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: 08

Members: Ömer Faruk Akgül, Öykü Irmak Hatipoğlu, Burak Öçalan

Contents

Contents....1
Deliverable 1....3
  Assumptions....3
  Entity Relationship Schema....4
    Schema....4
    Description....6
  Relational Schema....8
    ER schema to Relational schema....8
    DDL....8
  General Comments....22
Deliverable 2....23
  Assumptions....23
  Data Loading/Cleaning....23
  Query Implementation....23
  General Comments....29
Deliverable 3....30
Assumptions....30
  Query Implementation....30
  Query Performance Analysis – Indexing....40
General Comments....51

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

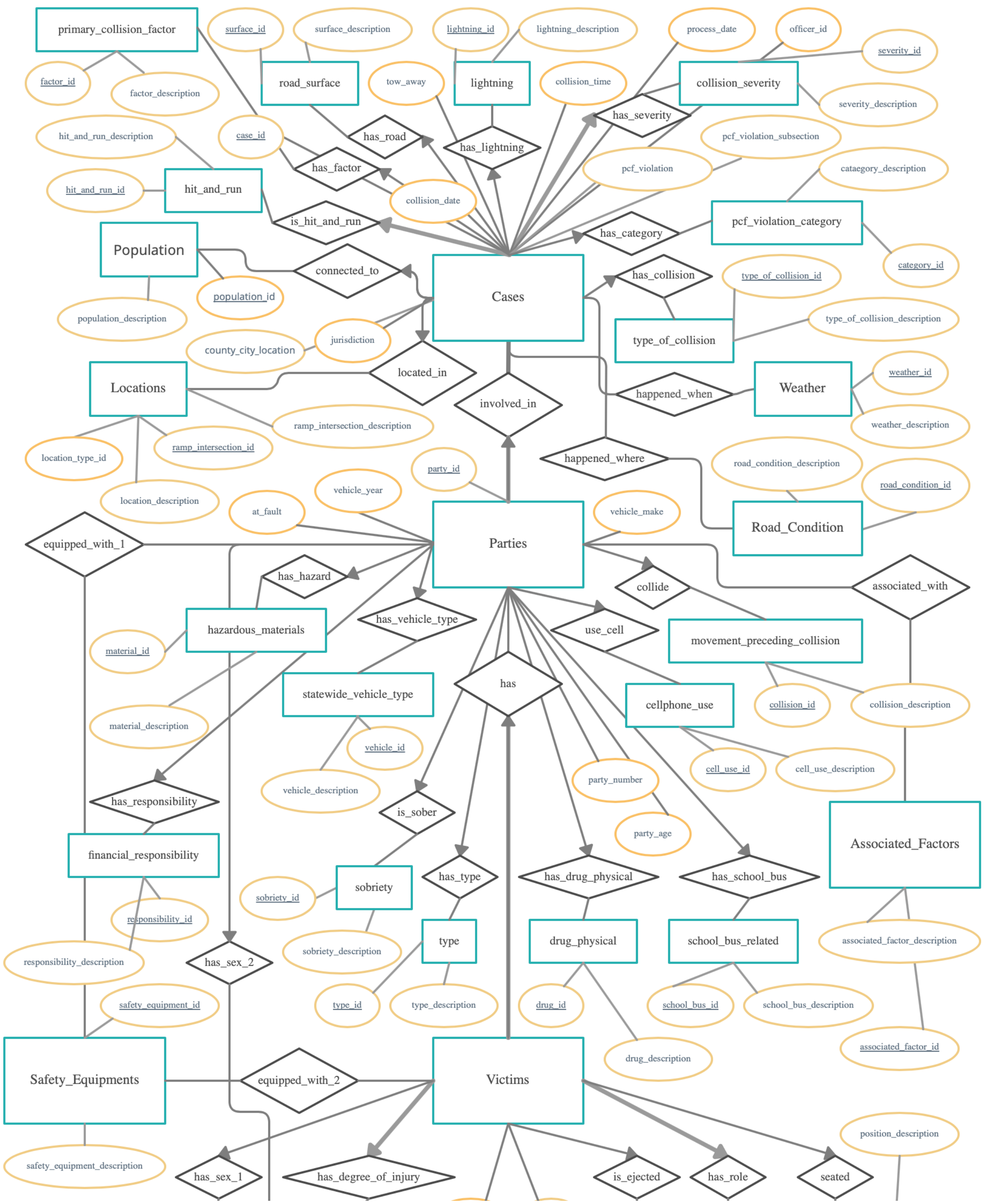
- We assumed one instance of “county_city_location” cannot correspond to more than one “jurisdiction” or “population” value.
- We assumed that a victim can belong to one and only party and has to have a party.
- If the columns in each csv file have absolutely no NULL values, we assumed that these attributes cannot have NULL values.
- “ramp_intersection” values 1-4 indicate the “location_type” is ramp, 5-6 indicate intersection, 7 indicates highway, and 8 indicates the “location_type” is empty.
- “party_number” and “case_id” are candidate keys for “Parties” entity.
- “vehicle_made” and “vehicle_year” can be NULL although the project description hasn’t mentioned so.

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/



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

- "Cases" entity corresponds to the collisions.csv. It has most of the attributes listed in the project description. However, there are some attributes of the same kind that can have multiple values such as "road_condition_x" and "weather_x" where x is a number. To handle these types of attributes, we opened a separate entity for these attributes called "Road_Condition" and "Weather". By this way, we do not have to have NULL values on the "Cases" entity for most of the "road_condition_2" attribute and also, we can have more than 2 road condition or weather for one case which is more flexible for further uses of this database. It also makes it easier to add different road conditions other than stated.
- "Parties" entity corresponds to the parties.csv. It has most of the attributes listed in the project description. The same situation as the "Cases" entity is present here with attributes "other_associated_factor_x", "party_safety_equipment_x". Likewise, we constructed two separate entities called "Associated_Factors" and "Safety_Equipments". Parties has a 4-nary relationship with one-to-many, at least one "Cases" via relationship "involved_in", with "Associated_Factors" via "associated_with", with one-to-many "Victims" via "has" and, with "Safety_Equipments" via "equipped_with_1".
- "Victims" entity corresponds to the victims.csv. It has most of the attributes listed in the project description. The same situation as the "Cases" entity is present here with attribute "victim_safety_equipment_x". Likewise, we used a separate entity called "Safety_Equipments" which we also used for "party_safety_equipment_x".
- We created a "Locations" entity which has "ramp_intersection_id" as a primary key and "ramp_intersection_description", "location_type_id" and, "location_description" as attributes.
- The "Safety_Equipments" entity has relationships with both "Victims" and "Parties" entities with the relationships "equipped_with_2" and "equipped_with_1" respectively. It also has a primary key named "safety_equipment_id" and an attribute named "safety_equipment_description".
- "Associated_Factors" is connected to "Parties" entity via "associated_with" relationship. It has a primary key called "associated_factor_id" and an attribute named "associated_factor_description".
- The entity "Road_Condition" is connected to the entity "Cases" via "happened_where" relationship. It has a primary key named "road_condition_id" and it has an attribute named "road_condition_description".
- "Weather" has a relationship with the entity "Cases" via "happened_where". It has a primary key "weather_id" and it has an attribute named "weather_description".

- “Primary_Collision_Factor” entity is connected to “Cases” via the “has_factor” relationship. It has the primary key “factor_id” and an attribute “factor_description”.
- “Road_surface” entity is connected to “Cases” via the “has_road” relationship. It has the primary key “factor_id” and an attribute “factor_description”.
- “Lightning” entity is connected to “Cases” via the “has_lightning” relationship. It has the primary key “lightning_id” and an attribute “lightning_description”.
- “Collision_Severity” entity is connected to “Cases” via the “has_severity” relationship. It has the primary key “severity_id” and an attribute “severity_description”.
- “Hit_And_Run” entity is connected to “Cases” via the “is_hit_and_run” relationship. It has the primary key “hit_and_run_id” and an attribute “hit_and_run_description”.
- “Population” entity is connected to “Cases” via the “connected_to” relationship. It has the primary key “population_id” and an attribute “population_description”.
- “Pcf_Violation_Category” entity is connected to “Cases” via the “has_category” relationship. It has the primary key “category_id” and an attribute “category_description”.
- “Type_Of_Collision” entity is connected to “Cases” via the “has_collision” relationship. It has the primary key “type_of_collision_id” and an attribute “type_of_collision_description”.
- “Hazardous_Materials” entity is connected to “Parties” via the “has_hazard” relationship. It has the primary key “material_id” and an attribute “material_description”.
- “Statewide_Vehicle_Type” entity is connected to “Parties” via the “has_vehicle_type” relationship. It has the primary key “vehicle_id” and an attribute “vehicle_description”.
- “Cellphone_Use” entity is connected to “Parties” via the “use_cell” relationship. It has the primary key “cell_use_id” and an attribute “cell_use_description”.
- “Movement_Preceding_Collision” entity is connected to “Parties” via the “collide” relationship. It has the primary key “collision_id” and an attribute “collision_description”.
- “Financial_Responsibility” entity is connected to “Parties” via the “has_responsiblity” relationship. It has the primary key “responsibility_id” and an attribute “responsibility_description”.
- “Sobriety” entity is connected to “Parties” via the “is_sober” relationship. It has the primary key “sobriety_id” and an attribute “sobriety_description”.
- “Type” entity is connected to “Parties” via the “has_type” relationship. It has the primary key “type_id” and an attribute “type_description”.

- “Drug_Physical” entity is connected to “Parties” via the “has_drug_physical” relationship. It has the primary key “drug_id” and an attribute “drug_description”.
- “School_Bus_Related” entity is connected to “Parties” via the “has_school_bus” relationship. It has the primary key “school_bus_id” and an attribute “school_bus_description”.
- “Sex” entity is connected to “Victims” and “Parties” via the “has_sex_1” and “has_sex_2” relationships respectively. It has the primary key “sex_id” and an attribute “sex_description”.
- “Degree_Of_Injury” entity is connected to “Victims” via the “has_degree_of_injury” relationship. It has the primary key “degree_id” and an attribute “degree_description”.
- “Ejected” entity is connected to “Victims” via the “is_ejected” relationship. It has the primary key “ejected_id” and an attribute “ejected_description”.
- “Role” entity is connected to “Victims” via the “has_role” relationship. It has the primary key “role_id” and an attribute “role_description”.
- “Seating_Position” entity is connected to “Victims” via the “seated” relationship. It has the primary key “position_id” and an attribute “position_description”.

Relational Schema

ER schema to Relational schema

When translating from ER to Relational Schema, most of the entities and relations from ER are being translated in a straightforward way. If there is a one to one relation between an entity and a relation, these are combined. For example, in our case, the combined entity-relation pairs are:

- Victim - has
- Party - involved-in
- Case - connected-to
- Case - located-in

Note that in our relational schema, we are currently not able to display the constraint: "Each case must have at least one party". Because there is no easy way to show that in relational schema.

DDL

CREATE TABLE Victims (

victim_id CHAR(16),

```
    party_id CHAR(16) NOT NULL,
    age SMALLINT,
    degree_of_injury CHAR NOT NULL,
    ejected CHAR,
    role_id CHAR NOT NULL,
    seating_position CHAR,
    sex CHAR,

    PRIMARY KEY(victim_id),
    FOREIGN KEY(party_id) REFERENCES Parties(party_id),
    FOREIGN KEY(degree_of_injury) REFERENCES Degree_Of_Injury(degree_id),
    FOREIGN KEY(ejected) REFERENCES Ejected(ejected_id),
    FOREIGN KEY(role_id) REFERENCES Roles(role_id),
    FOREIGN KEY(seating_position) REFERENCES Seating_Position(position_id),
    FOREIGN KEY(sex) REFERENCES Sex(sex_id)
)

CREATE TABLE Degree_Of_Injury (
    degree_id CHAR,
    degree_description VARCHAR,

    PRIMARY KEY(degree_id)
)

CREATE TABLE Sex (
```

```
    sex_id CHAR,
    sex_description VARCHAR,

    PRIMARY KEY(sex_id)
)

CREATE TABLE Ejected (
    ejected_id CHAR,
    ejected_description VARCHAR,

    PRIMARY KEY(ejected_id)
)

CREATE TABLE Roles (
    role_id CHAR,
    role_description VARCHAR,

    PRIMARY KEY(role_id)
)

CREATE TABLE Seating_Position (
    position_id CHAR,
    position_description VARCHAR,

    PRIMARY KEY(position_id)
```

```
)

CREATE TABLE Safety_Equipments (
    safety_equipment_id CHAR,
    safety_equipment_description VARCHAR,

    PRIMARY KEY(safety_equipment_id)
)

CREATE TABLE Equipped_With_2 (
    victim_id CHAR(16),
    safety_equipment_id VARCHAR,

    PRIMARY KEY(victim_id,safety_eq),
    FOREIGN KEY(victim_id) REFERENCES Victims(victim_id),
    FOREIGN KEY(safety_equipment_id) REFERENCES
Safety_Equipments(safety_equipment_id)
)

CREATE TABLE Parties (
    party_id CHAR(16),
    case_id CHAR(16) NOT NULL,
    at_fault BOOLEAN NOT NULL,
    cellphone_use CHAR,
    financial_responsibility CHAR,
```

```
    hazardous_materials CHAR,
    movement_preceding_collision CHAR,
    age SMALLINT,
    drug_physical CHAR,
    party_number SMALLINT NOT NULL,
    sex CHAR,
    sobriety CHAR,
    party_type CHAR,
    school_bus_related CHAR,
    statewide_vehicle_type CHAR,
    vehicle_make VARCHAR,
    vehicle_year SMALLINT,

    PRIMARY KEY(party_id),
    FOREIGN KEY(case_id) REFERENCES Cases(case_id),
    FOREIGN KEY(cellphone_use) REFERENCES Cellphone_Use(cell_use_id),
    FOREIGN KEY(hazardous_materials) REFERENCES Hazardous_Materials(material_id),
    FOREIGN KEY(financial_responsibility) REFERENCES
Financial_Responsibility(financial_responsibility_id),
    FOREIGN KEY(movement_preceding_collision) REFERENCES
Movement_Preceding_Collision(collision_id),
    FOREIGN KEY(drug_physical) REFERENCES Drug_Physical(drug_id),
    FOREIGN KEY(sex) REFERENCES Sex(sex_id),
    FOREIGN KEY(sobriety) REFERENCES Sobriety(sobriety_id),
    FOREIGN KEY(party_type) REFERENCES Party_Type(type_id),
    FOREIGN KEY(school_bus_related) REFERENCES School_Bus_Related(school_bus_id),
```

```
    FOREIGN KEY(statewide_vehicle_type) REFERENCES
Statewide_Vehicle_Type(vehicle_id),
)

CREATE TABLE Hazardous_Materials (
    material_id CHAR,
    material_description VARCHAR,

    PRIMARY KEY(material_id)
)

CREATE TABLE Population (
    population_id CHAR,
    population_description VARCHAR,

    PRIMARY KEY(population_id)
)

CREATE TABLE Statewide_Vehicle_Type (
    vehicle_id CHAR,
    vehicle_description VARCHAR,

    PRIMARY KEY(vehicle_id)
)
```

```
CREATE TABLE Sobriety (
   sobriety_id CHAR,
   sobriety_description VARCHAR,

   PRIMARY KEY(sobriety_id)
)

CREATE TABLE Party_Type (
   type_id CHAR,
   type_description VARCHAR,

   PRIMARY KEY(type_id)
)

CREATE TABLE Drug_Physical (
   drug_id CHAR,
   drug_description VARCHAR,

   PRIMARY KEY(drug_id)
)

CREATE TABLE School_Bus_Related (
   school_bus_id CHAR,
   school_bus_description VARCHAR,
```

```
    PRIMARY KEY(school_bus_id)
)

CREATE TABLE Cellphone_Use (
    cell_use_id CHAR,
    cell_use_description VARCHAR,

    PRIMARY KEY(cell_use_id)
)

CREATE TABLE Movement_Preceding_Collision (
    collision_id CHAR,
    collision_description VARCHAR,

    PRIMARY KEY(collision_id)
)

CREATE TABLE Equipped_With_1 (
    party_id CHAR(16),
    safety_equipment_id CHAR,

    PRIMARY KEY(party_id,safety_equipment_id),
    FOREIGN KEY(party_id) REFERENCES Parties(party_id),
    FOREIGN KEY(safety_equipment_id) REFERENCES
Safety_Equipments(safety_equipment_id)
```

```
)

CREATE TABLE Associated_Factors (
    associated_factor_id CHAR,
    associated_factor_description VARCHAR,

    PRIMARY KEY(associated_factor_id)
)

CREATE TABLE Associated_With (
    party_id CHAR(16),
    associated_factor_id CHAR,

    PRIMARY KEY(party_id,associated_factor_id),
    FOREIGN KEY(party_id) REFERENCES Parties(party_id),
    FOREIGN KEY(associated_factor_id) REFERENCES
Associated_Factors(associated_factor_id)
)

CREATE TABLE Cases (
    case_id CHAR(16)
    collision_date DATE NOT NULL,
    collision_severity CHAR NOT NULL,
    collision_time TIME,
    hit_and_run CHAR NOT NULL,
```

```
   lightning CHAR,
   officier_id, CHAR(16),
   primary_collision_factor CHAR,
   process_date DATE NOT NULL,
   road_surface CHAR,
   tow_away BOOLEAN,
   type_of_collision CHAR,
   pcf_violation_id CHAR(8),
   pcf_violation_category CHAR,
   pcf_violation_subsection CHAR,
   county_city_location CHAR(8) NOT NULL,
   location_type CHAR,
   jurisdiction CHAR(4),
   population CHAR,



   PRIMARY KEY(case_id)
   FOREIGN KEY(location_type) REFERENCES Locations(ramp_intersection_id),
   FOREIGN KEY(hit_and_run) REFERENCES Hit_And_Run(hit_and_run_id),
   FOREIGN KEY(primary_collision_factor) REFERENCES
Primary_Collision_Factor(factor_id),
   FOREIGN KEY(road_surface) REFERENCES Road_Surface(surface_id),
   FOREIGN KEY(lightning) REFERENCES Lightning(lightning_id),
   FOREIGN KEY(collision_severity) REFERENCES Collision_Severity(severity_id),
   FOREIGN KEY(pcf_violation_category) REFERENCES
PCF_Violation_Category(category_id),
```

```
    FOREIGN KEY(type_of_collision) REFERENCES Type_Of_Collision(type_of_collision_id),
    FOREIGN KEY(population) REFERENCES Population(population_id),
)

CREATE TABLE Hit_And_Run (
    hit_and_run_id CHAR,
    hit_and_run_description VARCHAR,

    PRIMARY KEY(hit_and_run_id)
)

CREATE TABLE Primary_Collision_Factor (
    factor_id CHAR,
    factor_description VARCHAR,

    PRIMARY KEY(factor_id)
)

CREATE TABLE Road_Surface (
    surface_id CHAR,
    surface_description VARCHAR,

    PRIMARY KEY(surface_id)
)
```

```
CREATE TABLE Lightning (
   lightning_id CHAR,
   lightning_description VARCHAR,

   PRIMARY KEY(lightning_id)
)

CREATE TABLE Collision_Severity (
   severity_id CHAR,
   severity_description VARCHAR,

   PRIMARY KEY(severity_id)
)

CREATE TABLE PCF_Violation_Category (
   category_id CHAR(2),
   category_description VARCHAR,

   PRIMARY KEY(category_id)
)

CREATE TABLE Type_Of_Collision (
   type_of_collision_id CHAR,
   type_of_collision_description VARCHAR,
```

```
    PRIMARY KEY(type_of_collision_id)
)

CREATE TABLE Locations (
    ramp_intersection_id CHAR,
    ramp_intersection_description VARCHAR,
    location_type_id CHAR,
    location_description VARCHAR,

    PRIMARY KEY(ramp_intersection_id)
)


CREATE TABLE Weather (
    weather_id CHAR,
    weather_description VARCHAR,

    PRIMARY KEY(weather_id)
)

CREATE TABLE Happened_When (
    case_id CHAR(16),
    weather_id CHAR,

    PRIMARY KEY(case_id,weather_id),
```

```
    FOREIGN KEY(case_id) REFERENCES Cases(case_id),
    FOREIGN KEY(weather_id) REFERENCES Weather(weather_id)
)

CREATE TABLE Road_Condition (
    road_condition_id CHAR,
    road_condition_description VARCHAR,

    PRIMARY KEY(road_condition_id)
)

CREATE TABLE Happened_Where (
    case_id CHAR(16),
    road_condition_id CHAR,

    PRIMARY KEY(case_id,road_condition_id),
    FOREIGN KEY(case_id) REFERENCES Cases(case_id),
    FOREIGN KEY(road_condition_id) REFERENCES Road_Condition(road_condition_id)
)

CREATE TABLE Financial_Responsibility (
    financial_responsibility_id CHAR,
    financial_responsibility_description VARCHAR,

    PRIMARY KEY(financial_responsibility_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/

)

General Comments

We did the ER model together by exchanging and negotiating ideas via Discord.

Ömer Faruk Akgül: Data cleaning.

Öykü Irmak Hatipoğlu: Wrote the report.

Burak Öçalan: Translation from ER to relational model.

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

A victim can't have both G and C safety equipment at the same time.

Data Loading/Cleaning

It is aimed to create a csv file for each table in order to easily load the data of the tables into the database. First, the csv files for our main tables were created by selecting the columns from the project dataset. For tables containing enumeration information, our csv files were created with the information in the project description document. Usually, our files were created using python's pandas library and sometimes using unix commands.

Some rows in csv files failed while loading to database system. Thereupon, a small number of data that did not comply with the table constraints were deleted from the tables (less than 5).

Query Implementation

Query a: List the year and the number of collisions per year. Suppose there are more years than just 2018.
Description of logic:
First, year of collisions is extracted from collision_date. Then, Cases table is grouped by year of collisions. Count of each group gives the number of collisions per year.
SQL statement
SELECT EXTRACT(YEAR FROM collision_time) AS Years, COUNT(*) AS Collisions
FROM Cases
GROUP BY EXTRACT(YEAR FROM collision_time)
Query result (if the result is big, just a snippet)
2007,497313
2002,540814
2006,494293
2003,534829
2001,519169
2004,533996
NULL,29645
2017,7
2005,527975
2018,21

Query b: Find the most popular vehicle make in the database. Also list the number of vehicles of that particular make.
Description of logic:
To find the most popular vehicle_make, I grouped the vehicle_makes with their corresponding count, then ordered the table by descending popularity order. When I fetched the first row only, it gives out the most popular vehicle_make with the number of the vehicles of this vehicle_make.
SQL statement
```
SELECT P.vehicle_make AS Vehicles, COUNT(*) AS Popularity
FROM Parties P
GROUP BY P.vehicle_make
ORDER BY Popularity DESC
FETCH FIRST 1 ROW ONLY
```
Query result (if the result is big, just a snippet)
FORD,1129701

Query c: Find the fraction of total collisions that happened under dark lightning conditions.
Description of logic:
Using a nested query, I made a table with the number of cases where the lightning_id of the case corresponds to the lightning_descritption that includes the keyword 'Dark'. Then, I also constructed another table with the count of all cases. Using these two tables, I made a final query that displayed the fraction of the total collisions happened when the lightning_description is dark. The fraction is (collisions in dark conditions) / (all collisions).
SQL statement
```
SELECT DarkCount.Cnt / AllCount.Cnt AS Fraction
FROM (SELECT COUNT(*) AS Cnt
      FROM Cases C, Lightning L
      WHERE C.lightning = L.lightning_id AND L.lightning_description LIKE '%Dark%')
DarkCount,
     (SELECT COUNT(*) AS Cnt
      FROM Cases) AllCount
```
Query result (if the result is big, just a snippet)
0.2798142065033161485586702997393736157792

Query d: Find the number of collisions that have occurred under snowy weather conditions.
Description of logic:
Weather entity connects to Cases entity via the happened_when relation. Using this information, I connected the weather_id that corresponds to the weather_description 'Snowing' to the cases using Happened_When entity. Then I take the count of the cases in which the weather_description is 'Snowing'.
SQL statement

SELECT COUNT(*)
FROM Weather W, Cases C, Happened_When H
WHERE W.weather_description LIKE '%Snowing%' AND W.weather_id = H.weather_id AND C.case_id = H.case_id

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

8530

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

Description of logic:

First, day of collisions is extracted from collision_time. Then, Cases table is grouped by day of collisions. Count of each group gives the number of collisions per day. Then the table is ordered with descending order according to the count of the collisions on each day. The first row is fetched to find the day of the week which has the highest number of collisions and its collision count.

SQL statement

SELECT TO_CHAR(COLLISION_TIME, 'DAY', 'NLS_DATE_LANGUAGE = ENGLISH') AS Days, COUNT(*) AS Collisions
FROM Cases
GROUP BY TO_CHAR(COLLISION_TIME, 'DAY', 'NLS_DATE_LANGUAGE = ENGLISH')
ORDER BY Collisions DESC
FETCH FIRST 1 ROWS ONLY

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

FRIDAY ,610673

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

Description of logic:

Entities Weather and Cases are connected via the relationship Happened_When. To get the weather types, I reached the weather_description attribute of the Weather entity and the number of collisions are found. Then the number of collisions are listed in descending order.

SQL statement

SELECT W.weather_description, COUNT(*) AS Collisions
FROM Weather W, Happened_When H, Cases C
WHERE W.weather_id = H.weather_id AND H.case_id = C.case_id
GROUP BY W.weather_description
ORDER BY Collisions DESC

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

Clear,2941042
Cloudy,548250
Raining,223752
Fog,21259

Wind,13952
Snowing,8530
Other,6960

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

Description of logic:

The requirements are simply applied to select from where query.

SQL statement

```
SELECT COUNT(*)
FROM Parties P, Happened_Where H, Road_Condition R, Financial_Responsibility F
WHERE P.at_fault = 1 AND P.financial_responsibility = 'Y' AND P.case_id = H.case_id AND
     H.road_condition_id = R.road_condition_id AND R.road_condition_description LIKE
'%Loose%'
       AND P.financial_responsibility = F.financial_responsibility_id AND
F.financial_responsibility_desc LIKE '%Yes%'
```

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

4803

Query h: Find the median victim age and the most common victim seating position.

Description of logic:

To display the median victim age and the most common victim seating position on the same query, using nested queries, I made two different tables from which I reached both of this information individually. In the first table for median victim age, I simply used the MEDIAN function and find the median of all of the victim ages. On the second table from the second nested query, I ordered the table with descending count of seating_position occurrences on the Victims entity and fetched the first row only to find the most common seating_position among the Victims. Then on the main query, I displayed the median age from the first nested query and the position_description of the most common seating_position I get from the second nested query. In the result, “Passengers” is the description of the seating position and “3.0” is the exact position of the “Passengers”.

SQL statement

```
SELECT t1.median_age, SP.position_description, SP.position_id
FROM (SELECT MEDIAN(age) AS median_age
   FROM VICTIMS) t1, (SELECT V.seating_position AS common_seating
                    FROM Victims V
                    GROUP BY V.seating_position
                    ORDER BY COUNT(*) DESC
                    FETCH FIRST 1 ROW ONLY) t2, seating_position SP
WHERE T2.common_seating = SP.position_id
```

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

25,Passengers,3.0

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

Description of logic:

First of all, we have assumed that a victim cannot have both G (Lap/Shoulder Harness Used) and C (Lap Belt Used) safety_equipment at the same time. In that case, we first count the number of rows having the string “Lap” and not having the string “Not”. By this way, we only take the lines that correspond to a victim having a lap belt. Then we divide it to total number of victims which we found from a nested query that gives the count of all of the victims and obtain the fraction.

SQL statement

```
SELECT LapCount.Cnt / AllCount.Cnt AS Fraction
FROM (SELECT COUNT(*) AS Cnt
      FROM Equipped_With_2 E, Safety_Equipments S
      WHERE E.safety_equipment_id = S.safety_equipment_id AND
            S.safety_equipment_description LIKE '%Lap%' AND
            S.safety_equipment_description NOT LIKE '%Not%') LapCount,
     (SELECT COUNT(*) AS Cnt
      FROM Victims) AllCount
```

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

0.7451813514652140260809076390998069895265

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

Description of logic:

In a nested query, I extracted the hours out of the case collision_time and the count of collisions happened in each hour. Then, in another nested query, I take the count of total cases. On the main query, using the nested queries, I found the percentage of the collisions happened on each hour of the day by multiplying the count of the collisions on each hour by 100 and dividing it to the total number of collisions.

SQL statement

```
SELECT T2.Hours, CAST(T2.Count * 100 AS FLOAT) / T3.Total
FROM (SELECT EXTRACT(HOUR FROM collision_time) AS Hours, COUNT(*) AS Count
    FROM Cases C
    GROUP BY EXTRACT(HOUR FROM collision_time)
    ORDER BY EXTRACT(HOUR FROM collision_time)) T2, (SELECT COUNT(*) AS Total
                                                     FROM Cases) T3
```

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

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/

0,1.908450700396023775564414085461310875l3

1,1.82982233578444300286400827392251680369

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/

2,1.8080445625984553822094352950004649187 5

3,1.154086037701376431392401759404816993 3

4,0.98130482846673057713545883674609074017 78

5,1.446712970036937930899479127866795067 62

6,2.6232836749353327921062777082061150682 1

7,5.1706849966096275701714652988448808095 1

8,5.2335985635913695853957872379530306993 2

9,4.0881040069471368345612444814687735008 3

10,4.2271174330394648051066023356865653705 7

11,4.8913802975588774740610680298483277334 6

12,5.7755415759712587770407350392679623127 6

13,5.7752696936593238504408027923400964964 7

14,6.5475785889416763502083434156357342535 3

15,7.7480477490591512595491865009344595061 2

16,7.3308715296261998846131468148171509887 5

17,7.9070717133098898278495577290431754549

18,6.3005191320864085488499106322840669896 3

19,4.4286366026456324009763837586207084056 8

20,3.4896366619159764027903825438505386804 2

21,3.2818641991353054951221594415754818706 2

22,2.8618604036582308835468243874083688638 2

23,2.38451662859408025204577845615435520119

NULL,0.8059951137310899054991460176582123955496

General Comments

Our ER Model and DDL has changed.

Ömer Faruk Akgül: Data loading and cleaning.

Öykü Irmak Hatipoğlu: Fixed the ER model. Wrote even numbered queries.

Burak Öçalan: Fixed DDL. Wrote even numbered queries. Wrote odd numbered queries.

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

Query Implementation

Query a: For the drivers of age groups: underage (less and equal to 18 years), young I [19, 21], young II [22,24], adult [24,60], elder I [61,64], elder II [65 and over), find the ratio of cases where the driver was the party at fault. Show this ratio as percentage and display it for every age group – if you were an insurance company, based on the results would you change your policies?

Description of logic:

-This query contains two subqueries.
-The first one separates the parties with at_fault of 1 according to age groups.
-The second one divides into groups regardless of at_fault.
-The main query finds the ratio by dividing the counts of the two corresponding subqueries with the same age group.

Generally, the younger the age, the higher the at_fault rate is. Therefore, the insurance rates should be higher for younger customers as their probability of being at_fault on an accident is higher than older people. This policy should also change when the customer who would like to be insured is over 65 years old. The insurance cost should be almost as high as 22-24 aged customers' insurance cost.

SQL statement

```
select  at_fault_all.interval, round(at_fault_one.party_count/at_fault_all.party_count,4)
from
(select p2.range as interval, count () as party_count
   from (
      select case
         when age between 0 and 18 then 'underage (0-18)'
         when age between 19 and 21 then 'young 1 (19-21)'
         when age between 22 and 24 then 'young 2 (22-24)'
         when age between 25 and 60 then 'adult (25-60)'
         when age between 61 and 64 then 'elder 1 (61-64)'
         else 'elder 2' end as range, p.at_fault
      from
         parties p
         where p.at_fault = 1 and p.age is not null and p.age <> 998 and p.age <> 999)
p2
   group by p2.range
   order by party_count desc) at_fault_one,
```

```
(select p2.range as interval, count () as party_count
from (
   select case
      when age between 0 and 18 then 'underage (0-18)'
      when age between 19 and 21 then 'young 1 (19-21)'
      when age between 22 and 24 then 'young 2 (22-24)'
      when age between 25 and 60 then 'adult (25-60)'
      when age between 61 and 64 then 'elder 1 (61-64)'
      else 'elder 2' end as range, p.at_fault
   from
      parties p
      where p.age is not null and p.age <> 998 and p.age <> 999) p2
group by p2.range
order by party_count desc) at_fault_all
where at_fault_one.interval = at_fault_all.interval
```

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

adult (25-60)   0.409
young 1 (19-21)   0.5721
young 2 (22-24)   0.5174
underage (0-18)   0.6358
elder 2   0.498
elder 1 (61-64)   0.3988

Query b: Find the top-5 vehicle types based on the number of collisions on roads with holes. List both the vehicle type and their corresponding number of collisions.

Description of logic:

1- 'happened_where', 'parties' and 'statewide_vehicle_type' are crossed.
2- the rows meeting the following conditions are filtered:
- having the same case_id for happened_where and parties table,
- road_condition_id is 'A', which means holes in the road
- nonnull statewide_vehicle_type values
- having the same vehicle_id for statewide_vehicle_type and parties table

3- Table is grouped based on vehicle types and length of these groups are calculaleted with count statement.
4- Result is ordered to find the top 5 vehicle types satisfying given condition.

SQL statement

```
select svt.vehicle_description, count(*) as collision_count
from happened_where hw, parties p, statewide_vehicle_type svt, road_condition rc
where hw.case_id = p.case_id and hw.road_condition_id = rc.road_condition_id and
rc.road_condition_description  like '%Holes%' and p.statewide_vehicle_type is not null and
svt.vehicle_id = p.statewide_vehicle_type
group by svt.vehicle_description
```

order by collision_count desc
fetch first 5 rows only

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

passenger car	10662
pickup or panel truck	2263
motorcycle or scooter	450
bicycle	430
truck or truck tractor with trailer	369

Query c: Find the top-10 vehicle makes based on the number of victims who suffered either a severe injury or were killed. List both the vehicle make and their corresponding number of victims.

Description of logic:

1- 'parties' and 'victims' tables are crossed.
2- rows with the same party_ids for parties and victims tables are filtered.
degree_of_injury is selected as either 1 or 2 to find victims suffering severy injury or killed.
3- null and empty values are filtered out.
4. table is grouped based on vehicle makes of parties.
5- Lengths of these groups are calculated and the table is reordered based on the descending numbers of these counts.
6- First 10 rows are fetched.

SQL statement

select p.vehicle_make, count(*) as vehicle_count
from parties p, victims v, degree_of_injury di
where p.party_id = v.party_id and v.degree_of_injury = di.degree_id and
(di.degree_description like '%killed%' or di.degree_description like '%severe%') and
p.vehicle_make is not null and p.vehicle_make != 'NOT STATED'
group by p.vehicle_make
order by vehicle_count desc
fetch first 10 rows only

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

FORD	13929
HONDA	12056
TOYOTA	10642
CHEVROLET	10420
NISSAN	3860
DODGE	3642
HARLEY-DAVIDSON	3410
SUZUKI	2482
YAMAHA	2105
MISCELLANEOUS	2048

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 d: Compute the safety index of each seating position as the fraction of total incidents where the victim suffered no injury. The position with the highest safety index is the safest, while the one with the lowest is the most unsafe. List the most safe and unsafe victim seating position along with its safety index.

Description of logic:

1- In this query, there are 2 subqueries of 2 main queries. The first main query finds the safest seat and the other main query finds the least safe one. Then, they are combined by 'union' statement.

2- The subqueries have the similar principles:

- 'victims' and 'seating_position' tables are crossed and the rows with 'no injury'('degree_of_injury'=0) are chosen. To convert the position_id enumeration to description, 'seating_position' table is used.
- resulting table is grouped based on 'seating_position' and 'position_description' pair.
- lengths of these groups are used for ordering.

3- To find the ratios, one more query is appended inside the select statement, which basically calculates the total number of victim seats with 'no injury'. The number returned by this query is used as the denominator of the ratio.

4- Round function is used to adjust decimals.

SQL statement

```
select round(ranking.seating_position, 0) as seat_pos, ranking.position_description,
round(ranking.seat_counts / (select count () from victims v2 where v2.degree_of_injury =
0 and v2.seating_position is not null), 3) as safety_index
from
   (select v.seating_position, sp.position_description, count() seat_counts
   from victims v, seating_position sp, degree_of_injury doi
   where v.degree_of_injury = doi.degree_id and doi.degree_description = 'no injury' and
v.seating_position is not null and sp.position_id = v.seating_position
   group by v.seating_position,  sp.position_description
   order by seat_counts asc
   fetch first 1 rows only) ranking

union

select round(ranking2.seating_position, 0) as seat_pos, ranking2.position_description,
round(ranking2.seat_counts / (select count () from victims v3 where v3.degree_of_injury
= 0 and v3.seating_position is not null), 3) as safety_index
from
   (select v4.seating_position, sp4.position_description, count() seat_counts
   from victims v4, seating_position sp4, degree_of_injury doi2
   where v4.degree_of_injury = doi2.degree_id and doi2.degree_description = 'no injury'
and v4.seating_position is not null and sp4.position_id = v4.seating_position
   group by v4.seating_position,  sp4.position_description
```

```
    order by seat_counts desc
    fetch first 1 rows only) ranking2
order by safety_index desc
```

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

3	Passengers	0.481
1	Driver	0.006

Query e: How many vehicle types have participated in at least 10 collisions in at least half of the cities?

Description of logic:

1- parties and cases tables were crossed and rows with the same case_id for both were selected.
2- The table was grouped by location and vehicle_type, and groups less than 10 in length were discarded. Thus, it was aimed to find vehicles that had at least 10 accidents in a particular city.
3- the resulting table is returned and used for the main query.
4- The returned table is regrouped according to vehicle_type. It was checked whether the counts found for each were more than half of the total number of distinct cities. For this, another subquery is used in the having statement.
5- This subquery basically calculates the number of distinct cities.
6- Length of resulting table is returned.

SQL statement

```
select count(*)
from
    (select collisions.statewide_vehicle_type, count(*) city_count
    from
        (select c.county_city_location, p.statewide_vehicle_type, count(*) times
        from cases c, parties p
        where c.case_id = p.case_id and c.county_city_location is not null and
p.statewide_vehicle_type is not null
        group by c.county_city_location, p.statewide_vehicle_type
        having count(c.case_id) >= 10
        order by times desc) collisions
    group by collisions.statewide_vehicle_type
    having 2 * count(*) >= (select count(distinct c2.county_city_location) from cases c2))
output
```

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

13

Query f: For each of the top-3 most populated cities, show the city location, population, and the bottom-10
collisions in terms of average victim age (show collision id and average victim age).

Description of logic:

To find the top-3 most populated cities, I wrote a nested query which look at the county_city_location attributes which have the highest population. Since population 9.0 means the city is unincorporated, I looked at the populations that are smaller than 9.0. Then I ordered the cities with descending order and fetched the first three rows to get the top-3 most populated cities. Using the result of this query, which is a table with top-3 most populated cities with their corresponding populations, I constructed another nested query which apart from what is in the first nested query, includes the case_ids, average victim age for each of the collisions happened in each of the cities, and an additional attribute “rank”. To calculate the average victim age for each case, I divided the sum of the victims ages in each collision to the count of the victims on a collision. I sorted this table with respect to ascending order of victim age averages for each collision. The additional attribute “rank” corresponds to the row number of each row however, assigned such that for each county_city_location, to enumeration starts from the beginning. By this way, on the main query, we are able to get the first 10 rows from each county_city_location by looking at their ranks. The first 10 rows for each city represents the collisions that happened on that city and that have the minimum victim age average. The final results consists of 30 rows, 10 row for each city.

SQL statement

```
SELECT Q.location, Q.population, Q.case, Q.average
FROM (SELECT CS3.county_city_location AS location, CS3.population AS population,
CS3.case_id AS case,       SUM(V.age) / COUNT(*) AS average, ROW_NUMBER()
OVER(PARTITION BY CS3.county_city_location
ORDER BY SUM(V.age) / COUNT(*), CS3.county_city_location) AS rank
        FROM Cases CS3, Victims V, Parties P, (SELECT CS.county_city_location AS
location1, CS.population
                                                   FROM Cases CS
WHERE CS.population IS NOT NULL AND TO_NUMBER(CS.population, '9.9') < 9.0
                                                   GROUP BY CS.county_city_location,
CS.population
                                                   ORDER BY CS.population DESC
                                                    FETCH FIRST 3 ROWS ONLY) C2
WHERE C2.location1 = CS3.county_city_location AND CS3.case_id = P.case_id AND
P.party_id = V.party_id
        GROUP BY CS3.county_city_location, CS3.population, CS3.case_id
        ORDER BY average, CS3.county_city_location ) Q
WHERE rank < 11
```

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

```
1005   ,7.0,2048203,0
1005   ,7.0,2376747,0
1005   ,7.0,0644343,0
1005   ,7.0,2981117,0
1005   ,7.0,0360320,0
1005   ,7.0,3238600,0
1005   ,7.0,3050857,0
```

1005 ,7.0,2661658,0
1005 ,7.0,2473424,0
1005 ,7.0,0682784,0
3313 ,7.0,1397749,0
3313 ,7.0,2114294,0
3313 ,7.0,2245599,0
3313 ,7.0,3381097,0
3313 ,7.0,0062070,0
3313 ,7.0,3443703,0
3313 ,7.0,0861990,0
3313 ,7.0,2130214,0
3313 ,7.0,2477271,0
3313 ,7.0,2668975,0
4313 ,7.0,2840344,0
4313 ,7.0,3209284,0
4313 ,7.0,2226892,0
4313 ,7.0,1978697,0
4313 ,7.0,1071697,0
4313 ,7.0,2034278,0
4313 ,7.0,1949045,0
4313 ,7.0,2632596,0
4313 ,7.0,2537633,0
4313 ,7.0,2034270,0

Query g: Find all collisions that satisfy the following: the collision was of type pedestrian and all victims were above 100 years old. For each of the qualifying collisions, show the collision id and the age of the eldest collision victim.

Description of logic:

We should find the collisions with type pedestrian and the collisions that only involve victims who are older than 100 years old. I wrote a nested query that constructs a table of cases and their corresponding victims. Then using this table, I looked at all ages on a collision that is larger than 100 but smaller than 126 as the ages in this database vary between 0 and 125 and other values have different meanings. Then, I also selected the cases which have type_of_collision_description that includes the string ‘Pedestrian’. I printed the final case_id’s and their corresponding maximum ages.

SQL statement

```
SELECT C.case_id, MAX(V2.victim_age)
FROM Cases C, (SELECT C2.case_id AS cid, V.age AS victim_age
              FROM Victims V, Cases C2, Parties P
              WHERE C2.case_id = P.case_id AND P.party_id = V.party_id ) V2,
TYPE_OF_COLLISION TC
WHERE V2.cid = C.case_id AND V2.victim_age > 100 AND V2.victim_age < 126
      AND C.type_of_collision = TC.type_of_collision_id AND
TC.type_of_collision_description LIKE '%Pedestrian%'
```

GROUP BY C.case_id
Query result (if the result is big, just a snippet)
0445265,101
0784061,102
0868472,103
1794459,104
2472739,103
0566220,102
0644226,103
0819020,101
0958765,102
1209166,101
1548445,102
1873000,102
3485436,101
2531557,103
0036446,110
0439197,102
0815100,101
1213340,121
0851026,106
1373664,101
3388544,105
0817210,102
0820619,101
0828116,102
1347636,101
0069198,101
1847678,104

Query h: Find the vehicles that have participated in at least 10 collisions. Show the vehicle id and number of
collisions the vehicle has participated in, sorted according to number of collisions (descending order).
What do you observe?
Description of logic:
Since we do not have a Vehicle entity, we will print vehicle_make, vehicle_year and, vehicle_description to represent the vehicle_id. First, we construct a table by a nested query with the constructs of vehicle_id and find the number of collisions each vehicle_id involved in. Then using this table, we eliminated the vehicle_id's which are involved in less than 10 collisions and we ordered this table with respect to the descending number of collisions order. We observe that generally Toyota, Ford, and Honda are the vehicle_makes and passenger car is the vehicle_description that make the most collisions.

SQL statement

```
SELECT V2.vehicle_make, V2.vehicle_year, V2.vehicle_type, V2.no_of_collision
FROM (SELECT P.vehicle_make, P.vehicle_year, V.vehicle_description AS vehicle_type,
COUNT(*) AS no_of_collision
	FROM Statewide_Vehicle_Type V, Cases C, Parties P
	WHERE C.case_id = P.case_id AND P.statewide_vehicle_type = V.vehicle_id AND
P.vehicle_make IS NOT NULL AND P.vehicle_year IS NOT NULL
	GROUP BY P.vehicle_make, P.vehicle_year, V.vehicle_description) V2
WHERE V2.no_of_collision > 10
GROUP BY V2.vehicle_make, V2.vehicle_year, V2.vehicle_type, V2.no_of_collision
ORDER BY V2.no_of_collision DESC
```

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

TOYOTA,2000.0,passenger car,52504
FORD,2000.0,passenger car,51943
HONDA,2000.0,passenger car,50284
FORD,1998.0,passenger car,49182
TOYOTA,2001.0,passenger car,47232
HONDA,2001.0,passenger car,45277
FORD,2001.0,passenger car,45236
TOYOTA,1999.0,passenger car,42941
HONDA,1998.0,passenger car,42091
FORD,1999.0,passenger car,41948
FORD,1995.0,passenger car,40246
HONDA,1997.0,passenger car,39210
FORD,1997.0,passenger car,38885
HONDA,1999.0,passenger car,38556
TOYOTA,2002.0,passenger car,38427
TOYOTA,1998.0,passenger car,38012
TOYOTA,1997.0,passenger car,37158
TOYOTA,2003.0,passenger car,35943
HONDA,2002.0,passenger car,35785
FORD,2002.0,passenger car,35460

Query i: Find the top-10 (with respect to number of collisions) cities. For each of these cities, show the city location and number of collisions.

Description of logic:

I grouped the query by county_city_location to find the count of collisions of each city. I ordered the query with respect to number of collisions in descending order and when I fetched the first 10 rows, I get the 10 cities that have the largest number of collisions and I displayed the county_city_location and their respective collision numbers.

SQL statement

```
SELECT CS.county_city_location AS location, COUNT(*) as no_of_collisions
FROM Cases CS
```

```
GROUP BY CS.county_city_location
ORDER BY no_of_collisions DESC
FETCH FIRST 10 ROWS ONLY
```

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

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

Query j: Are there more accidents around dawn, dusk, during the day, or during the night? In case lighting
information is not available, assume the following: the dawn is between 06:00 and 07:59, and dusk between 18:00 and 19:59 in the period September 1 - March 31; and dawn between 04:00 and 06:00, and dusk between 20:00 and 21:59 in the period April 1 - August 31. The remaining corresponding times are night and day. Display the number of accidents, and to which group it belongs, and make your conclusion based on absolute number of accidents in the given 4 periods.

Description of logic:

To find the day periods, I first looked at the lightning attributes of all of the collisions on a nested query that gives a table with 1 column. If the corresponding lightning_description in Lightning entity for the lightning attribute on the Cases entity is “Dark”, then the value of the new attribute “time_period” is “night”. The new attribute is also constructed in another nested query. If the corresponding lightning_description is “Daylight” then the value of “time_period” is “day”. Apart from these, on other occasions (when the lightning is NULL or when the lightning_description is “dusk – dawn”), the time_period is determined by the month and the hour of the collision. According to the description of this query, we assigned values to the “time_period” attribute. There were some cases when the lightning_description said “dusk-dawn” and the collision_time suggested its day or night, we choose to use the value collision_time indicated as the lightning_description is a subjective attribute whereas the collision_time give us more concrete description of the collision. Then, using the new query Q that include the new attribute time_period, I write the main query in which the query is grouped by the time_period so that it can also display the corresponding count of each of the time_period. Then I ordered the query according to the descending order of count of each time period. The result is a table consist of 4 pay periods (day, night, dusk, and dawn) and the null values. And looking at the top row, we can see that the time period that has the most collision number is “day”.

SQL statement

```
SELECT Q.time_period, COUNT(*)
FROM (SELECT C.collision_time, L.lightning_description,
    (CASE WHEN L.lightning_description LIKE '%Dark%' THEN 'night'
      WHEN L.lightning_description LIKE '%Daylight%' THEN 'day'
      ELSE (CASE WHEN EXTRACT(MONTH FROM C.collision_time) BETWEEN 4 AND 8
THEN
            (CASE WHEN EXTRACT(HOUR FROM C.collision_time) BETWEEN 4 AND 5
THEN 'dawn'
              WHEN EXTRACT(HOUR FROM C.collision_time) BETWEEN 6 AND 19
THEN 'day'
              WHEN EXTRACT(HOUR FROM C.collision_time) BETWEEN 20 AND 21
THEN 'dusk'
              ELSE 'night' END)
            WHEN EXTRACT(MONTH FROM C.collision_time) IS NULL THEN NULL
            ELSE (CASE WHEN EXTRACT(HOUR FROM C.collision_time) BETWEEN 6
AND 7 THEN 'dawn'
              WHEN EXTRACT(HOUR FROM C.collision_time) BETWEEN 8 AND 17
THEN 'day'
              WHEN EXTRACT(HOUR FROM C.collision_time) BETWEEN 18 AND 19
THEN 'dusk'
              ELSE 'night' END) END) END) AS time_period
  FROM Cases C, Lightning L
  WHERE L.lightning_id = C.lightning) Q
GROUP BY Q.time_period
ORDER BY COUNT(*) DESC
```

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

day,2559367

night,1037292

dawn,28888

dusk,25705

,1030

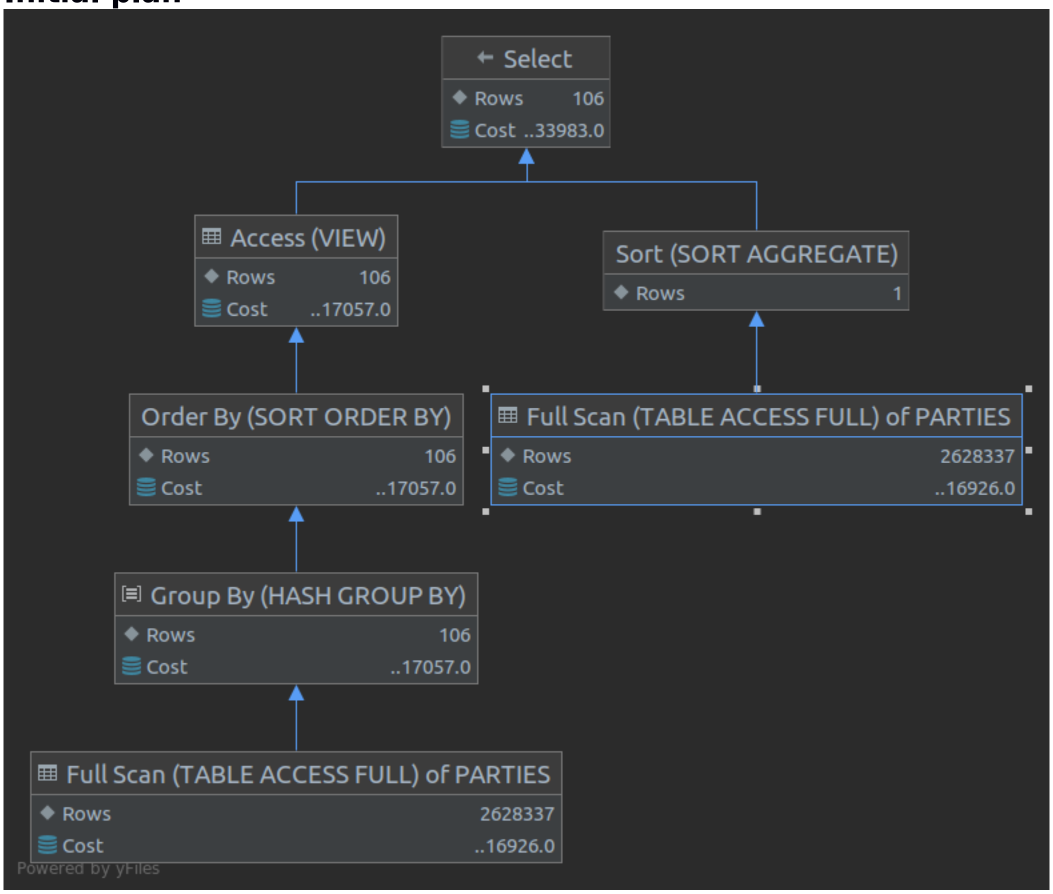
Query Performance Analysis – Indexing

Query A
Initial Running time/IO: 6 s 312 ms / 33650
Optimized Running time/IO: 2 s 237 ms / 8648

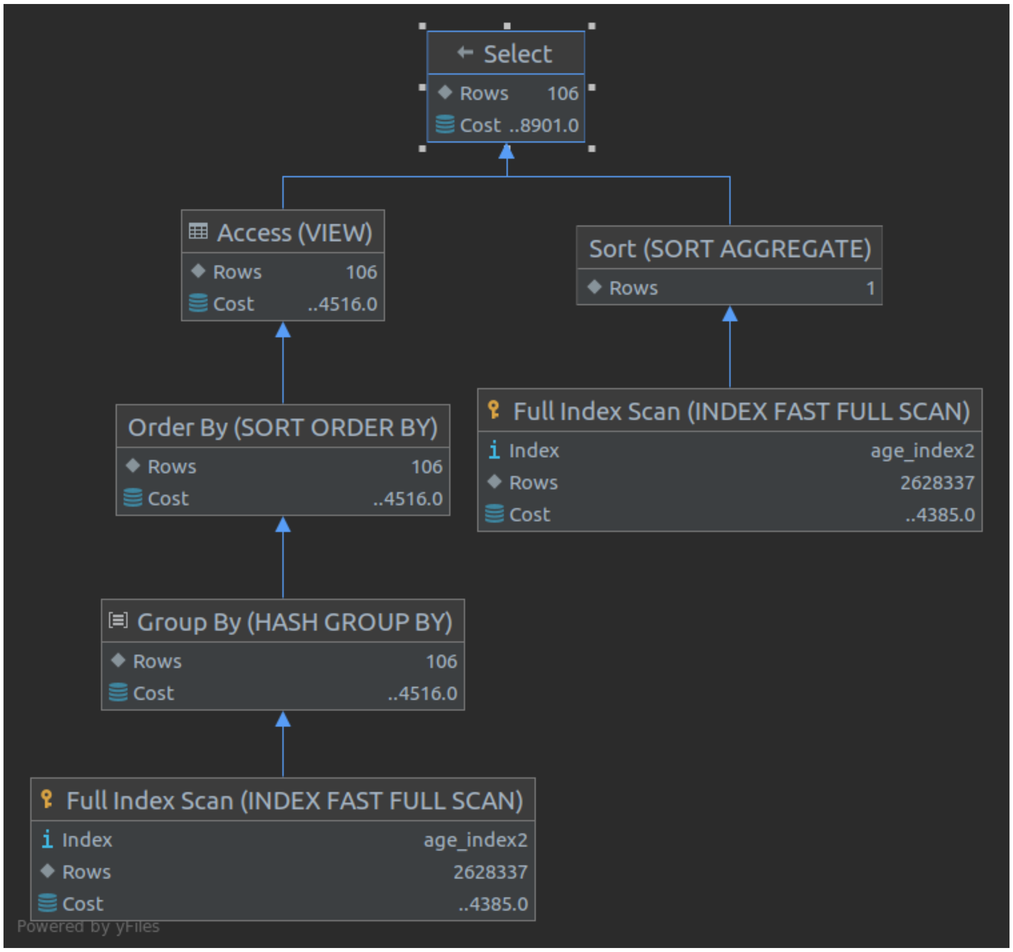
DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Explain the improvement: Normally, the selection of party age ranges requires a full scan of party table. We created a B+ tree index on Parties table with party age and at fault. In this way, the selection of party age ranges can be done by full index scan, which is way cheaper than full scan of parties table. Note that we added at fault to indexing to create an index-only table. Initial plan cost is 33983 while improved plan cost is 8901, which justifies the differences in running time.

Initial plan

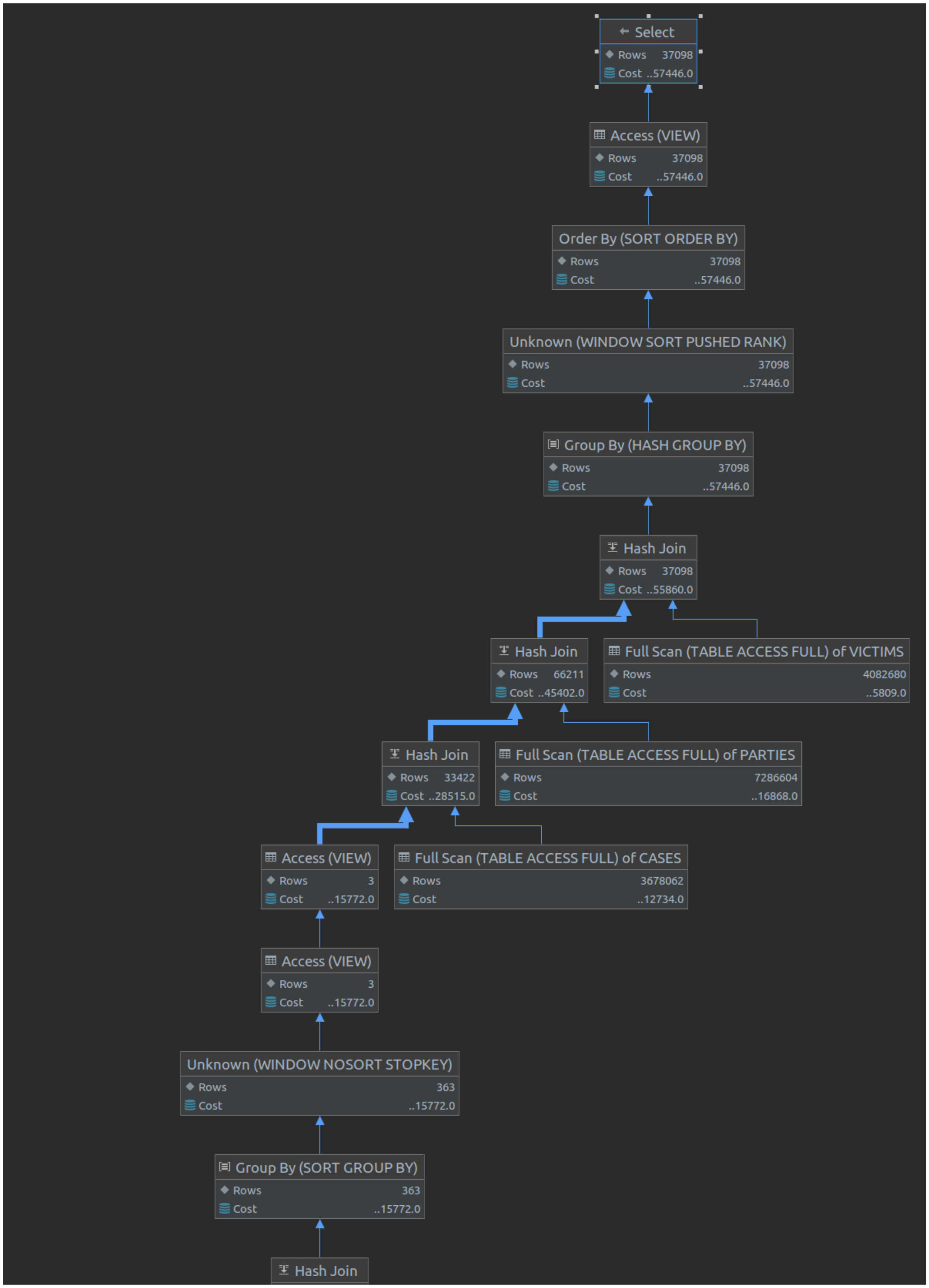


Improved plan



Query F
Initial Running time/IO: 10 s 663 ms / 57182
Optimized Running time/IO: 6 s 422 ms / 41649
Explain the improvement: In the original query, several full scans on Cases is done to search for county-city-location. This consumes a considerable amount of time. We created a bitmapped index on Cases table with county-city-location. This improvement converts the full scans on Cases to index scans, which are more efficient. Initial plan cost is 57446 while improved plan cost is 41845, which justifies the differences in running time.
Initial plan

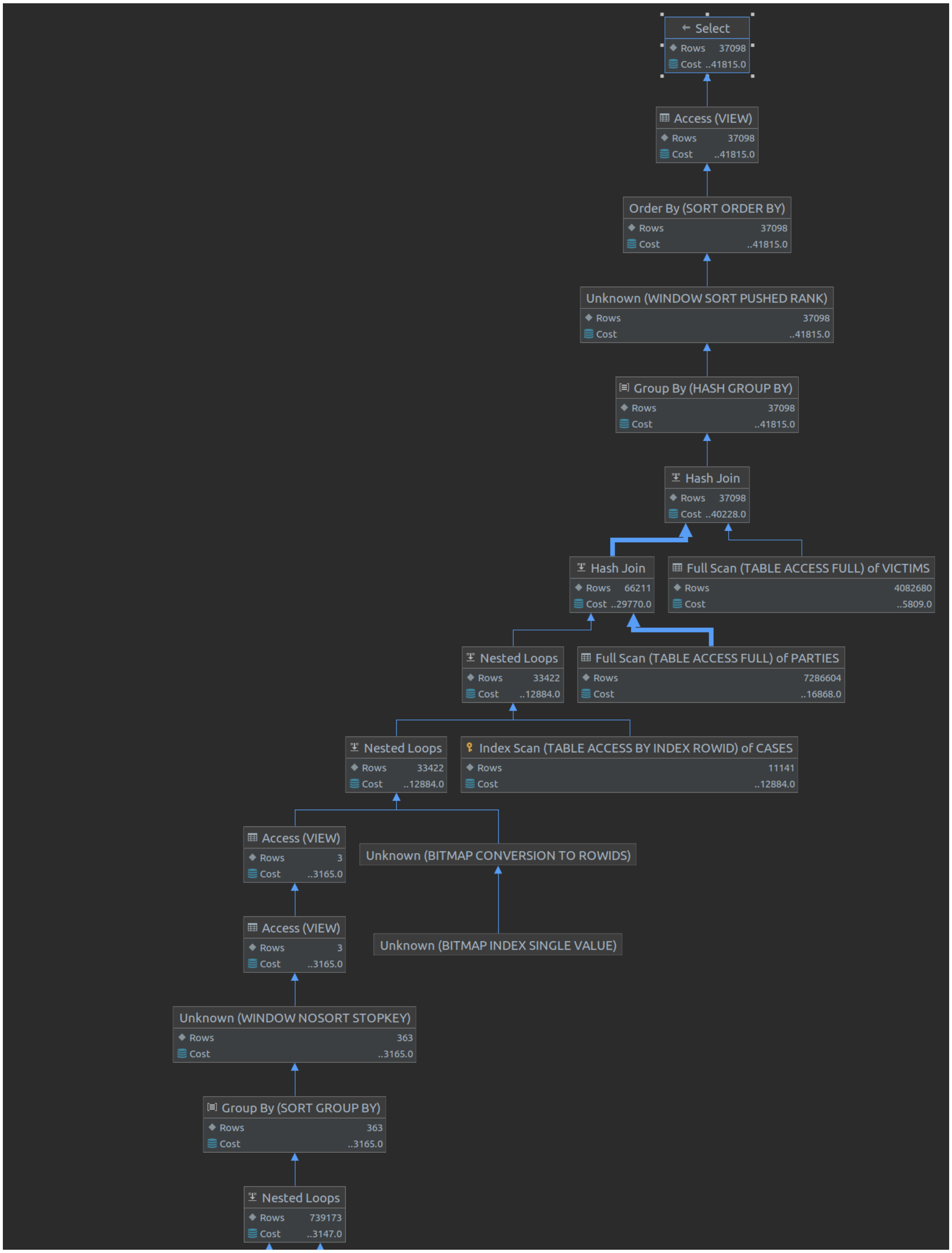
DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/



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/

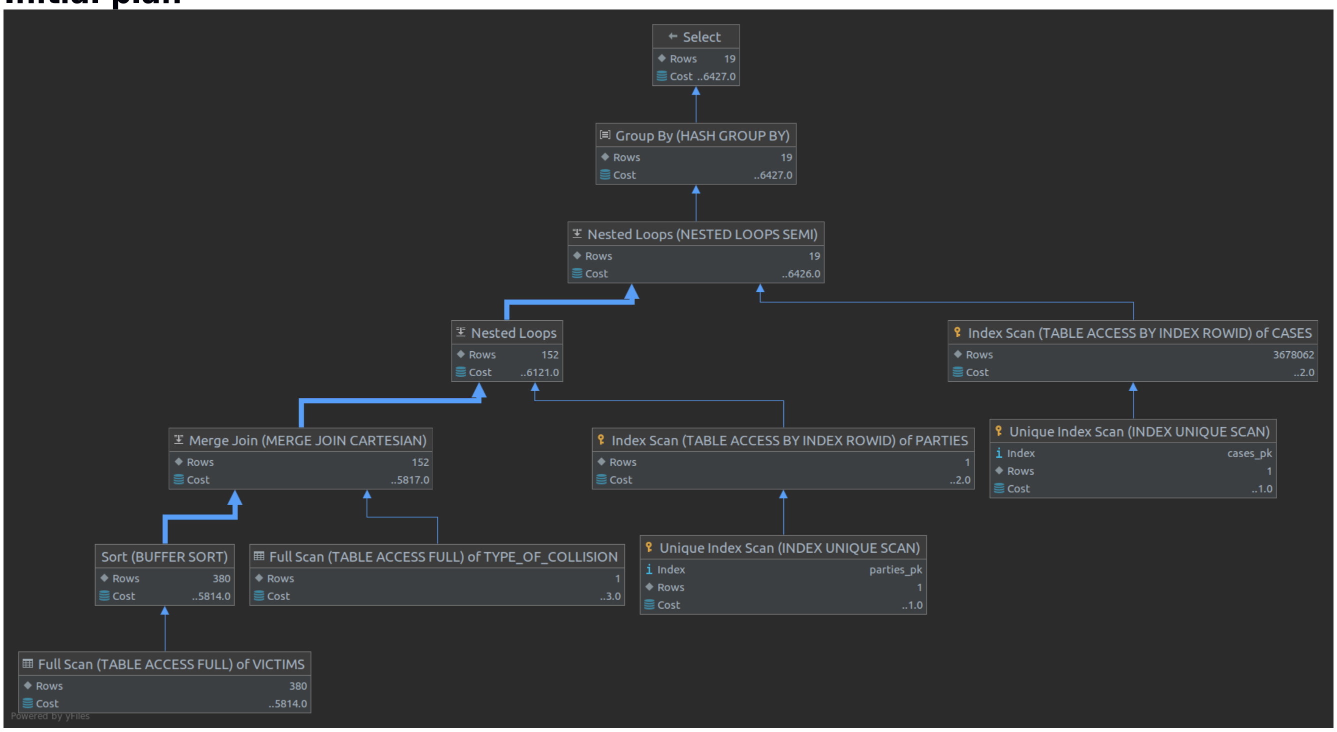
Improved plan

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/



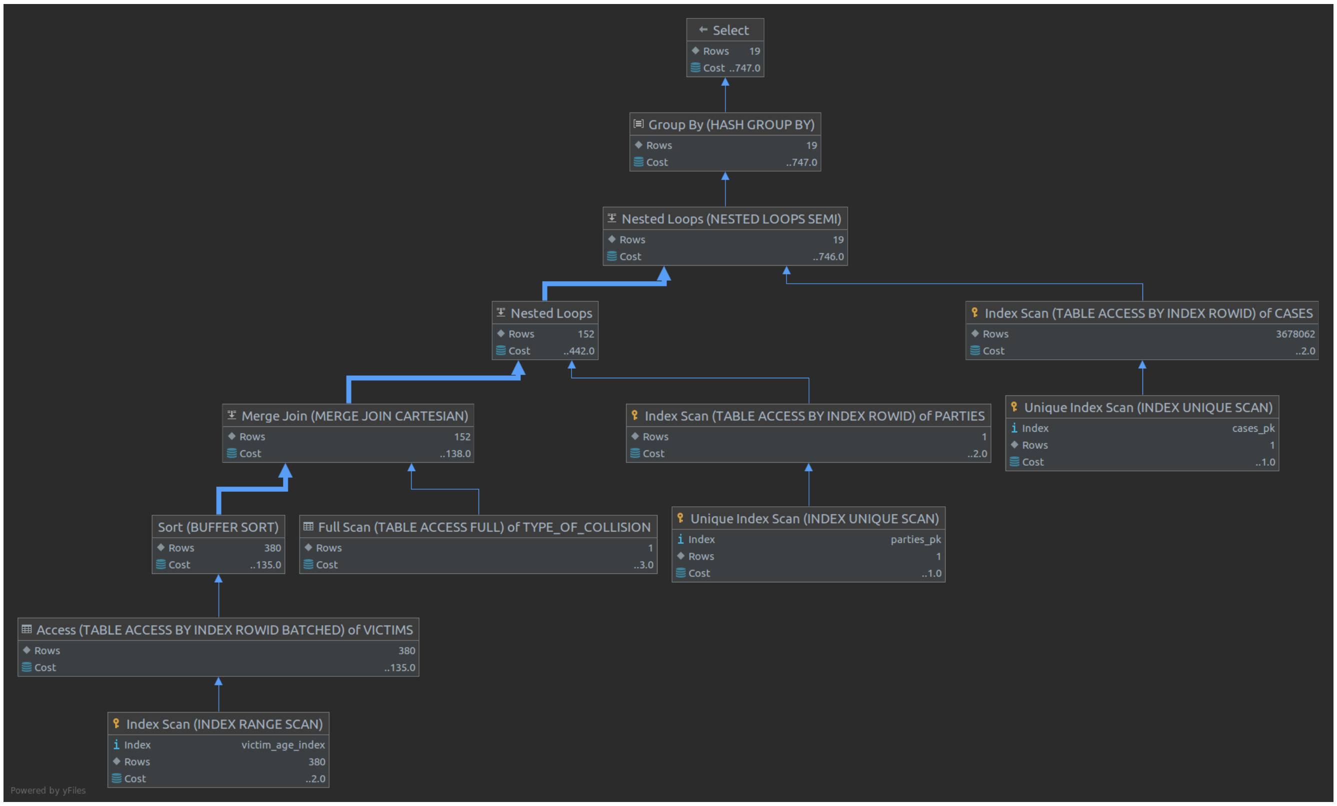
DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Query G
Initial Running time/IO: 881 ms / 6386
Optimized Running time/IO: 170 ms / 746
Explain the improvement: Normally, the selection of victim age range requires a full scan of victim table. We created a B+ tree index on Victims table with victim age. In this way, the selection of victim age range can be done by index range scan of victim age index, which is way cheaper than full scan of victim table. Initial plan cost is 6427 while improved plan cost is 747, which justifies the differences in running time.
Initial plan



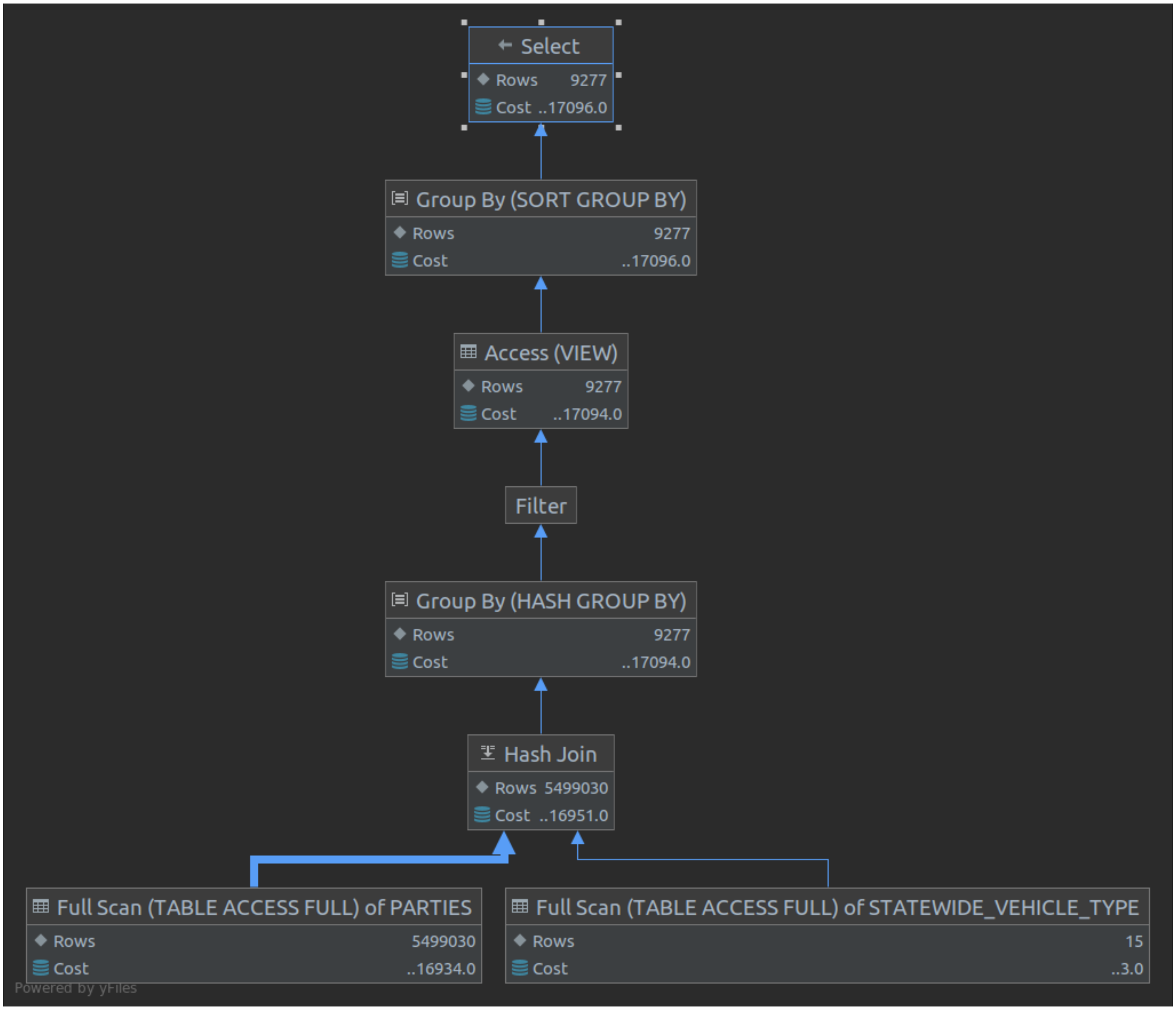
Improved plan

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/



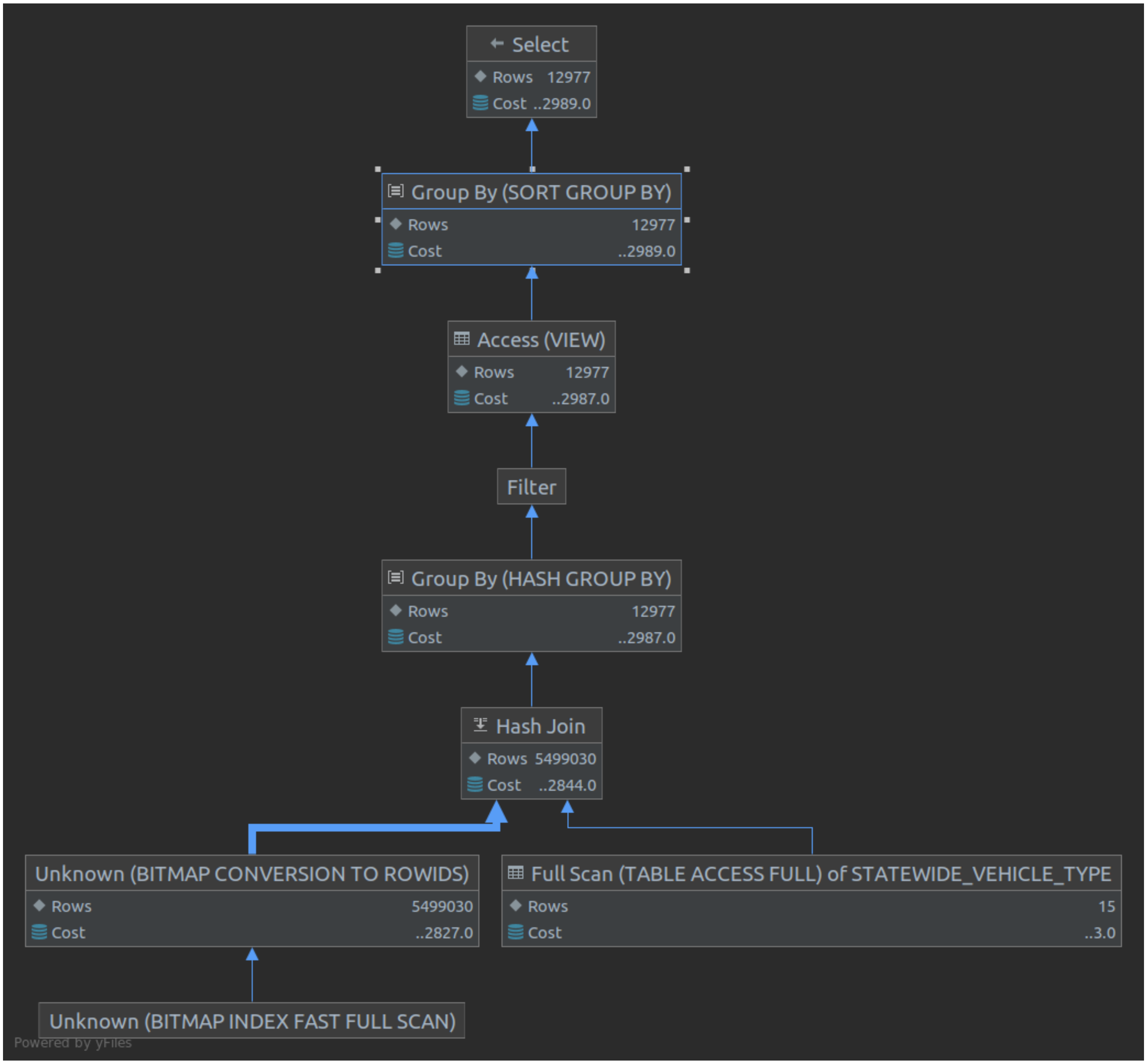
Query H
Initial Running time/IO: 4 s 623 ms / 16828
Optimized Running time/IO: 1 s 341 ms / 2829
Explain the improvement: In this query, parties are grouped by their vehicle information and some counting is done. No other information in parties is used. So we created a bitmapped index on Parties with vehicle info, which is statewide-vehicle-type, vehicle-make and vehicle-year. Instead of full scan on Parties table for vehicle info, this index enabled the query to get the neccesary info with only index fast full scan, which is more faster. Initial plan cost is 17096 while improved plan cost is 2989, which justifies the differences in running time.
Initial plan

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/



Improved plan

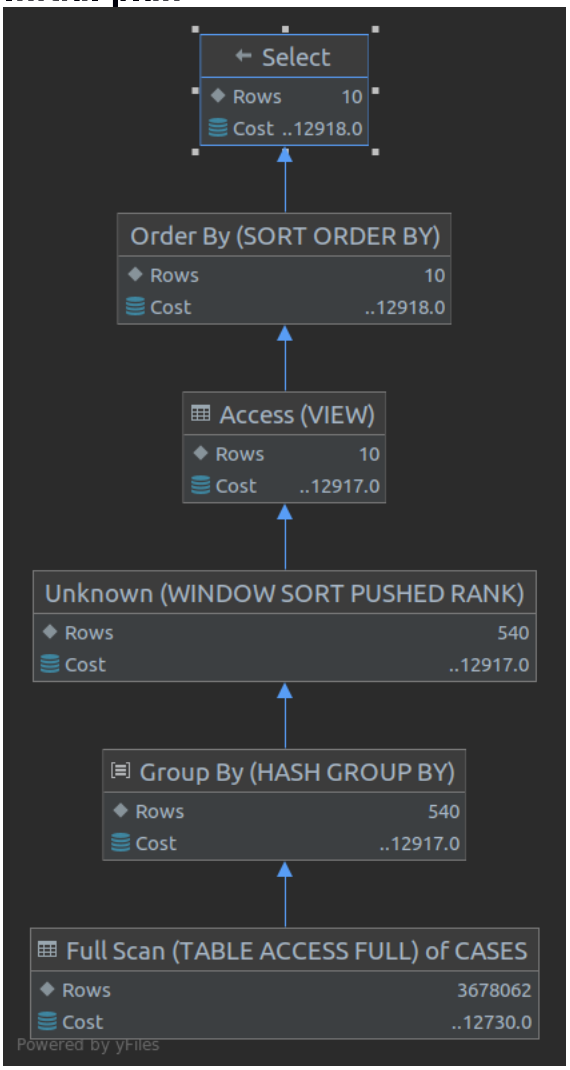
DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/



Query I
Initial Running time/IO: 1 s 340 ms / 12681
Optimized Running time/IO: 135 ms / 543
Explain the improvement: Normally, full scan of the whole Cases table is required to count the location types. We used bitmapped index on Cases table with county_city_location column. In this way, index fast full scan is enough to get location types and counts. Index fast full scan is faster than full scan of Cases, so improvement is

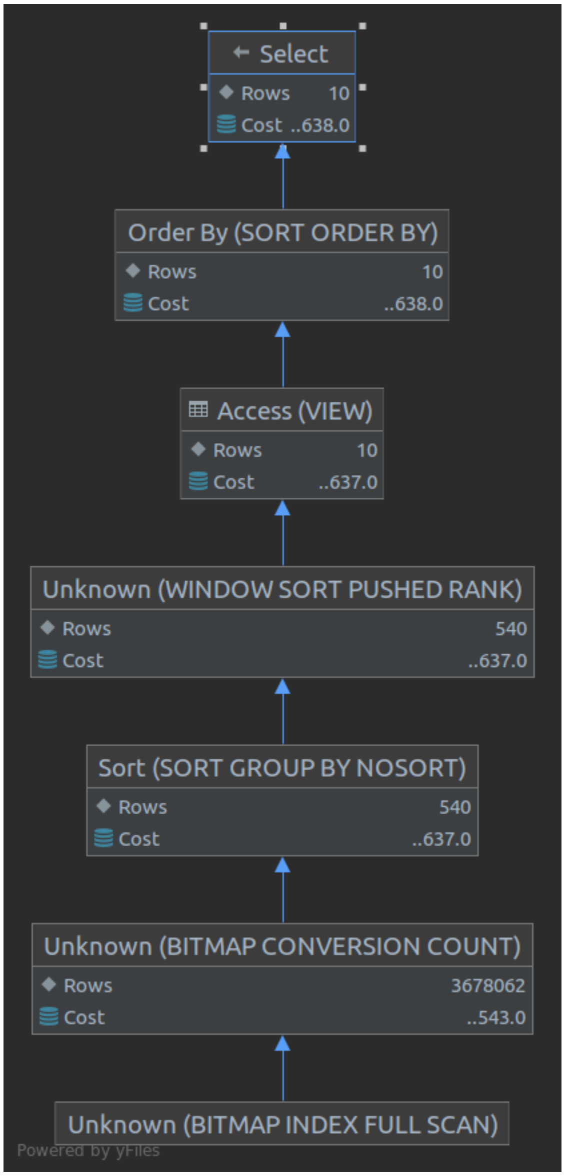
gained from this. Initial plan cost is 12918 while improved plan cost is 638, which justifies the differences in running time.

Initial plan



Improved plan

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 updated the queries from Deliverable 2 and wrote their query results. We deleted the "Cities" entity and opened a new entity for population_id and population_description named "Population". We included the attributes jurisdiction, county_city_location and, population into the Cases entity.

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/

Ömer Faruk Akgül: Wrote the queries from query a to query e.

Öykü Irmak Hatipoğlu: Wrote the queries from query f to query j.

Burak Öçalan: Query performance analysis and indexing.