | 1030 | 1 | 3 | 4 |



Execution of the DELETE Trigger

Contributions

| Name | UB ID | Contributions |
|----------------------|----------|---------------|
| Kaviyaa Vasudevan | 50443082 | 33% |
| Suriya Badrinath | 50442121 | 33% |
| Rakshokini Umasankar | 50442672 | 33% |