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: G34

Members: Siran Li, Ke Wang, Ningwei Ma

Contents

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

Deliverable 3

Changes

Based on the feedback, we have made some changes including ER model, DDL, data processing and query implementation of Deliverable 2. We have quoted some of the feedback and have responded to it.

ER MODEL

Modifications respective to Feedback on M2 (the modifications are based on the feedback on ER:

-- 'Yes, I think that the "condition" entity could be merged with "case". It doesn't give you any specific advantage to keep these two entities as separate.'

Removed the 'condition' entity and moved all its attributes to the 'case' entity.

-- 'For example, you can create a new entity (and the corresponding table) for "road conditions", with road condition as an attribute and the case_id acts as a foreign key referred from the collision table. Note that in this case, both the case_id and the road_condition serve as the PK.'

Changed the attributes of 'road_en' entity into 'case_id' and 'road_con', the pairs of them serve as primary keys.

-- 'Shouldn't you have a foreign key reference of the primary keys of party into this table? Also, I don't think factors can exist without a party. Lastly, I don't think you need a separate table for the "have" relationship b/w party and other_fac_en, it is absorbed as a single table for 'other_fac_en'.'

Changed the attributes of 'other_fac_en' entity into 'party_id' and 'other_fac', both of them serve as primary keys. Replaced the attribute 've_num' of 'vehicle' entity by 'party_id'.

--'It is indeed correct to have a single "safety_equipment" entity in the ER, however, during the translation I think you would need two separate tables: safetey_equipment for party and victim. Similar to the previous point and with a very similar explanation, I don't think you need separate tables for the relationships "have_ps" and "have_vs".'

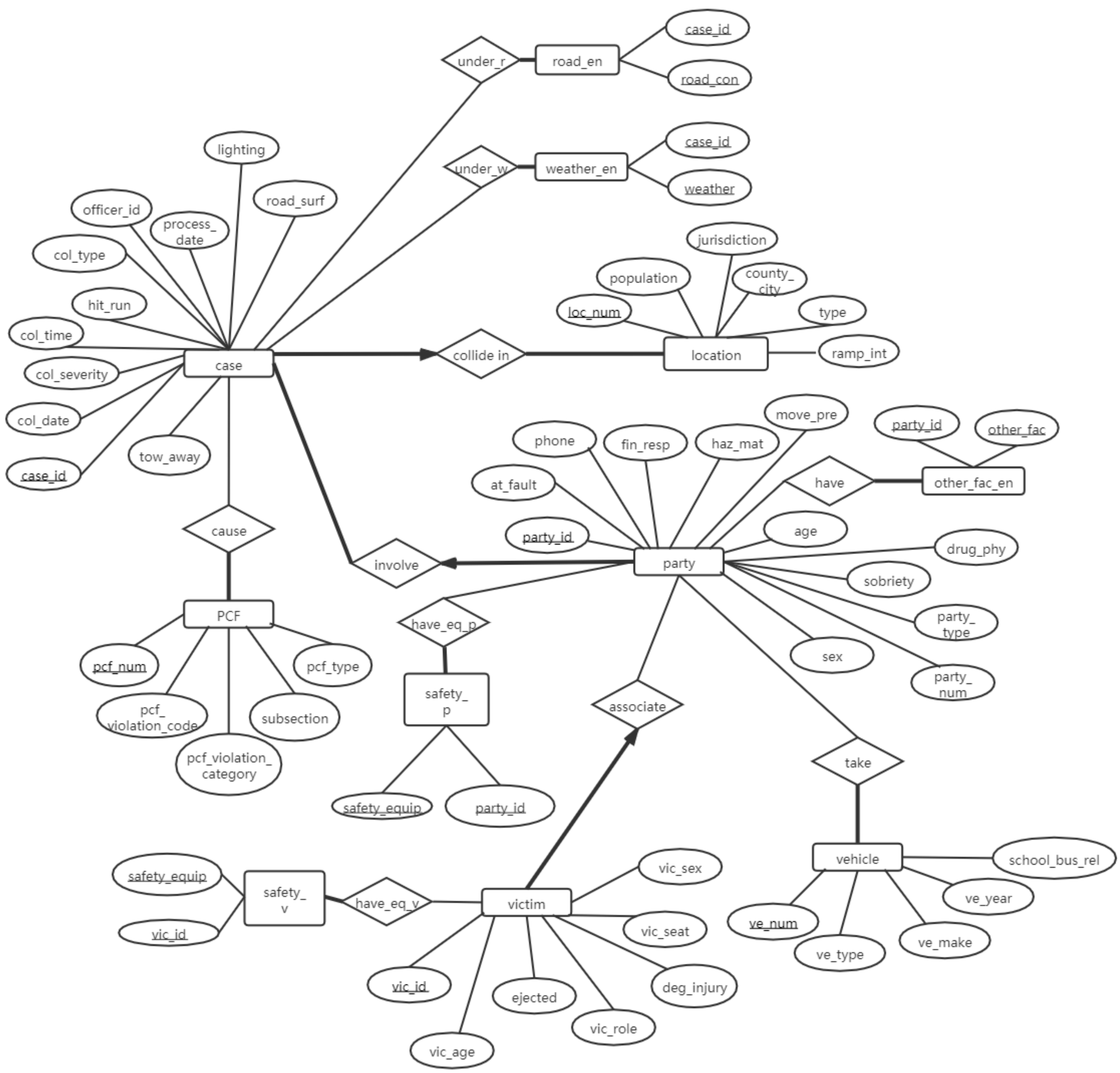
For this feedback, we decided to separate the 'other_fac_en' entity into two entities in the ER model as well.

Separated the 'safety_equip_en' entity into two entities 'safety_p' (the safety equipment for party) and 'safety_v' (the safety equipment for victim').

For 'safety_p', its attributes are changed to 'safety_equip' and 'party_id', both of them serve as primary keys.

For 'safety_v', its attributes are changed to 'safety_equip' and 'vic_id' both of them serve as primary keys.

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



DDL

In response to deliverable 2's feedback, we have modified the DDL.

--' I don't think you need a separate table for the "have" relationship b/w party and other_fac_en, have_ps, have_vs, under_r, under_w, take'

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/

We deleted the relational tables such as under_w, under_r, have, have_ps, have_vs, and merged them to the independent entity tables including weather_en, other_fac_en, safety_equipment, road_en, respectively.

Now in the relational table, such conditions (weather, road condition, other_fac, safety_p, safety_v, road_con) will be like an attribute and the case_id/party_id/victim_id will act as a foreign key referred from the corresponding table. We use the pairs of (xx_id, xx_condition) as the primary key to ensure that they are unique.

--'other_fac_en should be added the 'on delete cascade' constraint. '

In these tables (other_fac_en, road_en, weather_en, safety_p, safety_v), we add "on delete cascade" constraint on the foreign key, because these record cannot exist without party/case/victim. Once the corresponding party/case/victim is deleted from the database, the record related to them would also be deleted.

--'Also, shouldn't ve_num be added as a foreign key in the party table? Lastly, similar to the above two points, I don't think you need a separate table for the "take" relationship.'

Each case happens under exactly one location and PCF and each party drives one kind of vehicle(by the data and our way of data processing). We deleted the relational table 'take' between party_involve-take-vehicle, but just added a column "ve_num" in the party_involve table referring to the "ve_num" of vehicle table. We also deleted the relation 'cause' table between PCF-casue-case and added a foreign key "pcf_num" to the case table referring to the "pcf_num" of PCF table.

--'Recall that party can/should not exist without a collision, and victim can/should not exist without a party.'

We added "on delete cascade" constraint on the "party_id" foreign key in the associate_victim table, and the "case_id" foreign key in the party_involve table, because party is the weak entity of the case, and the victim if the weak entity of the party (by our assumption and data processing).

--'Since party is a weak entity of case, shouldn't you have a foreign key reference of the primary keys of party into this table?'

We modeled the party a weak entity of the case entity. We put the foreign key reference of the case_id in the party entity, based on this assumption: party has exactly one participation in this relationship, but one case may be related to many parties. So, we think it might be more reasonable to put the foreign key to case in the party table. And thus we did not put foreign key refering to party in the case table, to avoid a reference loop.

--'I just want to make sure that the duplicates are not because of the datatype issue (you are modeling case_id as integer), and are genuine duplicates'

We changed the type of "case_id" to VARCHAR2, because there may be several zeros in front of the number, which is also a part of the identity. Now there is no duplicated case_id. (We did not change the type of party_id and vict_id, because they do not have duplicates)

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/

-- 'Yes, I think that the "condition" entity could be merged with "case". It doesn't give you any specific advantage to keep these two entities as separate.'

We have merged the condition table into the case table, with table condition's two attributes becoming the attributes of case now.

We have 11 tables now, and the DDL is attached below. Modified parts are marked in yellow.

==

```
CREATE TABLE PCF(

pcf_num INTEGER,

pcf_violation_code INTEGER,

pcf_violation_category VARCHAR2(50),

subsection VARCHAR2(3),

pcf_type VARCHAR2(50),

PRIMARY KEY (pcf_num)

);


CREATE TABLE Vehicle(

ve_num INTEGER,

ve_type VARCHAR2(50),

ve_make VARCHAR2(30),

ve_year INTEGER,

school_bus_rel VARCHAR2(5),

PRIMARY KEY (ve_num)

);


CREATE TABLE Location(

loc_num INTEGER,
```

```
population INTEGER,
county_city INTEGER,
loc_type VARCHAR2(20),
ramp_int VARCHAR2(10),
PRIMARY KEY (loc_num)
);

CREATE TABLE Case(
case_id VARCHAR2(30),
loc_num INTEGER NOT NULL,
pcf_num INTEGER NOT NULL,
col_severity VARCHAR2(30),
col_time DATE,
col_date DATE,
hit_run VARCHAR2(30),
jurisdiction INTEGER,
officer_id VARCHAR2(10),
process_date DATE,
tow_away INTEGER,
col_type VARCHAR2(30),
lighting VARCHAR2(50),
road_surf VARCHAR2(10),
PRIMARY KEY (case_id),
FOREIGN KEY (loc_num) REFERENCES Location(loc_num),
FOREIGN KEY(pcf_num) REFERENCES PCF(pcf_num)
```

```
);

CREATE TABLE Party_involve(
party_id INTEGER,
case_id VARCHAR2(30) NOT NULL,
at_fault INTEGER,
phone VARCHAR2(3),
fin_resp VARCHAR2(3),
haz_mat VARCHAR2(3),
move_pre VARCHAR2(30),
age INTEGER,
drug_phy VARCHAR2(3),
sobriety VARCHAR2(3),
party_type VARCHAR2(15),
party_num INTEGER,
sex VARCHAR2(6),
ve_num INTEGER,
PRIMARY KEY (party_id),
FOREIGN KEY (case_id) REFERENCES Case(case_id)
ON DELETE CASCADE,
FOREIGN KEY (ve_num) REFERENCES Vehicle(ve_num)
);

CREATE TABLE Associate_victim(
vic_id INTEGER,
```

```
party_id INTEGER NOT NULL,

vic_age INTEGER,

ejected INTEGER,

vic_role INTEGER,

deg_injury VARCHAR2(50),

vic_seat INTEGER,

vic_sex VARCHAR2(6),

PRIMARY KEY (vic_id),

FOREIGN KEY (party_id) REFERENCES Party_involve(party_id)

ON DELETE CASCADE

);


CREATE TABLE Other_fac_en(

party_id INTEGER,

other_fac VARCHAR2(3),

PRIMARY KEY (party_id, other_fac),

FOREIGN KEY (party_id) REFERENCES Party_involve(party_id)

ON DELETE CASCADE

);


CREATE TABLE road_en(

case_id VARCHAR2(30),

road_con VARCHAR2(20),

PRIMARY KEY (case_id, road_con),

FOREIGN KEY (case_id) REFERENCES Case(case_id)
```

```
ON DELETE CASCADE
);

CREATE TABLE Weather_en(
case_id VARCHAR2(30),
weather VARCHAR2(20),
PRIMARY KEY (case_id, weather),
FOREIGN KEY (case_id) REFERENCES Case(case_id)
ON DELETE CASCADE
);

CREATE TABLE safety_p(
party_id INTEGER,
safety_equip VARCHAR2(3),
PRIMARY KEY (party_id, safety_equip),
FOREIGN KEY (party_id) REFERENCES Party_involve(party_id)
ON DELETE CASCADE
);

CREATE TABLE safety_v(
vic_id INTEGER,
safety_equip VARCHAR2(3),
PRIMARY KEY (vic_id, safety_equip),
FOREIGN KEY (vic_id) REFERENCES Associate_victim(vic_id)
ON DELETE CASCADE);
```

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

==

Query Implementation of Deliverable 2

Query 1:

Description of logic:

List the year-number of collisions per year. We use "group by" to group case by year (extracted from col_date) and count the number of cases of each year.

SQL statement

SELECT EXTRACT (YEAR FROM col_date) AS YEAR, count(*) AS N_collisions
FROM case
GROUP BY EXTRACT (YEAR FROM col_date)
ORDER BY YEAR ASC

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

YEAR	N_COLLISIONS
2001	522562
2002	544741
2003	538954
2004	538295
2005	532725
2006	498850
2007	501908
2017	7
2018	21

Query 2:

Description of logic:

In the "take" table, group entries by "ve_make" and count the number of parties of each "ve_make", then find the max count and the corresponding "ve_make". Before that we need to use "ve_number" to know the "ve_make", so we first join table "vehicle" and "party_involve". To illustrate the whole row of the most popular, we fetch first 1 row only.

SQL statement

SELECT VE_MAKE, COUNT(*) AS N_COLLISION

```
FROM (VEHICLE INNER JOIN PARTY_INVOLVE ON VEHICLE.VE_NUM = PARTY_INVOLVE.VE_NUM)
GROUP BY VE_MAKE
ORDER BY N_COLLISION DESC
FETCH FIRST 1 ROWS ONLY
```

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

VE_MAKE	N_VEHICLE
FORD	1129701

Query 3:

Description of logic:

In the lighting attribute of condition, find the description that contains “dark”, and count the fraction of cases that occur in such condition, and keep 2 significant digits..

SQL statement

```
SELECT ROUND(NOM/(SELECT COUNT(*) FROM CASE),2)
FROM
  (SELECT COUNT(*) AS NOM
  FROM CASE
  WHERE CASE.LIGHTING LIKE '%dark%')
```

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

FRACTION
0.28

Query 4:

Description of logic:

Find the number of collisions that have occurred under snowy weather. We count the number of entries that have weather_con = ‘snowing’ in the table “weather_en”

SQL statement

```
SELECT COUNT(*)
FROM WEATHER_EN
WHERE WEATHER LIKE '%snowing%'
```

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

N_COLLISIONS
8530

Query 5:

Description of logic:

Group by collisions by which day they are during a week, and count the total number of collisions of that day, then find the row of highest number of cases. We use TO_CHAR (COL_DATE, 'D') to extract the day of the week.

SQL statement

```
SELECT TO_CHAR(COL_DATE, 'D') AS WEEK_DAY, COUNT(*) AS N_COLLISONS
FROM CASE
GROUP BY TO_CHAR(COL_DATE, 'D')
ORDER BY N_COLLISONS DESC
FETCH FIRST 1 ROWS ONLY
```

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

WEEK_DAY	N_COLLISIONS
6	614853

Query 6:

Description of logic:

List all weather types and their corresponding number of collisions in ascending order of the collisions.We group cases by "weather" and list "weather" and the count number.

SQL statement

```
SELECT WEATHER, COUNT(*) AS N_COLLISION
FROM WEATHER_EN
GROUP BY WEATHER
ORDER BY N_COLLISION
```

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

WEATHER	N_COLLISION
other	6960
snowing	8530
wind	13952
fog	21259
raining	223752
cloudy	548250
clear	2941042

Query 7:

Description of logic:

Count the number of parties that are at-fault, with financial responsibility and loose material. We first extract the "road_num" of "road_loose", and find which parties are associated with such road condition. We filter the "party_id" table who is at fault and with financial responsibility. Finally we count the number of the selected parties.

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

SQL statement

```
  SELECT COUNT(*) AS N_PARTIES
  FROM PARTY_INVOLVE P, ROAD_EN R
  WHERE P.CASE_ID = R.CASE_ID AND P.AT_FAULT = 1 AND P.FIN_RESP = 'Y' AND R.ROAD_CON LIKE '%loose
material%'
```

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

N_PARTIES
4803

Query 8:

Description of logic:

Find the median victim age: we directly use the "MEDIAN" function of SQL from the associate_victim table.

Find the most common victim seating position. We group the victims with seating position, and count the number of victims of each vic_seat, order them in the descending order of this number and find the max.

SQL statement

8.a

```
SELECT MEDIAN(vic_age) AS MEDIAN_VIC_AGE
FROM ASSOCIATE_VICTIM v2;
```

8.b

```
SELECT VIC_SEAT AS MOST_COMMON_SEAT_POSITION
FROM
  (SELECT COUNT(vic_seat) AS count, vic_seat
   FROM associate_victim v2
   GROUP BY vic_seat
   ORDER BY count DESC)
FETCH FIRST 1 ROWS ONLY;
```

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

8.a

MEDIAN_VIC_AGE
25

8.b

MOST_COMMON_SEAT_POSITION
3

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

Query 9:

Description of logic:

Fraction of all participants (victims + parties) that have been victims using a belt. All participants refer to both parties and victims, so our denominator is the sum of number of all victims and parties. We first extract the 'vic_id's who use belt using table have_vs and safety_equip_en. Then we count the unique 'vic_id's and use this number as the numerator. Finally we get the fraction and keep 2 significant digits.

SQL statement

```
SELECT ROUND(A.FRACTION,3) AS fraction
FROM(SELECT DISTINCT
     (SELECT  COUNT(vic_id) AS count
        FROM
          (SELECT h1.vic_id as vic_id
             FROM SAFETY_V H1
             WHERE H1.SAFETY_EQUIP like '%C%') v_belt)/
     ((SELECT COUNT(party_id) FROM party_involve)
     +(SELECT COUNT(vic_id) FROM associate_victim)) as fraction
      FROM party_involve) a
```

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

FRACTION
0.011

Query 10:

Description of logic:

Compute the fraction of collisions happening for each hour of the day, and display the ratio as percentage for all the hours of the day. We first use cast(col_time as timestamp) to extract the hour in which the case occurred. Then we group the cases by the specific hour and count the number of the cases, then order them by the number. We also directly calculate the count of total cases. Then we divide the count number of each hour by the total number to get each fraction.

SQL statement

```
SELECT DISTINCT
EXTRACT(hour from cast(col_time as timestamp)) as hour, CONCAT( ROUND((COUNT(*)/(SELECT COUNT(*)
FROM CASE)*100.0),2),'%') as FRACTION
FROM CASE
   GROUP BY EXTRACT(hour from cast(col_time as timestamp))
   ORDER BY hour ASC
```

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/

HOUR	*FRACTION*
0	*1.91%*
1	*1.83%*
2	*1.81%*
3	*1.15%*
4	*0.98%*
5	*1.45%*
6	*2.62%*
7	*5.17%*
8	*5.23%*
9	*4.09%*
10	*4.23%*
11	*4.89%*
12	*5.78%*
13	*5.78%*
14	*6.55%*
15	*7.75%*

16	*7.33%*
17	*7.91%*
18	*6.30%*
19	*4.43%*
20	*3.49%*
21	*3.28%*
22	*2.86%*
23	*2.38%*
(null)	*0.81%*

Data processing

We made modifications based on the following feedbacks:

-- 'How are you generated identifiers: e.g. pcf_num, loc_num, con_num, etc.?'

For identifiers of pcf_num, ve_num and loc_num (now con_num has been deleted), we used Pandas to generate indexes on the dataframes (reset_index(drop=True)). Thus the indexes are all natural numbers and unique. We generated these indexes to serve as keys for respective entities.

-- 'How are you handling attributes with missing values, if any? Especially, the attributes that are important and shouldn't/can't be simply ignored.'

Case_id, party_id and victim_id do not and should not have null values.

For road_en, we first extracted all distinct road_en values and dropped the null value, then joined it with the 2 columns of road condition in the *collisions2018.csv* to get the not null (case_id, road_con) pairs, and dropped the duplicates. In this way if a collision's road condition is unavailable, it will not appear in the road_en table, and all the pairs are unique. The same method also works for weather_en, other_fac_en, safety_p and safety_v.

For vehicle, we first extracted all the distinct tuples of their attributes, and we did not drop the null values, so there will not be null value in the party_involve table for ve_num, but there is a row of "ve_num | null | null | null | null" is the vehicle table. It is the same for location and PCF table.

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

Based on our assumptions on M1 and M2, we made the following further assumptions.

-- 'I just want to make sure that the duplicates are not because of the datatype issue (you are modeling case_id as integer), and are genuine duplicates. Please reanalyze this part.'

As stated in previous parts, we changed the type of case_id into string now, which fixed this problem. The duplicates are indeed caused by the datatype issue, and now the problem has been fixed.

*-- (Feedback on M1) '"Party_number refers to the specific party of a particular case, so party_number + case_id is unique for each party, playing the same role as party_id." * Have you verified this from the data?'*

Furthermore, after fixing this problem, it supports our earlier assumption "case_id+party_num" uniquely identifies 'party_id'.

Query Implementation

Query a:

Description of logic:

We first select parties whose "party_type" is 'driver' and the "age" is not null. Then we just project columns of "age" and "at_fault" and classify them into different age groups. If the party is at fault, the value of "at_fault" is 1, otherwise it is 0 (if it's not null), so we can just calculate the sum of "at_fault" of each age group, and then divide the sum by the count of each age group to get the ratio

SQL statement

SELECT

(CASE WHEN AGE<=18 THEN 'underage'

WHEN AGE BETWEEN 19 and 21 then 'young I'

when AGE BETWEEN 22 AND 24 THEN 'young II'

when AGE BETWEEN 24 AND 60 THEN 'adult'

when AGE BETWEEN 61 AND 64 THEN 'elder I'

else 'elder II' END) as AGE_RANGE,

CONCAT(100*ROUND(SUM(AT_FAULT)/COUNT (AT_FAULT), 3), '%') as ratio

FROM (SELECT AGE, AT_FAULT FROM PARTY_INVOLVE WHERE PARTY_TYPE like '%driver%' and AGE IS NOT NULL and AT_FAULT IS NOT NULL)

group by (CASE WHEN AGE<=18 THEN 'underage'

WHEN AGE BETWEEN 19 and 21 then 'young I'

when AGE BETWEEN 22 AND 24 THEN 'young II'

when AGE BETWEEN 24 AND 60 THEN 'adult'

when AGE BETWEEN 61 AND 64 THEN 'elder I'

ELSE 'elder II' END)

order by RATIO DESC

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

AGE_RANGE	RATIO
underage	64.70%
young I	58%
young II	51.90%
elder II	50.50%
adult	41.00%
elder I	40.10%

We found the underage, young I and young II group has the 3 highest at_fault ratio. If we were an insurance company, I will raise premiums for young people under 24 years old and the elder over 65 years old.

Query b:

Description of logic:

We join the party_involve table with vehicle on ve_num to get the vehicle details for each party, then we join it with road_en (whose road_con contains “holes”) on case_id to get the road_condition of the case that the party is involved. If in a case there are vehicles with the same type, we count 1 for how many collisions this vehicle_type participates. Finally we group the cases based on the ve_type and calculate the total number of cases of each group, order them by the count and fetch the top 5 type.

SQL statement

```
SELECT VE_TYPE, COUNT(*) AS COUNT
FROM
   (SELECT DISTINCT VEHICLE.VE_TYPE, PARTY_INVOLVE.CASE_ID
   FROM PARTY_INVOLVE, VEHICLE, ROAD_EN
   WHERE ROAD_EN.ROAD_CON LIKE '%holes%'
   AND VE_TYPE IS NOT NULL
   AND ROAD_EN.CASE_ID = PARTY_INVOLVE.CASE_ID
   AND VEHICLE.VE_NUM = PARTY_INVOLVE.VE_NUM)
GROUP BY VE_TYPE
ORDER BY COUNT DESC
FETCH FIRST 5 ROWS ONLY
```

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

VE_TYPE	COUNT
passenger car	6940
pickup or panel truck	2017
motorcycle or scooter	432
bicycle	417
truck or truck tractor with trailer	351

Query c:

Description of logic:

We first select the victims whose deg_injury is like “severe injury” or “killed”, and then join it with the table party_involve on the party_id and then table vehicle (select those whose ve_make is not null) on ve_num to get [vic_id|ve_make]. We group the table by ve_make and count the number of victims of each ve_make. Finally we order the table by the count and fetch the top-10 vehicle makes and their number of victims.

SQL statement

SELECT VE_MAKE, COUNT(ASSOCIATE_VICTIM.VIC_ID) AS COUNT

FROM VEHICLE, PARTY_INVOLVE, ASSOCIATE_VICTIM

WHERE VEHICLE.VE_NUM = PARTY_INVOLVE.VE_NUM

AND ASSOCIATE_VICTIM.PARTY_ID = PARTY_INVOLVE.PARTY_ID

AND (ASSOCIATE_VICTIM.DEG_INJURY LIKE '%severe injury%' OR ASSOCIATE_VICTIM.DEG_INJURY LIKE '%killed%')

AND VEHICLE.VE_MAKE IS NOT NULL

GROUP BY VE_MAKE

ORDER BY COUNT DESC

FETCH FIRST 10 ROWS ONLY

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

VE_MAKE	COUNT
FORD	13924
HONDA	12060
TOYOTA	10639
CHEVROLET	10418
NISSAN	3860
DODGE	3641

NOT STATED	3603
HARLEY-DAVIDSON	3410
SUZUKI	2482
YAMAHA	2105

Query d:

Description of logic:

We assume that the fraction is calculated by: (# victims with no injury and of one seat place) / (# victims of this seat place). We don't consider the number of collisions because if there are 2 victims seating in same/different position in one collision, it is ambiguous to count for 1 or 2 collisions, and the collision can be of several seating positions.

We partition the table by vic_seat, and add a column for count of victims of one kind of vic_seat. Then we select the tuples whose deg_injury is like "no injury", and calculate the sum of 1/count of each vic_seat group. The denominator is the total count and the nominator is the count of the no injured, so we get the safety_index of each vic_seat. To get the max and min safety_index and their corresponding seat, we union two tables, one is for the min and one is for the max.

SQL statement

```
(SELECT vic_seat, ROUND(SUM(1.0/denom),2) AS SAFETY_INDEX

FROM

   (SELECT VIC_SEAT, DEG_INJURY, COUNT(DEG_INJURY) OVER (PARTITION BY VIC_SEAT)
AS DENOM

   FROM ASSOCIATE_VICTIM)

WHERE DEG_INJURY LIKE '%no injury%'

GROUP BY VIC_SEAT

ORDER BY SAFETY_INDEX DESC

FETCH FIRST 1 ROWS ONLY)
```

```
UNION ALL

(SELECT vic_seat, ROUND(SUM(1.0/denom),2) AS SAFETY_INDEX
FROM
   (SELECT VIC_SEAT, DEG_INJURY, COUNT(DEG_INJURY) OVER (PARTITION BY VIC_SEAT) AS DENOM
   FROM ASSOCIATE_VICTIM)
WHERE DEG_INJURY LIKE '%no injury%'
GROUP BY VIC_SEAT
ORDER BY SAFETY_INDEX
FETCH FIRST 1 ROWS ONLY)
```

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

VIC_SEAT	SAFETY__INDEX
5	0.839
1	0.009

Query e:

Description of logic:

We assume that the query is to find ve_types that participate in at least 10 collisions of EACH city and in at least half of the cities. We join the vehicle and party to know what type of the vehicle every party is driving, and then join it with table case and location to know in which city the collisions happen. There may be two same ve_type in the same collision, and we count 1 for the #collision of this ve_type, so for each [ve_type, county_city], we calculate the total number of collisions as

CASE_NUM. After selecting [ve_type, county_city] tuples where the CASE_NUM >= 10, we group the table by ve_type and select the number of those whose count(county_city) >= total number of cities / 2

SQL statement

```
SELECT COUNT(VE_TYPE) AS NUM_VE_TYPE
FROM
   (SELECT VE_TYPE, COUNT(COUNTY_CITY) AS CITY_NUM
   FROM
      (SELECT DISTINCT VE_TYPE, COUNTY_CITY, COUNT(*) AS CASE_NUM
      FROM VEHICLE, PARTY_INVOLVE, CASE, LOCATION
      WHERE VEHICLE.VE_NUM=PARTY_INVOLVE.VE_NUM
         AND PARTY_INVOLVE.CASE_ID=CASE.CASE_ID
         AND LOCATION.LOC_NUM=CASE.LOC_NUM
         GROUP BY VE_TYPE, COUNTY_CITY)CITY
   WHERE CASE_NUM >=10
   GROUP BY VE_TYPE) COUNT_CITY,
   (SELECT COUNT(DISTINCT COUNTY_CITY) AS TOTAL_CITY FROM LOCATION) TOTAL
WHERE CITY_NUM>= TOTAL_CITY/2
```

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

NUM_VE_TYPE
13

Query f:

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 of logic:

To find the most populated cities, we use the order by function defined by ourselves to order the population level of cities descendingly, and fetch the first 3 distinct county_city and the corresponding population. In our model, each case has a loc_num, but several location entries can have the same county_city, so we need to join the 3-row table with the location and the case table to get [county_city|population|case_id], and then join it with party_involve and associate_victim table to get [county_city|population|case_id|vic_d|vic_age]. We group the table over case_id and add a column of average victim age of each case. Then we partition the table over county_city, and add a row_number() column in the order of victim age, and select the youngest 10.

SQL statement

```
SELECT COUNTY_CITY, POPULATION, CASE_ID, AVG_AGE
FROM
   (SELECT COUNTY_CITY, POPULATION, CASE_ID, AVG_AGE, ROW_NUMBER()
OVER(PARTITION BY COUNTY_CITY ORDER BY AVG_AGE ASC) AS ROW_NUMBER
   FROM
      (SELECT COL_CITY.COUNTY_CITY, COL_CITY.POPULATION, CASE.CASE_ID,
AVG(VIC_AGE) AS AVG_AGE
      FROM
         (SELECT DISTINCT COUNTY_CITY, POPULATION
            FROM LOCATION
            order by (case population
               when 7 then 0
               when 6 then 1
               when 5 then 2
               when 4 then 3
               when 3 then 4
               when 2 then 5
               when 1 then 6
               when 9 then 7
               when 0 then 8
               else 9 end)
            FETCH FIRST 3 ROWS ONLY) COL_CITY,
            LOCATION, CASE, PARTY_INVOLVE, ASSOCIATE_VICTIM
      WHERE COL_CITY.county_city=LOCATION.county_city
         AND CASE.LOC_NUM=LOCATION.LOC_NUM
         AND PARTY_INVOLVE.CASE_ID=CASE.CASE_ID
         AND ASSOCIATE_VICTIM.PARTY_ID=PARTY_INVOLVE.PARTY_ID
         AND VIC_AGE IS NOT NULL
         GROUP BY COL_CITY.COUNTY_CITY, COL_CITY.POPULATION,
CASE.CASE_ID)AGE)RANK_AGE
WHERE ROW_NUMBER<=10
ORDER BY COUNTY_CITY, ROW_NUMBER
```

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

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

COUNTY_CITY	POPULATION	CASE_ID	AVG_AGE
109	7	1050440	0
109	7	2858865	0
109	7	2489862	0
109	7	1572294	0
109	7	739484	0
109	7	491424	0
109	7	3553733	0
109	7	3486455	0
109	7	3012670	0
109	7	2856537	0
3801	7	984042	0
3801	7	2945655	0
3801	7	2613850	0
3801	7	1686308	0
3801	7	1783897	0
3801	7	353302	0
3801	7	2903949	0
3801	7	1895167	0
3801	7	1490755	0
3801	7	994429	0
4313	7	1702989	0
4313	7	2530732	0
4313	7	2427381	0
4313	7	2034278	0
4313	7	1585055	0
4313	7	3391590	0
4313	7	2034270	0
4313	7	1506715	0
4313	7	1322740	0
4313	7	1239869	0

Query g:

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 of logic:

We first select case with col_type like “pedestrian” to get the “PE_CASE” table, then we join PE_CASE with party_involve and associate_victim table to get the corresponding victims of the case. We select the tuples that satisfies the min(vic_age) > 100 and find the eldest victim age of each collision.

SQL statement

```
SELECT DISTINCT CASE_ID, MAX(VIC_AGE) OVER(PARTITION BY CASE_ID) AS ELDEST

FROM

   (SELECT PE_CASE.CASE_ID, VIC_AGE, MIN(VIC_AGE) OVER (PARTITION BY
PE_CASE.CASE_ID)AS MIN_AGE

   FROM

      (SELECT CASE_ID

      FROM CASE

      WHERE COL_TYPE LIKE '%pedestrian%') PE_CASE, PARTY_INVOLVE, ASSOCIATE_VICTIM

   WHERE PARTY_INVOLVE.CASE_ID=PE_CASE.CASE_ID

      AND ASSOCIATE_VICTIM.PARTY_ID=PARTY_INVOLVE.PARTY_ID)CASE_AGE

WHERE MIN_AGE>100

ORDER BY CASE_ID
```

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

CASE_ID	ELDEST
36446	110
69198	101

439197	102
445265	101
566220	102
644226	103
784061	102
817210	102
820619	101
828116	102
851026	106
868472	103
1209166	101
1213340	121
1347636	101
1373664	101
1548445	102

1847678	104
2472739	103
2531557	103

Query h:

Description of logic:

We first group the party_involve table by the ve_num to get [ve_num | N_COLLISION] table VE_COL, then we join the VE_COL with the vehicle table on the ve_num to get how many collisions a kind of vehicle has participated in. We select those who has participated in more than 9 collisions. Because we assign each vehicle with same [ve_type, ve_make, ve_year] a unique ve_num, so we finally use [ve_type, ve_make, ve_year] to denote the vehicle id.

SQL statement

```
WITH

VE_COL AS

   (

   SELECT VE_NUM, COUNT(case_id) AS N_COLLISION

   FROM PARTY_INVOLVE

   GROUP BY VE_NUM

   )

SELECT VEHICLE.VE_TYPE || ', ' || VEHICLE.VE_MAKE || ', ' || VEHICLE.VE_YEAR AS VE_ID,
VE_COL.N_COLLISION AS N_COLLISION

FROM (VEHICLE INNER JOIN VE_COL ON VE_COL.VE_NUM = VEHICLE.VE_NUM)

WHERE N_COLLISION > 9 AND VEHICLE.VE_MAKE IS NOT NULL AND VEHICLE.VE_TYPE IS
NOT NULL AND VEHICLE.VE_TYPE != 'pedestrain'

ORDER BY N_COLLISION DESC
```

FETCH FIRST 20 ROWS ONLY

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

VE_ID	N_COLLISION
passenger car, TOYOTA, 2000	52469
passenger car, FORD, 2000	51915
passenger car, HONDA, 2000	50255
passenger car, FORD, 1998	49148
passenger car, TOYOTA, 2001	47205
passenger car, HONDA, 2001	45243
passenger car, FORD, 2001	45203
passenger car, TOYOTA, 1999	42907
passenger car, HONDA, 1998	42063
passenger car, FORD, 1999	41918
passenger car, FORD, 1995	40225
passenger car, HONDA, 1997	39182
passenger car, FORD, 1997	38852
passenger car, HONDA, 1999	38528
passenger car, TOYOTA, 2002	38403
passenger car, TOYOTA, 1998	37982
passenger car, TOYOTA, 1997	37134
passenger car, TOYOTA, 2003	35919
passenger car, HONDA, 2002	35759
passenger car, FORD, 2002	35433

Observation:

The vehicles participated in most collisions are all passenger cars. Besides, these vehicles are all made by large car manufactures such as Totota, Ford and Honda.

Query i:

Description of logic:

Because we assign each kind of location a loc_num but several loc_num have the same county_city, so we first join location and case, then group the table by "county_city" to count how many collisions happen in the same county_city (if it is not null). Then we order them by number of collisions and fetch the top-10 county_city.

SQL statement

SELECT COUNTY_CITY, COUNT(CASE_ID) as N_COLLISION FROM

LOCATION, CASE

WHERE LOCATION.LOC_NUM = CASE.LOC_NUM and LOCATION.COUNTY_CITY IS NOT NULL

GROUP BY COUNTY_CITY

ORDER BY N_COLLISION DESC

FETCH FIRST 10 ROWS ONLY

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

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

Query j:

Description of logic:

We focus on the "group by" function. First we group by lighting of the case, there are "DAY", "NIGHT" and others, in the dataset, lighting conditions of "dawn and dusk" are ambiguous, so we just distinguish them by the col_date and col_time, neglecting the value. If the lighting is not "DAY" or

"NIGHT", we group them by the date and time, based on different seasons. After grouping, we count the number of collisions of each lighting condition, and order them by the count.

SQL statement

```
WITH

COL_PERIOD AS

   (

   SELECT

   CASE

   WHEN CASE.LIGHTING LIKE '%day%' THEN 'DAY'

   WHEN CASE.LIGHTING LIKE '%dark%' THEN 'NIGHT'

   ELSE

         (CASE

            WHEN EXTRACT(MONTH FROM COL_TIME)>8 OR EXTRACT(MONTH FROM
COL_TIME)<4 THEN

               (CASE

               WHEN EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))>5 AND
EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))<8 THEN 'DAWN'

               WHEN EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))>17 AND
EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))<20 THEN 'DUSK'

               WHEN EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))>7 AND
EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))<18 THEN 'DAY'

               ELSE 'NIGHT'

               END)

            WHEN EXTRACT(MONTH FROM COL_TIME)>3 OR EXTRACT(MONTH FROM
COL_TIME)<9 THEN

               (CASE
```

```
                WHEN EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))>3 AND
EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))<6 THEN 'DAWN'

                WHEN EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))>19 AND
EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))<22 THEN 'DUSK'

                WHEN EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))>5 AND
EXTRACT(HOUR FROM CAST(COL_TIME AS TIMESTAMP))<20 THEN 'DAY'

                ELSE 'NIGHT'

                END)

          ELSE NULL

          END)

   END AS PERIOD

   FROM CASE

   )

SELECT PERIOD, COUNT(*) AS N_COLLISION

FROM COL_PERIOD

WHERE PERIOD IS NOT NULL

GROUP BY PERIOD

ORDER BY N_COLLISION DESC
```

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

PERIOD	N_COLLISION
DAY	2575153
NIGHT	1041614
DAWN	30317
DUSK	28092

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

Query Performance Analysis – Indexing

Query 2:

<Initial Running time/IO: 3371ms/37073

Optimized Running time/IO:1511ms/24804

Explain the improvement:

Added the following index:

create index ve_case_ix on party_involve(case_id, ve_num)

Necessities:

From the initial plan, we can find there are three tables joining: ROAD_EN, VEHICLE, PARTY_INVOLVE and each of them involves a full file scan. We only need [case_id, ve_num] of the party_involve table, the whole content of the road_en table, [ve_num, ve_type] of the vehicle table. Since road_con and case_id are both primary keys of the road_en table, there is no need to add indexes in the road_en tables. So we build indexes on [case_id, ve_num] of the party_involve table to reduce full scan and optimize.

For this query, we added a joint index 've_case_ix' on <case_id, ve_num> on entity 'party_involve'. In this case, when doing hash join, the system can do a fast full scan on 'party_involve' instead of a full scan.

In our case, the index 've_case_ix' is relevant when we want to group the collisions according to the type of vehicles. So a fast full scan using a B-tree index is useful.

Influence on the cost:

After applying the two indexes, we could find that the total cost is reduced from 37073 to 24804. The cost of access of party_involve is reduced from 19124 to 6855, and thus the hash join of party_involve and road_en is reduced, and then the hash join of vehicle and the party_involve is also reduced due to the less cost of scanning the table party_involve. Now the cost of scanning the party_involve is still a big part of the total cost, but the cost of right semi join and the unique operation becomes the dominant part.

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/

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5	37073
SORT		ORDER BY	5	37073
VIEW	SYS.null		5	37072
Filter Predicates				
from$_subquery$_005.rowlimit_$$_rownumber<=5				
WINDOW		SORT PUSHED RANK	15	37072
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=5				
HASH		GROUP BY	15	37072
VIEW			301205	35412
HASH		UNIQUE	301205	35412
HASH JOIN			301205	31478
Access Predicates				
VEHICLE.VE_NUM=PARTY_INVOLVE.VE_NUM				
TABLE ACCESS	VEHICLE	FULL	21932	68
Filter Predicates				
VE_TYPE IS NOT NULL				
HASH JOIN		RIGHT SEMI	362485	31409
Access Predicates				
ROAD_EN.CASE_ID=PARTY_INVOLVE.CASE_ID				
TABLE ACCESS	ROAD_EN	FULL	182607	3040
Filter Predicates				
ROAD_EN.ROAD_CON LIKE '%holes%'				
TABLE ACCESS	PARTY_INVOLVE	FULL	7286606	19124

Improved plan>

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5	24804
SORT		ORDER BY	5	24804
VIEW	SYS.null		5	24803
Filter Predicates				
from$_subquery$_005.rowlimit_$$_rownumber<=5				
WINDOW		SORT PUSHED RANK	15	24803
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=5				
HASH		GROUP BY	15	24803
VIEW			301205	23143
HASH		UNIQUE	301205	23143
HASH JOIN			301205	19209
Access Predicates				
VEHICLE.VE_NUM=PARTY_INVOLVE.VE_NUM				
TABLE ACCESS	VEHICLE	FULL	21932	68
Filter Predicates				
VE_TYPE IS NOT NULL				
HASH JOIN		RIGHT SEMI	362485	19140
Access Predicates				
ROAD_EN.CASE_ID=PARTY_INVOLVE.CASE_ID				
TABLE ACCESS	ROAD_EN	FULL	182607	3040
Filter Predicates				
ROAD_EN.ROAD_CON LIKE '%holes%'				
INDEX	VE_CASE_IX	FAST FULL SCAN	7286606	6855

Query 5:

<Initial Running time/IO: 21.494s/48986

Optimized Running time/IO: 14.703s /24096

Explain the improvement:

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/

Added the following index:

create index ve_case_ix on party_involve(case_id, ve_num)

create index case_loc_ix on case(case_id, loc_num)

create index county_city_ix on location(loc_num, county_city)

Necessities:

From the initial plan, we can find there are four tables joining: PARTY_INVOLVE, VEHICLE, CASE, LOCATION and each of them involves a full file scan. We only need [case_id, ve_num] of the party_involve table, [ve_num, ve_type] of the vehicle table, [case_id, loc_num] of the case table and [loc_num, county_city] of the location table. So we build indexes on [case_id, ve_num] of the party_involve table, indexes on [case_id, loc_num] of the case table, and indexes on [loc,num, county_city] of the location table to reduce full scan and optimize.

For this query, we added a joint index 've_case_ix' on <case_id, ve_num> on entity 'party_involve', and 'case_loc_ix' on <case_id, loc_num> on entity 'case'. In this case, when doing hash join, the system can do a fast full scan on 'party_involve' and 'case' instead of a full scan.

The index 've_case_ix' is relevant when we want to group the collisions according to the type of vehicles, and the index 'case_loc_ix' is used for the grouping of collisions according to the county_city code. In this case it is indeed useful to index the 'case_ id' according to 've_num' and 'loc_num' as a B-tree. It could indeed search the desired result binarily.

We can see that the index 've_case_ix' has been used twice for a fast full scan for the matching of vehicle and case, which reduces the loss greatly, and improves the quality of the query.

Distribution of cost:

Now for the optimized plan, the majority cost is spent on doing group by based on [ve_type, county_city] for 'vehicle' entity and the join of case and party_involve table. After applying the three indexes, we could find that the total cost is reduced from 48986 to 24096. The cost of access of party_involve is reduced from 19124 to 6855, and cost of accessing the case table is reduced from 16023 to 3406, then the cost of the join of them is also reduced. Due to the case_loc_ix index, the join of the location and the case is also reduced. The county_city_ix index plays a small role in the optimization.

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/

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	48986
SORT		AGGREGATE	1	
NESTED LOOPS			1	48986
VIEW			1	8
SORT		AGGREGATE	1	
VIEW	SYS.VM_NWVW_1		540	8
HASH		GROUP BY	540	8
TABLE ACCESS	LOCATION	FULL	4362	7
VIEW			1	48978
Filter Predicates				
CITY_NUM>=TOTAL_CITY/2				
SORT		GROUP BY	15	48978
VIEW			287	48978
FILTER				
Filter Predicates				
COUNT(*)>=10				
SORT		GROUP BY	287	48978
HASH JOIN			7251269	48787
Access Predicates				
LOCATION.LOC_NUM=CASE.LOC_NUM				
TABLE ACCESS	LOCATION	FULL	4362	7
HASH JOIN			7251269	48761
Access Predicates				
VEHICLE.VE_NUM=PARTY_INVOLVE.VE_NUM				
TABLE ACCE	VEHICLE	FULL	26266	68
HASH JOIN			7286606	48674
Access Predicates				
PARTY_INVOLVE.CASE_ID=CASE.CASE_ID				
TABLE A	CASE	FULL	3678063	16023
TABLE A	PARTY_INVOLVE	FULL	7286606	19124

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/

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	24096
SORT		AGGREGATE	1	
NESTED LOOPS			1	24096
VIEW			1	6
SORT		AGGREGATE	1	
VIEW	SYS.VM_NWVW_1		540	6
HASH		GROUP BY	540	6
INDEX	COUNTY_CITY_IX	FAST FULL SCAN	4362	5
VIEW			1	24090
Filter Predicates				
CITY_NUM>=TOTAL_CITY/2				
SORT		GROUP BY	15	24090
VIEW			287	24090
FILTER				
Filter Predicates				
COUNT(*)>=10				
SORT		GROUP BY	287	24090
HASH JOIN			7251269	23898
Access Predicates				
LOCATION.LOC_NUM=CASE.LOC_NUM				
INDEX	COUNTY_CITY_IX	FAST FULL SCAN	4362	5
HASH JOIN			7251269	23875
Access Predicates				
VEHICLE.VE_NUM=PARTY_INVOLVE.VE_NUM				
TABLE ACCE	VEHICLE	FULL	26266	68
HASH JOIN			7286606	23788
Access Predicates				
PARTY_INVOLVE.CASE_ID=CASE.CASE_ID				
NESTED			7286606	23788
STA				
	CASE_LOC_IX	FAST FULL SCAN	3678063	3406
IND	VE_CASE_IX	RANGE SCAN	2	6855
Access Predicates				
PARTY_INVOLVE.CASE_ID=CASE.CASE_ID				
INDEX	VE_CASE_IX	FAST FULL SCAN	7286606	6855

Query 6:

<Initial Running time/IO: 5 measurements shows initial average run time is 3641.6 ms / 47735

Optimized Running time/IO: 5 measurements shows optimized running time is 2980 ms / 37102

Explain the improvement:

Added the following index:

create index loc_in_case on case(loc_num)

Necessities: From the initial plan, we can find there are five tables joining: COL_CITY LOCATION, CASE, PARTY_INVOLVE, ASSOCIATE_VICTIM and each of them involves a full file scan. We only need [case_id, loc_num] of the case table, [[party_id, case_id] of the party_involve table, [party_id, vic_age] of the associate_victim table, [loc_num, county_city] of the location table, and we have selection predicates on these fields. So we try to build indexes on these columns to reduce full scan and optimize.

We tried several combinations of the indexes of these columns, but finally we found this one is useful in the timing, though other keys like location(county_city) (for selection during the join), party_involve(case_id, loc_num) (for utilizing the clustered index) will also reduce the cost and change the plan by using index.

With the loc_in_case of the case table, the hash join of table case and location is completed by index, and it also utilizes the clustered case_id row number as index, which is very nice . While joining and accessing the needed columns, they can just use the index range scan to check if the predicates are satisfied, thus the IO is reduced.

Influence on the cost: Cost is related to the CPU and IO. From the optimized plan, we can find that the total cost is reduced from 47735 to 37104, and the cost of case table access is reduced from 16023 to 225, and the cost of location access is reduced from 7 to 2, and the total cost of their join is reduced from 16048 to 5417 (this is why the total cost is reduced). There are also two hash joins becoming nested join, both because of the index scan. Maybe the optimizer chooses finds the indexed nested loop more efficient.

However, the cost of party_involve full scan is still large and plays an important role in the total cost. We can reduce its cost by creating an index on party_involve(party_id, case_id) and it indeed helps to reduce the cost, but because the running time suffers, we did not use it. From the plan we can also find that the group cost and join cost if dominated by the file access cost.

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/

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			21893	47735
SORT		ORDER BY	21893	47735
VIEW			21893	47376
Filter Predicates				
ROW_NUMBER<=10				
WINDOW		SORT PUSHED RANK	21893	47376
Filter Predicates				
ROW_NUMBER() OVER (PARTITION BY from$_subquery$_006.COUNTY_CITY ORDER BY SUM(VIC_AGE)/COUNT(VIC_AGE))<=10				
HASH		GROUP BY	21893	47376
HASH JOIN			21893	46673
Access Predicates				
ASSOCIATE_VICTIM.PARTY_ID=PARTY_INVOLVE.PARTY_ID				
HASH JOIN			40481	35147
Access Predicates				
PARTY_INVOLVE.CASE_ID=CASE.CASE_ID				
HASH JOIN			20434	16048
Access Predicates				
CASE.LOC_NUM=LOCATION.LOC_NUM				
HASH JOIN			24	16
Access Predicates				
from$_subquery$_006.COUNTY_CITY=LOCATION.COUNTY_CITY				
VIEW	SYS.null		3	9
Filter Predicates				
from$_subquery$_006.rowlimit_$$_rownumber<=3				
WINDOW		SORT PUSHED RANK	3055	9
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY from$_subquery$_005.rowlimit_$_0)<=3				
VIEW	SYS.null		3055	8
HASH		UNIQUE	3055	8
TABLE ACCESS	LOCATION	FULL	4362	7
TABLE ACCESS	LOCATION	FULL	4362	7
TABLE ACCESS	CASE	FULL	3678063	16023
TABLE ACCESS	CASE	FULL	3678063	16023
TABLE ACCESS	PARTY_INVOLVE	FULL	7286606	19080
TABLE ACCESS	ASSOCIATE_VICTIM	FULL	3940663	7460
Filter Predicates				
VIC_AGE IS NOT NULL				

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/

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			21893	37104
SORT		ORDER BY	21893	37104
VIEW			21893	36745
Filter Predicates				
ROW_NUMBER<=10				
WINDOW		SORT PUSHED RANK	21893	36745
Filter Predicates				
ROW_NUMBER() OVER (PARTITION BY from$_subquery$_006.COUNTY_CITY ORDER BY SUM(VIC_AGE)/COUNT(VIC_AGE))<=10				
HASH		GROUP BY	21893	36745
HASH JOIN			21893	36042
Access Predicates				
ASSOCIATE_VICTIM.PARTY_ID=PARTY_INVOLVE.PARTY_ID				
HASH JOIN			40481	24516
Access Predicates				
PARTY_INVOLVE.CASE_ID=CASE.CASE_ID				
NESTED LOOPS			20434	5417
NESTED LOOPS			20434	5417
HASH JOIN			24	16
Access Predicates				
from$_subquery$_006.COUNTY_CITY=LOCATION.COUNTY_CITY				
VIEW	SYS.null		3	9
Filter Predicates				
from$_subquery$_006.rowlimit_$$_rownumber<=3				
WINDOW		SORT PUSHED RANK	3055	9
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY from$_subquery$_005.rowlimit_$_0)<=3				
VIEW	SYS.null		3055	8
HASH		UNIQUE	3055	8
TABLE ACCESS	LOCATION	FULL	4362	7
TABLE ACCESS	LOCATION	FULL	4362	7
INDEX	LOC_IN_CASE	RANGE SCAN	843	2
Access Predicates				
Access Predicates				
CASE.LOC_NUM=LOCATION.LOC_NUM				
TABLE ACCESS	CASE	BY INDEX ROWID	843	225
TABLE ACCESS	PARTY_INVOLVE	FULL	7286606	19080
TABLE ACCESS	ASSOCIATE_VICTIM	FULL	3940663	7460
Filter Predicates				
VIC_AGE IS NOT NULL				

Query 8:

<Initial Running time/IO: 2.478s / 19386

Optimized Running time/IO: 1.492s / 5998

Necessity of indexing:

This query requires us to a group by operation on the ‘ve_num’ attribute on ‘party_involve’ entity. However, there are no indexes for the ‘ve_num’ attribute for ‘party_involve’, and group by in this case can consume much cost. Therefore, it is necessary to add an index to ‘ve_num’ for ‘party_involve’.

Explain the improvement:

Added the following index:

create index ve_party_ix on party_involve(party_id, ve_num)

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/

For this query, we added a joint index 've_party_ix' on <party_id, ve_num> on entity 'party_involve'. In this case, when doing group by, the system can use the 've_party_ix', which reduces the cost for group by from 19317 to 5929.

Distribution of cost:

Now for the optimized plan, the majority cost is spent on doing group by based on 've_num' for 'party_involve' entity. Little cost is spent on accessing the 'vehicle' entity (because it can access by its primary key, the index).

Initial plan

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			20	19386
VIEW	SYS.null		20	19386
Filter Predicates				
from$_subquery$_005.rowlimit_$$_rownumber<=20				
WINDOW		SORT PUSHED RANK	1320	19386
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY INTERNAL_FUNCTION(VE_COL.N_COLLISION) DESC)<=20				
HASH JOIN			1320	19385
Access Predicates				
VE_COL.VE_NUM=VEHICLE.VE_NUM				
NESTED LOOPS			1320	19385
NESTED LOOPS				
STATISTICS COLLECT				
VIEW			1320	19317
HASH		GROUP BY	1320	19317
Filter Predicates				
COUNT(*)>9				
TABLE ACCE	PARTY_INVOLVE	FULL	7286606	19124
INDEX	SYS_C00224265	UNIQUE SCAN		
Access Predicates				
VE_COL.VE_NUM=VEHICLE.VE_NUM				
TABLE ACCESS	VEHICLE	BY INDEX ROWID	1	68
Filter Predicates				
AND				
VEHICLE.VE_TYPE IS NOT NULL				
VEHICLE.VE_MAKE IS NOT NULL				
VEHICLE.VE_TYPE<>'pedestrain'				
TABLE ACCESS	VEHICLE	FULL	17752	68
Filter Predicates				
AND				
VEHICLE.VE_TYPE IS NOT NULL				
VEHICLE.VE_MAKE IS NOT NULL				
VEHICLE.VE_TYPE<>'pedestrain'				

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/

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			20	5998
VIEW	SYS.null		20	5998
Filter Predicates				
from$_subquery$_005.rowlimit_$$_rownumber<=20				
WINDOW		SORT PUSHED RANK	1320	5998
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY INTERNAL_FUNCTION(VE_COL.N_COLLISION) DESC)<=20				
HASH JOIN			1320	5997
Access Predicates				
VE_COL.VE_NUM=VEHICLE.VE_NUM				
NESTED LOOPS			1320	5997
NESTED LOOPS				
STATISTICS COLLECT				
VIEW			1320	5929
HASH		GROUP BY	1320	5929
Filter Predicates				
COUNT(*)>9				
INDEX	VE_PARTY_IX	FAST FULL SCAN	7286606	5736
INDEX	SYS_C00224265	UNIQUE SCAN		
Access Predicates				
VE_COL.VE_NUM=VEHICLE.VE_NUM				
TABLE ACCESS	VEHICLE	BY INDEX ROWID	1	68
Filter Predicates				
AND				
VEHICLE.VE_TYPE IS NOT NULL				
VEHICLE.VE_MAKE IS NOT NULL				
VEHICLE.VE_TYPE<>'pedestrain'				
TABLE ACCESS	VEHICLE	FULL	17752	68
Filter Predicates				
AND				
VEHICLE.VE_TYPE IS NOT NULL				
VEHICLE.VE_MAKE IS NOT NULL				
VEHICLE.VE_TYPE<>'pedestrain'				

Query 9:

<Initial Running time/IO: 4.343s / 16227

Optimized Running time/IO: 1.457s / 2260

Necessity of indexes:

Here in the query, we require the system to select from two different entities, given an equality condition ‘loc_num’. Hence, the system has to do a join on the two entities. However, it only has a primary key ‘loc_num’ for ‘location’ entity, and has no indexes for the ‘case’ entity. In this case, adding an index to ‘loc_num’ for ‘case’ entity is necessary to reduce the costs for the join.

Explain the improvement:

Added the following indexes:

create index loc_in_case on case(loc_num)

For this query, we added an index ‘loc_in_case’ on ‘loc_num’ for ‘case’ entity.

We can see from the query plan that the index ‘loc_in_case’ has been used twice.

Based on these indexes, when the system is doing hash join for ‘location’ entity and ‘case’ entity on ‘location.loc_num = case.loc_num’ (to find the name of the cities respective to the location numbers), it can do a fast full scan on ‘case’ entity, instead of a full scan, which greatly reduces the cost from 16023 to 2056.

Distribution of cost:

From the optimized query plan, we can see very little cost is spent on scanning the table ‘location’, the majority cost is spent on doing range scan on ‘case’ tabel, fulfilling the equality constraint. Despite this, it still saves much cost compared to the original plan.

Initial plan

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			10	16227
SORT		ORDER BY	10	16227
VIEW	SYS.null		10	16226
Filter Predicates				
from$_subquery$_003.rowlimit_$$_rownumber<=10				
WINDOW		SORT PUSHED RANK	540	16226
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=10				
HASH		GROUP BY	540	16226
HASH JOIN			3678063	16039
Access Predicates				
LOCATION.LOC_NUM=CASE.LOC_NUM				
TABLE ACCESS	LOCATION	FULL	4362	7
Filter Predicates				
LOCATION.COUNTY_CITY IS NOT NULL				
TABLE ACCESS	CASE	FULL	3678063	16023

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/

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			10	2258
SORT		ORDER BY	10	2258
VIEW	SYS.null		10	2257
Filter Predicates				
from$_subquery$_003.rowlimit_$$_rownumber<=10				
WINDOW		SORT PUSHED RANK	540	2257
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=10				
HASH		GROUP BY	540	2257
HASH JOIN			3678063	2070
Access Predicates				
LOCATION.LOC_NUM=CASE.LOC_NUM				
NESTED LOOPS			3678063	2070
STATISTICS COLL				
INDEX	COUNTY_CITY_IX	FAST FULL SCAN	4362	5
Filter Predicates				
LOCATION.COUNTY_CITY IS NOT NULL				
INDEX	LOC_IN_CASE	RANGE SCAN	843	2056
Access Predicates				
LOCATION.LOC_NUM=CASE.LOC_NUM				
INDEX	LOC_IN_CASE	FAST FULL SCAN	3678063	2056

General Comments

In this project, we finished the design of our database system, from designing the ER diagram, preparing data to implement the queries, and optimizing the execution. We successfully compelted the tasks requried in the project, a very comprehensive and practical project.
Many thanks for the help from the professors and TAs!