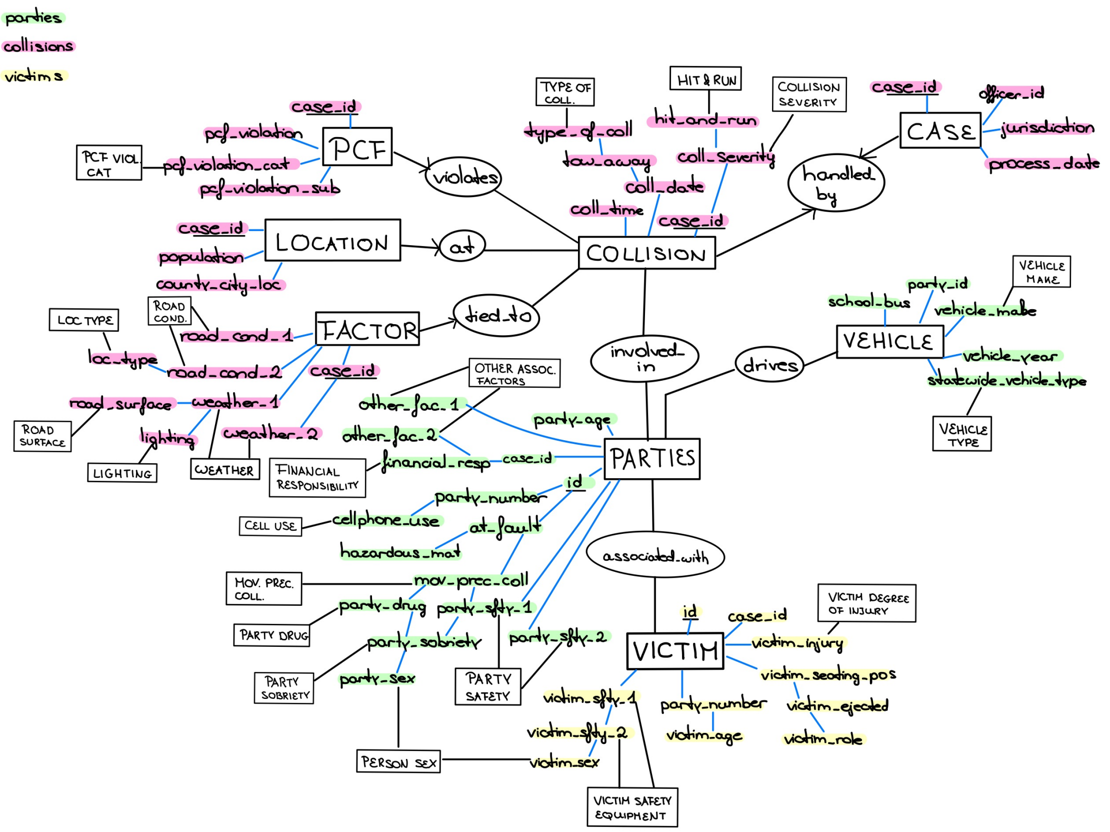
ER Model



Data constraints

Participation constraints

- Partial participation from `Collision` in the `involved in` relationship: `case_id 10` has no involved party.
- 'Exactly one' participation from `Party` in the `involved in` relationship: Every party has an associated `case_id` that is unique.
- Partial participation from `Party` in the 'associated with' relationship: Some `case_id` have no victim e.g. `case_id 1,2`.
- 'Exactly one' participation from `Victim` in the `associated with` relationship: Given a `case_id`, if there is a victim, there is a party (unique).

Additional constraints

- In the project description, each attribute that is nullable is clearly stated:

```
> Blank or - - Not Stated
```

Hence we can mark as `NOT NULL` every other attribute, except for `PRIMARY KEY`s, which are implicitely so.

- Since we are using a star schema, we cannot express the following constraints in the SQL code: every collision has a location, a factor, a pcf and a case. Likewise, every party has a vehicle and a party context. Finally, every victim has a victim context.

- As said in the moodle forum, a victim is not a party.

Design choices

Star schema

We decided to cluster attributes into separate entities following a star schema. Some groups are obvious, others are debatable.

The obvious groups are:

- For the `Collision` entity, some attributes form the logical groups `Pcf`, `Location`, `Factor`, `Case`.
- Similarly, for the `Victim` entity, the only logical group is `Vehicle`.

We also decided to add two less obvious groups:

- For the `Party` entity, attributes which are orthogonal to the collision are not stored in a separate entity (age, sex, ...)

  Attributes which are about the context of the collision are stored in a `PartyContext` entity.

- Similarly for the `Victim` entity, we have a `VictimContext` entity.

Attribute types in the SQL code

- In the project description, some attributes are enums: they can only take on specific pre-defined values. Therefore, we can let them be `INTEGER` and have lookup tables when we dump the `.csv`s into a `SQL` database.

  The alternative would be to let them be `VARCHAR`. The problem with this approach is that determining the max length means looking up the max number of characters for each attribute.

  Granted: creating lookup tables would require the same amount of work; however it leads to substantial data compression.

  Note that these attributes are the same that are `nullable`.

- Similarly, `tow_away` from `Collisions` can be translated from a `float` (`0.0` or `1.0`) to a `BIT`.

- The rest of the attributes are clearly `INTEGER` from the project description as well as upon inspection of the values in the `.csv` files.

SQL code

Collision

Main Tables

```
CREATE TABLE Collisions(case_id NUMERIC(19),
                        collision_date DATE NOT NULL,
                        collision_time TIME,
                        type_of_collision INTEGER,
                        collision_severity INTEGER NOT NULL,
                        hit_and_run INTEGER NOT NULL,
                        tow_away BIT,
                        FOREIGN KEY(type_of_collision) REFERENCES
TypeOfCollision(id),
                        FOREIGN KEY(collision_severity) REFERENCES
CollisionSeverity(id),
                        FOREIGN KEY(hit_and_run) REFERENCES HitAndRun(id),
                        PRIMARY KEY(case_id))
```

```
CREATE TABLE Pcfs(case_id NUMERIC(19) NOT NULL,
                    pcf_violation INTEGER,
                    pcf_violation_category INTEGER,
                    pcf_violation_subsection CHAR(1),
                    FOREIGN KEY(pcf_violation_category) REFERENCES
PcfViolationCategory(id),
                    FOREIGN KEY(case_id) REFERENCES Collisions(case_id))
```

```
CREATE TABLE Locations(case_id NUMERIC(19) NOT NULL,
                        population INTEGER,
                        county_city_location INTEGER NOT NULL,
                        FOREIGN KEY(case_id) REFERENCES
Collisions(case_id))
```

```
CREATE TABLE Factors(case_id NUMERIC(19) NOT NULL,
                        location_type INTEGER,
                        lighting INTEGER,
                        road_condition_1 INTEGER,
                        road_condition_2 INTEGER,
                        road_surface INTEGER,
                        weather_1 INTEGER,
                        weather_2 INTEGER,
                        FOREIGN KEY(location_type) REFERENCES
LocationType(id),
                        FOREIGN KEY(lighting) REFERENCES Lighting(id),
                        FOREIGN KEY(road_condition_1) REFERENCES
RoadCondition(id),
                        FOREIGN KEY(road_condition_2) REFERENCES
```

```
RoadCondition(id),
                        FOREIGN KEY(road_surface) REFERENCES
RoadSurface(id),
                        FOREIGN KEY(weather_1) REFERENCES Weather(id),
                        FOREIGN KEY(weather_2) REFERENCES Weather(id),
                        FOREIGN KEY(case_id) REFERENCES
Collisions(case_id))
```

```
CREATE TABLE Cases(case_id NUMERIC(19) NOT NULL,
                    process_date DATE NOT NULL,
                    officer_id VARCHAR(8),
                    jurisdiction INTEGER,
                    FOREIGN KEY(case_id) REFERENCES Collisions(case_id))
```

Satelite Enum Tables

```
CREATE TABLE CollisionSeverity(id INTEGER AUTO_INCREMENT,
                   description VARCHAR(25) NOT NULL UNIQUE,
                   PRIMARY KEY(id))
```

```
CREATE TABLE HitAndRun(id INTEGER AUTO_INCREMENT,
            description VARCHAR(20) NOT NULL UNIQUE,
            PRIMARY KEY(id))
```

```
CREATE TABLE Lighting(id INTEGER AUTO_INCREMENT,
               description VARCHAR(39) NOT NULL UNIQUE,
               PRIMARY KEY(id))
```

```
CREATE TABLE LocationType(id INTEGER AUTO_INCREMENT,
               description VARCHAR(20) NOT NULL UNIQUE,
               PRIMARY KEY(id))
```

```
CREATE TABLE PcfViolationCategory(id INTEGER AUTO_INCREMENT,
                   description VARCHAR(70) NOT NULL UNIQUE,
                   PRIMARY KEY(id))
```

```
CREATE TABLE PrimaryCollisionFactor(id INTEGER AUTO_INCREMENT,
                     description VARCHAR(25) NOT NULL UNIQUE,
```

```
                        PRIMARY KEY(id))
```

```
CREATE TABLE RoadCondition(id INTEGER AUTO_INCREMENT,
                      description VARCHAR(20) NOT NULL UNIQUE,
                      PRIMARY KEY(id))
```

```
CREATE TABLE RoadSurface(id INTEGER AUTO_INCREMENT,
                    description VARCHAR(15) NOT NULL UNIQUE,
                    PRIMARY KEY(id))
```

```
CREATE TABLE TypeOfCollision(id INTEGER AUTO_INCREMENT,
                        description VARCHAR(15) NOT NULL UNIQUE,
                        PRIMARY KEY(id))
```

```
CREATE TABLE Weather(id INTEGER AUTO_INCREMENT,
                    description VARCHAR(10) NOT NULL UNIQUE,
                    PRIMARY KEY(id))
```

Party

Main Tables

```
CREATE TABLE Parties(id INTEGER,
                        case_id NUMERIC(19) NOT NULL,
                        party_number INTEGER NOT NULL,
                        finanicial_responsibility INTEGER,
                        party_age INTEGER,
                        party_sex INTEGER,
                        at_fault BIT NOT NULL,
                        cellphone_use INTEGER,
                        hazardous_materials CHAR(1),
                        movement_preceding_collision INTEGER,
                        other_associate_factor_1 INTEGER,
                        other_associate_factor_2 INTEGER,
                        party_drug_physical INTEGER,
                        party_safety_equipment_1 INTEGER,
                        party_safety_equipment_2 INTEGER,
                        party_sobriety INTEGER,
                        PRIMARY KEY(id, case_id),
                        FOREIGN KEY(finanicial_responsibility) REFERENCES
FinancialResponsability(id),
                        FOREIGN KEY(party_sex) REFERENCES PersonSex(id),
```

```
                        FOREIGN KEY(cellphone_use) REFERENCES
CellphoneUse(id),
                        FOREIGN KEY(movement_preceding_collision)
REFERENCES MovementPrecedingCollision(id),
                        FOREIGN KEY(other_associate_factor_1) REFERENCES
OtherAssociatedFactors(id),
                        FOREIGN KEY(other_associate_factor_2) REFERENCES
OtherAssociatedFactors(id),
                        FOREIGN KEY(party_drug_physical) REFERENCES
PartyDrugPhysical(id),
                        FOREIGN KEY(party_safety_equipment_1) REFERENCES
PartySafetyEquipement(id),
                        FOREIGN KEY(party_safety_equipment_2) REFERENCES
PartySafetyEquipement(id),
                        FOREIGN KEY(party_sobriety) REFERENCES
PartySobriety(id),
                        FOREIGN KEY(case_id) REFERENCES
Collisions(case_id))
```

```
CREATE TABLE Vehicles(party_id INTEGER NOT NULL,
                        school_bus_related BIT,
                        statewide_vehicle_type INTEGER,
                        vehicle_make INTEGER,
                        vehicle_year INTEGER,
                        FOREIGN KEY(statewide_vehicle_type) REFERENCES
StatewideVehiculeType(id),
                        FOREIGN KEY(vehicle_make) REFERENCES
VehiculeMake(id),
                        FOREIGN KEY(party_id) REFERENCES Parties(id))
```

Satelite Enum Tables

```
CREATE TABLE CellphoneUse(id INTEGER AUTO_INCREMENT,
               description CHAR(1) NOT NULL UNIQUE,
                CHECK (description BETWEEN 'B' AND 'D'),
               PRIMARY KEY(id))
```

```
CREATE TABLE FinancialResponsibility(id INTEGER AUTO_INCREMENT,
                      description CHAR(1) NOT NULL UNIQUE,
                                      CHECK (description IN ('N', 'Y',
'O', 'E')),
                          PRIMARY KEY(id))
```

```
CREATE TABLE MovementPrecedingCollision(id INTEGER AUTO_INCREMENT,
                        description VARCHAR(100) NOT NULL UNIQUE,
                        PRIMARY KEY(id))
```

```
CREATE TABLE OtherAssociatedFactors (id INTEGER AUTO_INCREMENT,
    description CHAR(1) NOT NULL UNIQUE,
    CHECK (description BETWEEN 'A' AND 'Z'),
    PRIMARY KEY(id))
```

```
CREATE TABLE PartyDrugPhysical(id INTEGER AUTO_INCREMENT,
                    description CHAR(1) NOT NULL UNIQUE,
                             CHECK (description IN ('E', 'F', 'H', 'I')),
                    PRIMARY KEY(id))
```

```
CREATE TABLE PartySafetyEquipement(id INTEGER AUTO_INCREMENT,
                    description CHAR(1) NOT NULL UNIQUE,
                                    CHECK (description BETWEEN 'A' AND
'Y'),
                    PRIMARY KEY(id))
```

The following can store `victim_sex` and `party_sex`

```
CREATE TABLE PersonSex(id INTEGER AUTO_INCREMENT,
                description VARCHAR(6) NOT NULL UNIQUE,
                        CHECK (description in ('male', 'female')),
                PRIMARY KEY(id))
```

```
CREATE TABLE PartySobriety(id INTEGER AUTO_INCREMENT,
                 description CHAR(1) NOT NULL UNIQUE,
                             CONSTRAINT sobriety_check CHECK (description
IN ('A', 'B', 'C', 'D', 'G', 'H')),
                 PRIMARY KEY(id))
```

```
CREATE TABLE PartyType(id INTEGER AUTO_INCREMENT,
             description VARCHAR(14) NOT NULL UNIQUE,
             PRIMARY KEY(id))
```

```
CREATE TABLE StatewideVehiculeType(id INTEGER AUTO_INCREMENT,
                    description VARCHAR(35) NOT NULL UNIQUE,
                    PRIMARY KEY(id))
```

```
CREATE TABLE VehiculeMake(id INTEGER AUTO_INCREMENT,
               description VARCHAR(28) NOT NULL,
               PRIMARY KEY(id))
```

Victim

Main Table

```
CREATE TABLE Victims(id INTEGER,
                        case_id NUMERIC(19) NOT NULL,
                        party_number INTEGER NOT NULL,
                        victim_age INTEGER,
                        victim_sex INTEGER,
                        victim_degree_of_injury INTEGER,
                        victim_ejected INTEGER,
                        victim_role INTEGER NOT NULL,
                        victim_safety_equipment_1 INTEGER,
                        victim_safety_equipment_2 INTEGER,
                        victim_seating_position INTEGER,
                        PRIMARY KEY(id, case_id),
                        FOREIGN KEY(victim_sex) REFERENCES PersonSex(id),
                        FOREIGN KEY(victim_safety_equipment_1) REFERENCES
VictimSafetyEquipment(id),
                        FOREIGN KEY(victim_safety_equipment_2) REFERENCES
VictimSafetyEquipment(id),
                        FOREIGN KEY(victim_degree_of_injury) REFERENCES
VictimDegreeOfInjury(id),
                        FOREIGN KEY(case_id) REFERENCES
Collisions(case_id))
```

Satelite Enum Tables

```
CREATE TABLE VictimSafetyEquipment(id INTEGER AUTO_INCREMENT,
                   description CHAR(1) NOT NULL UNIQUE,
                                  CHECK(description BETWEEN 'A' AND 'Z'),
                   PRIMARY KEY(id))
```

```
CREATE TABLE VictimDegreeOfInjury(id INTEGER AUTO_INCREMENT,
                   description VARCHAR(30) NOT NULL UNIQUE,
```

```
                    PRIMARY KEY(id))
```

The following can store `victim_sex` and `party_sex`.

```
CREATE TABLE PersonSex(id INTEGER AUTO_INCREMENT,
              description VARCHAR(6) NOT NULL UNIQUE,
                     CHECK (description in ('male', 'female')),
              PRIMARY KEY(id))
```

Design modifications

Justifications

We made certain design modifications to respond to the feedback that we got after the last deliverable.

> Think about the repeated information stored in rows of your table with your current entity modeling. As a thought experiment, if 1000 accidents took place in the same county_city_location "Lausanne", it would not be efficient to repeatedly store "Lausanne" 1000 times in the rows of our collision table. We'd prefer to store it only once as a single row in a separate location entity, and reference it. Apply this logic to your modeling structure, starting with the feedback below.

Firstly, to avoid having to store certain strings over and over again, we created satellite enum tables to which we reference from within our main entities.

> Consider modeling "other_associated_factors" as a separate entity. Consider modeling road conditions and weather as separate entities.

The satellite table modeling also responded to this point of the feedback by making weather, road conditions and other_associated_factors a separate entity.

To be consistant throughout the columns, we had to model any column like the ones mentioned as separate entities. There are too much of these to mention here.

> There is no strong benefit to creating "Party Contexts" and "Victim Contexts" entities separate from your Party and Victim entities, because all this requires is more joining at query time

We removed the Party Context and Victim Context entities and simply merged their attributes to the ones in the Party and Victim entities.

> Please justify your decision to model safety equipment-related information as fields of victims and party instead of migrating them to a separate 'Safety Equipment' entity.

We decided to keep safety equipment related attributes inside the Party and Victim entities instead of migrating them to a separate safety equipment entity. This is because since these attributes exist individually for victims and parties, it seemed more logical to us to keep them with those entities rather than creating a new entity that would hold information for different types of participants in the collision.

> Strong vs. weak entities: consider converting currently-modeled-as-strong, non-uniquely identifiable entities to weak ones. For example, parties can be modeled as a weak entity, as they only exist in the context of a collision.

We noticed that both Victim and Party entities were modeled as strong entities. In order to turn these into weak entities, as suggested, we added a foreign key : the case id.

Disclaimer

Last second we realized that having `DECIMAL(19)` as the type of `case_id` was not suitable.

Although `pandas` indicated `float64` as the type for `case_id`, it seems like there are leading `0` for some values of this column in `collisions.csv`.

Therefore in the future we will have to change the `case_id` type to `VARCHAR(M)` where `M` is the longest string representing a `case_id`. We will also need to load the data in a different way, overriding `pandas` assumed type for `case_id`.

This mistake led us to have some collisions dropped as well their associated victims and parties. The queries below will therefore probably yield slightly less reliable values than expected.

How we did the import

The following is implementation details that we inserted for completion.

The database server

We decided to run `MySQL` locally after failing to get acceptable performances on AWS/GCP.

The method to dump data

- After having declared every table, we simply performed a row-wise import, using `pd.iterrows` and `INSERT INTO` commands.
- Sattelite tables were populated on the fly.
- `id`s from such Sattelite tables were cached during import for performance.

Data cleaning choices

Collisions

- `Collisions` has three columns with dirty values.
- The columns are `hit_and_run`, `pcf_violation_category` and `road_surface`. The corresponding values are `D`, `21804` and `H`.
- The above anomalies each occur once in the table.

- Droping a collision means dropping the associated parties and victims. In our case, we dropped 1 victim and 2 parties.
- Surprisingly, it was a single row combining the above three anomalies. The corresponding `case_id` was `2816618`.

Parties

- `Parties` has two columns with dirty values.
- In `party_drug_physical` there is an undefined `G` which is converted into a `NULL` in the table. Even if this value is the most prevalant in the table, we decided to replace with `NULL` as we cannot interpret it.
- In `cellphone_use`, undefined values `1`, `2` and `3` are converted into `B`, `C` and `D` respectively. We made this choice because of similar frequency as well as similar order.

Victims

- In the table `Victims`, `victim_degree_of_injury` has an undefined `7` which is replaced by `NULL`. This occurs once throughout the table.

Queries

As asked, we only included the first 20 rows or less.

Query 1

List the year and the number of collisions per year. Suppose there are more years than just 2018.

Command

```
SELECT
  EXTRACT(
    YEAR
    FROM
      c.collision_date
  ) as year,
  COUNT(*) as collisions_count
FROM
  Collisions c
GROUP BY
  EXTRACT(
    YEAR
    FROM
      c.collision_date
  )
```

Result

year	collisions_count
2002	544741
2003	538953
2004	538294
2005	532724
2006	498849
2007	501908
2017	7
2018	21
2001	518985

Query 2

Find the most popular vehicle make in the database. Also list the number of vehicles of that particular make.

Command

```
CREATE VIEW MostPopularVehicle AS (
  SELECT
    COUNT(p.id) as number_of_vehicles,
    v.vehicle_make as brand_id
  FROM
    Parties p,
    Vehicles v
  WHERE
    p.id = v.party_id
  GROUP BY
    v.vehicle_make
  ORDER BY
    number_of_vehicles DESC
  LIMIT
    1
)

SELECT
  vm.description as brand,
  vp.number_of_vehicles as vehicle_count
FROM
  VehiculeMake vm,
  MostPopularVehicle vp
WHERE
  vm.id = 1
```

Result

brand	vehicle_count
FORD	1129719

Query 3

Find the fraction of total collisions that happened under dark lighting conditions.

Command

```
SELECT
  (
    SELECT
      COUNT(*)
    FROM
      (
        SELECT
          c.case_id
        FROM
          Collisions c,
          Factors f,
          Lighting l
        WHERE
          c.case_id = f.case_id
          AND f.lighting = l.id
          AND l.description LIKE 'dark%'
      ) AS DarkCols
  ) / (
    SELECT
      COUNT(*)
    FROM
      Collisions
  )
```

Result

fraction
0.2802

Query 4

Find the number of collisions that have occurred under snowy weather conditions.

Command

```
SELECT
  COUNT(snowy.case_id)
```

```
FROM
  Collisions snowy,
  Factors f,
  Weather w
WHERE
  snowy.case_id = f.case_id
  AND (
    f.weather_1 = w.id
    OR f.weather_2 = w.id
  )
  AND w.description LIKE 'snowing'
```

Result

count
8542

Query 5

Compute the number of collisions per day of the week, and find the day that witnessed the highest number of collisions. List the day along with the number of collisions.

Command

```
SELECT
  DAYOFWEEK(collision_date) as day,
  COUNT(*) as count
FROM
  Collisions
GROUP BY
  day
ORDER BY
  counts desc
LIMIT
  1
```

Result

day	count
6	614143

Query 6

List all weather types and their corresponding number of collisions in descending order of the number of collisions.

Command

```
SELECT
  weather,
  SUM(counts) as count
FROM
  (
    SELECT
      f.weather_1 as weather,
      COUNT(*) as counts
    FROM
      Factors f
    GROUP BY
      f.weather_1
    UNION ALL
    SELECT
      f.weather_2 as weather,
      COUNT(*) as counts
    FROM
      Factors f
    GROUP BY
      f.weather_2
  ) AS WeatherList
GROUP BY
  weather
ORDER BY
  counts DESC
```

Result

weather	count
None	3593584
1	2942023
2	548420
6	223797
3	21289
7	13958
4	8542
5	6965

Query 7

Find the number of at-fault collision parties with financial responsibility and loose material road conditions.

Command

```
SELECT
  COUNT(*)
FROM
  Parties p,
  Factors f
WHERE
  p.at_fault = 1
  AND p.financial_responsibility IN (
    SELECT
      id
    FROM
      FinancialResponsibility
    WHERE
      description = 'Y'
  )
  AND f.case_id = p.case_id
  AND (
    f.road_condition_1 IN (
      SELECT
        id
      FROM
        RoadCondition
      WHERE
        description = 'loose material'
    )
    OR f.road_condition_2 IN (
      SELECT
        id
      FROM
        RoadCondition
      WHERE
        description = 'loose material'
    )
  )
```

Result

count
4818

Query 8

Find the median victim age and the most common victim seating position.

Command

```
CREATE INDEX index_victim_age ON Victims(victim_age)

SET @rowindex := -1;

SELECT
   AVG(d.age) as median
FROM
   (SELECT @rowindex:=@rowindex + 1 AS rowindex,
           v.victim_age AS age
    FROM Victims v
    ORDER BY v.victim_age) AS d
WHERE
d.rowindex IN (FLOOR(@rowindex / 2), CEIL(@rowindex / 2));

SELECT
  victim_seating_position as seating_position
FROM
  Victims
GROUP BY
  victim_seating_position
ORDER BY
  COUNT(*) DESC
LIMIT
  1
```

Result

median
24

and

seating_position
3

Query 9

What is the fraction of all participants that have been victims of collisions while using a belt?

Query

```
SELECT
  (
    (
```

```
        SELECT
          COUNT(V.case_id)
        FROM
          Victims V,
          VictimSafetyEquipment S
        WHERE
          (
            V.victim_safety_equipment_1 = S.id
            OR V.victim_safety_equipment_2 = S.id
          )
          AND S.description IN ('C', 'E', 'G')
      ) / (
        SELECT
          COUNT(*)
        FROM
          Victims
      )
    )
```

Result

fraction
0.7495

Query 10

Compute and the fraction of the collisions happening for each hour of the day (for example, x% at 13, where 13 means period from 13:00 to 13:59). Display the ratio as percentage for all the hours of the day.

Command

```
SELECT
   CASE WHEN collision_time BETWEEN '00:00:00'
   AND '00:59:00' THEN '0' WHEN collision_time BETWEEN '01:00:00'
   AND '01:59:00' THEN '1' WHEN collision_time BETWEEN '02:00:00'
   AND '02:59:00' THEN '2' WHEN collision_time BETWEEN '03:00:00'
   AND '03:59:00' THEN '3' WHEN collision_time BETWEEN '04:00:00'
   AND '04:59:00' THEN '4' WHEN collision_time BETWEEN '05:00:00'
   AND '05:59:00' THEN '5' WHEN collision_time BETWEEN '06:00:00'
   AND '06:59:00' THEN '6' WHEN collision_time BETWEEN '07:00:00'
   AND '07:59:00' THEN '7' WHEN collision_time BETWEEN '08:00:00'
   AND '08:59:00' THEN '8' WHEN collision_time BETWEEN '09:00:00'
   AND '09:59:00' THEN '9' WHEN collision_time BETWEEN '10:00:00'
   AND '10:59:00' THEN '10' WHEN collision_time BETWEEN '11:00:00'
   AND '11:59:00' THEN '11' WHEN collision_time BETWEEN '12:00:00'
   AND '12:59:00' THEN '12' WHEN collision_time BETWEEN '13:00:00'
   AND '13:59:00' THEN '13' WHEN collision_time BETWEEN '14:00:00'
   AND '14:59:00' THEN '14' WHEN collision_time BETWEEN '15:00:00'
```

```
    AND '15:59:00' THEN '15' WHEN collision_time BETWEEN '16:00:00'
    AND '16:59:00' THEN '16' WHEN collision_time BETWEEN '17:00:00'
    AND '17:59:00' THEN '17' WHEN collision_time BETWEEN '18:00:00'
    AND '18:59:00' THEN '18' WHEN collision_time BETWEEN '19:00:00'
    AND '19:59:00' THEN '19' WHEN collision_time BETWEEN '20:00:00'
    AND '20:59:00' THEN '20' WHEN collision_time BETWEEN '21:00:00'
    AND '21:59:00' THEN '21' WHEN collision_time BETWEEN '22:00:00'
    AND '22:59:00' THEN '22' ELSE '23' END AS hour_ranges,
    count(1) / (SELECT COUNT(*) FROM Collisions) as count
  FROM
    Collisions
  GROUP BY
    hour_ranges
```

Result

hour_ranges	count
15	0.0775
19	0.0443
7	0.0517
11	0.0489
17	0.0790
16	0.0733
8	0.0523
6	0.0262
12	0.0578
23	0.0319
22	0.0286
10	0.0423
2	0.0181
14	0.0655
1	0.0183
9	0.0409
18	0.0630
13	0.0578
21	0.0328
20	0.0349