Technical Exercise

InnovateEDU

Name: Manikandan Ganesh

Email: karthickgm27@gmail.com

Phone: 312-874-8660

LinkedIn: https://www.linkedin.com/in/manikandan-ganesh-02919963

GitHub: https://github.com/manikandanganesh2791

Objective

The purpose of this task is to make a clean attendance record dataset which is probably going to be stacked into another sole system.

The word" clean" indicates four things here:

1. The final data set must be straightforward and all the department in the organization must find it simple to interpret and utilize.

2. The structure of my design must be flexible so that, if the business rules changes, the design can accommodate the change easily.

3. The solution must open doors for more data.

4. The data set must be consistent across all the areas.

Taking these factors into consideration, I chose to take a shot at the given unclean data set from three unique systems (c, k, s) and change them into a solitary clean data set fulfilling all the above given requirements.

Data Cleaning

I first started off by looking at the data set of all the given three different systems and I observed the presence of duplicate records in the attendance_s.csv file.

Removing Duplicates in attendance_s.csv file

I could observe the presence of redundant data (like the one given below) carrying the same ID value in the attendance_s.csv file.

ID	CardSourceID	CardHolderID	DateIn	ReportField1	ReportField2	DateOut	HoursWorked
4071	73	218696326	2/27/2017 10:38			NULL	0
4071	73	218696326	2/27/2017 10:38			NULL	0
4072	73	201706009	2/27/2017 10:39			NULL	0
4072	73	201706009	2/27/2017 10:39			NULL	0

So, I removed these duplicate entries from the data set. Also, I restructured the table format by having separate fields for Date and Time and having only those records for which the value of CardSourceID is 73. My final table after cleaning the entire data set had the following structure:

ID	CardSourceID	CardHolderID	DateIn	InTime	ReportField1	ReportField2	DateOut	Out Time	HoursWorked
4839	73	226730190	3/21/2017	9:09			NULL	NULL	0
4838	73	223198193	3/21/2017	9:05			3/23/2017	15:48	54.717
7045	73	221142714	NULL	NULL			4/6/2017	15:54	0

I used Python to scrap and cleanse the data set to obtain a new clean source record from the “s” system, in the prescribed format. The code I used to clean the set is given below (code is available in the **cleaning_for_attendance_s_intermediate.ipynb file**).

Python Code:

```
target_file = open('attendance_s_clean.csv', 'w')
attendance_s = open('attendance_s.csv', 'r')
row_s = attendance_s.read().splitlines()
row_list = []
record_dict = {}
date_split_list = []
date_val = ""
target_line = ""

#Method list_val is used to clean the attendance_s.csv file by returning only
#the set of rows having unique Id and eliminates the redundant Id values.
#Also, this method parses the datein and dateout column values by stripping
#the date of the format mm/dd/yyyy and removing the time values from it.
```

```
14.
15. def list_val(row_list):
16.     result = []
17.     for i in range(len(row_list)):
18.         date_val = ""
19.         time_val = ""
20.         date_split_list = []
21.         if(i != 0):
22.             if(i==3):
23.                 if(row_list[i] != 'NULL'):
24.                     date_split_list=row_list[i].split(" ")
25.                     date_val=date_split_list[0]
26.                     time_val=date_split_list[1]
27.                     result.append(date_val)
28.                     result.append(time_val)
29.                 else:
30.                     result.append(row_list[i])
31.                     result.append(row_list[i])
32.             elif(i==6):
33.                 if(row_list[i] != 'NULL'):
34.                     date_split_list=row_list[i].split(" ")
35.                     date_val=date_split_list[0]
36.                     time_val=date_split_list[1]
37.                     result.append(date_val)
38.                     result.append(time_val)
39.                 else:
40.                     result.append(row_list[i])
41.                     result.append(row_list[i])
42.             else:
43.                 result.append(row_list[i])
44.     return result
45.
46. #Loop that forms the dictionary 'record_dict' to store the records from
47. #attendance_s.csv file having the key as Id and values as a list carrying
48. #CardSourceId, CardHolderId, DateIn, DateOut and HoursWorked values
49. #for CardSourceId '73'
50. for line in row_s[1:]:
51.     row_list = line.split(",")
52.     if(row_list[1] == '73'):
53.         if(row_list[0] in record_dict):
54.             continue
55.         record_dict[row_list[0]] = list_val(row_list)
56. column_header = 'ID'+','+'CardSourceID'+','+"CardHolderID"+','+"DateIn"+','+"InTime"+',
    '+"ReportField1"+','+"ReportField2"+','+"DateOut"+','+"OutTime"+','+"HoursWorked"
57. target_file.write(column_header)
58. target_file.write("\n")
59. for key, value in record_dict.items():
60.     target_line = ""
61.     target_line+=key
62.     for i in range(len(value)):
63.         target_line+=","+value[i]
64.     target_file.write(target_line)
65.     target_file.write("\n")
66. target_file.close()
```

After cleaning the data set, I decided to dump the entries into a MySQL table exclusively for attendance_s.csv entries. Before creating a table, I first created a database named “**attendance**” inside the MySQL engine using the script –

CREATE DATABASE attendance;

The purpose of creating the database is to have a common store and easy accessibility for all the tables I will be creating in this exercise.

I created a table exclusive for attendance_s.csv file entries inside the **attendance** database using the script,

```
drop table if exists attendance_s;
create table attendance_s(
ID int,
CardSourceID int,
CardHolderID int,
DateIn varchar(12),
InTime varchar(5),
ReportField1 varchar(100),
ReportField2 varchar(100),
DateOut varchar(12),
OutTime varchar(5),
HoursWorked int);
```

I dumped the values from the attendance_s_clean.csv (cleaned version of attendance_s.csv) into the MySQL table using the following command:

```
LOAD DATA LOCAL INFILE '~/attendance_s_clean.csv' INTO TABLE attendance_s
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

attendance_c and **attendance_k** files structure was more appropriate than the **attendance_s** file. So, I decided to first dump the entire contents of **attendance_c.csv** file into **attendance_c** table and **attendance_k.csv** file into **attendance_k** table using the following queries.

```
drop table if exists attendance_c;
create table attendance_c(
ID int not null auto_increment,
CardHolderID int,
Date varchar(12),
Status varchar(25),
PRIMARY KEY(ID));
LOAD DATA LOCAL INFILE '/vagrant_data/attendance_c.csv'
INTO TABLE attendance_c
FIELDS TERMINATED BY ','
```

```
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
(@col1,@col2,@col3) set CardHolderID=@col1,Date=@col2,Status=@col3;
```

One unique criteria to be noted down while dumping the contents from attendance_c.csv into the table is, unlike the attendance_s.csv file, attendance_c.csv file did not have a primary key field. To overcome this problem, I used an **AUTO INCREMENTED** primary key field and customized the process of dumping using temporary variables (@col1, @col2, @col3).

```
drop table if exists attendance_k;
create table attendance_k(
ID int not null auto_increment,
CardHolderID int,
Date varchar(12),
Status varchar(50),
PRIMARY KEY(ID));
LOAD DATA LOCAL INFILE '/vagrant_data/attendance_k.csv'
INTO TABLE attendance_k
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(@col1,@col2,@col3,@col4) set CardHolderID=@col1,Status=@col2,Date=@col4;
```

Once I dumped all the records from the csv files into the corresponding database tables, I decided to design the system which is consistent and at the same time, flexible to any additional business logic.

System Design

After reading the business requirement documentation, I decided to integrate the data from three different systems (c, k, s) and load them together into a single table that can hold relationship with all the system entities as well as be flexible to new business logic/ideas. I gave the name **attendance_main** to my final table and decided on having five columns. The structure of my final table is given below:

attendance_main				
Column Name	**Data Type**	**Primary Key**	**Auto Increment**	**Description**
ID	int	YES	YES	Primary Key field
CardHolderID	Int	NO	NO	Identifies the name of the person for whom the event is logged
Date	varchar(12)	NO	NO	Event date field
Status	varchar(35)	NO	NO	The attendance status field

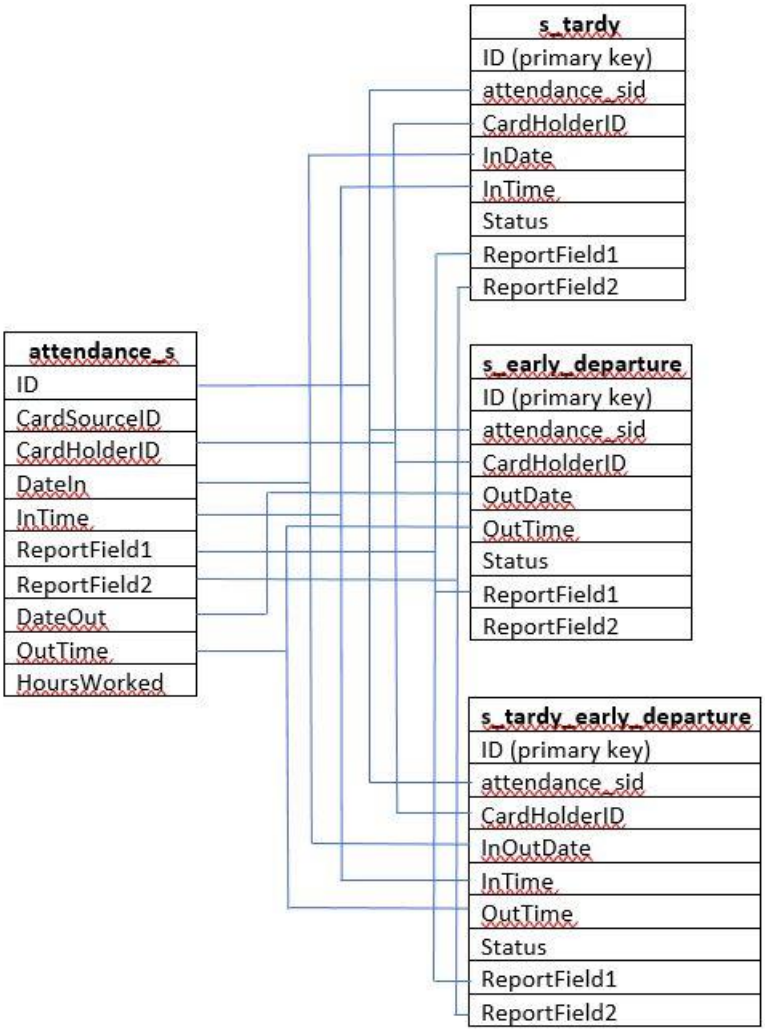
The final table will be loaded from the three given source tables. But before loading the entries directly, I decided to further break down the records from attendance_s table into three separate dimension tables having records based on DateIn and DateOut entries. The reason I chose to move ahead with this step is because of the given business logic which inferred,

- if cardsourceid = 73 and there is a value for datein then the event should be labeled “tardy”
- if cardsourceid = 73 and there is a value for dateout then the event should be labeled “Early departure”

However, the main type of inconsistency or uncertainty that I could observe from the given logic was, there were entries having values in both datein and dateout fields. In such cases, I decided to label the event status as “**tardy and Early departure**” with respect to the following conditions:

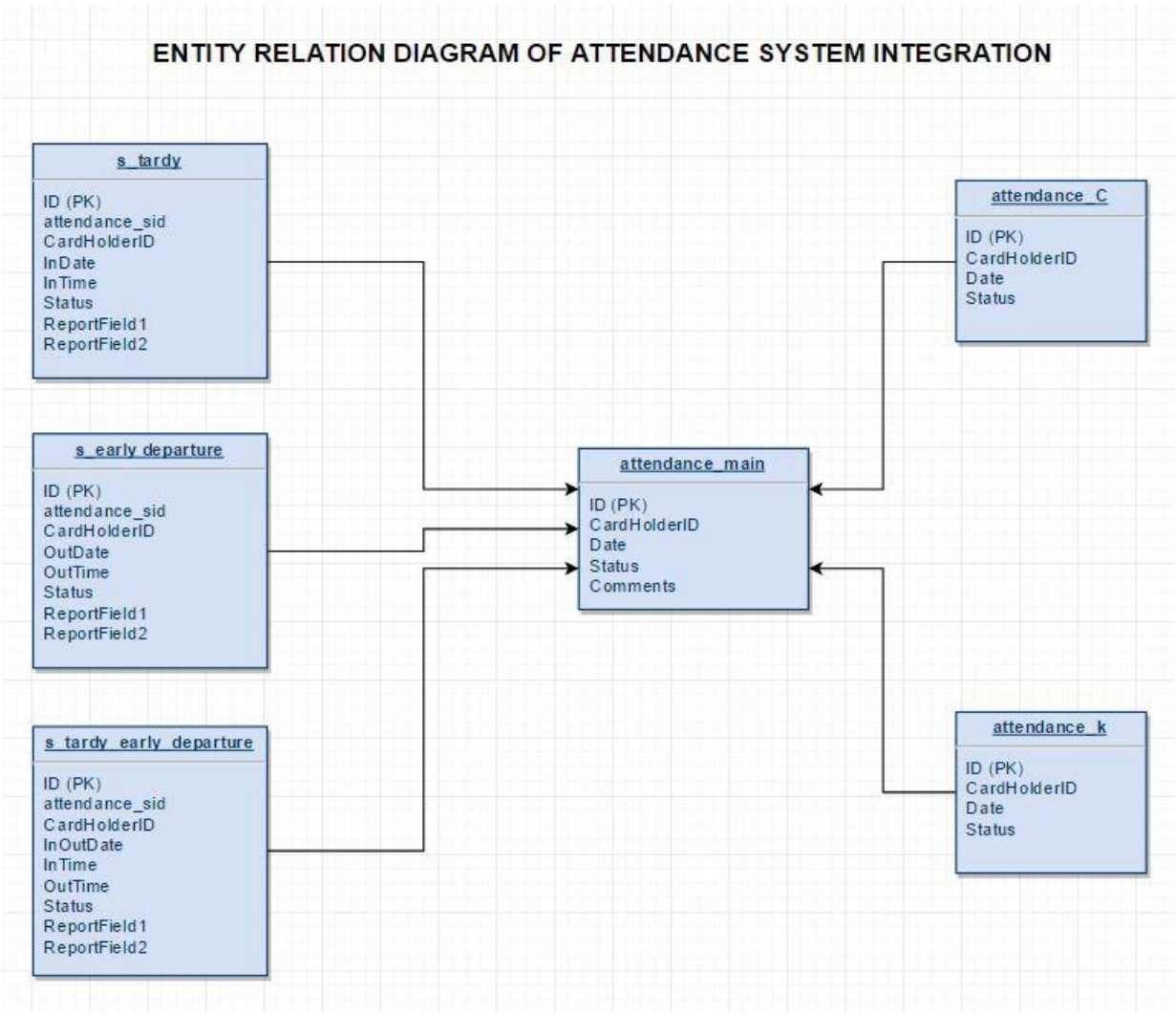
- if the values in datein and dateout fields are equal, then the event status can be labelled “**tardy and Early departure**”
- if the value in datein field is not equal to the value in dateout field, then the event status for the specific date (datein value) can be labelled “**tardy**” and the even status for the corresponding dateout value can be labelled “**Early departure**”

The breakdown structure of attendance_s into different dimension tables is given below:



The first table **tardy_s** will be carrying all the records corresponding to the status "**tardy**". **s_early_departure** table will be carrying records pointing to "**Early departure**". The final table **s_tardy_early_departure** will contain records corresponding to both "**tardy and Early departure**".

ENTITY RELATION DIAGRAM OF ATTENDANCE SYSTEM INTEGRATION



IMPLEMENTATION

The first step of in the implementation of my design is creating the dimension tables from the attendance_s.

The SQL queries that I used to create the tables are:

```
drop table if exists s_tardy;
create table s_tardy(
ID int not null auto_increment,
attendance_sid int,
CardHolderID int,
InDate varchar(12),
InTime varchar(5),
Status varchar(5),
ReportField1 varchar(100),
ReportField2 varchar(100),
PRIMARY KEY(ID));

drop table if exists s_early_departure;
create table s_early_departure(ID int not null auto_increment,
attendance_sid int,
CardHolderID int,
OutDate varchar(12),
OutTime varchar(5),
Status varchar(15),
ReportField1 varchar(100),
ReportField2 varchar(100),
PRIMARY KEY(ID));

drop table if exists s_tardy_early_departure;
create table s_tardy_early_departure(ID int not null auto_increment,
attendance_sid int,
CardHolderID int,
InOutDate varchar(12),
InTime varchar(5),
OutTime varchar(5),
Status varchar(25),
ReportField1 varchar(100),
ReportField2 varchar(100),
PRIMARY KEY(ID));
```

I used the following SQL queries with the specific conditions (as mentioned earlier) to load the data from **attendance_s** table into the corresponding dimension tables:

```
insert into s_tardy (attendance_sid, CardHolderID, InDate, InTime, Status, ReportField1, ReportField2)
SELECT ID, CardHolderID, DateIn, InTime, "tardy", ReportField1, ReportField2
from attendance_s
where (DateIn IS NOT NULL and DateOut IS NULL) || ((DateIn IS NOT NULL and DateOut IS NOT NULL) and (DateIn != DateOut));

insert into s_early_departure(attendance_sid, CardHolderID, OutDate, OutTime, Status,ReportField1,ReportField2)
select ID, CardHolderID, DateOut, OutTime, "Early departure", ReportField1, ReportField2
from attendance_s
where (DateOut IS NOT NULL and DateIn IS NULL) || ((DateOut IS NOT NULL and DateIn IS NOT NULL) and (DateOut != DateIn));

insert into s_tardy_early_departure(attendance_sid,CardHolderID, InOutDate, InTime, OutTime, Status,ReportField1,ReportField2)
select ID, CardHolderID, DateIn, InTime, OutTime, "tardy and Early departure", ReportField1, ReportField2
from attendance_s
where (DateOut IS NOT NULL and DateIn IS NOT NULL) and (DateIn = DateOut);
```

After loading the data from the “s” system based on the three categories (tardy, early departure and tardy & early departure), I decided to check for inconsistencies in the integrated data set.

For inconsistency check I looked out for data that are repetitive or that are negating each other in terms of attendance status. For example, let us consider the records given in the table below:

ID	CardSourceID	CardHolderID	DateIn	InTime	Report Field1	Report Field2	DateOut	OutTime	Hours Worked
4359	73	201706009	3/15/2017	12:33			NULL	NULL	0
4303	73	201706009	NULL	NULL			3/15/2017	11:35	0

If you closely observe the records given above, it is uncertain under which category the record comes under. It apparently appears in both s_tardy table with attendance_sid as 4359 and in s_early_departure table with attendance_sid as

4303. The status in s_tardy table will remain as **tardy** and the status s_early_departure table will remain as **Early departure**. So, while loading this data into the main table, the attendance status for a specific person for a specific day will be both **tardy** and **Early departure** which by itself is highly inconsistent. To overcome this problem, I decided to have an intermediate bridge table to clean the highly inconsistent records within the table and load it into the final main table. I named the intermediate bridge table as **attendance_s_intermediate.** I created and loaded the records into the intermediate table using the following queries:

```
drop table if exists attendance_s_intermediate;
create table attendance_s_intermediate(
ID int not null auto_increment,
CardHolderID int,
Event_Date varchar(12),
Status varchar(25),
PRIMARY KEY(ID));

insert into attendance_s_intermediate(CardHolderID, Event_Date, Status)
select CardHolderID, InDate, Status from s_tardy
union all
select CardHolderID, OutDate, Status from s_early_departure
union all
select CardHolderID, InOutDate, Status from s_tardy_early_departure;
```

From the intermediate table, I was also able to calculate error percentage in the **s** system using the following query,

```
select (select count(*)
from
(select CardHolderID, Date, count(*)
from attendance_main
group by CardHolderID, Date
having count(*) >= 2) as A)/
(select count(*)
from
attendance_main) * 100 as Error_Percentage;
```

The result that I obtained was **3.4731%** which means the success rate of **s** system was **96.52%**.

```
mysql> select (select count(*)
    -> from
    -> (select CardHolderID, Event_Date, count(*)
    -> from attendance_s_intermediate
    -> group by CardHolderID, Event_Date
    -> having count(*) >= 2) as A)/
    -> (select count(*)
    -> from
    -> attendance_s_intermediate)* 100 as Error_Percentage;
+------------------+
| Error_Percentage |
+------------------+
|           3.4731 |
+------------------+
1 row in set (0.01 sec)
```

My first step in cleaning the intermediate table is identifying the inconsistent records using the following query:

select CardHolderID, Event_Date, count(*)
from attendance_s_intermediate
group by CardHolderID, Event_Date having count(*) >= 2;

I could identify 29 sets of records that were inconsistent in the table.

```
mysql> select CardHolderID, Date , count(*)
    -> from attendance_main
    -> group by CardHolderID, Date having count(*) >=2;
+--------------+-----------+----------+
| CardHolderID | Date      | count(*) |
+--------------+-----------+----------+
|    201706009 | 3/15/2017 |        2 |
|    214313231 | 3/15/2017 |        2 |
|    215453226 | 3/13/2017 |        2 |
|    215675604 | 3/15/2017 |        2 |
|    215732025 | 3/13/2017 |        2 |
|    216144675 | 3/13/2017 |        2 |
|    216564161 | 3/15/2017 |        2 |
|    216863860 | 3/15/2017 |        2 |
|    217182989 | 3/15/2017 |        2 |
|    217197920 | 4/6/2017  |        2 |
|    217197938 | 3/23/2017 |        2 |
|    217197938 | 4/6/2017  |        2 |
|    218468627 | 3/15/2017 |        2 |
|    218509479 | 3/23/2017 |        2 |
|    218512499 | 3/23/2017 |        8 |
|    218696326 | 3/7/2017  |        2 |
|    220012637 | 3/15/2017 |        2 |
|    220117584 | 3/16/2017 |        2 |
|    220126635 | 4/6/2017  |        2 |
|    220377972 | 3/15/2017 |        2 |
|    220982136 | 4/6/2017  |        2 |
|    221373053 | 3/15/2017 |        2 |
|    221855265 | 3/15/2017 |        2 |
|    222467359 | 4/6/2017  |        2 |
|    222537730 | 3/15/2017 |        2 |
|    222581233 | 3/15/2017 |        2 |
|    223241829 | 3/15/2017 |        2 |
|    223276627 | 3/15/2017 |        2 |
|    232981423 | 3/15/2017 |        2 |
+--------------+-----------+----------+
29 rows in set (0.00 sec)
```

To clean all the 29 set of records and tag them to an appropriate attendance status, I created an additional table called **combined_name** to update the intermediate table inconsistent records. To flag the rows of inconsistent entries, I added an additional column called **duplicate_check** in my intermediate table and

update it with **True** in rows where inconsistent records were present and update it with **False** in rows where there is no sight of inconsistent records.

```
alter table attendance_s_intermediate
add column duplicate_check varchar(5);

create table combined_name
(ID int not null auto_increment,
name varchar(250),
PRIMARY KEY(ID));

insert into combined_name(name)
select concat(CardHolderID,'|',Event_Date) from attendance_s_intermediate;

update attendance_s_intermediate
set duplicate_check = 'True'
where concat(CardHolderID, '|', Event_Date)in
(select name
from combined_name
group by name
having count(*) >=2);

update attendance_s_intermediate
set duplicate_check = 'False'
where concat(CardHolderID, '|', Event_Date) not in
(select name
from combined_name
group by name
having count(*) >=2);
```

After updating the flag column, I exported the inconsistent entries (entries having duplicate_check column value as **True**) into a separate csv file called **intermediate_inconsistent.csv**. The consistent entries (entries having duplicate_check column value as **False**) on the other hand were exported into another csv file called **intermediate_consistent.csv**. The queries used to export the values are:

```
select * from attendance_s_intermediate where duplicated_check = 'True'
into outfile '/var/lib/mysql-files/intermediate_inconsistent.csv'
fields terminated by ','
enclosed by '"'
lines terminated by '\n';
```

select * from attendance_s_intermediate where duplicate_check = 'False'
into outfile '/var/lib/mysql-files/ intermediate_consistent.csv'
fields terminated by ','
enclosed by '"'
lines terminated by '\n';

I again switched back to Python for tagging the datasets to a relevant status and writing them back into a more cleaner intermediate table. I used the following Python code to clean the intermediate set:

```
target_file= open('attendance_s_consistent.csv', 'w')
inconsistent_main = open('intermediate_inconsistent.csv','r')
consistent_main = open('intermediate_consistent.csv','r')
row_inconsistent = inconsistent_main.read().splitlines()
row_consistent = consistent_main.read().splitlines()
row_inconsistent_list = []
row_consistent_list=[]
row_inconsistent_add_list=[]
row_inconsistent_result=[]
id_date_set=set()
flag = False
line_string = ""
for i in row_inconsistent:
    row_inconsistent_list.append(i.split(","))
row_inconsistent_result=[]
for i in range(len(row_inconsistent_list)-1):
    if row_inconsistent_list[i][1]+'|'+row_inconsistent_list[i][2] not in id_date_set:

        date_val=row_inconsistent_list[i][2]
        id_val=row_inconsistent_list[i][1]
        status_val = row_inconsistent_list[i][3]
        j=i+1
        while(j<len(row_inconsistent_list)):
            flag = False
            if row_inconsistent_list[j][1] == id_val and row_inconsistent_list[j][2] ==
 date_val:
                if row_inconsistent_list[j][3] != status_val:
                    flag = True
                    row_inconsistent_add_list=[]
                    row_inconsistent_add_list.append(row_inconsistent_list[j][1])
                    row_inconsistent_add_list.append(row_inconsistent_list[j][2])
                    row_inconsistent_add_list.append('tardy and Early departure')
                    row_inconsistent_result.append(row_inconsistent_add_list)
                    id_date_set.add(row_inconsistent_list[j][1]+'|'+row_inconsistent_li
st[j][2])
                    break
                else:
                    j+=1
            else:
                flag = True
                row_inconsistent_add_list=[]
                row_inconsistent_add_list.append(row_inconsistent_list[j-1][1])
                row_inconsistent_add_list.append(row_inconsistent_list[j-1][2])
```

```
                row_inconsistent_add_list.append(row_inconsistent_list[j-1][3])
                row_inconsistent_result.append(row_inconsistent_add_list)
                id_date_set.add(row_inconsistent_list[j-
1][1]+'|'+row_inconsistent_list[j-1][2])
                break
        if flag == False:
            row_inconsistent_add_list=[]
            row_inconsistent_add_list.append(id_val)
            row_inconsistent_add_list.append(date_val)
            row_inconsistent_add_list.append(status_val)
            row_inconsistent_result.append(row_inconsistent_add_list)
            id_date_set.add(id_val+'|'+date_val)
            break

column_header = 'CardHolderID' + ',' + 'Event_Date' + ',' + 'Status'
target_file.write(column_header)
target_file.write('\n')
for i in row_consistent:
    line_string = ""
    row_consistent_list=i.split(",")
    for j in range(len(row_consistent_list)):
        if j not in (0,3,4):
            line_string+=row_consistent_list[j]+','
        elif j == 3:
            line_string+=row_consistent_list[j]
    target_file.write(line_string)
    target_file.write('\n')
for i in row_inconsistent_result:
    line_string = ""
    for j in range(len(i)):
        if j != 2:
            line_string+=i[j]+','
        else:
            line_string+=i[j]
    target_file.write(line_string)
    target_file.write('\n')
target_file.close()
inconsistent_main.close()
consistent_main.close()
```

After cleaning the set, I decided to dump the entries from the csv file into a cleaner MySQL table for **s** system called **attendance_s_consistent**.

```
drop table if exists attendance_s_consistent;
create table attendance_s_consistent(
ID int not null auto_increment,
CardHolderID int,
Event_Date varchar(12),
Status varchar(25),
PRIMARY KEY(ID));
LOAD DATA LOCAL INFILE '/vagrant_data/attendance_s_consistent.csv'
INTO TABLE attendance_s_consistent
FIELDS TERMINATED BY ','
```

```
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(@col1,@col2,@col3) set CardHolderID=@col1,Event_Date=@col2,Status=@col3;
```

Now, the data from the **"s"** system became highly consistent, clean and they are resting inside the table **attendance_s_consistent**.

Now the next step in my implementation phase is to check for the consistency of records in the **attendance_c** table.

In this step, I used the same methodology as I used for the consistency check in **attendance_s_intermediate** table.

First, I checked for entries which had multiple recordings for the same card holder ID for a specific date using the following query. I could observe **2665** records that were inconsistent.

From this, I calculated the error percentage in the **c** system using the following query:

```
select (select count(*) from
(select combined_name, count(*)
from attendance_c
group by combined_name
having count(*) >= 2) as A)/(select count(*) from attendance_c) * 100 as Error_Percentage;
```

The error percentage was recorded as 4.5223% and that means, the success rate of the **c** system was **95.47%**.

To remove the inconsistency and make the Error Percentage as 0, I used a slightly different approach from **attendance_s_intermediate**. The SQL queries are given below:

```
alter table attendance_c
add column combined_name varchar(100);

update attendance_c
set combined_name = concat(CardHolderID, '|', Date);
```

```
drop table if exists combined_name_c;
create table combined_name_c (name varchar(100));
insert into combined_name_c(name)
select combined_name
from attendance_c
group by combined_name
having count(*) >= 2

select * from attendance_c
where combined_name in
(select name from combined_name_c)
limit 1
into outfile '/var/lib/mysql-files/attendance_c_inconsistent.csv'
fields terminated by ','
lines terminated by '\n';

select * from attendance_c
where combined_name not in
(select * from combined_name_c)
into outfile '/var/lib/mysql-files/attendance_c_consistent.csv'
fields terminated by ','
lines terminated by '\n';
```

On executing the above queries, I could obtain two separate csv files named **attendance_c_inconsistent** and **attendance_c_consistent** carrying records that are inconsistent and consistent respectively.

By inconsistency, I mean a set of records having multiple rows for the same CardHolderID and same Date. Sometimes, the status value remains throughout the repetitive rows and for some records the status change. Let us consider the following case for example:

External ID	eventdate	EventType
221229073	12/16/2016	Tardy
221229073	12/16/2016	In Attendance

On closely observing the External ID, eventdate and EventType columns, for the same ID and for the same date, EventType values are different. It is highly ambiguous to find under which category could this record be coming under.

So, I decided to clean up my attendance_c table using the following Python code to get two separate lists, one with all the inconsistent records and the other data set with consistent records (records that are not redundant and redundant records maintaining the same status).

```
c_target_file = open('attendance_c_intermediate.csv', 'w')
c_target_inconsistent = open('attendance_c_inconsistentlist.csv', 'w')
attendance_c_inconsistent = open('attendance_c_inconsistent.csv', 'r')
attendance_c_consistent = open('attendance_c_consistent.csv', 'r')
row_c_inconsistent = attendance_c_inconsistent.readlines()
row_c_consistent = attendance_c_consistent.readlines()
row_list_inc = []
row_list_inc_add=[]
row_list_inconsistent_final = []
row_list_consistent_final = []
row_list_con=[]
row_list_con_add=[]
dict_inc = {}
flag = False
line_string = ""
row_add_inc_list=[]
row_inc_result_list=[]

for i in row_c_inconsistent:
    row_list_inc_add=[]
    a = i.replace('\r','').strip()
    row_list_inc = a.split(',')
    if row_list_inc[4] in dict_inc:
        row_list_inc_add.append(row_list_inc[1])
        row_list_inc_add.append(row_list_inc[2])
        row_list_inc_add.append(row_list_inc[3])
        dict_inc[row_list_inc[4]].append(row_list_inc_add)
    else:
        row_list_inc_add.append(row_list_inc[1])
        row_list_inc_add.append(row_list_inc[2])
        row_list_inc_add.append(row_list_inc[3])
        dict_inc[row_list_inc[4]] = []
        dict_inc[row_list_inc[4]].append(row_list_inc_add)
for i,j in dict_inc.items():
    row_list_inc_add = j
    for k in range(len(row_list_inc_add)):
        flag = False
        if k == 0:
            prime_status = row_list_inc_add[k][2]
        else:
            if row_list_inc_add[k][2] != prime_status:
                flag = True
                row_list_inconsistent_final.append([(row_list_inc_add[k][0]+',' + row_l
ist_inc_add[k][1])])
                break
    if(flag==False):
        row_list_consistent_final.append(row_list_inc_add[0])
column_header = "CardHolderID" + ',' + "Date"
c_target_inconsistent.write(column_header)
c_target_inconsistent.write('\n')
for i in range(len(row_list_inconsistent_final)):
    row_list_inc_add=row_list_inconsistent_final[i]
    line_string=""
```

```
    line_string+=row_list_inc_add[0]
    c_target_inconsistent.write(line_string)
    c_target_inconsistent.write('\n')
c_target_inconsistent.close()
column_header = "CardHolderID" + ',' + "Date" + ',' + "Status"
c_target_file.write(column_header)
c_target_file.write('\n')
i_count=0
for i in row_c_consistent:
    i_count+=1
    a = i.replace('\r','').strip()
    row_list_con=a.split(',')
    line_string=""
    for j in range(len(row_list_con)):
        if j in (1,2):
            if j==1:
                a = row_list_con[j].replace('"''"',',')
                line_string+=a+','
            else:
                line_string+=row_list_con[j]+','
        elif j == 3:
            line_string+=row_list_con[j]
    c_target_file.write(line_string)
    c_target_file.write('\n')
for i in row_list_consistent_final:
    a = i[0].replace('"','')
    line_string=a+','+i[1]+','+i[2]
    c_target_file.write(line_string)
    c_target_file.write('\n')
c_target_file.close()
```

The above cleans up the data by looking at inconsistencies in terms of status and those which are differing in status will under one booth called "**attendance_c_inconsistent.csv**" and those which are not will go under another booth called "**attendance_c_intermediate.csv**".

After obtaining these two data sets, I decided to use the consistent one (**attendance_c_intermediate.csv**) in my MySQL data model and I dumped them using the following query:

```
drop table if exists attendance_c_intermediate;
create table attendance_c_intermediate(
ID int not null auto_increment,
CardHolderID int,
Event_Date varchar(12),
Status varchar(50),
PRIMARY KEY(ID));
LOAD DATA LOCAL INFILE '/vagrant_data/attendance_c_intermediate.csv'
INTO TABLE attendance_c_intermediate
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
```

IGNORE 1 LINES
(@col1,@col2,@col3) set CardHolderID=@col1,Event_Date=@col2,Status=@col3;

The next set of process that I took in my implementation is the final cleaning of **attendance_k** file removing all the inconsistent data and keeping only consistent records.

After eye balling the records from attendance_k file, I could identify blank records in the ExternalID column which apparently made CardHolderID column in my MySQL table to be 0.

So, I decided to remove the rows having 0 values in CardHolderID using the following query:

```
delete from attendance_k
where CardHolderID LIKE '0';
```

After removing the null values, I decided to calculate the error percentage in my k system using the following query:

```
select (select count(*)
from
(select CardHolderID, Date, count(*)
from attendance_k
group by CardHolderID, Date
having count(*) >= 2) as A)/
(select count(*)
from
attendance_k)* 100 as Error_Percentage;
```

I obtained the result as 0.0478%

```
mysql> select (select count(*)
    -> from
    -> (select CardHolderID, Date, count(*)
    -> from attendance_k
    -> group by CardHolderID, Date
    -> having count(*) >= 2) as A)/
    -> (select count(*)
    -> from
    -> attendance_k)* 100 as Error_Percentage;
+------------------+
| Error_Percentage |
+------------------+
|           0.0478 |
+------------------+
1 row in set (0.00 sec)
```

To remove the inconsistencies and nullify the error percentage, I exported the records into a csv file and used the Python code (given below) on the csv.

```
select *
from attendance_k
into outfile '/var/lib/mysql-files/attendance_k_inconsistent.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';

select *
from attendance_k
where concat(CardHolderID,'|',Date) IN
(select concat(CardHolderID,'|',Date)
from attendance_k
group by concat(CardHolderID,'|',Date)
having count(*)>=2)
into outfile '/var/lib/mysql-files/inconsistent_k.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

```
target_file_k = open('attendance_k_intermediate.csv', 'w')
attendance_k_inconsistent = open('inconsistent_k.csv','r')
attendance_k = open('attendance_k_inconsistent.csv','r')
row_k=attendance_k.readlines()
row_k_inc=attendance_k_inconsistent.readlines()
row_k_inc_list=[]
row_k_con_list=[]
row_k_list=[]
row_k_set=set()
row_k_dict={}
line_string=""
flag = False
for i in row_k_inc:
    row_k_list=[]
    a = i.replace('\r','').strip()
    row_k_list=a.split(",")
    if row_k_list[1]+'|'+row_k_list[2] not in row_k_set:
        row_k_set.add(row_k_list[1]+'|'+row_k_list[2])
    row_k_inc_list.append(row_k_list)
    if row_k_list[1]+'|'+ row_k_list[2] not in row_k_dict:
        row_k_dict[row_k_list[1]+'|'+row_k_list[2]] = []
        row_k_dict[row_k_list[1]+'|'+row_k_list[2]].append(row_k_list[3])
    else:
        row_k_dict[row_k_list[1]+'|'+row_k_list[2]].append(row_k_list[3])
for i in row_k:
    row_k_list=[]
    a = i.replace('\r','').strip()
    row_k_list=a.split(',')
    if row_k_list[1]+'|'+row_k_list[2] not in row_k_set:
        line_string=row_k_list[1]+','+row_k_list[2]+','+row_k_list[3]
        target_file_k.write(line_string)
        target_file_k.write('\n')
row_k_inc_list=[]
```

```
row_k_con_dup_list=[]
for k,v in row_k_dict.items():
    flag = False
    prime_status = v[0]
    for i in range(1,len(v)):
        if v[i] != prime_status:
            flag = True
            row_k_inc_list.append(k)
            break
    if flag == False:
        row_k_con_dup_list.append([k])
        row_k_con_dup_list.append(v[1])
        for i in range(len(row_k_con_dup_list)):
            id_dateval = row_k_con_dup_list[0]
            fetch_list=id_dateval.split("|")
            fetch_list.append(row_k_con_dup_list[1])
            line_string=""
            line_string=fetch_list[0]+','+fetch_list[1]+','+fetch_list[2]
            target_file_k.write(line_string)
            target_file_k.write('\n')
```

After cleaning the data set, I dumped the entire consistent set into an intermediate table in MySQL data store which I named as **attendance_k_intermediate**.

```
drop table if exists attendance_k_intermediate;
create table attendance_k_intermediate(
ID int not null auto_increment,
CardHolderID int,
Date varchar(12),
Status varchar(50),
PRIMARY KEY(ID));
LOAD DATA LOCAL INFILE '/vagrant_data/attendance_k_intermediate.csv'
INTO TABLE attendance_k_intermediate
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(@col1,@col2,@col3) set CardHolderID=@col1,Date=@col2,Status=@col3;
```

At this stage, the process of forming a highly consistent data sets from all the three different systems has been completed. The next step that I was aiming for is to dump the entries from the intermediate tables into the final table which will be my final highly consistent and flexible set.

While doing this process, I made decision on which intermediate set must be given a top priority for loading the data into the final table. For this reason, I used the calculated Error Percentage values to make a trade-off on which one is better

than the other. The Error Percentage comparison and ultimately deciding on the intermediate set can be clearly understood from the following table:

Intermediate Set	Error Percentage	Priority
attendance_k_intermediate	0.0478	**1**
attendance_s_intermediate	3.4731	**2**
attendance_c_intermediate	4.5223	**3**

So, I decided to load the table from attendance_k_intermediate first, followed by attendance_s_intermediate so that the values present in the attendance_k_intermediate will be given the top most priority while loading if the same value is encountered in the attendance_s_intermediate. I used the following query to avoid redundancy in the final table according to the priority that I assumed from the Error Percentage value:

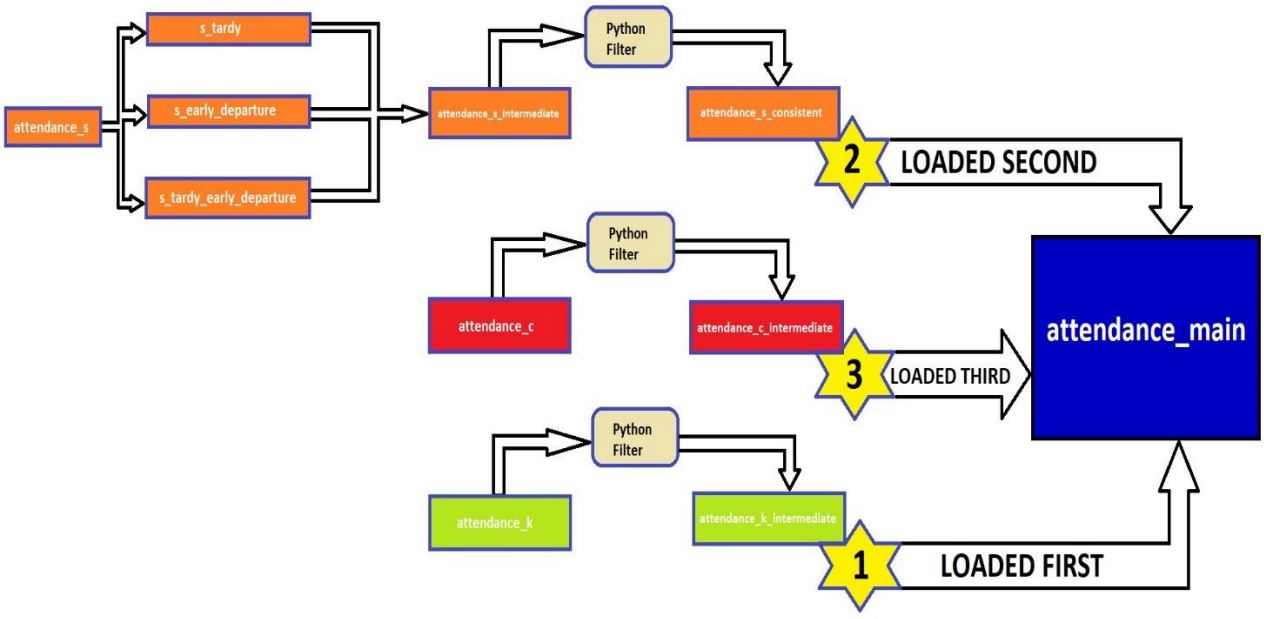
```
drop table if exists attendance_main;
create table attendance_main(
ID int not null auto_increment,
CardHolderID int,
Date varchar(12),
Status varchar(50),
PRIMARY KEY(ID));

insert into attendance_main(CardHolderID,Date,Status)
select CardHolderID, Date, Status from attendance_k_intermediate
union all
select CardHolderID, Event_Date, Status from attendance_s_intermediate
where concat(CardHolderID,'|',Event_Date) not in
(select concat(CardHolderID,'|',Date) from attendance_k_intermediate)
union all
select CardHolderID, Event_Date, Status from attendance_c_intermediate
where concat(CardHolderID,'|',Event_Date) not in
(select concat(CardHolderID,'|',Date) from attendance_k_intermediate
union all
select concat(CardHolderID,'|',Event_Date) from attendance_s_intermediate);
```

After populating the final table, I could observe a total of around 5953 matching rows of consistent data from all the three sets put together which means there are 5953 rows creating conflicts in my final merge. I tackled this problem by taking the priority (given above) into consideration and I call it my **conflict resolution method**. I observed this value of **5953** using the following query which takes the difference of total records present in all the three intermediate sets and the total records present in my final main data set after elimination.

```
select (select count(*) from attendance_k_intermediate) +
(select count(*) from attendance_s_intermediate) +
(select count(*) from attendance_c_intermediate) as total_number_records;
```

```
select (select count(*) from attendance_k_intermediate) +
(select count(*) from attendance_s_intermediate) +
(select count(*) from attendance_c_intermediate) -
(select count(*) from attendance_main) as records_eliminated;
```

The whole process of implementation can be described in one single diagram given below:



ANALYSIS

According to the task objectives, there were couple of analysis to be done in my final data set:

1. What day of the year had the highest number of absences?
2. Are specific days of the week more likely to result in absences/tardies?

To obtain result for both the questions, I used the following SQL queries on my final data set:

Query to identify the day of the year having the highest number of absences:

```
(select Date, count(Status) as Count_of_Absences
from
(select STR_TO_DATE(Str_Date, '%m/%d/%Y') as Date,Status
from
(select REPLACE(Date,'"','') as Str_Date, Status
from attendance_main
where Status LIKE '%Absen%') as A) as B
where YEAR(Date) = 2016
group by Date
order by count(Status) desc
limit 1)
union all
(select Date, count(Status) as count_of_absent
from
(select STR_TO_DATE(Str_Date, '%m/%d/%Y') as Date,Status
from
(select REPLACE(Date,'"','') as Str_Date, Status
from attendance_main
where Status LIKE '%Absen%') as A) as B
where YEAR(Date) = 2017
group by Date
order by count(Status) desc
limit 1);
```

In my final data set, I could observe records under two consecutive years (2016 and 2017). So, I used the above query to identify the day for both the years and I obtained the following result:

Date	Day	Count_of_Absences
2016-12-22	Thursday	97
2017-03-15	Wednesday	136

```
mysql> (select Date,
    -> case
    -> when DAYOFWEEK(Date) = 1 then 'Sunday'
    -> when DAYOFWEEK(Date) = 2 then 'Monday'
    -> when DAYOFWEEK(Date) = 3 then 'Tuesday'
    -> when DAYOFWEEK(Date) = 4 then 'Wednesday'
    -> when DAYOFWEEK(Date) = 5 then 'Thursday'
    -> when DAYOFWEEK(Date) = 6 then 'Friday'
    -> when DAYOFWEEK(Date) = 7 then 'Saturday'
    -> end as Day,
    -> count(Status) as Count_of_Absent
    -> from
    -> (select STR_TO_DATE(Str_Date, '%m/%d/%Y') as Date,Status
    -> from
    -> (select REPLACE(Date,'"','') as Str_Date, Status
    -> from attendance_main
    -> where Status LIKE '%Absen%') as A) as B
    -> where YEAR(Date) = 2016
    -> group by Date
    -> order by count(Status) desc
    -> limit 1)
    -> union all
    -> (select Date,
    -> case
    -> when DAYOFWEEK(Date) = 1 then 'Sunday'
    -> when DAYOFWEEK(Date) = 2 then 'Monday'
    -> when DAYOFWEEK(Date) = 3 then 'Tuesday'
    -> when DAYOFWEEK(Date) = 4 then 'Wednesday'
    -> when DAYOFWEEK(Date) = 5 then 'Thursday'
    -> when DAYOFWEEK(Date) = 6 then 'Friday'
    -> when DAYOFWEEK(Date) = 7 then 'Saturday'
    -> end as Day,
    -> count(Status) as count_of_absent
    -> from
    -> (select STR_TO_DATE(Str_Date, '%m/%d/%Y') as Date,Status
    -> from
    -> (select REPLACE(Date,'"','') as Str_Date, Status
    -> from attendance_main
    -> where Status LIKE '%Absen%') as A) as B
    -> where YEAR(Date) = 2017
    -> group by Date
    -> order by count(Status) desc
    -> limit 1);
+------------+-----------+-----------------+
| Date       | Day       | Count_of_Absent |
+------------+-----------+-----------------+
| 2016-12-22 | Thursday  |              97 |
| 2017-03-15 | Wednesday |             136 |
+------------+-----------+-----------------+
2 rows in set (0.04 sec)
```

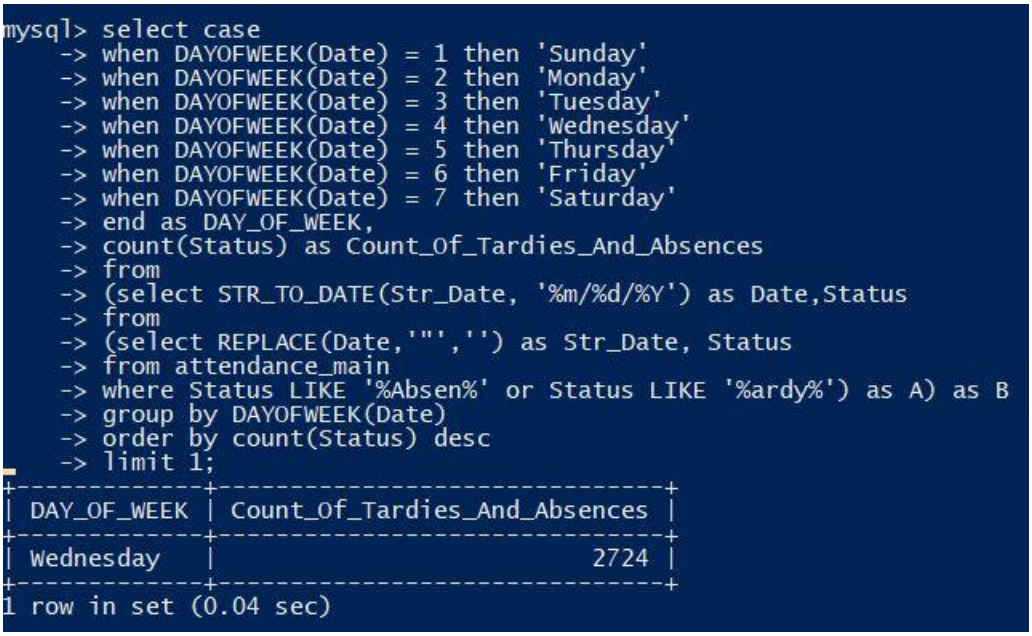
Query to identify the specific day of the week to result in absence/tardies:

```
select case
        when DAYOFWEEK(Date) = 1 then 'Sunday'
        when DAYOFWEEK(Date) = 2 then 'Monday'
        when DAYOFWEEK(Date) = 3 then 'Tuesday'
```

```
        when DAYOFWEEK(Date) = 4 then 'Wednesday'
        when DAYOFWEEK(Date) = 5 then 'Thursday'
        when DAYOFWEEK(Date) = 6 then 'Friday'
        when DAYOFWEEK(Date) = 7 then 'Saturday'
end as DAY_OF_WEEK,
count(Status) as Count_Of_Tardies_And_Absences
from
(select STR_TO_DATE(Str_Date, '%m/%d/%Y') as Date,Status
from
(select REPLACE(Date,'"','') as Str_Date, Status
from attendance_main
where Status LIKE '%Absen%' or Status LIKE '%ardy%') as A) as B
group by DAYOFWEEK(Date)
order by count(Status) desc
limit 1;
```

On executing the above query, I obtained the following result:

DAY_OF_WEEK	Count_of_Tardies_And_Absences
Wednesday	2724



Recommendations

The first and the foremost recommendation that I would like to give to the school administrator to simplify the attendance maintenance is to have a consistent database system that follows the properties of ACID. Here, I mainly insist of having a suitable mechanism that takes in the record of each individual card holders appropriately at the end of every single working day. For example, the records coming out of **S** system had several inconsistencies to it apart from having an unclean structure to itself. One of possible recommendation that I would like to give for maintaining the **S** system is to change the structure of the table as given below:

attendance_s	
Column Name	**Data Type**
ID	INTEGER (PK)
CardSourceID	INTEGER
CardHolderID	INTEGER
DateIn	VARCHAR OR DATE
InTime	VARCHAR OR DATETIME
DateOut	VARCHAR OR DATE
OutTime	VARCHAR OR DATE
HoursWorked	INTEGER
Comments	VARCHAR

By having this structure, even when there is an inconsistency in the records entered, it can be rectified on looking at the values entered in the **Comments** field. The records from this table can be loaded across the already mentioned dimension tables by using a scheduler or a trigger every night after the school hours to track down inconsistencies as well as to have an ease of maintenance.

Likewise, the final table that I formed can also have an additional comments field, where the comment for a student status can be written and later used for analytical purpose. Time fields are not necessary in the final table (for reducing the complexity) as the time variable can be fetched from the intermediate tables for a specific date using **JOINS**.

ADDITIONAL INFORMATION

Now, all the records in my final table are consistent. Even after this phase, I could observe certain conflicts in the degree of success of my final table. It is not 100% accurate because the attendance record for all the dates given in the List of School days table is not tracked and there are some missing dates in the source as well as target table. I used the following python code to identify the missing dates:

```
attendance_source = open('attendance_main.csv','r')
attendance_main=open('List of school days .csv','r')
row_source = attendance_source.readlines()
row_main=attendance_main.read().splitlines()
row_source_list=[]
row_source_set = set()
row_source_targetlist=[]
for i in row_source:
    a = i.replace('\r','')
    row_source_list=a.split(',')
    b = row_source_list[2].replace('"','')
    if b not in row_source_set:
        row_source_set.add(b)

for i in row_main[1:]:
    if i not in row_source_set:
        row_source_targetlist.append(i)
print row_source_targetlist
```

I got the following result:

```
['8/30/2016', '9/1/2016', '2/9/2017', '3/14/2017', '4/10/2017']
```

/So, my final table does not carry values for the dates,

1. **8/30/2016**
2. **9/1/2016**
3. **2/9/2017**
4. **3/14/2017**
5. **4/10/2017**

On reverse engineering to check for the presence of these dates in all the source tables (**attendance_s.csv, attendance_c.csv, attendance_k.csv**), I couldn't find values for these dates. Hence, the final table couldn't track down records for these days.

DELIVERABLES

Thus, my final deliverable files are listed below:

1. **From the S System (Available in S System folder)**
   a. s_tardy.csv
   b. s_early_departure.csv
   c. s_tardy_early_departure.csv
   d. attendance_s_intermediate.csv
   e. attendance_s_consistent.csv
   f. cleaning_for_attendance_s_intermediate.ipynb
   g. cleaning_for_attendance_s_consistent.ipynb

2. **From the C System (Available in C System folder)**
   a. attendance_c_intermediate.csv
   b. cleaning_for_attendance_c_intermediate.ipynb

3. **From the K System (Available in K System folder)**
   a. attendance_k_intermediate.csv
   b. cleaning_for_attendance_k_intermediate.ipynb

4. attendance_main_missing_dates.ipynb for identifying the missing dates

5. **Inconsistent Records (Available in Inconsistent Records folder)**
   From the **S** system:
   - attendance_k_empty.csv has records whose ExternalID column is NULL
   - attendance_c_inconsistentlist.csv has records that are inconsistent in the C system
   - inconsistent_k.csv has records that are inconsistent from the S system

6. **SQL Scripts.txt** file has all the SQL scripts that I used for data modeling and analysis

Final Table – attendance_main.csv