Databases Project – Spring 2021

Team No: 65

Members: Zad ABI FADEL, Loïc HOUMARD, Jonas BLANC

Contents

Contents 1

Deliverable 1 2

Assumptions 2

Entity Relationship Schema 2

Schema 2

Description 2

Relational Schema 2

ER schema to Relational schema 2

DDL 2

General Comments 2

Deliverable 2 3

Assumptions 3

Data Loading/Cleaning 3

Query Implementation 3

General Comments 3

Deliverable 3 4

Assumptions 4

Query Implementation 4

Query Performance Analysis – Indexing 4

General Comments 4

# 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.

## Entity Relationship Schema

### Schema

Diagram

Description automatically generated

### 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.

## Relational Schema

### ER schema to Relational schema

Diagram, schematic

Description automatically generated

### 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  (  **id         char**(1), *-- check if if is one of letter*  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Type\_of\_collision  (  **id         char**(1), *--check char between a & h*  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Collision\_severity  (  **id         int CHECK** (0 <= **id and id** <= 4),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Hit\_and\_run  (  **id         char**(1),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Primary\_collision\_factor  (  **id         char**(1),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Pcf\_violation\_category  (  **id         int CHECK** ((0 <= **id and id** <= 24)),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Ramp\_intersection  (  **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**)  );  **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**)  );  **CREATE TABLE** Party\_sobriety  (  **id         char**(1),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Party\_type  (  **id         int**,  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Statewide\_vehicle\_type  (  **id         char**(1),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Cellphone\_use  (  **id         char**(1),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  *-- Relations with party: Many to many*  **CREATE TABLE** Other\_associated\_factor  (  **id         char**(1),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Financial\_responsibility  (  **id         char**(1),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  *-- Parties*  **CREATE TABLE** Parties  (  **id                              int**,  *-- Attributes*  **hazardous\_materials             char**(1),  **party\_age                       int**,  **party\_sex                       char**(1),  *-- relation to collision*  **collision\_case\_id               char**(64) **not null references** Collisions (**case\_id**),  **financial\_responsibility\_id     char**(1) **references** Financial\_responsibility (**id**),  **school\_bus\_related              char**(1),  **at\_fault                        char**(1)  **not null**,  *-- referenced ids*  **movement\_preceding\_collision\_id char**(1) **references** Movement\_preceding\_collision (**id**),  **party\_drug\_physical\_id          char**(1) **references** Party\_drug\_physical (**id**),  **party\_sobriety\_id               char**(1) **references** Party\_sobriety (**id**),  **party\_type\_id                   int references** Party\_type (**id**),  **statewide\_vehicle\_type\_id       char**(1) **references** Statewide\_vehicle\_type (**id**),  **vehicle\_make                    varchar**(150),  **vehicle\_year                    int**,  **cellphone\_use\_id                char**(1) **default 'D' references** Cellphone\_use (**id**), *--default 'D'  makes it faster*  *-- key*  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Party\_equipped\_with\_safety\_equipment  (  **party\_id            int     not null references** Parties (**id**) **on delete cascade**,  **safety\_equipment\_id char**(1) **not null references** Safety\_equipment (**id**) **on delete cascade**,  **PRIMARY KEY** (**party\_id**, **safety\_equipment\_id**)  );  **CREATE TABLE** Party\_associated\_with\_safety\_other\_associated\_factor  (  **party\_id                   int     not null references** Parties (**id**) **on delete cascade**,  **other\_associated\_factor\_id char**(1) **not null references** Other\_associated\_factor (**id**) **on delete cascade**,  **PRIMARY KEY** (**party\_id**, **other\_associated\_factor\_id**)  );  *---Parties end---*  *---Victims start---*  **CREATE TABLE** Victim\_degree\_of\_injury  (  **id         int CHECK** (0 <= **id and id** <= 7), *-- can we make sure id and def are consistent*  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Victim\_seating\_position  (  **id         int**, *--can we check if id is number or char?*  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Victim\_role  (  **id         int CHECK** (1 <= **id and id** <= 6),  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Victim\_ejected  (  **id         int CHECK** (0 <= **id and id** <= 3), *--make sure entity is still created if id is null*  **definition varchar**(150) **not null**,  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Victims  (  **id                         int**,  **victim\_age                 int**,  **victim\_sex                 char**(1),  **unborn                     char**(1),  *--- referenced ids--*  **victim\_degree\_of\_injury\_id int not null references** Victim\_degree\_of\_injury (**id**),  **victim\_seating\_position\_id int references** Victim\_seating\_position (**id**),  **victim\_role\_id             int not null references** Victim\_role (**id**),  **victim\_ejected\_id          int references** Victim\_ejected (**id**),  **party\_id                   int not null REFERENCES** Parties (**id**),  **PRIMARY KEY** (**id**)  );  **CREATE TABLE** Victim\_equipped\_with\_safety\_equipment  (  **victim\_id           int     not null references** Victims (**id**) **on delete cascade**,  **safety\_equipment\_id char**(1) **not null references** Safety\_equipment (**id**) **on delete cascade**,  **PRIMARY KEY** (**victim\_id**, **safety\_equipment\_id**)  );  *---Victims end---* |

## General Comments

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

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

# Deliverable 2

## Assumptions

## Data Loading/Cleaning

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

**Collisions.csv:**

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

officer\_id:

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

**Parties.csv**

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

cellphone\_use:

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

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

1 : 24787 in % : 0.009 B : 38932 in % : 0.018

2 : 39114 in % : 0.015 C : 795475 in % : 0.377

3 : 2572993 in % : 0.976 D : 1274423 in % : 0.604

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

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

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

vehicle\_make:

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

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

"DODG" =>  "DODGE"

"HOND" => "HONDA"

"MERCEDES BENZ" => "MERCEDES-BENZ"

"MAZD" => "MAZDA"

"TOYTA" =>  "TOYOTA"

"MISCELLANEOUS" , "NOT STATED" => None

party\_drug\_physical:

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

**Victims**:

victim\_age and pregnancy:

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

## Assumptions For Queries

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

## Query Implementation

***Description of logic:***

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

***SQL statement***

|  |
| --- |
| SELECT EXTRACT(YEAR FROM C.COLLISION\_DATE) AS YEAR, COUNT(\*) AS NUMBER\_COLLISIONS  FROM COLLISIONS C  GROUP BY EXTRACT(YEAR FROM C.COLLISION\_DATE)  ORDER BY EXTRACT(YEAR FROM C.COLLISION\_DATE) ASC; |

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

|  |  |
| --- | --- |
| YEAR | NUMBER\_COLLISIONS |
| 2001 | 522562 |
| 2002 | 544741 |
| 2003 | 538954 |
| 2004 | 538295 |
| 2005 | 532725 |
| 2006 | 498850 |
| 2007 | 501908 |
| 2017 | 7 |
| 2018 | 21 |

**Query 2:**

***Description of logic:***

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

***SQL statement***

|  |
| --- |
| SELECT P.VEHICLE\_MAKE, COUNT(\*) AS NUMBER\_VEHICLE  FROM PARTIES P  GROUP BY P.VEHICLE\_MAKE  ORDER BY COUNT(\*) DESC  FETCH FIRST 1 ROW ONLY; |

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

|  |  |
| --- | --- |
| VEHICLE\_MAKE | NUMBER\_VEHICLE |
| FORD | 1129701 |

**Query 3:**

***Description of logic:***

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

***SQL statement***

|  |
| --- |
| SELECT ROUND(A.NUMBER\_COLLISIONS\_UNDER\_DARK/A.TOTAL\_NUMBER\_COLLISIONS, 3) AS FRACTION\_UNDER\_DARK  FROM(     SELECT         (SELECT COUNT(\*)             FROM COLLISIONS C             WHERE C.LIGHTING\_ID IN                 (   SELECT L.ID                     FROM LIGHTING L                     WHERE LOWER(L.DEFINITION) LIKE '%dark%')) AS NUMBER\_COLLISIONS\_UNDER\_DARK,             (SELECT COUNT(\*) FROM COLLISIONS) AS TOTAL\_NUMBER\_COLLISIONS     FROM DUAL  )A; |

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

|  |
| --- |
| FRACTION\_DARK |
| 0.28 |

**Query 4:**

***Description of logic:***

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

***SQL statement***

|  |
| --- |
| SELECT COUNT(\*) AS NUMBER\_COLLISIONS\_SNOWY\_WEATHER  FROM COLLISION\_WITH\_WEATHER CWW  WHERE CWW.WEATHER\_ID IN     (   SELECT W.ID         FROM WEATHER W         WHERE LOWER(W.DEFINITION) LIKE '%snow%'); |

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

|  |
| --- |
| NUMBER\_COLLISIONS\_SNOWY\_WEATHER |
| 8530 |

**Query 5:**

***Description of logic:***

This query should retrieve the number of collisions that happen every day of the week. For that, we first groupy by the day using the built-in function “TO\_CHAR(date, ‘DAY’)” and count the number of entries. 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 |

**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 |

**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; |

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

|  |  |
| --- | --- |
| VICTIM\_AGE\_MEDIAN | MOST\_COMMON\_VICTIM\_SEATING\_POSITION |
| 25 | Passengers |

**Query 9:**

***Description of logic:***

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

***Remarks***

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

***SQL statement***

|  |
| --- |
| SELECT ROUND(A.NUMBER\_VICTIM\_WITH\_BELT / (A.TOTAL\_VICTIM + A.TOTAL\_PARTIES), 3) AS FRACTION\_WITH\_BELT  FROM  (     SELECT         (SELECT COUNT(\*)         FROM VICTIMS V         WHERE V.ID IN             (   SELECT VEWSE.VICTIM\_ID                 FROM VICTIM\_EQUIPPED\_WITH\_SAFETY\_EQUIPMENT VEWSE                 WHERE VEWSE.SAFETY\_EQUIPMENT\_ID IN                     (   SELECT SE.ID                         FROM SAFETY\_EQUIPMENT SE                         WHERE LOWER(SE.DEFINITION) LIKE '%belt use%'))) AS NUMBER\_VICTIM\_WITH\_BELT,         (SELECT COUNT(\*) FROM VICTIMS) AS TOTAL\_VICTIM,         (SELECT COUNT(\*) FROM PARTIES) AS TOTAL\_PARTIES         FROM DUAL  )A; |

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

|  |
| --- |
| FRACTION\_WITH\_BELT |
| 0.011 |

**Query 10:**

***Description of logic:***

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

***Remark:***

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

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

***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 |

## 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.

# 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.

## 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 descending to have a better vision of the results and see which categories 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 |

**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 |
| HARLEY-DAVIDSON | 3410 |
| SUZUKI | 2482 |
| YAMAHA | 2105 |
| GMC | 1837 |

**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,          (              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 |

**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 |

**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                        (                            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 |

**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 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; |

***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 |

**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; |

***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 |

**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 |

**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'                     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             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 |

## 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.

**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.

***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:

|  |
| --- |
| 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.

***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)"

**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 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):

---------------------------------------------------

"   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.

***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)"

**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.

***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:

|  |
| --- |
| 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.

***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.