Databases Project - Spring 2021

Team 23
Noureddine Gueddach, Benedek Hauer and Sebastian Velez

May 1, 2021

Deliverable 1

Assumptions

We assumed that each victim is associated with exactly one party and exactly one collision. A party can be associated with multiple victims (if, for example, there were multiple passengers in the vehicle driven by the party), but can only be involved in exactly one collision. We assumed that a collision involves at least a party.

Entity Relationship Schema

1. Schema

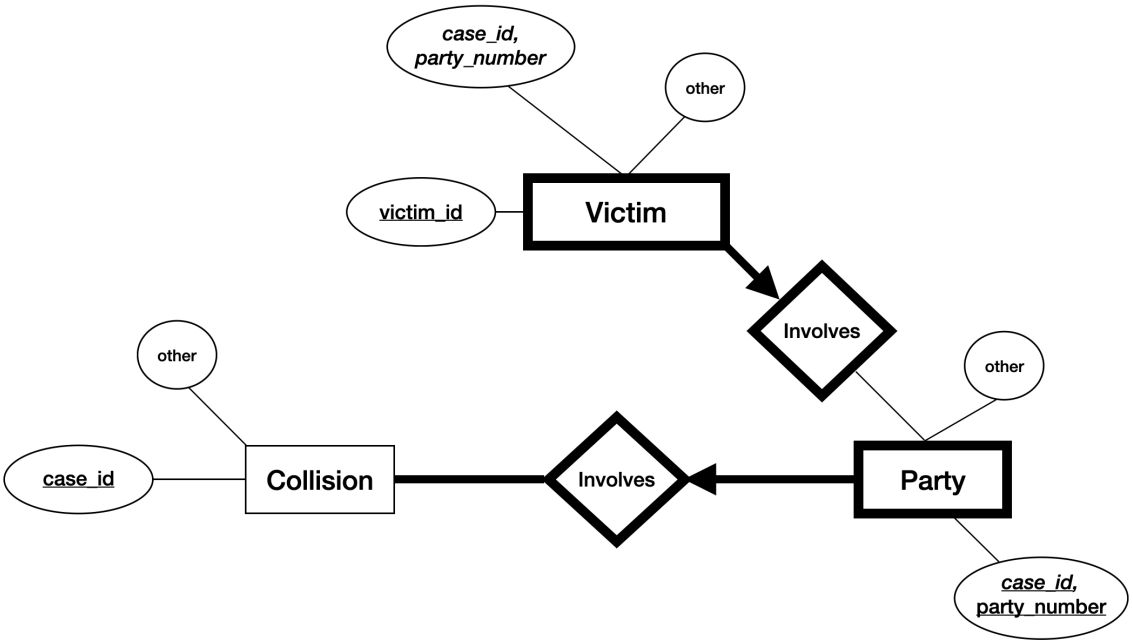

Figure 1: *ER schema, where underlined text corresponds to a primary key, and italic text corresponds a foreign key. The "other" fields references other entities and attributes. They can be found below. We decided to make multiple figures for visualization purposes.*

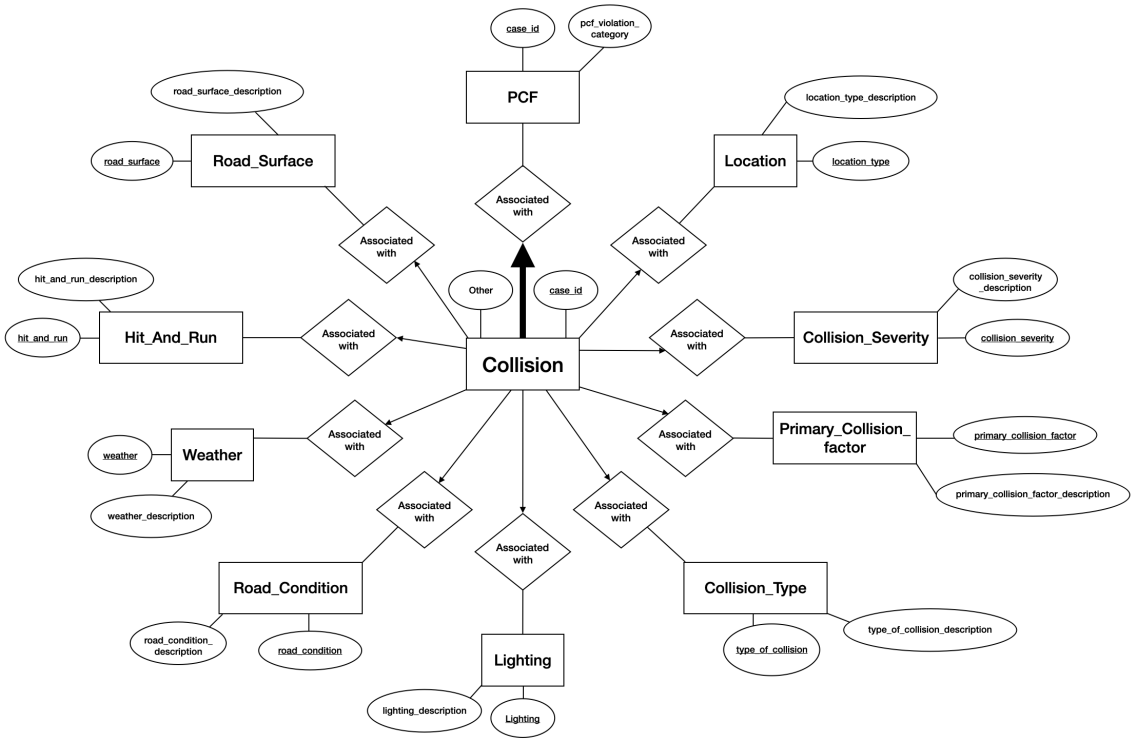

Figure 2: *Collisions and other entities.*

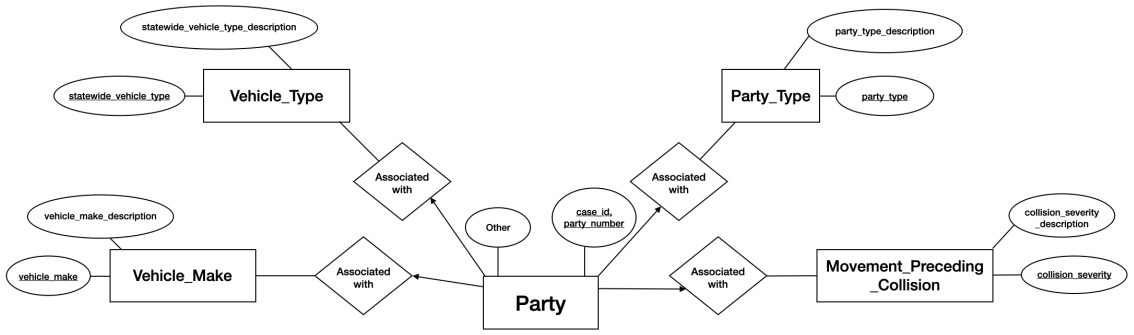

Figure 3: *Parties and other entities.*

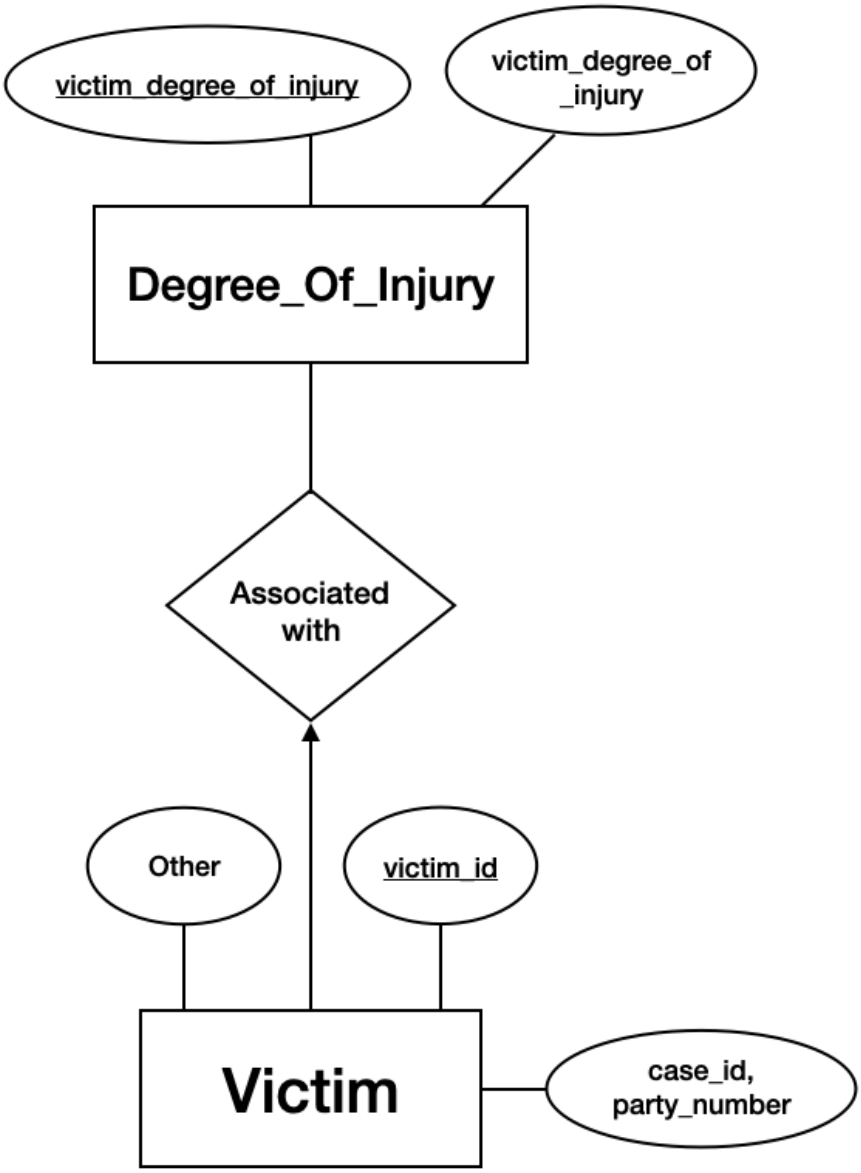

Figure 4: *Victims and other entities.*

2. Description :

At first, we thought each person was uniquely identifiable (which would correspond to victim_id and party_id), by a social security number or any other unique id, and thus, that the same person could be involved in multiple collisions. We had decided to have a party table, which only contained a way to identify a person. From there, party was separated in two tables in ISA relationships: a party could be an active party (a driver for example) and/or a victim.

We then learned that a person was not identifiable, so we had to change our scheme: a victim couldn't be directly associated with a party using only the id. Instead, a victim is associated with a party and a collision with the case_id and the party_number. Using both those attributes, we can obtain the party_id with

which the victim is involved.

We observed that multiple collisions could have the same primary collision factor (PCF), and thus decided to make it into an entity.

We realized that multiple collisions shared the same attributes description, like weather, road surface, road condition and others. Same goes for the victim and party tables. We decided to make those descriptions into other entities in order to save space, by avoiding repeating the same descriptions over and over again.

Relational Schema

1. ER schema to Relational schema :
Note in the following, we underline the fields corresponding to a primary key and put the foreign keys in *italic*.

Collision(case_id : **integer**,
pcf_violation : **int**,
collision_severity: **integer**,
collision_date : **date**,
collision_time : **string**,
county_city_location : **integer**,
jurisdiction : **char(4)**,
hit_and_run : **char**,
lighting : **char**,
location_type : **char**,
officer_id : **varchar(8)**,
pcf_violation_subsection : **char**,
population : **integer**,
primary_collision_factor : **char**,
process_date : **date**,
ramp_intersection : **integer**,
road_condition_1 : **char**,
road_condition_2 : **char**,
road_surface : **char**,
tow_away : **char**,
type_of_collision : **char**,
weather_1 : **char**,
weather_2 : **char**)

PCF(pcf_violation : **integer**,
pcf_violation_category : **varchar(33)**)

Collision_Severity(
collision_severity: **integer**,
collision_severity_description: **varchar(20)**)

Hit_And_Run(
hit_and_run: **char**,
hit_and_run_description: **varchar(15)**)

Lighting(
lighting: **char**,
lighting_description: **varchar(39)**)

Location(
location_type : **char**,
location_type_description : **varchar(12)**

Primary_Collision_Factor(
primary_collision_factor: **char**,
primary_collision_factor_description: **varchar(22)**)

Road_Condition(
road_condition: **char**,
road_condition_description: **varchar(14)**)

Road_Surface(
road_surface: **char**,
road_surface_description: **varchar(8)**)

Collision_Type(
type_of_collision: **char**,
type_of_collision_description: **varchar(10)**)

Weather(
weather: **char**,
weather_description: **varchar(7)**)

Party(*case_id :* **integer**,
party_number : **integer**,
age : **integer**,
sex : **char**,
at_fault : **char**,
cellphone_use : **char**,
financial_responsibility : **char**,
hazardous_materials : **char**,
movement_preceding_collision : **char**,
other_associated_factor_1 : **char**,
other_associated_factor_2 : **char**,
party_drug_physical : **char**,
safety_equipment_1 : **char**,
safety_equipment_2 : **char**,
party_sobriety : **char**,
party_type : **integer**,
school_bus_related : **char**,
statewide_vehicle_type : **char**,
vehicle_make : **integer**,
vehicle_year : **integer**)

Movement_Preceding_Collision(
movement_preceding_collision : **char**,
movement_preceding_collision_description : **varchar(26)**)

Party_Type(
party_type : **integer**,
party_type_description : **varchar(14)**)

Vehicle_Type(
statewide_vehicle_type : **char**,
statewide_vehicle_type_description : **varchar(35)**)

Vehicle_Make(
vehicle_type : **integer**,
vehicle_type_description : **varchar(28)**)

Victim(victim_id : **integer**,
case_id : **integer**,
party_number : **integer**,
safety_equipment_1 : **char**,
safety_equipment_2 : **char**,
victim_degree_of_injury : **integer**,
victim_ejected : **integer**,
victim_role : **integer**,
victim_seating_position : **char**,
victim_age : **integer**,
victim_sex : **char**)

Degree_Of_Injury(
victim_degree_of_injury : **integer**,
victim_degree_of_injury_description : **varchar(24)**)

2. DDL :

```
CREATE TABLE Collision (
    case_id integer not null,
    pcf_violation integer not null,
    collision_date date,
    collision_time varchar(20),
    collision_severity integer,
    county_city_location integer,
    hit_and_run char,
    jurisdiction char(4),
    lighting char,
    location_type char,
    officer_id varchar(8),
    pcf_violation_subsection char,
    population integer,
    primary_collision_factor char,
```

```
        process_date date,
        ramp_intersection integer,
        road_condition_1 char,
        road_condition_2 char,
        road_surface char,
        tow_away char,
        type_of_collision char,
        weather_1 char,
        weather_2 char,
        primary key (case_id),
        foreign key (pcf_violation) references PCF(pcf_violation)
        foreign key (collision_severity) references Collision_Severity(collision_severity)
        foreign key (hit_and_run) references Hit_And_Run(hit_and_run)
        foreign key (lighting) references Lighting(lighting)
        foreign key (location_type) references Location(location_type)
        foreign key (primary_collision_factor) references
            Primary_Collision_Factor(primary_collision_factor)
        foreign key (road_condition_1) references Road_Condition(road_condition)
        foreign key (road_condition_2) references Road_Condition(road_condition)
        foreign key (road_surface) references Road_Surface(road_surface)
        foreign key (type_of_collision) references Collision_Type(type_of_collision)
        foreign key (weather_1) references Weather(weather)
        foreign key (weather_2) references Weather(weather)
)


CREATE TABLE PCF (
        pcf_violation integer,
        pcf_violation_category varchar(35),
        primary key (pcf_violation)
)

CREATE TABLE Collision_Severity(
        collision_severity : integer not null,
        collision_severity_description : varchar(20),
        primary key (collision_severity)
)

CREATE TABLE Hit_And_Run(
        hit_and_run : char not null,
        hit_and_run_description : varchar(15),
        primary key (hit_and_run)
)

CREATE TABLE Lighting(
        lighting : char not null,
        lighting_description : varchar(39),
        primary key (lighting)
)
```

```
CREATE TABLE Location(
        location_type : char not null,
        location_type_description : varchar(12),
        primary key (location_type)
)

CREATE TABLE Primary_Collision_Factor(
        primary_collision_factor : char not null,
        primary_collision_factor_description : varchar(22),
        primary key (primary_collision_factor)
)

CREATE TABLE Road_Condition(
        road_condition : char not null,
        road_condition_description : varchar(14),
        primary key (road_condition)
)

CREATE TABLE Road_Surface(
        road_surface : char not null,
        road_surface_description : varchar(8),
        primary key (road_surface)
)

CREATE TABLE Collision_Type(
        type_of_collision : char not null,
        type_of_collision_description : varchar(10),
        primary key (type_of_collision)
)

CREATE TABLE Weather(
        weather : char not null,
        weather_description : varchar(7),
        primary key (weather)
)

CREATE TABLE Party (
        case_id integer not null,
        party_number integer not null,
        age integer,
        sex char,
        at_fault char,
        cellphone_user char,
        financial_responsibility char,
        hazardous_materials char,
        movement_preceding_collision char,
        other_associated_factor_1 char,
        other_associated_factor_2 char,
        party_drug_physical char,
        safety_equipment_1 char,
```

```
        safety_equipment_2 char,
        party_sobriety char,
        party_type integer,
        school_bus_related char,
        statewide_vehicle_type char,
        vehicle_make integer,
        vehicle_year integer,
        primary key (case_id, party_number),
        foreign key (case_id) references Collision(case_id) on delete cascade,
        foreign key (movement_preceding_collision) references
            Movement_Preceding_Collision(movement_preceding_collision),
        foreign key (party_type) references Party_Type(party_type),
        foreign key (statewide_vehicle_type) references
            Vehicle_Type(statewide_vehicle_type),
        foreign key (vehicle_make) references Vehicle_Make(vehicle_make)
)

CREATE TABLE Movement_Preceding_Collision(
        movement_preceding_collision : char not null,
        movement_preceding_collision_description : varchar(26),
        primary key (movement_preceding_collision)
)

CREATE TABLE Party_Type(
        party_type : integer not null,
        party_type_description : varchar(14),
        primary key (party_type)
)

CREATE TABLE Vehicle_Type(
        statewide_vehicle_type : char not null,
        statewide_vehicle_type_description : varchar(35),
        primary key (statewide_vehicle_type)
)

CREATE TABLE Vehicle_Make(
        vehicle_make : integer not null,
        vehicle_make_description : varchar(28),
        primary key (vehicle_make)
)

CREATE TABLE Victim (
        victim_id integer not null,
        case_id integer not null,
        party_number integer not null,
        safety_equipment_1 char,
        safety_equipment_2 char,
        victim_degree_of_injury integer,
        victim_ejected integer,
        victim_role integer,
```

```
        victim_seating_position char,
        victim_age integer,
        victim_sex char,
        primary key (victim_id),
        foreign key (case_id, party_number) references Party(case_id, party_number)
        on delete cascade,
        foreign key (victim_degree_of_injury) references
            Degree_Of_Injury(victim_degree_of_injury)
)

CREATE TABLE Degree_Of_Injury(
        victim_degree_of_injury : integer not null,
        victim_degree_of_injury_description : varchar(24),
        primary key (victim_degree_of_injury)
)
```

General Comments

We basically spent two sessions discussing the model, but Mr. Gueddach first described the schema, which Mr. Velez drew using the course conventions. Mr. Gueddach also did the transition from the ER schema to the relational schema. Then, Mr. Hauer did the DDL. As mentioned previously, we first misunderstood the meaning of "id" in both the victim and party tables, which led to a wrong schema.

With the comments we received from the first deliverable, we updated the ER model, the relational schema, and the DDLs.

Deliverable 2

Changes made since the first deliverable

We decided to change our ER model to separate in a better way the different attributes. We still have the three main entities: Victim, Collision and Party, but we added a few others, which are basically dictionaries containing textual descriptions. We realized that all the different descriptions could be transformed into chars or integers, so that the main tables contain less amount of data, meaning less storage used, and everything should become more efficient. The updated ER model can be found in the first deliverable, in the corresponding section. We also updated the assumptions, the description of the relational schema, the DDLs and the general comments.

SQL queries

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

```
SELECT
    EXTRACT(YEAR FROM collision_date) as "Year",
    COUNT(case_id) as "Number of collisions"
```

```
FROM Collision
GROUP BY (EXTRACT(YEAR FROM collision_date))
ORDER BY "Year";
```

	Year	Number of collisions
1	2001	522562
2	2002	544741
3	2003	538954
4	2004	538295
5	2005	532725
6	2006	498850
7	2007	501908
8	2017	7
9	2018	21

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

```
WITH res AS(
SELECT vehicle_make vm, COUNT(vehicle_make) cvm
FROM Party
GROUP BY vehicle_make
HAVING COUNT(vehicle_make) =
    (SELECT MAX(vehicle_count)
    FROM (SELECT vehicle_make, COUNT(vehicle_make) vehicle_count
    FROM Party
    GROUP BY vehicle_make)))
SELECT vehicle_make_description "Most popular vehicle",
    (SELECT cvm FROM res) "Quantity"
FROM Vehicle_Make WHERE vehicle_make = (SELECT vm FROM res);
```

	Most popular vehicle	Quantity
1	FORD	1129701

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

```
//A = 'daylight' and B = 'dusk'
    SELECT
        count1/count2 ratio
    FROM
        (SELECT CAST(COUNT(case_id) AS float) count1
```

```
 FROM Collision
 WHERE lighting != 'A' AND lighting != 'B'),
(SELECT CAST(COUNT(*) AS float) count2
 FROM Collision);
```

	RATIO
1	0.28682325452282899993828273197060517995 48

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

```
// D = snowing
    SELECT
        count(case_id) as "Number of collisions"
    FROM Collision
    WHERE weather_1='D'
    OR weather_2='D';
```

	Number of collisions
1	8530

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.

```
SELECT
    TO_CHAR(collision_date, 'DAY') as "Day",
    COUNT(case_id) as "Number of collisions"
FROM Collision
GROUP BY TO_CHAR(collision_date, 'DAY')
ORDER BY COUNT(case_id) DESC;
```

	Day	Number of collisions
1	FRIDAY	614853
2	THURSDAY	536813
3	WEDNESDAY	536068
4	TUESDAY	535743
5	MONDAY	516799
6	SATURDAY	509498
7	SUNDAY	428289

```
SELECT
    TO_CHAR(collision_date, 'DAY') as "Day",
    COUNT(case_id) as "Number of collisions"
FROM Collision
GROUP BY TO_CHAR(collision_date, 'DAY')
HAVING count(case_id)=
    (SELECT  MAX(count(case_id))
    FROM Collision
    GROUP BY TO_CHAR(collision_date, 'DAY'));
```

	Day	Number of collisions
1	FRIDAY	614853

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

```
WITH res AS(
    SELECT weather_1 w1, weather_2 w2, COUNT(case_id) cn
    FROM Collision
    GROUP BY weather_1, weather_2)
SELECT
    We1.weather_description as "Weather 1",
    We2.weather_description as "Weather 2",
    R.cn as "Number of collisions"
FROM res R, Weather We1, Weather We2
ORDER BY(R.cn) DESC;
```

	Weather 1	Weather 2	Number of collisions
1	clear	(null)	2924614
2	cloudy	(null)	452975
3	raining	(null)	138924
4	cloudy	raining	81729
5	(null)	(null)	23018
6	fog	(null)	18411
7	clear	wind	8498
8	clear	cloudy	6155
9	snowing	(null)	5693
10	other	(null)	4153
11	cloudy	snowing	2255
12	cloudy	wind	2049
13	cloudy	fog	1908
14	wind	(null)	1575
15	raining	wind	1463
16	cloudy	other	1179
17	clear	other	925
18	clear	raining	519
19	raining	fog	429
20	raining	other	397

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

```
// B = loose material
    SELECT count(*) as "Number of collisions"
    FROM Collision c, Party p
    WHERE
        p.case_id=c.case_id
        AND p.at_fault='T'
        AND financial_responsibility='Y'
        AND
            (c.road_condition_1='B'
            OR c.road_condition_2='B');
```

	Number of collisions
1	4803

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

```
SELECT
    MEDIAN(victim_age) "Median age",
    STATS_MODE(victim_seating_position) "Most common position"
FROM Victim;
```

	Median age	Most common position
1	25	3

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

```
WITH dataset AS (
    SELECT safety_equipment_1, safety_equipment_2
    FROM Party
    UNION
    SELECT safety_equipment_1, safety_equipment_2
    FROM Victim
)
SELECT count1/count2 "Ratio" FROM
(SELECT CAST(COUNT(*) AS float) count1
 FROM dataset
 WHERE safety_equipment_1 = 'C' OR
       safety_equipment_2 = 'C'),
(SELECT CAST(COUNT(*) AS float) count2
 FROM dataset);
```

	Ratio
1	0.096842105263157894736842105263157894736

10. Compute 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.

```
WITH total_count AS (SELECT CAST(COUNT(*) AS float) val
FROM Collision)
SELECT SUBSTR(collision_time, 1, 2) "Collision hour",
       ROUND(100*(count(*) / (SELECT val FROM total_count)), 2)
            || '%' "%"
       FROM Collision
       GROUP BY(SUBSTR(collision_time, 1, 2))
       ORDER BY(SUBSTR(collision_time, 1, 2));
```

	Collision hour	%
1	00	1.91%
2	01	1.83%
3	02	1.81%
4	03	1.15%
5	04	.98%
6	05	1.45%
7	06	2.62%
8	07	5.17%
9	08	5.23%
10	09	4.09%
11	10	4.23%
12	11	4.89%
13	12	5.78%
14	13	5.78%
15	14	6.55%
16	15	7.75%
17	16	7.33%
18	17	7.91%
19	18	6.3%
20	19	4.43%

Design choices

Since in our new design, we identify a party with the pair (case_id, party_number) (which was a candidate ke y), we no longer needed the party_id attribute so we dropped it. Though we kept the synthetic victim_id as a primary key in the Victims table (since it was the only candidate key we found), we made sure that Victim was a weak Entity of Party (referenced using the pair (case_id, party_number)). Similarly, following the feedback we got from the first deliverable, we also made Party a weak Entity of Collision.

Apart from redesigning our ER model, data cleaning was a big part of this second milestone. One of the issues we encountered was that some case_id's were too large to be stored in an integer, even though there are only about 3 million different case_id's. To solve this, we decided to relabel the whole case_id column with new contiguous integers. Doing so meant that we also had to remap all the case_id's in Parties and Victims to the new values.

Because the data was dirty, we had to properly clean each field (removing unwanted symbols, handling null values, reformatting some fields etc...) This involved read-

ing in the data as strings using Python scripts and outputting new clean versions. A difficulty we encountered was to properly handle cases with null values without loss of information, noticeably in the pcf case. Indeed, we encountered many rows with null pcf_violation (which is our primary key for the pcf table) but different pcf_description. We solved this by adding new pcf_violation values, one for each different pcf_description.

As stated earlier, we decided to create quite a few new tables to store the textual descriptions of some attributes. In order to do so, we scanned the 3 original csv files and created synthetic keys for each unique textual description appearing in the tables. We then had to replace the original textual fields in the original csv's by the newly created keys and mark them as foreign keys. By doing so, the queries then returned the new keys instead of the textual descriptions, which made it harder to interpret the results.

At first we had decided to keep it as is, since in a real-world application, one can simply store a dictionary of the mappings between keys and textual description in the application using the database, with no performance loss. However, we ended up modifying the queries to display the textual information instead of the keys, as it was a good exercise and made it easier to present the results in this report (though it made the queries quite a bit more complex). In our first updated implementation, the query scheme involved doing joins on the wanted data with the corresponding textual description tables; however we soon realized that this architecture made some queries terribly slow. Our next idea was to join with the textual tables only once the original query had been executed, which brought back the performance we had again.