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.

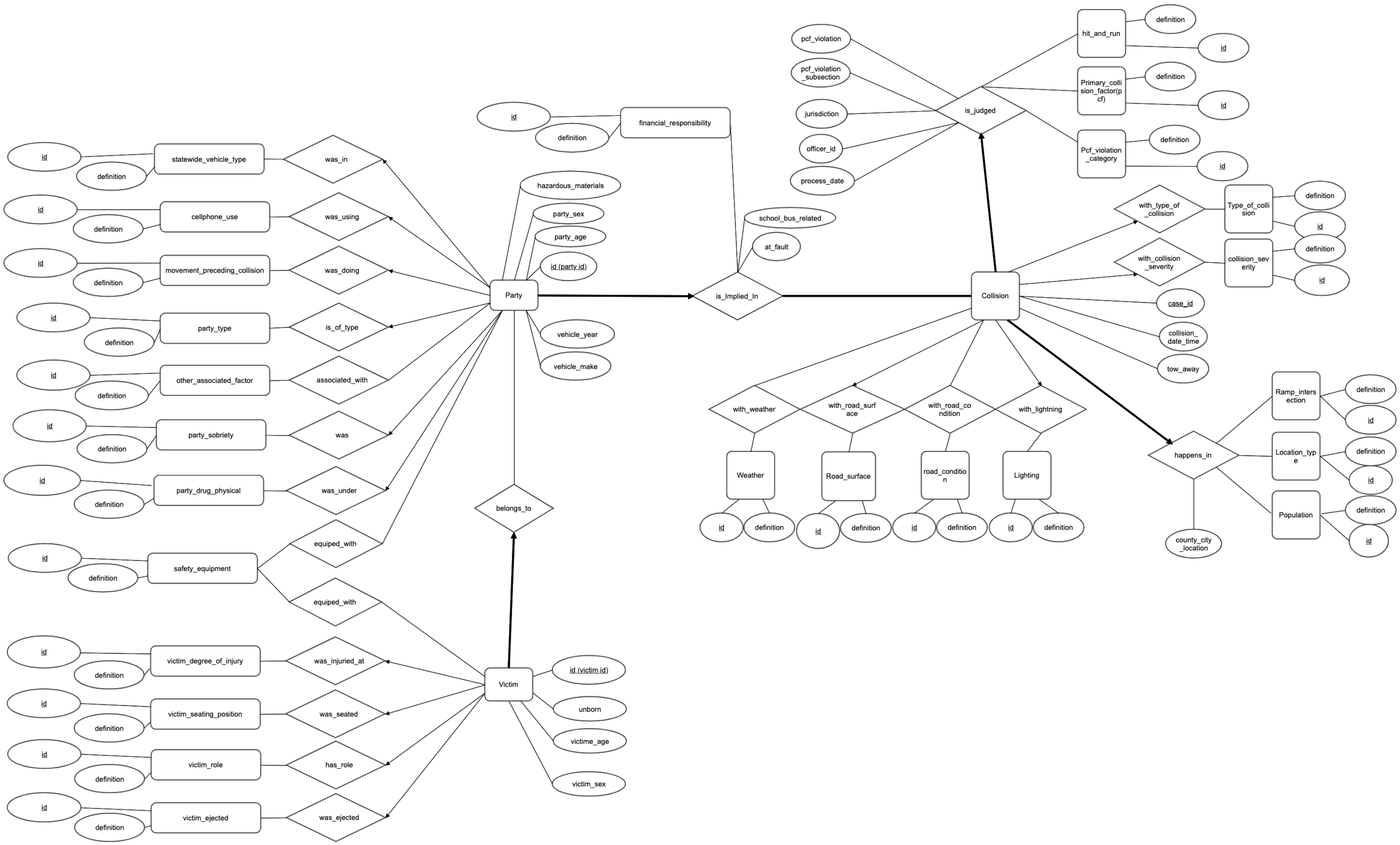
On integrity:

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/

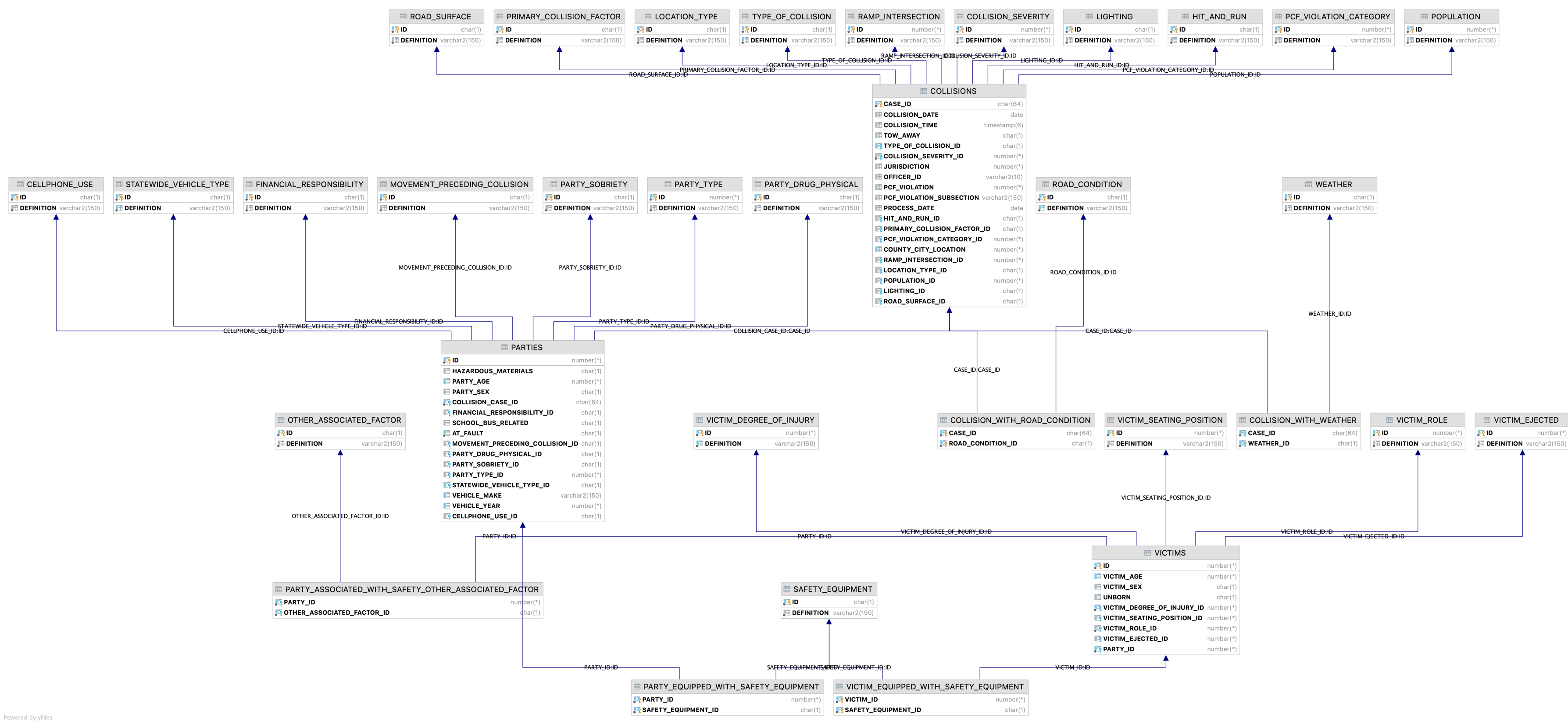
Description

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.
After the first milestone, we also decided to remove the condition table which we had kept, because we found it easier to implement in the data cleaning process and because our associated TA advised us to do so. Indeed, on our older schema, we had to create a custom key for condition and bind it through a relation which was more complicated and didn’t bring much. The only utility of the condition table was to make the star schema easier to understand, but in practice it didn’t bring much.

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

```
---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)
);


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
(
```

```sql
   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
(
```

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/

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


CREATE TABLE Ramp_intersection
(
   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),
   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)
);
```

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
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),
   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)
);
```

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
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)
```

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/

```
);


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,
```

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/

```
    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.

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 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.

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.

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/

EPFL

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.

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

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 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

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 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

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 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. To retrieve the top 1 only, we first sort by the number of collisions and fetch the first row only.

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')
ORDER BY COUNT(*) DESC
FETCH FIRST 1 ROW ONLY;
```

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

WEEKDAY	NUMBER_COLLISIONS
FRIDAY	614853

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

```
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, FINANCIAL_RESPONSIBILITY FR, COLLISIONS COL,
COLLISION_WITH_ROAD_CONDITION CWRC, ROAD_CONDITION RC
WHERE P.AT_FAULT = 'T'
AND P.FINANCIAL_RESPONSIBILITY_ID = FR.ID
AND LOWER(FR.DEFINITION) LIKE '%yes%'
AND P.COLLISION_CASE_ID = COL.CASE_ID
AND COL.CASE_ID = CWRC.CASE_ID
AND CWRC.ROAD_CONDITION_ID = RC.ID
AND LOWER(RC.DEFINITION) LIKE '%loose material%';
```

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

NUMBER_AT_FAULT_WITH_FIN_REP_LOOSE_MAT
4803

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 8:

Description of logic:

This query should retrieve the median age and the most common victim seating position. Since these 2 pieces of information 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;
```

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 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.

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 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

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 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.

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

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

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/

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

The assumptions/choices we made for our queries are the following:

- Round the decimal values returned by the queries to 3 decimal numbers to make them more readable.
- We decided to use the definitions of the small tables instead of directly using their ID because we found this way of querying the information easier to understand and cleaner. However, it comes with some cost since we must join the small tables everytime.
- For the 3rd query, we decided to discard the vehicle makes which were null. The reason is that null was the fourth most represented "vehicle make" and we found that this information was not really relevant since we were looking for real vehicle make (if a brand would like to make some statistics or know where they are on the list, they wouldn't care about the null values which don't give much useful information).
- For the 4th query, we understood " fraction of total incidents" as the number of incidents where no injury happened for a given seating position divided by the total number of victims seated at this particular position only (and not the total number of victims). It made more sense to us and a post on the forum seemed to agree with this assumption.
- For the 6th query, since many cities had the same population type (over 250'000) and we couldn't know the exact population from the data, we just took the 3 first results that the database returned for this category.
- For the 6th query, we decided to keep the case_ids where some ages were unknown (null), but not considering these ages in the computation of the average. This means for example that if we have an accident with people of age (10,40, null), the average would be 25 since null would be ignored. We could have dropped these entries instead of accepting them and ignore the null values for computation only, but we found that it was a good approach to count the accidents where only partial values were given as well to limit the data we're dropping. However, depending on why we would like to know this query, it could be useful to discard these entries.
- For the 8th query, since we didn't use any vehicle id in our diagram, we decided to use the vehicle type, vehicle make and vehicle year as the id, because they represent all the available information we have about the vehicles.
- For the last query (10th), we first based our classification on the lighting information when they were clear enough (daylight for day, dark for night). For the dusk-dawn case, we based ourselves on the time and the month and when we had inconsistent data (dusk-dawn at 12:00 for example), we discarded it. When the lighting was null, we tried to infer the period based on the time and the date only when it was possible and discarded the data otherwise.

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 Implementation

Query 1:

Description of logic:

This query should retrieve the ratio of cases where the driver was at fault for different age groups. For this, we first did two subqueries: a first one which counts all the parties that fall in each age category where we simply discarded all the parties having null for age and grouped by the age category using a case on the age and a second one quite similar where we took only the parties being at fault for each age group following the same logic. We then returned the age category and divided each of the two results to have the ratio of parties at fault. We also decided to sort it in descending order to have a better vision of the results and clearly see which are the categories that were the most often at fault. As we can see in the results and as we might have expected, underage, young people and elder people (elder 2) tend to be more often at fault. Therefore, as an insurance company, it would make sense to make young and old people pay more than middle-aged adults for their insurance.

SQL statement

```
SELECT FAULT.age_range, ROUND(NUMBER_AT_FAULT / TOTAL_NUMBER, 3)
as RATIO_AT_FAULT
FROM (SELECT case
                when P.PARTY_AGE <= 18 then 'Underage'
                when P.PARTY_AGE between 19 and 21 then 'young 1'
                when P.PARTY_AGE between 22 and 24 then 'young 2'
                when P.PARTY_AGE between 24 and 60 then 'adult'
                when P.PARTY_AGE between 61 and 64 then 'elder 1'
                when P.PARTY_AGE >= 65 then 'elder 2' end as
age_range,
            COUNT(*)                                        AS
NUMBER_AT_FAULT
     FROM PARTIES P
     WHERE P.AT_FAULT = 'T'
       and P.PARTY_AGE IS NOT NULL
     group by (case
                   when P.PARTY_AGE <= 18 then 'Underage'
                   when P.PARTY_AGE between 19 and 21 then 'young
1'
                   when P.PARTY_AGE between 22 and 24 then 'young
2'
                   when P.PARTY_AGE between 24 and 60 then 'adult'
                   when P.PARTY_AGE between 61 and 64 then 'elder
1'
                   when P.PARTY_AGE >= 65 then 'elder 2'
```

```
        END)) FAULT,

    (SELECT case
                 when P.PARTY_AGE <= 18 then 'Underage'
                 when P.PARTY_AGE between 19 and 21 then 'young 1'
                 when P.PARTY_AGE between 22 and 24 then 'young 2'
                 when P.PARTY_AGE between 24 and 60 then 'adult'
                 when P.PARTY_AGE between 61 and 64 then 'elder 1'
                 when P.PARTY_AGE >= 65 then 'elder 2' end as
age_range,
            COUNT(*)                                    AS
TOTAL_NUMBER
     FROM PARTIES P
     WHERE P.PARTY_AGE IS NOT NULL
     group by (case
                    when P.PARTY_AGE <= 18 then 'Underage'
                    when P.PARTY_AGE between 19 and 21 then 'young
1'
                    when P.PARTY_AGE between 22 and 24 then 'young
2'
                    when P.PARTY_AGE between 24 and 60 then 'adult'
                    when P.PARTY_AGE between 61 and 64 then 'elder
1'
                    when P.PARTY_AGE >= 65 then 'elder 2'
        END)) TOTAL
WHERE TOTAL.age_range = FAULT.age_range
ORDER BY NUMBER_AT_FAULT / TOTAL_NUMBER DESC;
```

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

AGE_RANGE	RATIO_AT_FAULT
underage	0.636
young 1	0.572
young 2	0.517
elder 2	0.498
adult	0.409
elder 1	0.399

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 2:

Description of logic:

This query should retrieve the top 5 vehicles having the most collisions on roads with holes. We first do a subquery where we retrieve the ID of the vehicle type and the corresponding number of collisions. For that, we join the parties, road conditions and the relation between them, counting only the ones having holes, grouping by the ID of the vehicle type. We also sort it in descending order and fetch the 5 first row only in order to keep the 5 biggest values. We then use this subquery to extract the definition instead of the ID.

SQL statement

```
SELECT SWT.DEFINITION, STATS_COLLISIONS_HOLE.NUMBER_OF_COLLISION
FROM STATEWIDE_VEHICLE_TYPE SWT,
    (SELECT P.STATEWIDE_VEHICLE_TYPE_ID AS SVT_ID, COUNT(*) AS
NUMBER_OF_COLLISION
     FROM PARTIES P,
          COLLISION_WITH_ROAD_CONDITION CWRC,
          ROAD_CONDITION RC
     WHERE P.STATEWIDE_VEHICLE_TYPE_ID IS NOT NULL
       AND P.COLLISION_CASE_ID = CWRC.CASE_ID
       AND CWRC.ROAD_CONDITION_ID = RC.ID
       AND RC.DEFINITION = 'Holes, Deep Ruts'
     GROUP BY P.STATEWIDE_VEHICLE_TYPE_ID
     ORDER BY COUNT(*) DESC
         FETCH FIRST 5 ROW ONLY
    ) STATS_COLLISIONS_HOLE
WHERE SWT.ID = STATS_COLLISIONS_HOLE.SVT_ID;
```

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

DEFINITION	NUMBER_OF_COLLISIONS_HOLE
Passenger Car/Station Wagon	10662
Pickup or Panel Truck	2263
Motorcycle/Scooter	450
Bicycle	430
Truck or Truck Tractor with Trailer	369

Query 3:

Description of logic:

This query should retrieve the top 10 vehicle makes with the most victims killed or with severe injuries. For this query, we first join the tables parties, victims and victim degree of injury. We then only take the rows where the degree of injury is either killed or severe injury and the vehicle make is not null (see our assumptions). We then group by the vehicle make, count the number of entries and sort it in descending order to be able to retrieve the top values only. Finally, we fetch the 10 first rows, in order to keep the top 10 only.

SQL statement

```
SELECT P.VEHICLE_MAKE, COUNT(*) AS
NUMBER_OF_VICTIMS_KILLED_OR_WITH_SEVERE_INJURIES
from PARTIES P,
    VICTIMS V,
    VICTIM_DEGREE_OF_INJURY VDOI
WHERE P.ID = V.PARTY_ID
 AND V.VICTIM_DEGREE_OF_INJURY_ID = VDOI.ID
 AND (VDOI.DEFINITION = 'Killed' OR VDOI.DEFINITION = 'Severe
Injury')
 and P.VEHICLE_MAKE is not NULL -- NULL is the 4th more
represented, not really interesting
group by P.VEHICLE_MAKE
order by COUNT(*) DESC
   FETCH FIRST 10 ROW ONLY;
```

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

VEHICLE_MAKE	NUMBER_OF_VICTIMS_KILLED_OR_WITH_SEVERE_INJURIES
FORD	13924
HONDA	12061
TOYOTA	10639
CHEVROLET	10418
NISSAN	3860
DODGE	3641

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/

HARLEY-DAVIDSON	3410
SUZUKI	2482
YAMAHA	2105
GMC	1837

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 4:

Description of logic:

This query should retrieve the safety index and the definition of the most safe and unsafe seating position. The safety factor is computed as the total number of victims having no injuries for a given position divided by the total number of victims at that position.

This query is done in two steps. We first create a table with every seating position and its corresponding safety factor. Then in a second query, we fetch only the row with the maximum and the row with the minimum safety factor.

For the creation of the table, we first run 2 subqueries. One that retrieves the definition of the position, and counts all the victims on that position. For that, we just join the tables for the victims and victims' degree of injuries and group by the seating position. We then extract the definition using the victim seating position table. The second subquery is almost equivalent except that we keep only the number of uninjured victims for each seating position. We then join the two subqueries on the definition of the seating position and compute the safety factor with a division of their respective count. To query the best and worst factors in this table, we then keep only the rows where the safety factor is either equivalent to the max or the min of the table, retrieved with 2 subqueries.

SQL statement

```
with SEATING_POSITION_TO_SAFETY_FACTOR AS (
   SELECT UNINJURED.DEFINITION,
          ROUND(UNINJURED.NUMBER_NO_INJURIES /
ALL_DEGREES.NUMBER_ALL_DEGREE_INJURIES, 3) AS SAFTEY_FACTOR
   FROM (
            SELECT VSP.DEFINITION,
SEATING_POSITION_NO_INJURIES.NUMBER_NO_INJURIES as NUMBER_NO_INJURIES
            FROM VICTIM_SEATING_POSITION VSP,
                 (
                     SELECT V.VICTIM_SEATING_POSITION_ID AS
VICTIM_SEATING_POSITION_ID, COUNT(*) AS NUMBER_NO_INJURIES
                     FROM VICTIMS V,
                          VICTIM_DEGREE_OF_INJURY VDOI
                     WHERE V.VICTIM_DEGREE_OF_INJURY_ID = VDOI.ID
                       AND VDOI.DEFINITION = 'No Injury'
                       AND V.VICTIM_SEATING_POSITION_ID is not NULL
                     GROUP BY V.VICTIM_SEATING_POSITION_ID)
SEATING_POSITION_NO_INJURIES
            WHERE VSP.ID =
SEATING_POSITION_NO_INJURIES.VICTIM_SEATING_POSITION_ID) UNINJURED,
```

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 VSP.DEFINITION,
                   GROUPED_SEATING_POSITIONS.NUMBER_ALL_DEGREE_INJURIES
as NUMBER_ALL_DEGREE_INJURIES
            FROM VICTIM_SEATING_POSITION VSP,
                 (
                     SELECT V.VICTIM_SEATING_POSITION_ID AS
VICTIM_SEATING_POSITION_ID,
                            COUNT(*)                     AS
NUMBER_ALL_DEGREE_INJURIES
                     FROM VICTIMS V,
                          VICTIM_DEGREE_OF_INJURY VDOI
                     WHERE V.VICTIM_DEGREE_OF_INJURY_ID = VDOI.ID
                       AND VICTIM_SEATING_POSITION_ID is not NULL
                     GROUP BY V.VICTIM_SEATING_POSITION_ID)
GROUPED_SEATING_POSITIONS
            WHERE VSP.ID =
GROUPED_SEATING_POSITIONS.VICTIM_SEATING_POSITION_ID) ALL_DEGREES

   WHERE UNINJURED.DEFINITION = ALL_DEGREES.DEFINITION)

SELECT *
FROM SEATING_POSITION_TO_SAFETY_FACTOR
WHERE SAFTEY_FACTOR = (SELECT MAX(SAFTEY_FACTOR) FROM
SEATING_POSITION_TO_SAFETY_FACTOR)
  OR SAFTEY_FACTOR = (SELECT MIN(SAFTEY_FACTOR) FROM
SEATING_POSITION_TO_SAFETY_FACTOR);
```

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

DEFINITION	SAFETY FACTOR
DRIVER	0.009
STATION WAGON REAR	0.825

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 5:

Description of logic:

This query should retrieve the number of vehicle types which have had at least 10 collisions in at least half of the cities. For this query, we first keep the vehicles/locations tuples having at least 10 collisions. Join the tables parties and collisions, group by the vehicle type and city location (removing the null values) and count the number of entries for each of these tuples and keep only the ones having at least 10 entries. We then group by the type of vehicle and count the number of entries, which correspond to the number of cities in which each vehicle type had at least 10 collisions. We then only keep those where this number is at least half of the cities. To count half the number of the cities, we used a subquery which counts every unique location and divides it by 2.

SQL statement

```
SELECT COUNT(*) AS NUMBER_OF_VEHICLE_TYPE
FROM (SELECT TYPE_CITY_TO_ACCIDENT_COUNT.TYPE
     FROM (SELECT P.STATEWIDE_VEHICLE_TYPE_ID AS TYPE,
C.COUNTY_CITY_LOCATION
           FROM PARTIES P,
                COLLISIONS C
           WHERE P.COLLISION_CASE_ID = C.CASE_ID
             AND C.COUNTY_CITY_LOCATION IS NOT NULL
             AND P.STATEWIDE_VEHICLE_TYPE_ID IS NOT NULL
           GROUP BY (P.STATEWIDE_VEHICLE_TYPE_ID,
C.COUNTY_CITY_LOCATION)
           HAVING COUNT(*) >= 10
          ) TYPE_CITY_TO_ACCIDENT_COUNT
     GROUP BY TYPE_CITY_TO_ACCIDENT_COUNT.TYPE
     HAVING COUNT(*) >= (SELECT COUNT(UNIQUE
(C.COUNTY_CITY_LOCATION)) / 2
                         FROM COLLISIONS C
     )
    ) TYPE_TO_CITY_COUNT;
```

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

NUMBER_OF_VEHICLE_TYPE
13

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 6:

Description of logic:

This query should retrieve the top 10 minimum average age cases for the 3 most populated cities, together with the city location, the population and the case id.

In order to do that, we rely on some subqueries. First, we compute the average victim age for each collision that happened in the 3 most populated cities. To get those 3 cities, we simply take 3 cities (no specific ordering) that have a population_id that corresponds to 'Incorporated (over 250000)'. Once we have the average victim age for each collision in the top 3 most populated cities, we label each resulting average victim age in ascending order. This will label the average victim age in each city. This allows us to start the count at 1 for each of the 3 cities. The last part of the query consists of taking the resulting rows that have a row_number less or equal to 10. This way we can show for each of the top-3 most populated cities the bottom 10 collisions in terms of average victim age.

Remark:

- We only showed the first 20 entries in the result as asked in the question.
- Due to our assumptions on the null values (see Assumptions), we only got 0 as age average. We would probably have had some non-zero values if we had discarded the cases where some ages were null, but as stated in the assumption, it made sense to us to discard as little data as possible.

SQL statement

```
with average_age(COLLISION_CASE_ID, COUNTY_CITY_LOCATION,
POPULATION_ID, V_AGE) as
        (
            SELECT distinct COLLISION_CASE_ID,
                            COUNTY_CITY_LOCATION,
                            POPULATION_ID,
                            avg(v.VICTIM_AGE) OVER (PARTITION BY
C.CASE_ID) as v_age
            FROM COLLISIONS C
                     INNER JOIN PARTIES on C.CASE_ID =
PARTIES.COLLISION_CASE_ID
                     inner join VICTIMS V on PARTIES.ID =
V.PARTY_ID
            WHERE C.COUNTY_CITY_LOCATION in (
                SELECT distinct COUNTY_CITY_LOCATION
                from COLLISIONS C
                         INNER JOIN POPULATION P ON P.ID =
C.POPULATION_ID
                where C.POPULATION_ID in
```

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 distinct (C.POPULATION_ID)
                          FROM COLLISIONS C
                          WHERE P.DEFINITION = 'Incorporated (over
250000)'
                      )
                    FETCH FIRST 3 ROWS ONLY
            )
        ),
    rws as (
        SELECT ROW_NUMBER() OVER (PARTITION BY
COUNTY_CITY_LOCATION
            ORDER BY V_AGE ASC ) AS Row_Number,
               COLLISION_CASE_ID,
               COUNTY_CITY_LOCATION,
               POPULATION_ID,
               V_AGE
        FROM average_age
    )
select COLLISION_CASE_ID, COUNTY_CITY_LOCATION, P.DEFINITION,
V_AGE as AVERAGE_VICTIM_AGE
from rws
        INNER JOIN POPULATION P ON P.ID = POPULATION_ID
where Row_Number <= 10
order by COUNTY_CITY_LOCATION, V_AGE asc;
```

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

COLLISION_CASE_ID	COUNTY_CITY_LOCATION	DEFINITION	AVERAGE_VICTIM_AGE
1838702	109	Incorporated (over 250000)	0
2727453	109	Incorporated (over 250000)	0
3486455	109	Incorporated (over 250000)	0

0059033	109	Incorporated (over 250000)	0
2295152	109	Incorporated (over 250000)	0
1336621	109	Incorporated (over 250000)	0
1231119	109	Incorporated (over 250000)	0
2737180	109	Incorporated (over 250000)	0
2506007	109	Incorporated (over 250000)	0
1377820	109	Incorporated (over 250000)	0
2715062	3019	Incorporated (over 250000)	0
2412373	3019	Incorporated (over 250000)	0
1994820	3019	Incorporated (over 250000)	0
1170908	3019	Incorporated (over 250000)	0

1825689	3019	Incorporated (over 250000)	0
3553649	3019	Incorporated (over 250000)	0
2072101	3019	Incorporated (over 250000)	0
2138547	3019	Incorporated (over 250000)	0
2674015	3019	Incorporated (over 250000)	0
3551315	3019	Incorporated (over 250000)	0

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 collisions of type pedestrian where all the victims were above 100 years old. We should then show only the `collision_id` and the `victim_age` of the oldest victim for each of them.

For this query, we first joined the 4 tables used (victims, parties, collisions and type of collisions) and kept only those that are of type pedestrian. We then grouped by the `case_id` and kept only the collisions where the minimum `victim_age` is above 100, to be sure that all victims were older than 100. We then returned the `case_id` and maximum `victim_age`.

Remark:

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

SQL statement

```
SELECT C.CASE_ID, MAX(V.VICTIM_AGE) AS AGE_MAX
FROM VICTIMS V, PARTIES P, COLLISIONS C, TYPE_OF_COLLISION TOC
WHERE V.PARTY_ID = P.ID
 AND P.COLLISION_CASE_ID = C.CASE_ID
 AND C.TYPE_OF_COLLISION_ID = TOC.ID
 AND TOC.DEFINITION = 'Vehicle/Pedestrian'
GROUP BY CASE_ID
HAVING MIN(V.VICTIM_AGE) > 100;
```

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

CASE_ID	AGE_MAX
2531557	103
0439197	102
1548445	102
1373664	101
1209166	101
1347636	101
0828116	102
0784061	102
1213340	121
0817210	102
0036446	110
3485436	101
0820619	101
0868472	103
1847678	104
0644226	103
0566220	102
3388544	105
2472739	103
0851026	106

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 8:

Description of logic:

This query should retrieve the vehicles which have participated in at least 10 collisions and their corresponding number of accidents. For the vehicle ID, see our assumptions.

For this query, we first join the parties and the vehicle types (to retrieve the definition). We then keep only the vehicles having not any null values (for the make, the year and the type) and group them together. We count them and only keep those that appear at least 10 times. We finally sort them in descending order.

We can observe that the type of the vehicles having the most collisions is always "Passenger Car/Station Wagon" which is quite logical since it represents the most common type. We can also observe that the make is always either TOYOTA, FORD or HONDA and the year between 1997 and 2002 for the top 20 vehicles in terms of collisions.

Remark:

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

SQL statement

```
SELECT SVT.DEFINITION, P.VEHICLE_MAKE, P.VEHICLE_YEAR, COUNT(*) AS
NUMBER_COLLISION
FROM PARTIES P,
    STATEWIDE_VEHICLE_TYPE SVT
WHERE P.STATEWIDE_VEHICLE_TYPE_ID IS NOT NULL
 AND P.VEHICLE_MAKE IS NOT NULL
 AND P.VEHICLE_YEAR IS NOT NULL
 AND P.STATEWIDE_VEHICLE_TYPE_ID = SVT.ID
GROUP BY (SVT.DEFINITION, P.VEHICLE_MAKE, P.VEHICLE_YEAR)
HAVING COUNT(*) >= 10
ORDER BY COUNT(*) DESC;
```

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

DEFINITION	VEHICLE_MAKE	VEHICLE_YEAR	NUMBER_COLLISION
Passenger Car/Station Wagon	TOYOTA	2000	52504
Passenger Car/Station Wagon	FORD	2000	51943
Passenger Car/Station Wagon	HONDA	2000	50284
Passenger Car/Station Wagon	FORD	1998	49182
Passenger Car/Station Wagon	TOYOTA	2001	47232
Passenger Car/Station Wagon	HONDA	2001	45277
Passenger Car/Station Wagon	FORD	2001	45236
Passenger Car/Station Wagon	TOYOTA	1999	42941
Passenger Car/Station Wagon	HONDA	1998	42091
Passenger Car/Station Wagon	FORD	1999	41948
Passenger Car/Station Wagon	FORD	1995	40246
Passenger Car/Station Wagon	HONDA	1997	39210
Passenger Car/Station Wagon	FORD	1997	38885
Passenger Car/Station Wagon	HONDA	1999	38556

Passenger Car/Station Wagon	TOYOTA	2002	38427
Passenger Car/Station Wagon	TOYOTA	1998	38012
Passenger Car/Station Wagon	TOYOTA	1997	37158
Passenger Car/Station Wagon	TOYOTA	2003	35943
Passenger Car/Station Wagon	HONDA	2002	35785
Passenger Car/Station Wagon	FORD	2002	35460

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 9:

Description of logic:

This query should retrieve the top 10 cities having the most collisions.

For that, we simply group by the county city location, count the number of entries. To retrieve the top 10 cities only, we sort our result in descending order and fetch the first 10 rows only.

SQL statement

```
SELECT COUNTY_CITY_LOCATION, COUNT(*) AS NUMBER_COLLISIONS
FROM COLLISIONS C
GROUP BY COUNTY_CITY_LOCATION
ORDER BY NUMBER_COLLISIONS DESC FETCH FIRST 10 ROWS ONLY;
```

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

COUNTY_CITY_LOCATION	NUMBER_COLLISIONS
1942	399582
1900	118446
3400	80191
3711	76867
109	72995
3300	61453
3404	58068
4313	57852
1941	53565
3801	48450

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 10:

Description of logic:

This query should retrieve the number of accidents for different time periods with different lighting conditions.

For this query, we decided to first take into account the lighting conditions that were not ambiguous, i.e. daylight for day and everything containing dark for night. For the dusk/dawn category, we looked at the time and month and the information given in the question to put them in the right category. We decided to discard all the data that was not consistent (for example if an accident had lighting 'Dusk-Dawn' but was at time 12:00 which is neither dusk nor dawn, we dropped it).

If the lighting condition was not given (null), we tried to infer the period based on the time only when it was possible or the time and the date when both were available.

SQL statement

```
SELECT TIME_PERIOD, COUNT(*) as NUMBER_ACCIDENT
FROM (
   SELECT CASE
       when l.DEFINITION = 'Daylight' then 'DAY_COLLISIONS'
       when l.DEFINITION like '%dark%' then 'NIGHT_COLLISIONS'
       when l.DEFINITION = 'Dusk - Dawn' then
           case
               when C.COLLISION_DATE is not null then
                   case
                       WHEN ((EXTRACT(MONTH FROM C.COLLISION_DATE)
BETWEEN '4' AND '8'
                               AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '20' AND '21')
                           OR (EXTRACT(MONTH FROM
C.COLLISION_DATE) NOT BETWEEN '4' AND '8'
                               AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '18' AND '19'))
                           THEN 'DUSK_COLLISIONS'
                       WHEN ((EXTRACT(MONTH FROM C.COLLISION_DATE)
BETWEEN '4' AND '8'
                               AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '4' AND '5')
                           OR (EXTRACT(MONTH FROM
C.COLLISION_DATE) NOT BETWEEN '4' AND '8'
                               AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '6' AND '7'))
                           THEN 'DAWN_COLLISIONS'
```

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/

```
                    end
            end

        else
            case
                when C.COLLISION_DATE is not null then
                    CASE
                        WHEN ((EXTRACT(MONTH FROM C.COLLISION_DATE)
BETWEEN '4' AND '8'
                                AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '20' AND '21')
                            OR (EXTRACT(MONTH FROM
C.COLLISION_DATE) NOT BETWEEN '4' AND '8'
                                AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '18' AND '19'))
                            THEN 'DUSK_COLLISIONS'
                        WHEN ((EXTRACT(MONTH FROM C.COLLISION_DATE)
BETWEEN '4' AND '8'
                                AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '4' AND '5')
                            OR (EXTRACT(MONTH FROM
C.COLLISION_DATE) NOT BETWEEN '4' AND '8'
                                AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '6' AND '7'))
                            THEN 'DAWN_COLLISIONS'
                        WHEN (EXTRACT(MONTH FROM C.COLLISION_DATE)
BETWEEN '4' AND '8'
                                AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '6' AND '19')
                            OR (EXTRACT(MONTH FROM
C.COLLISION_DATE) NOT BETWEEN '4' AND '8'
                                AND EXTRACT(HOUR FROM
C.COLLISION_TIME) BETWEEN '8' AND '17')
                            THEN 'DAY_COLLISIONS'
                        ELSE 'NIGHT_COLLISIONS'
                    end
            else
                case
                    when extract(hour from C.COLLISION_TIME) > 7
                        and extract(hour from C.COLLISION_TIME) <
18 then 'DAY_COLLISIONS'
                    when extract(hour from C.COLLISION_TIME) < 4
                        and extract(hour from C.COLLISION_TIME) >
21 then 'NIGHT_COLLISIONS'
                end
```

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/

```
           end
       end as TIME_PERIOD
   FROM COLLISIONS C
   left outer join LIGHTING L on C.LIGHTING_ID = L.ID
    )
where TIME_PERIOD is not null
GROUP BY TIME_PERIOD
ORDER BY NUMBER_ACCIDENT DESC;
```

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

TIME_PERIOD	NUMBER_ACCIDENT
DAY_COLLISIONS	2607362
NIGHT_COLLISIONS	628870
DUSK_COLLISIONS	305720
DAWN_COLLISIONS	64534

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 Performance Analysis – Indexing

We observed that the running time for a query varies for each run. Therefore we took the mean of five runs for the initial and optimized time. However, as time can vary a lot between different runs, we tried to rather use the cost of the plan to gain more insightful information about the quality of our optimization.

Query 1

Initial time: 4,5s

Optimized time: 2,5s

Explain the improvement:

We created the following 2 indexes:

```
CREATE INDEX PARTIES_IDX_PARTY_AGE on PARTIES(PARTY_AGE);
CREATE INDEX PARTIES_IDX_AT_FAULT_PARTY_AGE on PARTIES(AT_FAULT,
PARTY_AGE);
```

In this query we only access the parties table, once over the `party_age` and once over the `party_age` and the `at_fault` attributes. So it makes perfect sense to create one index for each of these accesses and we can indeed see on the optimized plan that both the `TABLE ACCESS FULL` have been replaced by `INDEX FAST FULL SCAN`. The latter means that all the needed attributes were present in the index. It only reads the index block by block and not the full table thus reducing the amount of IO.

Initial plan:

```
--------------------------------------------------------------------------------
| Id  | Operation               | Name     | Rows   | Bytes | Cost  (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |          |   112 |   4256 | 60028    (1)| 00:00:03 |
|   1 |  SORT ORDER BY          |          |   112 |   4256 | 60028    (1)| 00:00:03 |
|*  2 |   HASH JOIN             |          |   112 |   4256 | 60027    (1)| 00:00:03 |
|   3 |    VIEW                 |          |   106 |   2014 | 29997    (1)| 00:00:02 |
|   4 |     HASH GROUP BY       |          |   106 |    530 | 29997    (1)| 00:00:02 |
|*  5 |      TABLE ACCESS FULL| PARTIES |  2808K|    13M| 29927    (1)| 00:00:02 |
|   6 |    VIEW                 |          |   106 |   2014 | 30030    (1)| 00:00:02 |
|   7 |     HASH GROUP BY       |          |   106 |    318 | 30030    (1)| 00:00:02 |
|*  8 |      TABLE ACCESS FULL| PARTIES |  6188K|    17M| 29868    (1)| 00:00:02 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

"   2 - access(""TOTAL"".""AGE_RANGE""=""FAULT"".""AGE_RANGE"")"
"   5 - filter(""P"".""PARTY_AGE"" IS NOT NULL AND ""P"".""AT_FAULT""='T')"
"   8 - filter(""P"".""PARTY_AGE"" IS NOT NULL)"
```

Improved plan:

```
--------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                              | Rows   | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                                   |   112  |  4256 |  7926   (5)| 00:00:01 |
|   1 |  SORT ORDER BY             |                                   |   112  |  4256 |  7926   (5)| 00:00:01 |
|*  2 |   HASH JOIN                |                                   |   112  |  4256 |  7925   (5)| 00:00:01 |
|   3 |    VIEW                    |                                   |   106  |  2014 |  4463   (4)| 00:00:01 |
|   4 |     HASH GROUP BY          |                                   |   106  |   530 |  4463   (4)| 00:00:01 |
|*  5 |      INDEX FAST FULL SCAN| PARTIES_IDX_AT_FAULT_PARTY_AGE |  2808K|    13M|  4393   (2)| 00:00:01 |
|   6 |    VIEW                    |                                   |   106  |  2014 |  3461   (6)| 00:00:01 |
|   7 |     HASH GROUP BY          |                                   |   106  |   318 |  3461   (6)| 00:00:01 |
|*  8 |      INDEX FAST FULL SCAN| PARTIES_IDX_PARTY_AGE             |  6188K|    17M|  3299   (1)| 00:00:01 |
--------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------


"   2 - access(""TOTAL"".""AGE_RANGE""=""FAULT"".""AGE_RANGE"")"
"   5 - filter(""P"".""PARTY_AGE"" IS NOT NULL AND ""P"".""AT_FAULT""='T')"
"   8 - filter(""P"".""PARTY_AGE"" IS NOT NULL)"
```

Query 2

Initial time: 1,5s

Optimized time: 2,2s

Explain the improvement:

We created the following 3 indexes:

```sql
CREATE INDEX
PARTIES_IDX_COLLISION_CASE_ID_STATEWIDE_VEHICLE_TYPE_ID on
PARTIES(STATEWIDE_VEHICLE_TYPE_ID, COLLISION_CASE_ID);
CREATE INDEX ROAD_CONDITION_IDX_DEFINITION_ID on
ROAD_CONDITION(DEFINITION, ID);
CREATE INDEX STATEWIDE_VEHICLE_TYPE_IDX_DEFINITION_ID on
STATEWIDE_VEHICLE_TYPE(DEFINITION, ID);
```

We created an index for every group of attributes per accessed table. An index on `COLLISION_WITH_ROAD_CONDITION` was not needed since the accessed tuple is the primary key of the table so it's already clustered on that index. Except for this case all the other `TABLE ACCESS FULL`, on the improved plan have been replaced by `INDEX FAST FULL SCAN`. The latter means that all the needed attributes were present in the index. It only reads the index block by block and not the full table thus reducing the amount of IO.

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/

Initial plan:

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                                |     4 |   252 |       | 65994   (1)| 00:00:03 |
|   1 |  SORT ORDER BY                 |                                |     4 |   252 |       | 65994   (1)| 00:00:03 |
|   2 |   MERGE JOIN                   |                                |     4 |   252 |       | 65993   (1)| 00:00:03 |
|   3 |    TABLE ACCESS BY INDEX ROWID| STATEWIDE_VEHICLE_TYPE          |    15 |   330 |       |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN            | SYS_C00207107                  |    15 |       |       |     1   (0)| 00:00:01 |
|*  5 |    SORT JOIN                   |                                |     5 |   205 |       | 65991   (1)| 00:00:03 |
|*  6 |     VIEW                       |                                |     5 |   205 |       | 65990   (1)| 00:00:03 |
|*  7 |      WINDOW SORT PUSHED RANK   |                                |    15 |  2325 |       | 65990   (1)| 00:00:03 |
|   8 |       HASH GROUP BY            |                                |    15 |  2325 |       | 65990   (1)| 00:00:03 |
|*  9 |        HASH JOIN               |                                |   806K|   119M|    43M| 65951   (1)| 00:00:03 |
|* 10 |         HASH JOIN              |                                |   456K|    38M|       |  9954   (1)| 00:00:01 |
|* 11 |          TABLE ACCESS FULL     | ROAD_CONDITION                 |     1 |    21 |       |     3   (0)| 00:00:01 |
|  12 |          TABLE ACCESS FULL     | COLLISION_WITH_ROAD_CONDITION  |  3652K|   233M|       |  9942   (1)| 00:00:01 |
|* 13 |         TABLE ACCESS FULL      | PARTIES                        |  6400K|   408M|       | 29906   (1)| 00:00:02 |
-------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------


"   5 - access(""SWT"".""ID""=""from$_subquery$_006"".""SVT_ID"")"
"       filter(""SWT"".""ID""=""from$_subquery$_006"".""SVT_ID"")"
"   6 - filter(""from$_subquery$_006"".""rowlimit_$$_rownumber""<=5)"
   7 - filter(ROW_NUMBER() OVER ( ORDER BY COUNT(*) DESC )<=5)
"   9 - access(""P"".""COLLISION_CASE_ID""=""CWRC"".""CASE_ID"")"
"  10 - access(""CWRC"".""ROAD_CONDITION_ID""=""RC"".""ID"")"
"  11 - filter(""RC"".""DEFINITION""='Holes, Deep Ruts')"
"  13 - filter(""P"".""STATEWIDE_VEHICLE_TYPE_ID"" IS NOT NULL)"
```

Improved plan:

```
-------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                                                 | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                                                      |     4 |   252 |       | 57549   (1)| 00:00:03 |
|   1 |  SORT ORDER BY               |                                                      |     4 |   252 |       | 57549   (1)| 00:00:03 |
|*  2 |   HASH JOIN                  |                                                      |     4 |   252 |       | 57548   (1)| 00:00:03 |
|*  3 |    VIEW                      |                                                      |     5 |   205 |       | 57547   (1)| 00:00:03 |
|*  4 |     WINDOW SORT PUSHED RANK|                                                        |    15 |  2325 |       | 57547   (1)| 00:00:03 |
|   5 |      HASH GROUP BY           |                                                      |    15 |  2325 |       | 57547   (1)| 00:00:03 |
|*  6 |       HASH JOIN              |                                                      |   806K|   119M|    43M| 57508   (1)| 00:00:03 |
|*  7 |        HASH JOIN             |                                                      |   456K|    38M|       |  9952   (1)| 00:00:01 |
|*  8 |         INDEX RANGE SCAN     | ROAD_CONDITION_IDX_DEFINITION_ID                     |     1 |    21 |       |     1   (0)| 00:00:01 |
|   9 |         TABLE ACCESS FULL    | COLLISION_WITH_ROAD_CONDITION                        |  3652K|   233M|       |  9942   (1)| 00:00:01 |
|* 10 |        INDEX FAST FULL SCAN| PARTIES_IDX_COLLISION_CASE_ID_STATEWIDE_VEHICLE_TYPE_ID |  6400K|   408M|       | 21465   (1)| 00:00:01 |
|  11 |    INDEX FULL SCAN           | STATEWIDE_VEHICLE_TYPE_IDX_DEFINITION_ID             |    15 |   330 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------


"   2 - access(""SWT"".""ID""=""from$_subquery$_006"".""SVT_ID"")"
"   3 - filter(""from$_subquery$_006"".""rowlimit_$$_rownumber""<=5)"
   4 - filter(ROW_NUMBER() OVER ( ORDER BY COUNT(*) DESC )<=5)
"   6 - access(""P"".""COLLISION_CASE_ID""=""CWRC"".""CASE_ID"")"
"   7 - access(""CWRC"".""ROAD_CONDITION_ID""=""RC"".""ID"")"
"   8 - access(""RC"".""DEFINITION""='Holes, Deep Ruts')"
"  10 - filter(""P"".""STATEWIDE_VEHICLE_TYPE_ID"" IS NOT NULL)"
```

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 3

Initial time: 4s

Optimized time: 1,7s

Explain the improvement:

We created the following 4 indexes:

```
CREATE INDEX PARTIES_IDX_VEHICLE_MAKE on PARTIES(VEHICLE_MAKE);
CREATE INDEX PARTIES_IDX_ID_VEHICLE_MAKE on PARTIES(VEHICLE_MAKE,
ID);
CREATE INDEX VICTIMS_IDX_PARTY_ID_VICTIM_DEGREE_OF_INJURY_ID on
VICTIMS(VICTIM_DEGREE_OF_INJURY_ID, PARTY_ID);
CREATE INDEX VICTIM_DEGREE_OF_INJURY_IDX_DEFINITION_ID on
VICTIM_DEGREE_OF_INJURY(DEFINITION, ID);
```

In this query, we access the party table once over the `vehicle_make` and the `id` and therefore our second index on those attributes improves the plan by replacing the `TABLE ACCESS FULL` with an `INDEX FAST FULL SCAN.` The latter means that all the needed attributes were present in the index. It only reads the index block by block and not the full table thus reducing the amount of IO.

The same improvement happens for the table victims and the attributes `VICTIM_DEGREE_OF_INJURY_ID` and `PARTY_ID` thanks to our third index. The victim degree of injury table is also accessed over its `id` and `definition` and therefore it made sense to use an index, which transforms the `TABLE ACCESS FULL` into an `INDEX RANGE SCAN.` Our first index doesn't change the plan, but it improved the cost a bit, probably because it is used during the group by.

Initial plan:

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                           |    10 |  1160 |       | 45925   (1)| 00:00:02 |
|   1 |  SORT ORDER BY               |                           |    10 |  1160 |       | 45925   (1)| 00:00:02 |
|*  2 |   VIEW                       |                           |    10 |  1160 |       | 45924   (1)| 00:00:02 |
|*  3 |    WINDOW SORT PUSHED RANK|                              |   209 |  8778 |       | 45924   (1)| 00:00:02 |
|   4 |     HASH GROUP BY            |                           |   209 |  8778 |       | 45924   (1)| 00:00:02 |
|*  5 |      HASH JOIN               |                           | 1360K|    54M|    53M| 45858   (1)| 00:00:02 |
|*  6 |       HASH JOIN              |                           | 1360K|    37M|       |  5283   (2)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL     | VICTIM_DEGREE_OF_INJURY   |     2 |    40 |       |     3   (0)| 00:00:01 |
|*  8 |        TABLE ACCESS FULL     | VICTIMS                   | 4082K|    35M|       |  5269   (1)| 00:00:01 |
|*  9 |       TABLE ACCESS FULL      | PARTIES                   | 6759K|    83M|       | 29909   (1)| 00:00:02 |
-----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
```

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/

```
"   2 - filter(""from$_subquery$_004"".""rowlimit_$$_rownumber""<=10)"
  3 - filter(ROW_NUMBER() OVER ( ORDER BY COUNT(*) DESC )<=10)
"   5 - access(""P"".""ID""=""V"".""PARTY_ID"")"
"   6 - access(""V"".""VICTIM_DEGREE_OF_INJURY_ID""=""VDOI"".""ID"")"
"   7 - filter(""VDOI"".""DEFINITION""='Killed' OR ""VDOI"".""DEFINITION""='Severe Injury')"
"   8 - filter(""V"".""VICTIM_DEGREE_OF_INJURY_ID"">=0 AND ""V"".""VICTIM_DEGREE_OF_INJURY_ID""<=7)"
"   9 - filter(""P"".""VEHICLE_MAKE"" IS NOT NULL)"
```

Improved plan:

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name                                                | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                                                     |    10 |  1160 |       | 20343   (1)| 00:00:01 |
|   1 |  SORT ORDER BY              |                                                     |    10 |  1160 |       | 20343   (1)| 00:00:01 |
|*  2 |   VIEW                      |                                                     |    10 |  1160 |       | 20342   (1)| 00:00:01 |
|*  3 |    WINDOW SORT PUSHED RANK  |                                                     |   209 |  8778 |       | 20342   (1)| 00:00:01 |
|   4 |     HASH GROUP BY           |                                                     |   209 |  8778 |       | 20342   (1)| 00:00:01 |
|*  5 |      HASH JOIN              |                                                     |  1360K|    54M|    53M| 20276   (1)| 00:00:01 |
|*  6 |       HASH JOIN             |                                                     |  1360K|    37M|       |  3021   (2)| 00:00:01 |
|   7 |        INLIST ITERATOR      |                                                     |       |       |       |            |          |
|*  8 |         INDEX RANGE SCAN    | VICTIM_DEGREE_OF_INJURY_IDX_DEFINITION_ID           |     2 |    40 |       |     1   (0)| 00:00:01 |
|*  9 |        INDEX FAST FULL SCAN| VICTIMS_IDX_PARTY_ID_VICTIM_DEGREE_OF_INJURY_ID |  4082K|    35M|       |  3010   (1)| 00:00:01 |
|* 10 |       INDEX FAST FULL SCAN | PARTIES_IDX_ID_VEHICLE_MAKE                         |  6759K|    83M|       |  6589   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------------------------


Predicate Information (identified by operation id):
---------------------------------------------------


"   2 - filter(""from$_subquery$_004"".""rowlimit_$$_rownumber""<=10)"
  3 - filter(ROW_NUMBER() OVER ( ORDER BY COUNT(*) DESC )<=10)
"   5 - access(""P"".""ID""=""V"".""PARTY_ID"")"
"   6 - access(""V"".""VICTIM_DEGREE_OF_INJURY_ID""=""VDOI"".""ID"")"
"   8 - access(""VDOI"".""DEFINITION""='Killed' OR ""VDOI"".""DEFINITION""='Severe Injury')"
"   9 - filter(""V"".""VICTIM_DEGREE_OF_INJURY_ID"">=0 AND ""V"".""VICTIM_DEGREE_OF_INJURY_ID""<=7)"
"  10 - filter(""P"".""VEHICLE_MAKE"" IS NOT NULL)"
```

Query 5

Initial time: 3 min 20 s

Optimized time: 55s

Explain the improvement:

We created the following 2 indexes:

```
CREATE INDEX
PARTIES_IDX_COLLISION_CASE_ID_STATEWIDE_VEHICLE_TYPE_ID on
PARTIES(STATEWIDE_VEHICLE_TYPE_ID, COLLISION_CASE_ID);
CREATE INDEX COLLISIONS_IDX_CASE_ID_COUNTY_CITY_LOCATION on
COLLISIONS(COUNTY_CITY_LOCATION, CASE_ID);
```

By creating an index on `(STATEWIDE_VEHICLE_TYPE_ID, COLLISION_CASE_ID)` and on `(COUNTY_CITY_LOCATION, CASE_ID)`, we are able to replace all the `TABLE ACCESS FULL` by `INDEX FAST FULL SCAN`. The latter means that all the needed attributes were present in the index. It only reads the index block by block and not the full table thus reducing the amount of IO.

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/

Initial plan:

```
----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |     1 |       |       | 87192   (1)| 00:00:04 |
|   1 |  SORT AGGREGATE              |              |     1 |       |       |            |          |
|   2 |   VIEW                       |              |     1 |       |       | 87192   (1)| 00:00:04 |
|*  3 |    FILTER                    |              |       |       |       |            |          |
|   4 |     HASH GROUP BY            |              |     1 |     2 |       | 87192   (1)| 00:00:04 |
|   5 |      VIEW                    |              |   287 |   574 |       | 87192   (1)| 00:00:04 |
|*  6 |       FILTER                 |              |       |       |       |            |          |
|   7 |        HASH GROUP BY         |              |   287 | 39032 |       | 87192   (1)| 00:00:04 |
|*  8 |         HASH JOIN            |              | 6400K|   830M|   284M| 87024   (1)| 00:00:04 |
|*  9 |          TABLE ACCESS FULL| COLLISIONS |  3678K|   242M|       | 19090   (1)| 00:00:01 |
|* 10 |          TABLE ACCESS FULL| PARTIES    |  6400K|   408M|       | 29906   (1)| 00:00:02 |
|  11 |     SORT AGGREGATE           |              |     1 |    13 |       |            |          |
|  12 |      VIEW                    | VM_NWVW_1    |   540 |  7020 |       | 19182   (1)| 00:00:01 |
|  13 |       SORT GROUP BY          |              |   540 |  2160 |       | 19182   (1)| 00:00:01 |
|  14 |        TABLE ACCESS FULL     | COLLISIONS |  3678K|    14M|       | 19088   (1)| 00:00:01 |
----------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

"   3 - filter(COUNT(*)>= (SELECT COUNT(""$vm_col_1"")/2 FROM  (SELECT "
"              ""C"".""COUNTY_CITY_LOCATION"" ""$vm_col_1"" FROM ""COLLISIONS"" ""C"" GROUP BY "
"              ""C"".""COUNTY_CITY_LOCATION"") ""VM_NWVW_1""))"
  6 - filter(COUNT(*)>=10)
"   8 - access(""P"".""COLLISION_CASE_ID""=""C"".""CASE_ID"")"
"   9 - filter(""C"".""COUNTY_CITY_LOCATION"" IS NOT NULL)"
"  10 - filter(""P"".""STATEWIDE_VEHICLE_TYPE_ID"" IS NOT NULL)"
```

Improved plan:

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                                                | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                                                     |     1 |       |       | 70847   (1)| 00:00:03 |
|   1 |  SORT AGGREGATE              |                                                     |     1 |       |       |            |          |
|   2 |   VIEW                       |                                                     |     1 |       |       | 70847   (1)| 00:00:03 |
|*  3 |    FILTER                    |                                                     |       |       |       |            |          |
|   4 |     HASH GROUP BY            |                                                     |     1 |     2 |       | 70847   (1)| 00:00:03 |
|   5 |      VIEW                    |                                                     |   287 |   574 |       | 70847   (1)| 00:00:03 |
|*  6 |       FILTER                 |                                                     |       |       |       |            |          |
|   7 |        HASH GROUP BY         |                                                     |   287 | 39032 |       | 70847   (1)| 00:00:03 |
|*  8 |         HASH JOIN            |                                                     | 6400K|   830M|   284M| 70679   (1)| 00:00:03 |
|*  9 |          INDEX FAST FULL SCAN| COLLISIONS_IDX_CASE_ID_COUNTY_CITY_LOCATION         | 3678K|   242M|       | 11187   (1)| 00:00:01 |
|* 10 |          INDEX FAST FULL SCAN| PARTIES_IDX_COLLISION_CASE_ID_STATEWIDE_VEHICLE_TYPE_ID | 6400K|   408M|       | 21465   (1)| 00:00:01 |
|  11 |     SORT AGGREGATE           |                                                     |     1 |    13 |       |            |          |
|  12 |      VIEW                    | VM_NWVW_1                                           |   540 |  7020 |       | 11278   (1)| 00:00:01 |
|  13 |       SORT GROUP BY          |                                                     |   540 |  2160 |       | 11278   (1)| 00:00:01 |
|  14 |        INDEX FAST FULL SCAN  | COLLISIONS_IDX_CASE_ID_COUNTY_CITY_LOCATION         | 3678K|    14M|       | 11185   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

"   3 - filter(COUNT(*)>= (SELECT COUNT(""$vm_col_1"")/2 FROM  (SELECT ""C"".""COUNTY_CITY_LOCATION"" ""$vm_col_1"" FROM ""COLLISIONS"" ""C"" GROUP "
"              BY ""C"".""COUNTY_CITY_LOCATION"") ""VM_NWVW_1""))"
  6 - filter(COUNT(*)>=10)
"   8 - access(""P"".""COLLISION_CASE_ID""=""C"".""CASE_ID"")"
"   9 - filter(""C"".""COUNTY_CITY_LOCATION"" IS NOT NULL)"
"  10 - filter(""P"".""STATEWIDE_VEHICLE_TYPE_ID"" IS NOT NULL)"
```

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

Initial time: 23,47s

Optimized time: 8,96s

Explain the improvement:

We created the following 4 indexes:

```
CREATE INDEX VICTIMS_IDX_PARTY_ID_VICTIM_AGE on
VICTIMS(VICTIM_AGE, PARTY_ID);
CREATE INDEX PARTIES_IDX_COLLISION_CASE_ID_ID on
PARTIES(COLLISION_CASE_ID, ID);
CREATE INDEX COLLISIONS_IDX_CASE_ID_TYPE_OF_COLLISION_ID on
COLLISIONS(TYPE_OF_COLLISION_ID, CASE_ID);
CREATE INDEX TYPE_OF_COLLISION_IDX_DEFINITION_ID on
TYPE_OF_COLLISION(DEFINITION, ID);
```

In the query we access the three main tables so creating an index for each of these accesses greatly improves the runtime. We also created an index on `(TYPE_OF_COLLISION)` since it's a rather small table it doesn't reduce the cost much, but it's still an improvement. By creating an index on parties we avoid a `TABLE ACCESS FULL` and replace it by an `INDEX FAST FULL SCAN`. The latter means that all the needed attributes were present in the index. It only reads the index block by block and not the full table thus reducing the amount of IO.

Initial plan:

```
----------------------------------------------------------------------------------------------------
| Id  | Operation               | Name              | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |                   | 25334 |  3958K|       |   102K  (1)| 00:00:05 |
|*  1 |  FILTER                 |                   |       |       |       |            |          |
|   2 |   HASH GROUP BY         |                   | 25334 |  3958K|   82M|   102K  (1)| 00:00:05 |
|*  3 |    HASH JOIN            |                   |   506K|    77M|   81M| 95864   (1)| 00:00:04 |
|   4 |     TABLE ACCESS FULL   | VICTIMS           |  4082K|    35M|       |  5259   (1)| 00:00:01 |
|*  5 |     HASH JOIN           |                   |   904K|   130M|   40M| 79565   (1)| 00:00:04 |
|*  6 |      HASH JOIN          |                   |   456K|    34M|       | 19082   (1)| 00:00:01 |
|*  7 |       TABLE ACCESS FULL| TYPE_OF_COLLISION |     1 |    13 |       |     3   (0)| 00:00:01 |
|   8 |       TABLE ACCESS FULL| COLLISIONS        |  3678K|   235M|       | 19069   (1)| 00:00:01 |
|   9 |      TABLE ACCESS FULL | PARTIES           |  7286K|   493M|       | 29872   (1)| 00:00:02 |
----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

"   1 - filter(MIN(""V"".""VICTIM_AGE"")>100)"
"   3 - access(""V"".""PARTY_ID""=""P"".""ID"")"
"   5 - access(""P"".""COLLISION_CASE_ID""=""C"".""CASE_ID"")"
```

```
"    6 - access(""C"".""TYPE_OF_COLLISION_ID""=""TOC"".""ID"")"
"    7 - filter(""TOC"".""DEFINITION""='Vehicle/Pedestrian')"
```

Improved plan:

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                                         | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                                              | 25334 |  3958K|       | 86094   (1)| 00:00:04 |
|   1 |  SORT ORDER BY             |                                              | 25334 |  3958K|    82M| 86094   (1)| 00:00:04 |
|*  2 |   FILTER                   |                                              |       |       |       |            |          |
|   3 |    HASH GROUP BY           |                                              | 25334 |  3958K|    82M| 86094   (1)| 00:00:04 |
|*  4 |     HASH JOIN              |                                              |   506K|    77M|    81M| 72404   (1)| 00:00:03 |
|   5 |      INDEX FAST FULL SCAN  | VICTIMS_IDX_PARTY_ID_VICTIM_AGE              |  4082K|    35M|       |  3063   (1)| 00:00:01 |
|*  6 |      HASH JOIN             |                                              |   904K|   130M|    40M| 58301   (1)| 00:00:03 |
|   7 |       NESTED LOOPS         |                                              |   456K|    34M|       |  5002   (1)| 00:00:01 |
|*  8 |        INDEX RANGE SCAN    | TYPE_OF_COLLISION_IDX_DEFINITION_ID          |     1 |    13 |       |     1   (0)| 00:00:01 |
|*  9 |        INDEX RANGE SCAN    | COLLISIONS_IDX_CASE_ID_TYPE_OF_COLLISION_ID  |   456K|    29M|       |  5001   (1)| 00:00:01 |
|  10 |       INDEX FAST FULL SCAN| PARTIES_IDX_COLLISION_CASE_ID_ID             |  7286K|   493M|       | 22688   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------------


Predicate Information (identified by operation id):
---------------------------------------------------


"   2 - filter(MIN(""V"".""VICTIM_AGE"")>100)"
"   4 - access(""V"".""PARTY_ID""=""P"".""ID"")"
"   6 - access(""P"".""COLLISION_CASE_ID""=""C"".""CASE_ID"")"
"   8 - access(""TOC"".""DEFINITION""='Vehicle/Pedestrian')"
"   9 - access(""C"".""TYPE_OF_COLLISION_ID""=""TOC"".""ID"")"
```

Query 8

Initial time: 3,5s

Optimized time: 3,24s

Explain the improvement:

We created the following 2 indexes:

```sql
CREATE INDEX
PARTIES_IDX_STATEWIDE_VEHICLE_TYPE_ID_VEHICLE_MAKE_VEHICLE_YEAR on
PARTIES(STATEWIDE_VEHICLE_TYPE_ID, VEHICLE_YEAR, VEHICLE_MAKE);
CREATE INDEX STATEWIDE_VEHICLE_TYPE_IDX_DEFINITION_ID on
STATEWIDE_VEHICLE_TYPE(DEFINITION, ID);
```

In this query, we access the party table over the `vehicle_make`, `vehicle_year` and the `statewide_vehicle_type_id` together and therefore our first index on those attributes improves the plan by replacing the `TABLE ACCESS FULL` with an `INDEX FAST FULL SCAN`. Our second index is on the statewide vehicle type table where we created an index on the `id` and the `definition` which are accessed in the same where clause. This index transforms the `TABLE ACCESS FULL` into an `INDEX FULL SCAN`. The latter means that all the needed attributes were present in the index. It only reads the index block by block and not the full table thus reducing the amount of IO.

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/

Initial plan:

```
-----------------------------------------------------------------------------------------------
| Id  | Operation            | Name                    | Rows  | Bytes | Cost  (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                         |  8935 |   305K| 30218   (2)| 00:00:02 |
|   1 |  SORT ORDER BY       |                         |  8935 |   305K| 30218   (2)| 00:00:02 |
|*  2 |   FILTER             |                         |       |       |            |          |
|   3 |    HASH GROUP BY     |                         |  8935 |   305K| 30218   (2)| 00:00:02 |
|*  4 |     HASH JOIN        |                         |  5415K|   180M| 29936   (1)| 00:00:02 |
|   5 |      TABLE ACCESS FULL| STATEWIDE_VEHICLE_TYPE |    15 |   330 |     3   (0)| 00:00:01 |
|*  6 |      TABLE ACCESS FULL| PARTIES                |  5415K|    67M| 29919   (1)| 00:00:02 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------


  2 - filter(COUNT(*)>=10)
"   4 - access(""P"".""STATEWIDE_VEHICLE_TYPE_ID""=""SVT"".""ID"")"
"   6 - filter(""P"".""STATEWIDE_VEHICLE_TYPE_ID"" IS NOT NULL AND ""P"".""VEHICLE_YEAR"" IS NOT "
"              NULL AND ""P"".""VEHICLE_MAKE"" IS NOT NULL)"
```

Improved plan:

```
--------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name                                                          | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                                                               | 12466 |   426K|  6730   (5)| 00:00:01 |
|   1 |  SORT ORDER BY         |                                                               | 12466 |   426K|  6730   (5)| 00:00:01 |
|*  2 |   FILTER               |                                                               |       |       |            |          |
|   3 |    HASH GROUP BY       |                                                               | 12466 |   426K|  6730   (5)| 00:00:01 |
|*  4 |     HASH JOIN          |                                                               |  5415K|   180M|  6449   (1)| 00:00:01 |
|   5 |      INDEX FULL SCAN   | STATEWIDE_VEHICLE_TYPE_IDX_DEFINITION_ID                      |    15 |   330 |     1   (0)| 00:00:01 |
|*  6 |      INDEX FAST FULL SCAN| PARTIES_IDX_STATEWIDE_VEHICLE_TYPE_ID_VEHICLE_MAKE_VEHICLE_YEAR |  5415K|    67M|  6434   (1)| 00:00:01 |
--------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------


  2 - filter(COUNT(*)>=10)
"   4 - access(""P"".""STATEWIDE_VEHICLE_TYPE_ID""=""SVT"".""ID"")"
"   6 - filter(""P"".""STATEWIDE_VEHICLE_TYPE_ID"" IS NOT NULL AND ""P"".""VEHICLE_YEAR"" IS NOT NULL AND ""P"".""VEHICLE_MAKE"" IS NOT NULL)"
```

General Comments

The queries for the last milestone were more complicated and took us quite a long time, but we managed to finish them on time.
The time split between the members was almost equal and we all participated in every task during the whole semester.