Databases Project – Spring 2021

Team No: G34

Members: Siran Li, Ke Wang, Ningwei Ma

Contents

[Deliverable 1 2](#__RefHeading___Toc806_482092919)

[Assumptions 2](#__RefHeading___Toc808_482092919)

[Entity Relationship Schema 3](#__RefHeading___Toc810_482092919)

[Schema 3](#__RefHeading___Toc812_482092919)

[Description 3](#__RefHeading___Toc814_482092919)

[Relational Schema 4](#__RefHeading___Toc816_482092919)

[ER schema to Relational schema 4](#__RefHeading___Toc818_482092919)

[DDL 4](#__RefHeading___Toc820_482092919)

[General Comments 4](#__RefHeading___Toc822_482092919)

[Deliverable 2 5](#__RefHeading___Toc824_482092919)

[Entity Relationship Schema (updated) 5](#__RefHeading___Toc826_482092919)

[Assumptions 6](#__RefHeading___Toc828_482092919)

[Relational Schema 7](#__RefHeading___Toc830_482092919)

[DDL (UPDATED) 7](#__RefHeading___Toc832_482092919)

[Data Loading/Cleaning 11](#__RefHeading___Toc834_482092919)

[Query Implementation 11](#__RefHeading___Toc836_482092919)

[General Comments 17](#__RefHeading___Toc838_482092919)

# Deliverable 1

## Assumptions

Each collision case has its unique case\_id.

Each case has a primary collision factor (pcf); Different cases could have the same pcf.

Each case has several parties.

Each case collides in exactly one location; Different cases could collide in same location.

Each collision happened under exactly one condition, among which there could be several weather and road conditions; Different cases could collide under same condition.

Each party has a unique party\_id.

Each party is involved in exactly one case; A case could involve several parties.

Each party may take a vehicle; Vehicles with same attributes are recognized as the same vehicle, and under this condition different parties can take the same vehicle.

“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.

Each victim has a unique vic\_id.

Each victim is associated with exactly a party in a case, by “party\_number + case\_id”. A party could be associated with 0 or several victims.

Each party of the case may have some other factors for the collision. Different parties may have same other factors.

Each party and victim may have their different safety\_equipment. Different parties and victims may have same safety\_equipment.

## Entity Relationship Schema

### Schema

### Description

<Describe all the choices you made for Entities and Relationships>

## Relational Schema

### ER schema to Relational schema

<Describe the transition from ER schema to Relational schema>

### DDL

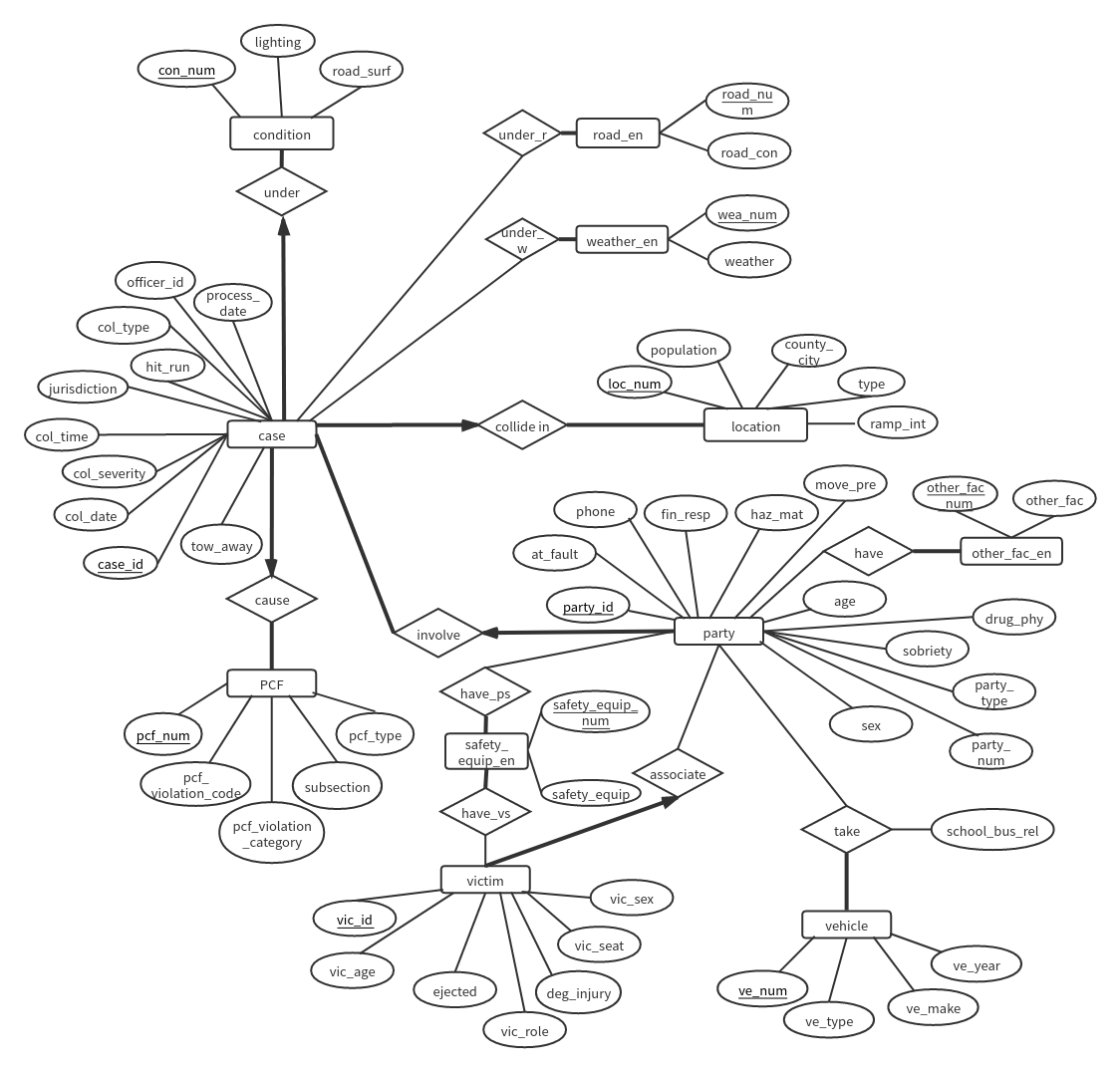
<Provide the DDL>

## General Comments

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

# Deliverable 2

## Entity Relationship Schema (updated)



## Assumptions

Based on the feedback on deliverable 1, we add these assumptions and explanations, along with some modifications.

1. We found that there are duplicates of case\_id in the collision2018.csv (6 of them), and since their total number is small, we recognize them as incorrectly registered data and removed them. After removing the duplicated terms in case, we have verified that in the parties2018.csv, each “party\_num + case\_id” corresponds to unique “party\_id”. So we consider party as the weak entity of case. In the victims2018.csv, each victim is associated with the specific party by unique “party\_num + case\_id”, and we also consider victim as the weak entity of party. Therefore, we merge the “victim-associated\_with-party” and “party-involved\_in-case” relationships into the tables of victim and party entity respectively to illustrate the weak entity constraints. Meanwhile, we added “ON DELETE CASCADE” to the DDL (based on DDL of deliverable 1) of victim and party entity to implement weak entity relationship.

2. For entities which may appear multiple times (weather, road\_condition...), two identical value may appear in both con\_1 and con\_2 columns, when we encounter this situation, we only record once in the “under\_condition”-like tables (relationship between party/case/victim and various condition).

3. We found that PCF related attributes (pcf\_violation, pcf\_violation\_category, pcf\_violation\_subsection, parimary\_collision\_factor) are quite similar and have close connection with each other, so we grouped them into one entity. Each case has exactly one corresponding PCF (entity) that caused it, which means each case participates exactly once in the relation. So, we changed the line between case and cause (in the ER model) into an thick arrow to illustrate the participation constraint. Meanwhile, in the DDL the cause relationship is merged into case, since case has total participation.

4. Each case happens under exactly one condition and location, so condition has 1-to-1 relationship with both case and the collide\_in relationship (the relationship connecting case and location). Thus, there is no need to use the aggregation in the previous ER model, and we directly connect condition to case. Besides, we merged the “case-under-condition” and “case-collide\_in-location” relationship into the case table, since case has total participation in these two relationships.

5. We connect the entity “road\_en” and “weather\_en” directly with entity “case” (instead of entity “condition”), because it is more simple, direct and efficient.

6. We move the attribute “jurisdiction” from the entity “location” to entity ”case” because jurisdiction has little to do with the collision scene.

7. We didn’t change the value of “null”, just dropped them when needed. For example, we don’t consider cases without weather conditions when querying weather and corresponding count of cases.

## Relational Schema

### DDL (UPDATED)

ALTER SESSION SET nls\_date\_format = 'YYYY-MM-DD HH24:MI:SS'

CREATE TABLE Other\_fac\_en(

other\_fac\_num INTEGER,

other\_fac VARCHAR2(3),

PRIMARY KEY (other\_fac\_num)

);

CREATE TABLE Safety\_equip\_en(

safety\_equip\_num INTEGER,

safety\_equip VARCHAR2(3),

PRIMARY KEY (safety\_equip\_num)

);

CREATE TABLE Vehicle(

ve\_num INTEGER,

ve\_type VARCHAR2(50),

ve\_make VARCHAR2(20),

ve\_year INTEGER,

PRIMARY KEY (ve\_num)

);

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 Location(

loc\_num INTEGER,

population INTEGER,

county\_city INTEGER,

loc\_type VARCHAR2(20),

ramp\_int VARCHAR2(10),

PRIMARY KEY (loc\_num)

);

CREATE TABLE Condition(

con\_num INTEGER,

lighting VARCHAR2(50),

road\_surf VARCHAR2(10),

PRIMARY KEY (con\_num)

);

CREATE TABLE Road\_en(

road\_num INTEGER,

road\_con VARCHAR2(20),

PRIMARY KEY (road\_num)

);

CREATE TABLE Weather\_en(

wea\_num INTEGER,

weather\_con VARCHAR2(20),

PRIMARY KEY (wea\_num)

);

CREATE TABLE Case(

case\_id INTEGER,

loc\_num INTEGER NOT NULL,

con\_num INTEGER NOT NULL,

pcf\_num INTEGER NOT NULL,

col\_date DATE,

col\_severity VARCHAR2(30),

col\_time DATE,

hit\_run VARCHAR2(30),

jurisdiction INTEGER,

officer\_id VARCHAR2(10),

process\_date DATE,

tow\_away INTEGER,

col\_type VARCHAR2(30),

PRIMARY KEY (case\_id),

FOREIGN KEY (loc\_num) REFERENCES Location(loc\_num),

FOREIGN KEY(con\_num) REFERENCES Condition(con\_num),

FOREIGN KEY(pcf\_num) REFERENCES PCF(pcf\_num)

);

CREATE TABLE Party\_involve(

party\_id INTEGER,

case\_id INTEGER NOT NULL,

at\_fault INTEGER,

phone VARCHAR2(3),

fin\_resp VARCHAR2(3),

haz\_mat VARCHAR2(3),

move\_pre VARCHAR2(3),

age INTEGER,

drug\_phy VARCHAR2(3),

sobriety VARCHAR2(3),

party\_type VARCHAR2(15),

party\_num INTEGER,

sex VARCHAR2(6),

PRIMARY KEY (party\_id),

FOREIGN KEY (case\_id) REFERENCES Case(case\_id)

);

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)

);

CREATE TABLE Have (

other\_fac\_num INTEGER,

party\_id INTEGER,

PRIMARY KEY (other\_fac\_num, party\_id),

FOREIGN KEY (party\_id) REFERENCES Party\_involve(party\_id),

FOREIGN KEY (other\_fac\_num) REFERENCES Other\_fac\_en(other\_fac\_num)

);

CREATE TABLE Have\_ps(

party\_id INTEGER,

safety\_equip\_num INTEGER,

PRIMARY KEY (party\_id, safety\_equip\_num),

FOREIGN KEY (party\_id) REFERENCES Party\_involve(party\_id),

FOREIGN KEY (safety\_equip\_num) REFERENCES Safety\_equip\_en(safety\_equip\_num)

);

CREATE TABLE Have\_vs(

vic\_id INTEGER,

safety\_equip\_num INTEGER,

PRIMARY KEY (vic\_id, safety\_equip\_num),

FOREIGN KEY (vic\_id) REFERENCES Associate\_victim(vic\_id),

FOREIGN KEY (safety\_equip\_num) REFERENCES Safety\_equip\_en(safety\_equip\_num)

);

CREATE TABLE Take(

ve\_num INTEGER,

party\_id INTEGER,

school\_bus\_rel VARCHAR2(5),

PRIMARY KEY (ve\_num, party\_id),

FOREIGN KEY (party\_id) REFERENCES Party\_involve(party\_id),

FOREIGN KEY (ve\_num) REFERENCES Vehicle(ve\_num)

);

CREATE TABLE Under\_r(

case\_id INTEGER,

road\_num INTEGER,

PRIMARY KEY (road\_num, case\_id),

FOREIGN KEY (road\_num) REFERENCES Road\_en(road\_num),

FOREIGN KEY (case\_id) REFERENCES Case(case\_id)

);

CREATE TABLE Under\_w(

case\_id INTEGER,

wea\_num INTEGER,

PRIMARY KEY (wea\_num, case\_id),

FOREIGN KEY (wea\_num) REFERENCES Weather\_en(wea\_num),

FOREIGN KEY (case\_id) REFERENCES Case(case\_id)

);

## Data Loading/Cleaning

In “milestone2.ipynb”, we dropped the duplicates of cases and then combine “party\_num” and “case\_id” in the victims2018.csv to assign “party\_id” to each victim. We extracted “road condition”, “weather condition”, “safety equipment”, “other associated factors” (where one case/party/victim may involve two values of them) to independent small tables and created the corresponding relationship tables. We concluded independent “location”, “PCF” and “condition” table from collisions2018.csv, added unique index to them and connected them with case by this index.We found that ‘TIME’ type will lead to error in query (using Oracle SQL), but we realized that ‘DATE’ datatype could include time. So we combined “collision\_date” into “collision\_time” (which now includes both time and date of the collision). Finally we got 17 tables corresponding to the 17 tables of the DLL. We used Oracle EPFL to import our data.

## Query Implementation

**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 | 544739 |
| 2003 | 538952 |
| 2004 | 538294 |
| 2005 | 532724 |
| 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 take. To illustrate the whole row of the most popular, we sort the table and take the first row.

***SQL statement***

SELECT \*

FROM

(

(SELECT ve\_make， COUNT(ve\_make) AS N\_VEHICLE

FROM (vehicle INNER JOIN take ON vehicle.ve\_num = take.ve\_num)

GROUP BY ve\_make)

ORDER BY N\_VEHICLE DESC

)

WHERE ROWNUM = 1

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

|  |  |
| --- | --- |
| VE\_MAKE | N\_VEHICLE |
| FORD | 1129700 |

**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. We count the total number of cases and cases under “dark” lighting, then calculate the fraction.

***SQL statement***

SELECT DISTINCT

CONCAT(ROUND((SELECT COUNT(\*) AS c

FROM case, condition

WHERE lighting LIKE '%dark%' AND case.con\_num = condition.con\_num)\*100 /

(SELECT COUNT (\*) FROM case), 2),'%') as fraction

FROM condition

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

|  |
| --- |
| FRACTION |
| 27.98% |

**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 “under\_w”

***SQL statement***

SELECT count(\*) AS N\_collisions

FROM

(SELECT \*

FROM under\_w, weather\_en

WHERE weather\_en.wea\_num = under\_w.wea\_num AND weather\_en.weather\_con = '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***

***(We consider grouping and finding the largest row as two tasks )***

***5.a***

SELECT TO\_CHAR(COL\_DATE, 'D') AS WEEK\_DAY, COUNT(\*) AS N\_COLLISONS

FROM CASE

GROUP BY TO\_CHAR(COL\_DATE, 'D')

ORDER BY TO\_CHAR(COL\_DATE, 'D') ASC

***5.b***

SELECT \*

FROM

(SELECT TO\_CHAR(COL\_DATE, 'D') AS WEEK\_DAY, COUNT(\*) AS N\_COLLISONS

FROM CASE

GROUP BY TO\_CHAR(COL\_DATE, 'D')

ORDER BY COUNT(\*) DESC)

WHERE ROWNUM = 1

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

***5.a***

|  |  |
| --- | --- |
| WEEK\_DAY | N\_COLLISIONS |
| 1 | 428287 |
| 2 | 516798 |
| 3 | 535742 |
| 4 | 536068 |
| 5 | 536813 |
| 6 | 614852 |
| 7 | 509497 |

***5.b***

|  |  |
| --- | --- |
| WEEK\_DAY | N\_COLLISIONS |
| 6 | 614852 |

**Query 6:**

***Description of logic:***

List all weather types and their corresponding number of collisions in descending order of the collisions. We first join the weather\_en table with the under\_w table to know which case is under which weather condition, then we group cases by weather\_con and list weather and the count number.

***SQL statement***

SELECT WEATHER\_CON AS WEATHER, COUNT(\*) AS COUNT

FROM WEATHER\_EN

INNER JOIN UNDER\_W

ON WEATHER\_EN.wea\_num = UNDER\_W.wea\_num

GROUP BY WEATHER\_CON

ORDER BY COUNT(\*) DESC

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

|  |  |
| --- | --- |
| WEATHER | COUNT |
| clear | 2941037 |
| cloudy | 548249 |
| raining | 223752 |
| fog | 21259 |
| wind | 13952 |
| snowing | 8530 |
| other | 6960 |

**Query 7:**

***Description of logic:***

Count the number of parties that are at-fault, with financial responsibility and loose material. The attributes “at-fault” and “fin\_resp” are in the table of party, we can filter them directly, but we need to find the road\_num connected with the case from the case\_id of party\_involve table and the under\_r table, then use road\_en table to know if the road condition is “road\_loose”.

We first extract the road\_num of “road\_loose”, then join it with the under\_r table to find which cases are under such road condition. Then we filter the party\_id with its case\_id of party\_involve table who is at fault and with financial responsibility. Finally we join the two table on the same case\_id and count the number of distinct parties.

***SQL statement***

SELECT COUNT(DISTINCT PARTY\_ID) AS N\_PARTIES

FROM

(SELECT CASE\_ID

FROM UNDER\_R

INNER JOIN (SELECT ROAD\_NUM

FROM ROAD\_EN

WHERE ROAD\_EN.ROAD\_CON = 'loose material') road\_loose

ON UNDER\_R.ROAD\_NUM = road\_loose.ROAD\_NUM) case\_loose

INNER JOIN (SELECT PARTY\_ID, CASE\_ID

FROM PARTY\_INVOLVE

WHERE AT\_FAULT = 1 AND FIN\_RESP = 'Y') party\_atfault

ON case\_loose.CASE\_ID = party\_atfault.CASE\_ID

***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)

WHERE rownum = 1;

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

***8.a***

|  |
| --- |
| MEDIAN\_VIC\_AGE |
| 25 |

***8.b***

|  |
| --- |
| MOST\_COMMON\_SEAT\_POSITION |
| 3 |

**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\_ids who use belt using table have\_vs and safety\_equip\_en. Then we count the unique vic\_ids and use this number as the numerator. Finally we get the fraction and format it to percentage.

***SQL statement***

SELECT CONCAT(ROUND(a.fraction\*100.0,2),'%') AS fraction

FROM(SELECT DISTINCT

(SELECT COUNT(vic\_id) AS count

FROM

(SELECT h1.vic\_id as vic\_id

FROM have\_vs h1, safety\_equip\_en s1

WHERE h1.safety\_equip\_num = s1.safety\_equip\_num

AND s1.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 |
| 1.06% |

**Query 10:**

***Description of logic:***

Compute the fraction of collisions happening for each hour of the day, and display as 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 calculate the total number of the cases. Then we divide the count number of each hour by the total number to get each percentage.

***SQL statement***

SELECT h\_count.hour, CONCAT(ROUND((h\_count.count/sum\_count.sum\*100.0),2),'%') AS fraction

FROM(SELECT

DISTINCT EXTRACT(hour from cast(col\_time as timestamp)) as hour, count(\*) as count

FROM case

GROUP BY EXTRACT(hour from cast(col\_time as timestamp))

ORDER BY hour ASC) h\_count,

(SELECT sum(h\_count.count) AS sum

FROM

(SELECT

DISTINCT EXTRACT(hour from cast(col\_time as timestamp)) as hour, count(\*) as count

FROM case

GROUP BY EXTRACT(hour from cast(col\_time as timestamp))

ORDER BY hour ASC) h\_count) sum\_count

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

|  |  |
| --- | --- |
| 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% |

## General Comments

We made necessary modifications based on the feedback of Deliverable 1, and achieved the required results in Deliverable 2 accordingly.

However, Besides, we also have several questions, about which we hope to get your feedback as well.

We found that TIME type will lead to error in query (in Oracle SQL) so we use DATE type. There are also other types like timestamp/datetime and so on, and we are not sure if this type is the best choice.

Is it better to combine “condition” into the case? For it has only two attributes now. We found that we may need more join operations based on this design, is it acceptable?