Benjamin Katz
BikeShare

1. “Count the number of stations in each city.”
The result set fields should be city, number of stations, and order by increasing “number of stations”, then ascending “city name”.

```
SELECT
        DISTINCT city,
        COUNT(station_id) num_stations
FROM
        station
GROUP BY
        city
ORDER BY
        num_stations ASC,
        city ASC
```

Palo Alto|5
Mountain View|7
Redwood City|7
San Jose|16
San Francisco|35

```
sqlite> SELECT
   ...> DISTINCT city,
   ...> COUNT(station_id) num_stations
   ...> FROM
   ...> station
   ...> GROUP BY
   ...> city
   ...> ORDER BY
   ...> num_stations ASC,
   ...> city ASC
   ...> ;
city|num_stations
Palo Alto|5
Mountain View|7
Redwood City|7
San Jose|16
San Francisco|35
```

2. “Find the percentage of bike trips made in each city.”
The result set fields should be city, percentage, ordered, first by decreasing percentage, then by ascending city name.
Note: by “percentage”, I mean a decimal fraction between 0 and 1, rounded to four decimal points. Consider using the Round function. (Alternatively: the “percentages” in the result should sum to 1.0).
Note: define a “trip made in a city” as a trip whose start station was in a given city or whose end station was in a given city. However:

if a trip begins and ends in the same trip, the trip is only counted once.

```
SELECT city, ROUND(SUM(trips)*1.0000/(SELECT SUM(total_trips) FROM(
SELECT

        COUNT(start.id)*1.0000 AS total_trips
FROM
        (trip LEFT JOIN station ON trip.start_station_id = station.station_id) start
        LEFT JOIN
        (trip LEFT JOIN station ON trip.end_station_id = station.station_id)
        end
        ON start.id = end.id
WHERE
         start.city = end.city
UNION ALL

SELECT
        COUNT(start.id)*2.0000 AS total_trips
FROM
        (trip LEFT JOIN station ON trip.start_station_id = station.station_id) start
        LEFT JOIN
        (trip LEFT JOIN station ON trip.end_station_id = station.station_id)
        end
        ON start.id = end.id
WHERE
         NOT start.city = end.city
)),4) perc
FROM(

SELECT
        start.city,
        COUNT(start.id) AS trips
FROM
        (trip LEFT JOIN station ON trip.start_station_id = station.station_id) start
        LEFT JOIN
        (trip LEFT JOIN station ON trip.end_station_id = station.station_id)
        end
        ON start.id = end.id
WHERE
         start.city = end.city
GROUP BY
```

```
        start.city

UNION ALL

SELECT
        start.city,
        COUNT(start.id) AS trips
FROM
        (trip LEFT JOIN station ON trip.start_station_id = station.station_id) start
        LEFT JOIN
        (trip LEFT JOIN station ON trip.end_station_id = station.station_id)
        end
        ON start.id = end.id
WHERE
         NOT start.city = end.city
GROUP BY
        start.city

UNION ALL

SELECT
        end.city,
        COUNT(end.id) AS trips
FROM
        (trip LEFT JOIN station ON trip.start_station_id = station.station_id) start
        LEFT JOIN
        (trip LEFT JOIN station ON trip.end_station_id = station.station_id)
        end
        ON start.id = end.id
WHERE
         NOT start.city = end.city
GROUP BY
        end.city)
GROUP BY
        city
ORDER BY
        perc DESC,
        city ASC;
```

San Francisco|0.8997
San Jose|0.0565
Mountain View|0.0277
Palo Alto|0.0108
Redwood City|0.0052

```
...>
...> SELECT
...> start.city,
...> COUNT(start.id) AS trips
...> FROM
...> (trip LEFT JOIN station ON trip.start_station_id = station.station_id) start
...> LEFT JOIN
...> (trip LEFT JOIN station ON trip.end_station_id = station.station_id)
...> end
...> ON start.id = end.id
...> WHERE
...>  NOT start.city = end.city
...> GROUP BY
...> start.city
...>
...> UNION ALL
...>
...> SELECT
...> end.city,
...> COUNT(end.id) AS trips
...> FROM
...> (trip LEFT JOIN station ON trip.start_station_id = station.station_id) start
...> LEFT JOIN
...> (trip LEFT JOIN station ON trip.end_station_id = station.station_id)
...> end
...> ON start.id = end.id
...> WHERE
...>  NOT start.city = end.city
...> GROUP BY
...> end.city)
...> GROUP BY
...> city
...> ORDER BY
...> perc DESC,
...> city ASC;
San Francisco|0.8997
San Jose|0.0565
Mountain View|0.0277
Palo Alto|0.0108
Redwood City|0.0052
```

3.)"Which stations are the most popular for bike trips?"
As you (by now) know, cities can contain multiple stations. In this query, you're determining, for each of the cities in the database, which station is most "popular" as defined by the number of visits to the station.
Note: define a station as "visited" if a trip either started or ended at that station. If a trip both started and ended at a given station, that trip is considered to be single visit to that station.
The result set fields should be city name, most popular station name, number of visits, ordered by ascending city name.

```
SELECT city, station_name, MAX(pop)
FROM(
SELECT stat, station_name, COUNT(stat) AS pop, city

FROM(
        SELECT start_station_id as stat, city, station_name
```

```
        FROM
                trip LEFT JOIN station ON trip.start_station_id = station.station_id
        WHERE
                trip.start_station_id != trip.end_station_id
        UNION ALL
        SELECT end_station_id as stat, city, station_name
        FROM
                trip LEFT JOIN station ON trip.end_station_id = station.station_id
        )
GROUP BY
        stat
)
GROUP BY
        city
ORDER BY
        city ASC;
```

Mountain View|Mountain View Caltrain Station|12735
Palo Alto|Palo Alto Caltrain Station|3534
Redwood City|Redwood City Caltrain Station|2654
San Francisco|San Francisco Caltrain (Townsend at 4th)|111738
San Jose|San Jose Diridon Caltrain Station|18782

```
sqlite> SELECT city, station_name, MAX(pop)
   ...> FROM(
   ...> SELECT stat, station_name, COUNT(stat) AS pop, city
   ...>
   ...> FROM(
   ...> SELECT start_station_id as stat, city, station_name
   ...> FROM
   ...>
Display all 183 possibilities? (y or n)
   ...> trip LEFT JOIN station ON trip.start_station_id = station.station_id
   ...> WHERE
   ...>
Display all 183 possibilities? (y or n)
   ...> trip.start_station_id != trip.end_station_id
   ...> UNION ALL
   ...> SELECT end_station_id as stat, city, station_name
   ...> FROM
   ...>
Display all 183 possibilities? (y or n)
   ...> trip LEFT JOIN station ON trip.end_station_id = station.station_id
   ...> )
   ...> GROUP BY
   ...> stat
   ...> )
   ...> GROUP BY
   ...> city
   ...> ORDER BY
   ...> city ASC;
Mountain View|Mountain View Caltrain Station|12735
Palo Alto|Palo Alto Caltrain Station|3534
Redwood City|Redwood City Caltrain Station|2654
San Francisco|San Francisco Caltrain (Townsend at 4th)|111738
San Jose|San Jose Diridon Caltrain Station|18782
```

4.)"Find the top ten days that have the highest average bike utilization."
For each of these "top ten" days, the result set should be the full
date, average bike utilization in seconds, ordered by decreasing average utilization.
Round "average utilization" to four decimal places, using the Round
function.
Important simplification: the query implementation will only consider bikes with an id ≤ 100.
This will be a constant in the implementation (details below)
Definition: average bike utilization on datei
is the sum of the durations of all the trips that happened on datei divided by the total
number of bikes (see above).
If a trip overlaps with datei
, but either starts before that date or
ends after that date, consider only the interval that overlaps (from
0:00 to 24:00) with that date.

Simplification: only the date that corresponds to a trip's start date or end date needs to be considered when calculating utilization.
Managing timestamps: you'll notice that SQLite loads its data from csv files (see setup.sql). SQLite stores timestamps as text: you can use datetime(timestamp string) to extract the timestamp out of the string and date(timestamp string) to extract the date out of the string. Consider using the strftime() function when computing the duration between two timestamps.
Very important: when determining the "trip duration", use trip.start_time and trip.end_time. Do not use trip.duration!

```
SELECT day, ROUND((SUM(dur)*60)/(SELECT COUNT(DISTINCT bike_id)
        FROM trip
        WHERE bike_id<101),4) AS util FROM(

SELECT date(start_time) AS day, SUM((strftime('%s', end_time) - strftime('%s',
start_time))/60.0) AS dur
        FROM trip
        WHERE bike_id<101 AND date(start_time) = date(end_time)
        GROUP BY date(start_time)
UNION ALL

SELECT date(start_time) AS day, SUM(((24*60*60)-(strftime('%s',
start_time))%(24*60*60))/60.0) AS dur
        FROM trip
        WHERE bike_id<101 AND date(start_time) != date(end_time)
        GROUP BY date(start_time)
UNION ALL


SELECT date(end_time) AS day, SUM(((strftime('%s', end_time))%(24*60*60))/60.0) AS dur
        FROM trip
        WHERE bike_id<101 AND date(start_time) != date(end_time)
        GROUP BY date(end_time)
)
GROUP BY day
ORDER BY util DESC
LIMIT 10
```

2014-07-13|3884.1758
2014-10-12|3398.9011
2014-08-29|2669.011
2014-06-23|2653.1868
2014-05-18|2618.2418
2014-05-17|2303.0769

2014-08-23|2262.8571
2013-11-02|2190.989
2015-07-01|2058.4615
2014-03-23|2045.2747

```
sqlite> SELECT day, ROUND((SUM(dur)*60)/(SELECT COUNT(DISTINCT bike_id)
   ...> FROM trip
   ...> WHERE bike_id<101),4) AS util FROM(
   ...>
   ...> SELECT date(start_time) AS day, SUM((strftime('%s', end_time) - strftime('%s', start_time))/60.0) AS dur
   ...> FROM trip
   ...> WHERE bike_id<101 AND date(start_time) = date(end_time)
   ...> GROUP BY date(start_time)
   ...> UNION ALL
   ...>
   ...> SELECT date(start_time) AS day, SUM(((24*60*60)-(strftime('%s', start_time))%(24*60*60))/60.0) AS dur
   ...> FROM trip
   ...> WHERE bike_id<101 AND date(start_time) != date(end_time)
   ...> GROUP BY date(start_time)
   ...> UNION ALL
   ...>
   ...>
   ...> SELECT date(end_time) AS day, SUM(((strftime('%s', end_time))%(24*60*60))/60.0) AS dur
   ...> FROM trip
   ...> WHERE bike_id<101 AND date(start_time) != date(end_time)
   ...> GROUP BY date(end_time)
   ...> )
   ...> GROUP BY day
   ...> ORDER BY util DESC
   ...> LIMIT 10;
day|util
2014-07-13|3884.1758
2014-10-12|3398.9011
2014-08-29|2669.011
2014-06-23|2653.1868
2014-05-18|2618.2418
2014-05-17|2303.0769
2014-08-23|2262.8571
2013-11-02|2190.989
2015-07-01|2058.4615
2014-03-23|2045.2747
```

5.)Are bikes being (incorrectly) recorded as being used in two trips concurrently?"
For each pair of conflict trips, the result-set should be the bike id, trip id #1, start time of trip id #1, end time of trip id #1, trip id #2, start time of trip id #2, end time of trip id #2.
The result-set should be sorted by increasing bike id #1, then by increasing trip id #1, then by increasing trip id #2.
Recording a bike as being used in two different trips, at the same time is a data-entry error. We want to detect whether the same bike is recorded as being used in pairs of overlapping intervals.
We'll reduce the size of the result-set by only checking for bikes with 100 ≤ id ≤ 200.
Note: as in other queries, you're allowed to assume that, for all trips, start time ≤ end time.
Suggestion: be careful to report each "conflict pair" only once.
Hint: we can define "overlapping" trips using only the start and end

times of both trips (i.e., no need for complication time duration arithmetic).

```
SELECT trip1.bike_id, trip1.id, trip1.start_time, trip1.end_time, trip2.id, trip2.start_time,
trip2.end_time
FROM
        trip trip1 CROSS JOIN trip trip2
WHERE strftime('%s', trip2.start_time) > strftime('%s', trip1.start_time) AND strftime('%s',
trip2.start_time) < strftime('%s', trip1.end_time) AND trip1.bike_id = trip2.bike_id AND
trip1.bike_id > 99 AND trip2.bike_id >99 AND trip1.bike_id <201 AND trip2.bike_id <201
ORDER BY trip1.bike_id ASC, trip1.id ASC, trip2.id ASC
```

144|815060|2015-06-19 21:26:00|2015-06-19 22:17:00|815073|2015-06-19 22:10:00|2015-06-19 22:17:00
158|576536|2014-12-15 15:05:00|2014-12-15 23:11:00|576591|2014-12-15 16:07:00|2014-12-15 16:17:00
158|576536|2014-12-15 15:05:00|2014-12-15 23:11:00|576604|2014-12-15 16:28:00|2014-12-15 16:40:00

```
sqlite> SELECT trip1.bike_id, trip1.id, trip1.start_time, trip1.end_time, trip2.id, trip2.start_time, tr
ip2.end_time
   ...> FROM
   ...> trip trip1 CROSS JOIN trip trip2
   ...> WHERE strftime('%s', trip2.start_time) > strftime('%s', trip1.start_time) AND strftime('%s', tri
p2.start_time) < strftime('%s', trip1.end_time) AND trip1.bike_id = trip2.bike_id AND trip1.bike_id > 99
 AND trip2.bike_id >99 AND trip1.bike_id <201 AND trip2.bike_id <201
   ...> ORDER BY trip1.bike_id ASC, trip1.id ASC, trip2.id ASC;
bike_id|id|start_time|end_time|id|start_time|end_time
144|815060|2015-06-19 21:26:00|2015-06-19 22:17:00|815073|2015-06-19 22:10:00|2015-06-19 22:17:00
158|576536|2014-12-15 15:05:00|2014-12-15 23:11:00|576591|2014-12-15 16:07:00|2014-12-15 16:17:00
158|576536|2014-12-15 15:05:00|2014-12-15 23:11:00|576604|2014-12-15 16:28:00|2014-12-15 16:40:00
sqlite>
```

6. “Find all the bikes that have been to more than one city”
The result-set should contain the bike id, the number of cities it has been to, and be ordered by decreasing number of cities, then by increasing bike id.
We define a bike as having “been to a city” whenever a trip involving the bike records either the start station or end station as being in that city.
Your write-up file should show only the first twenty tuples of the result set, but also state the total number of tuples in that result set (no need to show me the query that produces the latter number).

```
SELECT bike_id, places FROM(
SELECT bike_id, COUNT(DISTINCT city) AS places
FROM(
```

```
        SELECT bike_id, city FROM
        (trip LEFT JOIN station ON trip.start_station_id = station.station_id)
        UNION ALL
        SELECT bike_id, city FROM
        (trip LEFT JOIN station ON trip.end_station_id = station.station_id))
GROUP BY bike_id
)
WHERE places >1
ORDER BY places DESC, bike_id ASC
LIMIT 20
```

15|5
25|5
27|5
31|5
43|5
51|5
56|5
59|5
64|5
69|5
76|5
90|5
94|5
116|5
119|5
123|5
136|5
137|5
158|5
164|5

346

```
sqlite> SELECT bike_id, places FROM(
   ...> SELECT bike_id, COUNT(DISTINCT city) AS places
   ...> FROM(
   ...> SELECT bike_id, city FROM
   ...> (trip LEFT JOIN station ON trip.start_station_id = station.station_id)
   ...> UNION ALL
   ...> SELECT bike_id, city FROM
   ...> (trip LEFT JOIN station ON trip.end_station_id = station.station_id))
   ...> GROUP BY bike_id
   ...> )
   ...> WHERE places >1
   ...> ORDER BY places DESC, bike_id ASC
   ...> LIMIT 20;
bike_id|places
15|5
25|5
27|5
31|5
43|5
51|5
56|5
59|5
64|5
69|5
76|5
90|5
94|5
116|5
119|5
123|5
136|5
137|5
158|5
164|5
```

```
   ...> ORDER B
   ...> LIMIT 2
COUNT(bike_id)
346
```