DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Databases Project – Spring 2021

Team No: 65

Members: Zad ABI FADEL, Loïc HOUMARD, Jonas BLANC

Contents

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Deliverable 1

Assumptions

On Identification:

Every party number should be unique within a collision. Every party_id, victim_id, case_id should be unique by its own within the corresponding .csv files.

On data:

We assumed that in the .csv files every field would be represented by its key or that we would make it so during the data cleaning phase. We assumed that every description could fit in 150 char. We assumed based on data that party_id, victim_id and case_id can be typed as integer.

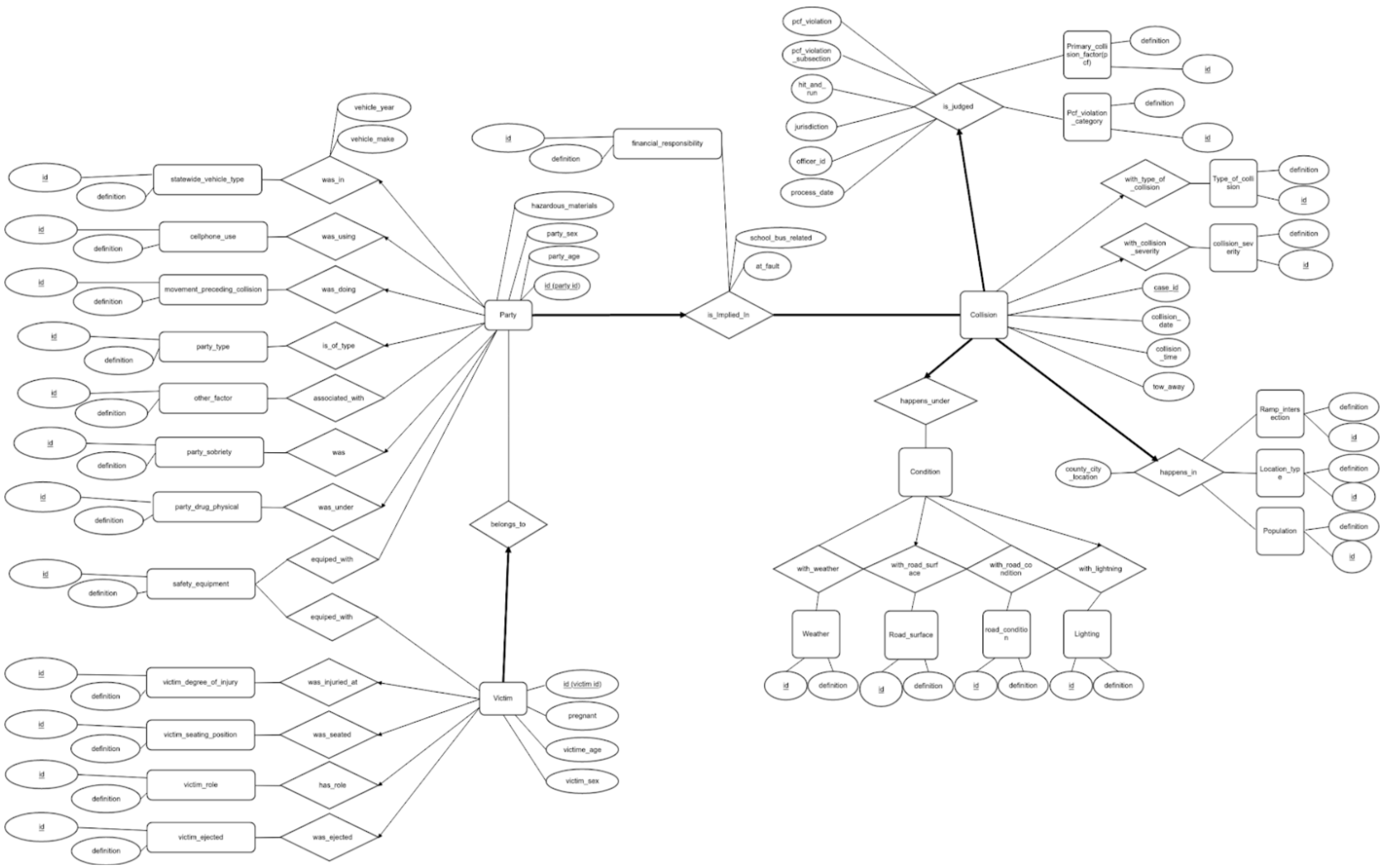
On integrity:

Every victim should be associated with an unique party. Every party should be implicated in a unique collision.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Entity Relationship Schema

Schema



DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

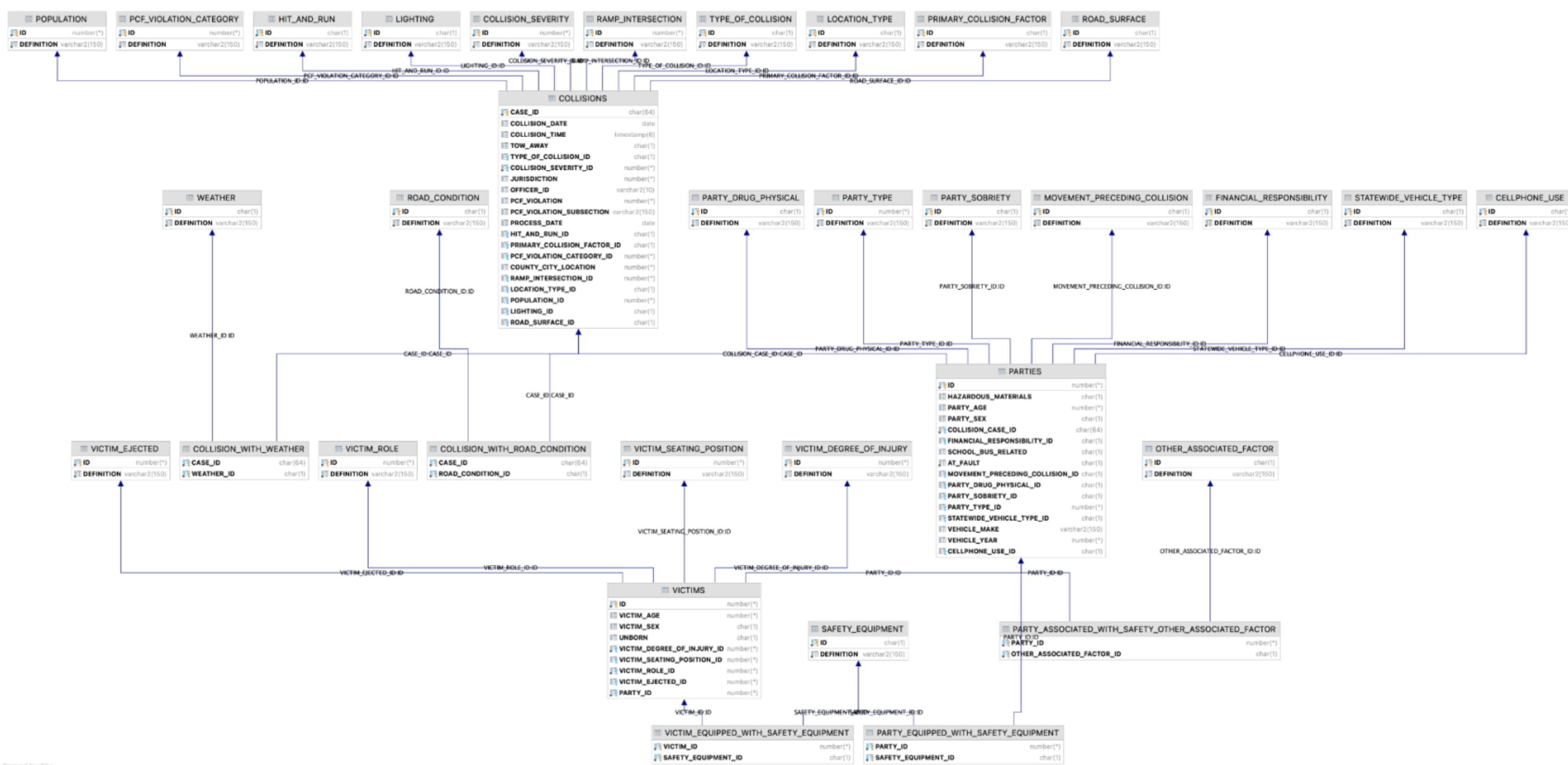
Description

For the ER diagram, we first decided to divide the attributes into 3 main entities called Victim, Party and Collision, because it seemed to us that they were the main actors in the model.
Then, we saw that it didn't make much sense to have only these 3 entities, because some attributes wouldn't be logically attributed to them. For example, it wouldn't make sense that a collision has an attribute population, because they are not directly correlated. Therefore, we tried to group attributes that logically belonged to a common idea together (star schema). For the collisions, we saw that there were many attributes related to the location of the collision, the conditions under which the collision happened and the legal part related to the collision. For the parties, many attributes were related to the vehicle. Hence, we wanted to add these 4 entities to our diagram (but finally modified it slightly, see below).
Also, after we spoke with some assistants, we realised that it would be a good idea to create entities for attributes that are lists with some finite non-logically predefined values (A:..., B:...). The reasons are the following: it would be easier to enforce the data we store to be cleaned and in the same format (it avoids to have one time 'a' and one time 'A' referencing to the same value) and it would make it more modulable and easier to change (if we realize that we would like to add/remove an option, we could simply add/remove one row in the table of the entity and add/invalidate these entries in the other table).
When there were many times the same attribute in the csv files (..._1 and ..._2), we also decided to create an entity. This has the advantage to be more modulable, since we could decide to add a third (..._3) attribute or even more of them in the future if we would like to slightly change the model. For that, we simply allowed the relation to have many of these new entities.
Finally, when we wanted to merge all our previous ideas together to construct the diagram, we found that creating the 4 entities mentioned above was not really practical because we would have to create these entities which now have no (or not many) attributes (since their corresponding attributes were often lists which we now model with an entity and bind through a relation), which makes them almost useless and increases the complexity of the diagram. Therefore, we decided to create N-ary relations directly to group the collision and all the attributes related to a given theme. This seems easier to understand and will create the same result in the database (since every attribute will finally be stored in the Collision table after the merging due to the many-to-one relation) when we translate it from the ER model to the SQL DDL commands.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Relational Schema

ER schema to Relational schema



DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

DDL

```sql
---Design implementations---

-- Boolean => char(1)

-- definition => varchar(150)

-- Table_name (First letter upper case then underscores)

-- One-to-Many (Store key in one)

-- No state is null, set key to null

-- In an entity: id is id of current entity, create new attribute
table_id for referenced id


---Collisions start---

CREATE TABLE Weather

(

   id          char(1), -- check if if is one of letter

   definition varchar(150) not null,

   PRIMARY KEY (id)

);


CREATE TABLE Road_surface

(

   id          char(1), -- check if if is one of letter

   definition varchar(150) not null,

   PRIMARY KEY (id)

);
```

```sql
CREATE TABLE Road_condition
(
   id          char(1), -- check if if is one of letter
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Lighting
(
   id          char(1), -- check if if is one of letter
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Type_of_collision
(
   id          char(1), --check char between a & h
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Collision_severity
(
   id          int CHECK (0 <= id and id <= 4),
   definition varchar(150) not null,
```

```
   PRIMARY KEY (id)
);

CREATE TABLE Hit_and_run
(
   id         char(1),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Primary_collision_factor
(
   id         char(1),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Pcf_violation_category
(
   id         int CHECK ((0 <= id and id <= 24)),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Ramp_intersection
```

```sql
(
   id          int CHECK (1 <= id and id <= 8),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Location_type
(
   id          char(1),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Population
(
   id          int CHECK (0 <= id and id <= 9),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Collisions
(
   case_id                    char(64),
   collision_date             date,
   collision_time             timestamp(6),
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
   tow_away                     char(1) CHECK (tow_away = 'T' or
tow_away = 'F'),

   type_of_collision_id         char(1) references
Type_of_collision (id),

   collision_severity_id        int not null references
Collision_severity (id),

   -- Relations is_judged

   jurisdiction                 int CHECK (0 <= jurisdiction and
jurisdiction <= 9999),

   officer_id                   varchar(10),

   pcf_violation                int,

   pcf_violation_subsection     varchar(150),

   process_date                 date,

   hit_and_run_id               char(1) references Hit_and_run
(id),

   primary_collision_factor_id char(1) references
Primary_collision_factor (id),

   pcf_violation_category_id    int references
Pcf_violation_category (id),

   -- Relations happens_in

   county_city_location         int,

   ramp_intersection_id         int references Ramp_intersection
(id),

   location_type_id             char(1) references Location_type
(id),

   population_id                int references Population (id),

   -- Relations happens_under

   lighting_id                  char(1) references Lighting (id),

   road_surface_id              char(1) references Road_surface
```

```
(id),
   PRIMARY KEY (case_id)
);


CREATE TABLE Collision_with_weather
(
   case_id    char(64) references Collisions (case_id) on delete
cascade,
   weather_id char(1) references Weather (id) on delete cascade,
   PRIMARY KEY (case_id, weather_id)
);


CREATE TABLE Collision_with_road_condition
(
   case_id           char(64) references Collisions (case_id) on
delete cascade,
   road_condition_id char(1) references Road_condition (id) on
delete cascade,
   PRIMARY KEY (case_id, road_condition_id)
);


---Collisions end---


CREATE TABLE Safety_equipment
(
   id         char(1),
```

```sql
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

---Parties start---

-- Related entities with party: one to many
CREATE TABLE Movement_preceding_collision
(
   id         char(1),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Party_drug_physical
(
   id         char(1),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Party_sobriety
(
   id         char(1),
   definition varchar(150) not null,
```

```
   PRIMARY KEY (id)
);


CREATE TABLE Party_type
(
   id          int,
   definition varchar(150) not null,
   PRIMARY KEY (id)
);


CREATE TABLE Statewide_vehicle_type
(
   id          char(1),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);


CREATE TABLE Cellphone_use
(
   id          char(1),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);
```

```
-- Relations with party: Many to many
CREATE TABLE Other_associated_factor
(
   id         char(1),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);


CREATE TABLE Financial_responsibility
(
   id         char(1),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);


-- Parties
CREATE TABLE Parties
(
   id                               int,
   -- Attributes
   hazardous_materials              char(1),
   party_age                        int,
   party_sex                        char(1),
   -- relation to collision
   collision_case_id                char(64) not null references
```

```
Collisions (case_id),

   financial_responsibility_id     char(1) references
Financial_responsibility (id),

   school_bus_related              char(1),

   at_fault                        char(1)  not null,

   -- referenced ids

   movement_preceding_collision_id char(1) references
Movement_preceding_collision (id),

   party_drug_physical_id          char(1) references
Party_drug_physical (id),

   party_sobriety_id               char(1) references
Party_sobriety (id),

   party_type_id                   int references Party_type (id),

   statewide_vehicle_type_id       char(1) references
Statewide_vehicle_type (id),

   vehicle_make                    varchar(150),

   vehicle_year                    int,

   cellphone_use_id                char(1) default 'D' references
Cellphone_use (id), --default 'D'  makes it faster

   -- key

   PRIMARY KEY (id)

);


CREATE TABLE Party_equipped_with_safety_equipment

(

   party_id            int     not null references Parties (id) on
delete cascade,

   safety_equipment_id char(1) not null references
```

```
Safety_equipment (id) on delete cascade,
   PRIMARY KEY (party_id, safety_equipment_id)
);


CREATE TABLE Party_associated_with_safety_other_associated_factor
(
   party_id                    int     not null references Parties
(id) on delete cascade,
   other_associated_factor_id char(1) not null references
Other_associated_factor (id) on delete cascade,
   PRIMARY KEY (party_id, other_associated_factor_id)
);
---Parties end---


---Victims start---
CREATE TABLE Victim_degree_of_injury
(
   id          int CHECK (0 <= id and id <= 7), -- can we make sure
id and def are consistent
   definition varchar(150) not null,
   PRIMARY KEY (id)
);


CREATE TABLE Victim_seating_position
(
   id          int, --can we check if id is number or char?
```

```
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Victim_role
(
   id          int CHECK (1 <= id and id <= 6),
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Victim_ejected
(
   id          int CHECK (0 <= id and id <= 3), --make sure entity
is still created if id is null
   definition varchar(150) not null,
   PRIMARY KEY (id)
);

CREATE TABLE Victims
(
   id                          int,
   victim_age                  int,
   victim_sex                  char(1),
   unborn                      char(1),
```

```
--- referenced ids--

   victim_degree_of_injury_id int not null references
Victim_degree_of_injury (id),

   victim_seating_position_id int references
Victim_seating_position (id),

   victim_role_id             int not null references Victim_role
(id),

   victim_ejected_id          int references Victim_ejected (id),

   party_id                   int not null REFERENCES Parties
(id),

   PRIMARY KEY (id)

);


CREATE TABLE Victim_equipped_with_safety_equipment

(

   victim_id           int     not null references Victims (id) on
delete cascade,

   safety_equipment_id char(1) not null references
Safety_equipment (id) on delete cascade,

   PRIMARY KEY (victim_id, safety_equipment_id)

);

---Victims end---
```

General Comments

In general, we found it pretty hard to create the ER diagram at first because there were a lot of attributes to proceed and understand and also because we didn't have much experience with this kind of work. But after having spent some time, we think that our implementation is now logical and should allow us to retrieve the information without having too many problems.

The allocation between the members was good, since we almost always worked together as a team. We first all took part in the elaboration of the ER diagram by concentrating us each on a CSV file and then talking with each other to see which attributes could belong together.
We then all wrote some of the SQL DDL commands to create the tables and wrote the report together.

Deliverable 2

Assumptions

Data Loading/Cleaning

We decided to clean the data in jupyter notebooks using pandas. We processed the data CSV by CSV then transferred the data using pickles for example to infer party_id from case_id and party_number. We used translation tables (python dictionary) to translate from description to id where it was needed, since we decided to create small entities for each for attributes that are lists with some finite non-logically predefined values. We generated the tables for such small entities by copying the data from the handout pdf file. For the relations with entities representing multiple attributes with the same mapping (with _1 and _2) we concatenate all the non null rows and drop the duplicates since they don't add any information.

Collisions.csv:

No major assumptions were needed to clean the collisions data. We chose to use timestamp as a type for all the date and time attributes. We first wanted to use a specific type for date only and one for time only, but we didn’t see any such data type available with Oracle DB, therefore we chose timestamp which is not ideal for our use case. For the collision_date the time is automatically set to 00:00. For the collision_time field we chose to set a fixed default date (2000-01-01) . We couldn’t merge both date and time in a single field because when one of them is missing, setting it to a default value would compromise the integrity of the data.

officer_id:

We decided to change the officer id ",66" to None because we had problems inserting it in the database due to the ','. We could have changed it to "66" (which is a valid value in the dataset), but since we were not sure that it was a typo, we found this assumption too strong and therefore we prefered to remove it.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Parties.csv

The data from parties had more dirty values. Here are the choices we did:

cellphone_use:

We realised that the values that are stored in the cellphone_use column {'1', '2', '3', 'B', 'C', 'D', nan} are different to the ones on the handout {'B', 'C', 'D', nan}. The values that are in the data but not in the handout {'1', '2', '3'} appear 2’636’894 times. We decided not to drop these values because they are a big chunk of the data (56%).

We needed to find a plausible mapping between the numbers and the letters. We opted to do it by doing a frequency analysis.

1 : 24787 in % : 0.009	B : 38932 in % : 0.018
2 : 39114 in % : 0.015	C : 795475 in % : 0.377
3 : 2572993 in % : 0.976	D : 1274423 in % : 0.604

As you can see, it is clear that 1 and B are those that appear the least, and 3 and D are those that appear most frequently.

Therefore, we concluded that the correct mapping is : 1 -> B, 2 -> C, 3 -> D

As we imported the data in the database we chose to replace the None value by "D" since D already means "No Cell Phone/Unknown" which is equal to "no value".

vehicle_make:

Since vehicle_make is an open field there are a lot of errors and inconsistency. We corrected the most obvious typos (see below) and made some brands consistent. We chose not to modify this field too much since we are not experts in vehicle_make and that's error prone to modify it manually. For example we decided not to remove values with "OTHER - ..." since they add information compared to a "None". Here are the typos and inconsistencies we corrected and:

"AMERICAN MOTORS" => "AMERICAN MOTORS (AMC)"

"DODG" => "DODGE"

"HOND" => "HONDA"

"MERCEDES BENZ" => "MERCEDES-BENZ"

"MAZD" => "MAZDA"

"TOYTA" => "TOYOTA"

"MISCELLANEOUS" , "NOT STATED" => None

party_drug_physical:

We noticed 585'062 rows of party_drug_physical with value "G" which is not a valid key. We decided to replace it by None since we had no way to guess what the correct value was.

Victims:

victim_age and pregnancy:

In order to clean the data and make querying easier, we decided to create a new field: unborn which is a boolean telling if the victim was born or not. We set unborn from the convention saying that if the age is a 999 then the victim is the fetus of a pregnant woman. Then we replaced the age 999 by None. We chose to replace it by None and not 0 because we thought it would make more sense and that it would be weird if the mean of age of a 30 years old pregnant woman is 15 years.

Assumptions For Queries

For the queries, we assumed that we could use all available built-in functions for Oracle database systems. These functions are EXTRACT, COUNT, MEDIAN, FETCH, TO_CHAR, LOWER and DUAL.

Query Implementation

Description of logic:

This query should retrieve the number of collisions per year. Therefore, we first group by the year that we extract with the built-in function "EXTRACT(YEAR from …)". We then count the number of entries per year. We decided to order it by year, ascending to make it clearer.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

SQL statement

```
SELECT EXTRACT(YEAR FROM C.COLLISION_DATE) AS YEAR, COUNT(*) AS
NUMBER_COLLISIONS

FROM COLLISIONS C

GROUP BY EXTRACT(YEAR FROM C.COLLISION_DATE)

ORDER BY EXTRACT(YEAR FROM C.COLLISION_DATE) ASC;
```

Query result (if the result is big, just a snippet)

YEAR	NUMBER_COLLISIONS
2001	522562
2002	544741
2003	538954
2004	538295
2005	532725
2006	498850

2007	501908
2017	7
2018	21

Query 2:

Description of logic:

This query should retrieve the most popular vehicle make and the number of vehicles for this make. We do this by first grouping by make and sorting it by the number of vehicles for each make. To retrieve the most popular make only, we use the “FETCH FIRST 1 ROW ONLY“ built-in function (which is equivalent to limit in MySQL).

SQL statement

```
SELECT P.VEHICLE_MAKE, COUNT(*) AS NUMBER_VEHICLE
FROM PARTIES P
GROUP BY P.VEHICLE_MAKE
ORDER BY COUNT(*) DESC
FETCH FIRST 1 ROW ONLY;
```

Query result (if the result is big, just a snippet)

VEHICLE_MAKE	NUMBER_VEHICLE
FORD	1129701

Query 3:

Description of logic:

This query should retrieve the fraction of collisions which happen under dark lighting. For that, we first query the lightning that contains "dark" in their definition (note that we could directly use the ID since we know it, but we found clearer and more robust to query it using the definition if we would like to use the same query later on, when the table could be modified and more than one field could be about dark weather). We then bind it to the lighting id stored in the collisions to count all the collisions with this weather type. We finally divide by the total number of collisions to have a fraction. We also decided to round the result to avoid having many useless digits.

SQL statement

```
SELECT
ROUND(A.NUMBER_COLLISIONS_UNDER_DARK/A.TOTAL_NUMBER_COLLISIONS, 3)
AS FRACTION_UNDER_DARK

FROM(

   SELECT

       (SELECT COUNT(*)

           FROM COLLISIONS C

           WHERE C.LIGHTING_ID IN

               (   SELECT L.ID

                   FROM LIGHTING L

                   WHERE LOWER(L.DEFINITION) LIKE '%dark%')) AS
```

```
NUMBER_COLLISIONS_UNDER_DARK,
           (SELECT COUNT(*) FROM COLLISIONS) AS
TOTAL_NUMBER_COLLISIONS
   FROM DUAL
)A;
```

Query result (if the result is big, just a snippet)

FRACTION_DARK
0.28

Query 4:

Description of logic:

This query should retrieve the number of collisions which happen under snowy weather. Just like before, we just query the ids in weather which contain "snow" in their definition and count all the entries of the relation which have this id.

SQL statement

```
SELECT COUNT(*) AS NUMBER_COLLISIONS_SNOWY_WEATHER
FROM COLLISION_WITH_WEATHER CWW
WHERE CWW.WEATHER_ID IN
   (   SELECT W.ID
```

```
        FROM WEATHER W
        WHERE LOWER(W.DEFINITION) LIKE '%snow%');
```

Query result (if the result is big, just a snippet)

NUMBER_COLLISIONS_SNOWY_WEATHER
8530

Query 5:

Description of logic:

This query should retrieve the number of collisions that happen every day of the week. For that, we first groupy by the day using the built-in function "TO_CHAR(date, 'DAY')" and count the number of entries.

SQL statement

```
SELECT TO_CHAR(C.COLLISION_DATE, 'DAY') AS WEEKDAY, COUNT(*) AS
NUMBER_COLLISIONS
FROM COLLISIONS C
GROUP BY TO_CHAR(C.COLLISION_DATE, 'DAY');
```

Query result (if the result is big, just a snippet)

WEEKDAY	NUMBER_COLLISIONS
MONDAY	516799
TUESDAY	535743
SUNDAY	428289
WEDNESDAY	536068
FRIDAY	614853
SATURDAY	509498
THURSDAY	536813

Query 6:

Description of logic:

This query should retrieve all the types of weather and their corresponding number of collisions, sorted in descending order. For that, we simply join the tables weather and collisions with weather, then group by the definition and count all the entries.

SQL statement

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
SELECT W.DEFINITION, COUNT(*) AS NUMBER_COLLISIONS
FROM WEATHER W, COLLISION_WITH_WEATHER CWW
WHERE W.ID=CWW.WEATHER_ID
GROUP BY W.DEFINITION
ORDER BY COUNT(*) DESC;
```

Query result (if the result is big, just a snippet)

DEFINITION	NUMBER_COLLISIONS
Clear	2941042
Cloudy	548250
Raining	223752
Fog	21259
Wind	13952
Snowing	8530
Other	6960

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Query 7:

Description of logic:

This query should retrieve all the at-fault collision parties with financial responsibility and loose material. For that, we had to check if the party is at fault in the table party and then check for the financial responsibility by using the id and extracting the ones having "yes" in their description and finally check the loose material by using the case id to retrieve the collision, then the road condition id from the relation table and finally take only the definition having "loose material" in it.

SQL statement

```
SELECT COUNT(*) AS NUMBER_AT_FAULT_WITH_FIN_REP_LOOSE_MAT
FROM PARTIES P
WHERE P.AT_FAULT='T'
AND P.FINANCIAL_RESPONSIBILITY_ID IN
    (   SELECT FR.ID
        FROM FINANCIAL_RESPONSIBILITY FR
        WHERE LOWER(FR.DEFINITION) LIKE '%yes%')
AND P.COLLISION_CASE_ID IN
    (   SELECT COL.CASE_ID
        FROM COLLISIONS COL
        WHERE COL.CASE_ID IN
            (   SELECT CWRC.CASE_ID
                FROM COLLISION_WITH_ROAD_CONDITION CWRC
                WHERE CWRC.ROAD_CONDITION_ID IN
                    (   SELECT RC.ID
```

```
                        FROM ROAD_CONDITION RC
                        WHERE LOWER(RC.DEFINITION) LIKE '%loose
material%')));
```

Query result (if the result is big, just a snippet)

NUMBER_AT_FAULT_WITH_FIN_REP_LOOSE_MAT
4608

Query 8:

Description of logic:

This query should retrieve the median age and the most common victim seating position. Since these 2 informations have not much to do with each other, we first wrote them individually and then used dual to write them together.

For the median age, we just used the built-in "MEDIAN" function.

For the most common victim seating position, we used the same trick as in query 2 which is to group by the seating position, sort by the number and keep the top row only.

SQL statement

```
SELECT
A.VICTIM_AGE_MEDIAN, A.MOST_COMMON_VICTIM_SEATING_POSITION
```

```
FROM
(
   SELECT
       (   SELECT MEDIAN(V.VICTIM_AGE)
           FROM VICTIMS V) AS VICTIM_AGE_MEDIAN,
       (   SELECT VSP.DEFINITION
           FROM VICTIM_SEATING_POSITION VSP
           WHERE VSP.ID IN
           (   SELECT V.VICTIM_SEATING_POSITION_ID
               FROM VICTIMS V
               GROUP BY V.VICTIM_SEATING_POSITION_ID
               ORDER BY COUNT(*) DESC
               FETCH FIRST 1 ROW ONLY)) AS
MOST_COMMON_VICTIM_SEATING_POSITION
   FROM DUAL
)A;
```

Query result (if the result is big, just a snippet)

VICTIM_AGE_MEDIAN	MOST_COMMON_VICTIM_SEATING_POSITION
25	Passengers

Query 9:

Description of logic:

This query should retrieve the fraction of victims who were using a belt along all the participants. For that, we first count all victims which have a belt and divide by the total number of victims and participants using DUAL to be able to divide them. We also decided to round the result to make it more readable.

Remarks

We found this query not very logical since a party represents a group of people and that a party could be already counted in the victim table, but not necessarily since we have no way to be sure whether a party only has victims or not. At first, we had only counted the total number of victims (instead of victims + parties), but after seeing this post https://moodle.epfl.ch/mod/forum/discuss.php?d=56137, point3, we decided to use the query shown below.

SQL statement

```
SELECT ROUND(A.NUMBER_VICTIM_WITH_BELT / (A.TOTAL_VICTIM +
A.TOTAL_PARTIES), 3) AS FRACTION_WITH_BELT

FROM

(

   SELECT

       (SELECT COUNT(*)

       FROM VICTIMS V

       WHERE V.ID IN

           (   SELECT VEWSE.VICTIM_ID

               FROM VICTIM_EQUIPPED_WITH_SAFETY_EQUIPMENT VEWSE

               WHERE VEWSE.SAFETY_EQUIPMENT_ID IN
```

```
                    (   SELECT SE.ID
                        FROM SAFETY_EQUIPMENT SE
                        WHERE LOWER(SE.DEFINITION) LIKE '%belt
use%'))) AS NUMBER_VICTIM_WITH_BELT,
       (SELECT COUNT(*) FROM VICTIMS) AS TOTAL_VICTIM,
       (SELECT COUNT(*) FROM PARTIES) AS TOTAL_PARTIES
       FROM DUAL
)A;
```

Query result (if the result is big, just a snippet)

FRACTION_WITH_BELT
0.011

Query 10:

Description of logic:

This query should retrieve the fraction of collisions that happen each hour of the day. For that, we simply group by the hour that we extract from the time using the EXTRACT(HOUR, time) built-in function, count the number of entries for each hour and divide by the total number of collisions.

Remark:

We decided to keep an entry when the hour was not specified with the fraction of accidents when the hour was unknown because we found it clearer this way.

We only showed the first 20 entries in the result as asked in the question.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

SQL statement

```
SELECT EXTRACT(HOUR FROM C.COLLISION_TIME) AS HOUR,
ROUND(COUNT(*)/(  SELECT COUNT(*) FROM COLLISIONS), 3) AS
FRACTION_COLLISIONS

FROM COLLISIONS C

GROUP BY EXTRACT(HOUR FROM C.COLLISION_TIME)

ORDER BY EXTRACT(HOUR FROM C.COLLISION_TIME) ASC;
```

Query result (if the result is big, just a snippet)

HOUR	FRACTION_COLLISIONS
0	0.019
1	0.018
2	0.018
3	0.012
4	0.01

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

5	0.014
6	0.026
7	0.052
8	0.052
9	0.041
10	0.042
11	0.049
12	0.058
13	0.058
14	0.065
15	0.077
16	0.073

17	0.079
18	0.063
19	0.044

General Comments

We didn't have to change our previous work on the ER diagram in part 1 too much and we were able to write the queries quite easily. However, it took us a lot of time to clean the data and we had some problems when we tried to import the data in the database.

We decided to work all together on the different tasks, each team member spent an equal amount of time.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Deliverable 3

Assumptions

<In this section write down the assumptions you made about the data. Write a sentence for each assumption you made>

Query Implementation

<For each query>
Query a:
Description of logic:
<What does the query do and how do I decide to solve it>
SQL statement
<The SQL statement>
Query result (if the result is big, just a snippet)
<The SQL statement result>

Query Performance Analysis – Indexing

<In this section, for 6 selected queries explain in detail why do you see given improvements (or not). For example, why building an index on certain field changed the plan and IO.>
Query 1
<Initial Running time/IO:
Optimized Running time/IO:
Explain the improvement:
Initial plan
Improved plan>

Query 2
<Initial Running time/IO:
Optimized Running time/IO:
Explain the improvement:
Initial plan
Improved plan>

General Comments

<In this section write general comments about your deliverable (comments and work allocation between team members>