London Bike Sharing Dataset

By Transport for London (TFL) On Kaggle

1. *Database:*

(1) London Bike Sharing Database from 04-01-2015 to 03-01-2017, which is organised by Hristo Mavrodiev on Kaggle. It was brought from the TfL's free transport data service, which granted the public the licence for free use.

(2) LINK: [London bike sharing dataset (kaggle.com)]

2. The Process based on the requirements

(1) To prepare the process, the first step is to bring the origin dataset.

➔ *In the 'bike_variation' schema, the name of the table is 'london_merged'*

```
19 •   DESC london_merged;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
timestamp	datetime	YES		NULL	
cnt	int	YES		NULL	
t1	double	YES		NULL	
t2	double	YES		NULL	
hum	double	YES		NULL	
wind_speed	double	YES		NULL	
weather_code	bigint	YES		NULL	
is_holiday	bigint	YES		NULL	
is_weekend	bigint	YES		NULL	
season	bigint	YES		NULL	

10 COLUMNS:

timestamp

cnt

t1, t2, hum

wind_speed, weather_code

is_holiday, is_weekend

season

(2) To make a new meaningful database, five tables are created with related columns: time_count, weather, season, holiday, weekend

USE: ***CREATE, INSERT INTO with SELECT clause***

to bring the columns data from the London_merged dataset

```
CREATE TABLE time_count(timestamp datetime, cnt int);
CREATE TABLE weather(timestamp datetime, t1 double, t2 double, hum double, weather_code int);
CREATE TABLE holiday(timestamp datetime, is_holiday int);
CREATE TABLE weekend(timestamp datetime, is_weekend int);
CREATE TABLE season(timestamp datetime, season int);
```

```
INSERT INTO time_count(timestamp,cnt) SELECT timestamp, cnt FROM london_merged;
INSERT INTO weather(timestamp,t1, t2, hum, weather_code) SELECT timestamp,t1, t2, hum, weather_code FROM london_merged;
INSERT INTO holiday(timestamp, is_holiday) SELECT timestamp, is_holiday FROM london_merged;
INSERT INTO weekend(timestamp, is_weekend) SELECT timestamp, is_weekend FROM london_merged;
INSERT INTO season(timestamp, season) SELECT timestamp, season FROM london_merged;
```

➔ The Primary key and the Foreign keys were set to create relations between tables. → *In the **ALTER** statement, using **ADD CONSTRAINTS***

```
ALTER TABLE time_count ADD CONSTRAINT pk_timestamp PRIMARY KEY(timestamp);
ALTER TABLE holiday ADD CONSTRAINT fk_timestamp FOREIGN KEY(timestamp) REFERENCES time_count(timestamp);
ALTER TABLE season ADD CONSTRAINT fk_timestampS FOREIGN KEY(timestamp) REFERENCES time_count(timestamp);
ALTER TABLE weekend ADD CONSTRAINT fk_timestampW FOREIGN KEY(timestamp) REFERENCES time_count(timestamp);
ALTER table weather add constraint fk_timeweather FOREIGN KEY(timestamp) REFERENCES time_count(timestamp);
```

(3) The view showing the overall data flow was created with five tables:

The initial dataset has 10 columns, but **the six columns are selected** to **create a view**, which is regarded as **more crucial** than the other four.

The timestamp column is a primary key in the time_count table. **The cnt column** is essential to understand the time-serial trends of the shared bike data.

➔ *Using **INNER JOIN ON**, **CREATE VIEW***

```
CREATE VIEW listAll_DESC AS
SELECT tc.timestamp, tc.cnt, wt.t1, se.season, hi.is_holiday, wk.is_weekend
FROM time_count AS tc
INNER JOIN weather AS wt ON tc.timestamp = wt.timestamp
INNER JOIN weekend AS wk ON tc.timestamp = wk.timestamp
INNER JOIN season AS se ON tc.timestamp = se.timestamp
INNER JOIN holiday AS hi ON tc.timestamp = hi.timestamp
ORDER BY tc.cnt DESC;

SELECT * FROM listAll_DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

timestamp	cnt	t1	season	is_holiday	is_weekend
2015-07-09 17:00:00	7860	23	1	0	0
2015-07-09 08:00:00	7531	14.5	1	0	0
2015-08-06 17:00:00	7208	22.5	1	0	0
2015-07-09 18:00:00	6913	22.5	1	0	0
2015-08-06 08:00:00	6585	19	1	0	0
2015-08-06 18:00:00	6394	21.5	1	0	0
2015-07-09 16:00:00	6033	23	1	0	0
2015-07-08 17:00:00	5560	20	1	0	0
2016-09-14 08:00:00	5422	22.5	2	0	0
2016-09-14 18:00:00	5345	26	2	0	0
2016-09-13 08:00:00	5342	20.5	2	0	0
2016-05-17 08:00:00	5322	13.5	0	0	0

(4) A stored function was set to represent the current state of the temperature. When the data is mainly composed of numerical data, it can be difficult to figure out the data.

➔ *Using* ***VIEW, INNER JOIN, ORDER BY***

```
CREATE VIEW listAll_WARMTH AS
SELECT tc.timestamp, tc.cnt, FEEL(wt.t1) AS temp_Feel, se.season, hi.is_holiday, wk.is_weekend
FROM time_count AS tc
INNER JOIN weather AS wt ON tc.timestamp = wt.timestamp
INNER JOIN weekend AS wk ON tc.timestamp = wk.timestamp
INNER JOIN season AS se ON tc.timestamp = se.timestamp
INNER JOIN holiday AS hi ON tc.timestamp = hi.timestamp
ORDER BY tc.cnt DESC;
```

timestamp	cnt	t1	season	is_holiday	is_weekend
2015-07-09 17:00:00	7860	23	1	0	0
2015-07-09 08:00:00	7531	14.5	1	0	0
2015-08-06 17:00:00	7208	22.5	1	0	0
2015-07-09 18:00:00	6913	22.5	1	0	0
2015-08-06 08:00:00	6585	19	1	0	0
2015-08-06 18:00:00	6394	21.5	1	0	0
2015-07-09 16:00:00	6033	23	1	0	0
2015-07-08 17:00:00	5560	20	1	0	0
2016-09-14 08:00:00	5422	22.5	2	0	0
2016-09-14 18:00:00	5345	26	2	0	0
2016-09-13 08:00:00	5342	20.5	2	0	0
2016-05-17 08:00:00	5322	13.5	0	0	0
2016-10-05 08:00:00	5322	14	2	0	0
2015-07-09 07:00:00	5309	13.5	1	0	0
2016-07-19 17:00:00	5304	31	1	0	0
2016-09-13 18:00:00	5297	30	2	0	0
2016-09-15 08:00:00	5295	20.5	2	0	0
2016-07-19 18:00:00	5282	28.5	1	0	0
2016-09-13 17:00:00	5189	31.5	2	0	0

Without a Function (ONLY numerical data)

timestamp	cnt	temp_Feel	season	is_holiday	is_weekend
2015-07-09 17:00:00	7860	WARM	1	0	0
2015-07-09 08:00:00	7531	COLD	1	0	0
2015-08-06 17:00:00	7208	COLD	1	0	0
2015-07-09 18:00:00	6913	COLD	1	0	0
2015-08-06 08:00:00	6585	COLD	1	0	0
2015-08-06 18:00:00	6394	COLD	1	0	0
2015-07-09 16:00:00	6033	WARM	1	0	0
2015-07-08 17:00:00	5560	COLD	1	0	0
2016-09-14 08:00:00	5422	COLD	2	0	0
2016-09-14 18:00:00	5345	WARM	2	0	0
2016-09-13 08:00:00	5342	COLD	2	0	0
2016-05-17 08:00:00	5322	COLD	0	0	0
2016-10-05 08:00:00	5322	COLD	2	0	0
2015-07-09 07:00:00	5309	COLD	1	0	0
2016-07-19 17:00:00	5304	WARM	1	0	0
2016-09-13 18:00:00	5297	WARM	2	0	0
2016-09-15 08:00:00	5295	COLD	2	0	0
2016-07-19 18:00:00	5282	WARM	1	0	0
2016-09-13 17:00:00	5189	WARM	2	0	0

With a Function

This function will help users understand data quickly in the future or re-utilise the columns in other tables to change numeric data to string data.

(5) **The subquery function was added** while doing the INNER JOIN to count the number by the season group. The initial season table listed the corresponding season by the timestamp.

A new intended query is to show both the number of shared bikes by season and the number of seasons by season group. (0: spring, 1: summer, 2: autumn)

And the colder, the fewer people ride a bike. This tendency was considered and applied to the condition in the HAVING clause as excepting the winter season.

➔ *Using **INNER JOIN, GROUP BY, HAVING, ORDER BY** and **Subquery!***

```
25 •  SELECT* FROM season;
26 •  SELECT se.season, seq.seasonTimes, sum(tc.cnt) AS totalRide
27    FROM season AS se
28    INNER JOIN (SELECT season, COUNT(season) AS seasonTimes
29                FROM season
30                GROUP BY season) AS seq ON se.season = seq.season
31    INNER JOIN time_count AS tc ON tc.timestamp = se.timestamp
32    GROUP BY se.season, seq.seasonTimes
33    HAVING se.season IN (0,1,2)
34    ORDER BY se.season ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

season	seasonTimes	totalRide
0	4394	4850236
1	4387	6424609
2	4303	5073040

(6) To efficient data management, **the event** was implemented. The event **drops the old data**, which is over 7 years from the current time. But it needed to **remove the foreign key** relation with the reference table and **set the 'ON DELETE CASCADE'.**

➔ After the event was triggered, the order of data was started from 2016-10-27.

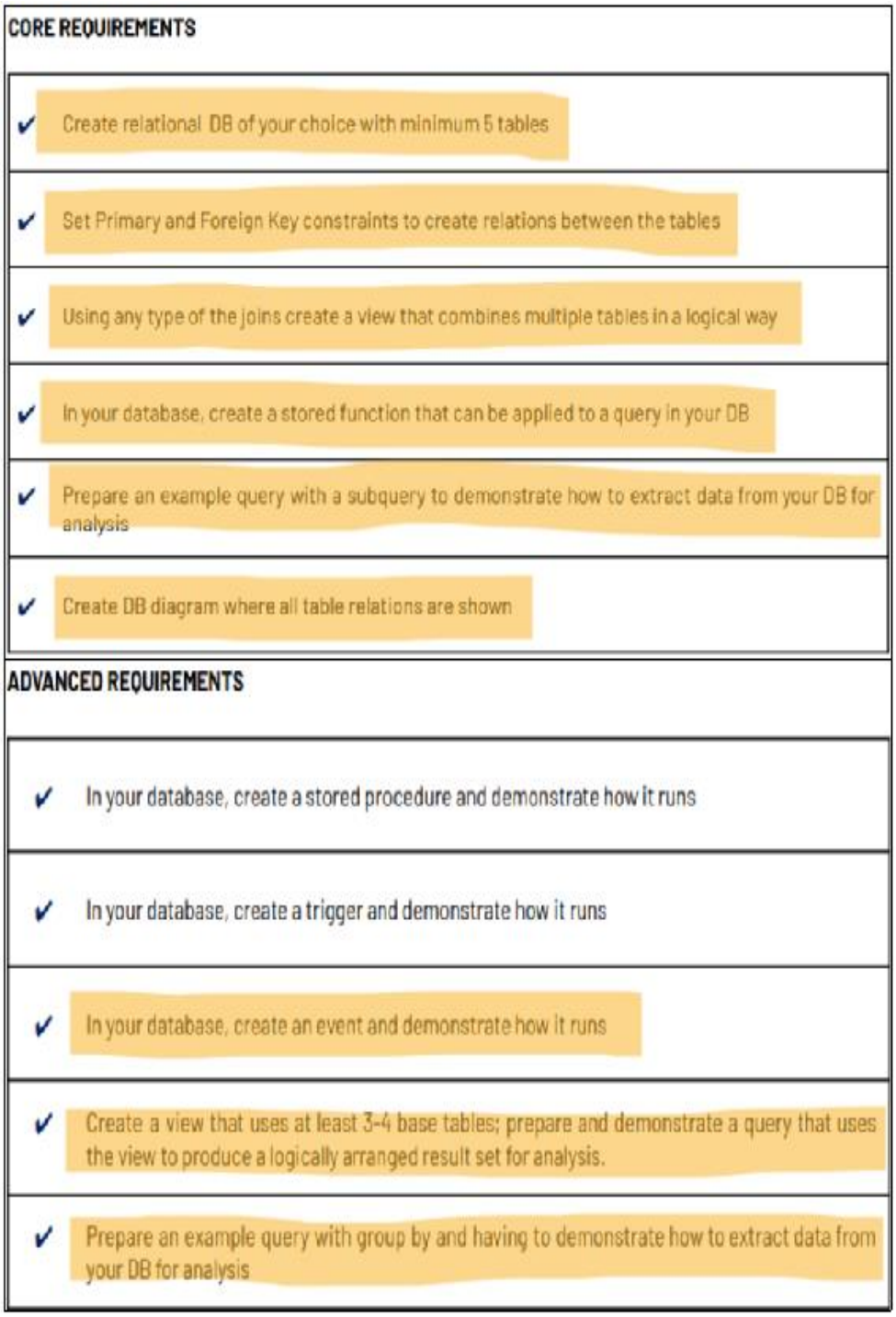
```
75    #---------EVENT ------------------------
76
77 •  SHOW VARIABLES LIKE 'event_scheduler';
78
79 •  CREATE EVENT remove_old ON SCHEDULE EVERY 1 YEAR
80    STARTS CURRENT_TIMESTAMP
81    DO
82    DELETE FROM time_count WHERE timestamp < DATE_SUB(NOW(), INTERVAL 7 YEAR);
83 •  SHOW CREATE EVENT remove_old;
84
85 •  SELECT* FROM weather ORDER BY timestamp ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

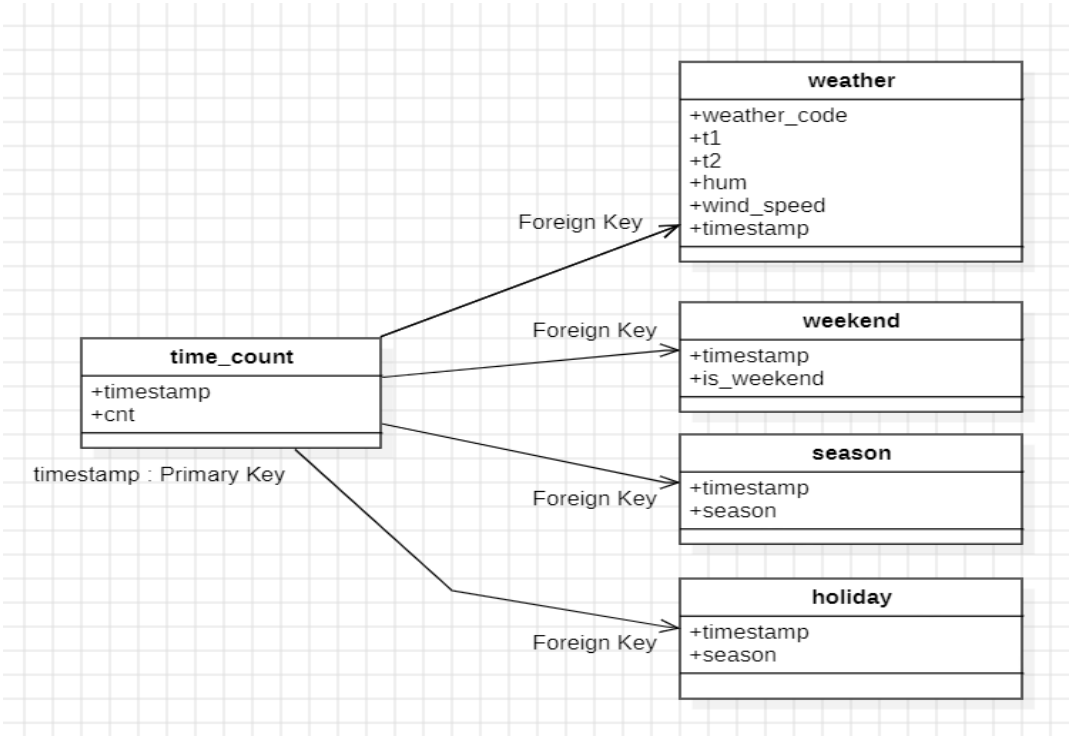
timestamp	t1	t2	hum	weather_code
2016-10-27 12:00:00	13	13	77	2
2016-10-27 13:00:00	15	15	63	2
2016-10-27 14:00:00	16	16	59	2
2016-10-27 15:00:00	16	16	61	2
2016-10-27 16:00:00	15	15	68	3

3. Reflection

(1) Requirements



(2) DB Diagram



(3) What I Achieved

While taking the database course, I could understand both the MySQL system and general SQL functionalities. I learnt the DDL, the DML, INDEX, how to analyse data using multiple operators, etc. This course made me organise and figure out the overall SQL system.

And when I made the SELECT statements in the past, I wrote the columns simply, not mentioning the table. However, I changed the way to write the queries.

(4) What I failed

I selected the project, which was about Transportation for London. This dataset focused on some unique data characteristics, and I've currently had a narrow view of the dataset. Due to these reasons, I didn't use the critical operator such as a LIKE and comparison operator.

And there are two advanced requirements that I wasn't able to meet.

➔ A Trigger and A stored Procedure

(5) What I will make up for in the future

I must review repeatedly what I have learnt,

and supplement the SQL background knowledge currently I have.

The function of the trigger and the procedure should be tried to implement using the dataset in the future.

I could make the new dataset and apply to the data visualisation platform such as Tableau and PowerBI.

4. The Code

```
USE bike_variation;

SHOW TABLES;

DESC london_merged;


CREATE TABLE time_count(timestamp datetime, cnt int);

CREATE TABLE weather(timestamp datetime, t1 double, t2 double, hum double, weather_code int);

CREATE TABLE holiday(timestamp datetime, is_holiday int);

CREATE TABLE weekend(timestamp datetime, is_weekend int);

CREATE TABLE season(timestamp datetime, season int);


INSERT INTO time_count(timestamp,cnt) SELECT timestamp, cnt FROM london_merged;

INSERT INTO weather(timestamp,t1, t2, hum, weather_code) SELECT timestamp,t1, t2, hum,
weather_code FROM london_merged;

INSERT INTO holiday(timestamp, is_holiday) SELECT timestamp, is_holiday FROM london_merged;

INSERT INTO weekend(timestamp, is_weekend) SELECT timestamp, is_weekend FROM
london_merged;

INSERT INTO season(timestamp, season) SELECT timestamp, season FROM london_merged;


SHOW TABLES;

SELECT* FROM holiday;

SELECT* FROM weekend;

SELECT* FROM season;

SELECT* FROM weather;

SELECT* FROM london_merged;

SELECT* FROM time_count;


ALTER TABLE time_count ADD CONSTRAINT pk_timestamp PRIMARY KEY(timestamp);
```

```
ALTER TABLE holiday ADD CONSTRAINT fk_timestamp FOREIGN KEY(timestamp) REFERENCES time_count(timestamp);

ALTER TABLE season ADD CONSTRAINT fk_timestampS FOREIGN KEY(timestamp) REFERENCES time_count(timestamp);

ALTER TABLE weekend ADD CONSTRAINT fk_timestampW FOREIGN KEY(timestamp) REFERENCES time_count(timestamp);

ALTER table weather add constraint fk_timeweather FOREIGN KEY(timestamp) REFERENCES time_count(timestamp);

#-- CREATE VIEW with multiple JOINs ---


CREATE VIEW listAll_DESC AS

SELECT tc.timestamp, tc.cnt, wt.t1, se.season, hi.is_holiday, wk.is_weekend

FROM time_count AS tc

INNER JOIN weather AS wt ON tc.timestamp = wt.timestamp

INNER JOIN weekend AS wk ON tc.timestamp = wk.timestamp

INNER JOIN season AS se ON tc.timestamp = se.timestamp

INNER JOIN holiday AS hi ON tc.timestamp = hi.timestamp

ORDER BY tc.cnt DESC;


SELECT * FROM listAll_DESC;

#---------------------------------------


#-- Checking missing data in the 'weekend' table -------


SELECT wk.timestamp, wk.is_weekend, tc.cnt

FROM weekend AS wk

LEFT JOIN time_count tc ON wk.timestamp = tc.timestamp

ORDER BY wk.is_weekend DESC;

#-----------------------------------------------------


#-- A new intended query by using subquery: season table------


SELECT* FROM season;

SELECT se.season, seq.seasonTimes, sum(tc.cnt) AS totalRide

FROM season AS se
```

```
INNER JOIN (SELECT season, COUNT(season) AS seasonTimes
                        FROM season
                        GROUP BY season) AS seq ON se.season = seq.season
INNER JOIN time_count AS tc ON tc.timestamp = se.timestamp
GROUP BY se.season, seq.seasonTimes
HAVING se.season IN (0,1,2)
ORDER BY se.season ASC;

SELECT* FROM season;
#---------------------------------------------------------------------

#---- Making a stored Function: numerical data --> string data --------

DELIMITER //
CREATE FUNCTION FEEL(t1 DOUBLE) RETURNS VARCHAR(10)
DETERMINISTIC
BEGIN
    DECLARE FEEL VARCHAR(10);
    IF t1 >= 23 THEN SET FEEL = 'WARM';
    ELSE SET FEEL = 'COLD';
    END IF;
     RETURN FEEL;
END//
DELIMITER ;

# ---------------- CHECKING THE FUNCTION to the NEW VIEW ----------------

CREATE VIEW listAll_WARMTH AS
SELECT tc.timestamp, tc.cnt, FEEL(wt.t1) AS temp_Feel, se.season, hi.is_holiday, wk.is_weekend
FROM time_count AS tc
INNER JOIN weather AS wt ON tc.timestamp = wt.timestamp
INNER JOIN weekend AS wk ON tc.timestamp = wk.timestamp
INNER JOIN season AS se ON tc.timestamp = se.timestamp
```

```
INNER JOIN holiday AS hi ON tc.timestamp = hi.timestamp

ORDER BY tc.cnt DESC;


SELECT* FROM listAll_DESC;

SELECT* FROM listAll_WARMTH;

#------------------------------------------------------------------------


#--- Making event (1) DROP A FOREIGN KEY AND SET DROP AUTOMATICALLY (2) MAKING AN
EVENT


ALTER TABLE weather ADD CONSTRAINT FOREIGN KEY(timestamp) REFERENCES
time_count(timestamp) ON DELETE CASCADE;

ALTER TABLE holiday ADD CONSTRAINT FOREIGN KEY(timestamp) REFERENCES
time_count(timestamp) ON DELETE CASCADE;

ALTER TABLE weekend ADD CONSTRAINT FOREIGN KEY(timestamp) REFERENCES
time_count(timestamp) ON DELETE CASCADE;

ALTER TABLE season    ADD CONSTRAINT FOREIGN KEY(timestamp) REFERENCES
time_count(timestamp) ON DELETE CASCADE;


#---------EVENT ------------------------


SHOW VARIABLES LIKE 'event_scheduler';


CREATE EVENT remove_old ON SCHEDULE EVERY 1 YEAR

STARTS CURRENT_TIMESTAMP

DO

DELETE FROM time_count WHERE timestamp < DATE_SUB(NOW(), INTERVAL 7 YEAR);

SHOW CREATE EVENT remove_old;


SELECT* FROM weather ORDER BY timestamp ASC;
```