From Csv to Query
Uber

Sérgio Costa

June 25, 2024

Contents

1 Query 1 - User **2**
1.1 Algorithm 3

2 Query 2 - Top drivers score **4**
2.1 Algorithm 5

3 Query 3 - Top users distance **6**
3.1 Algorithm 7

4 Query 4 - City average price **8**
4.1 Algorithm 8

5 Query 5 - Between dates average price **10**
5.1 Algorithm 10

6 Query 6 - City between dates average distance **12**
6.1 Algorithm 12

1 Query 1 - User

The goal is to get some user information about his score, money spent and also the total amount of rides.

Prompt
1 <username>
Output
name;gender;age;average_score;total_rides;total_spent

The users catalog shall be reached by the user username.
a) Array ordered by user usernames;
b) HashTable with hash by user usernames [1].

Catalog	**info.**	**func.**
users.csv	username	Primary key
users.csv	name	Get
users.csv	gender	Get
users.csv	birth_date	Calculate

After that, name and gender will come directly. The age is given facing the current date.

The rides catalog shall be ordered by user username.
a) Array ordered by user username.

Catalog	**info.**	**func.**
rides.csv	username	Get All
rides.csv	score_user	Sum
rides.csv	tip	Sum
rides.csv	distance	Get
rides.csv	driver_id	Foreign key

Per each entry we shall accumulate his score and tip. Given the driver of the ride we shall search his car class and multiply the tax of it by the distance of the ride.

The drivers catalog shall be reached by the driver id.
a) Array ordered by driver ids;
b) HashTable with hash by driver ids [2].

[1] Note that only exists one username per user.
[2] Note that only exists one id per driver.

Catalog	**info.**	**func.**
drivers.csv	id	Primary key
drivers.csv	car_class	Convert & Sum

Once founded, the driver's car class shall be taken in consideration for the total amount spent on that particular ride; summit to the current total amount.

1.1 Algorithm

```
// Gather the sampels
rides_slice = get_user_rides ( rides,
                                username );

// Throughout the samples
for ( int i=0; i < toal ( rides_slice ); i++ )
{
    // Car class
    driver = get_driver (rides_slice[i].driver_id);
    // Car tax
    tax = get_tax ( driver.car_class )

    // Total
    total += ( tax * rides_slice[i].distance )
                        +
                rides_slice[i].tip;

    // Score
    score += rides_slice[i].score_user;
}

score /= toal ( rides_slice );
```

2 Query 2 - Top drivers score

The goal is to list a rank of drivers by them average score.

In case of tie the top shall dispose the most recent ride first. If tie occurs, shall be ordered by driver id.

Prompt
2 <N>
Output
id;name;average_score

To answer the query we shall address the following structure.

```
structure query_2
{
    long driver_id;

    float score;
    char *date;

    char *name;
};
```

For general purposes the structure shall fit in an array with much space as the total of drivers in the system catalog.

```
Array *rank_query_2;
```

The rides catalog shall be accessed by their driver id.

Catalog	**info.**	**func.**
rides.csv	driver_id	Get All
rides.csv	score_driver	Sum
rides.csv	date	Get

For each entry we shall sum driver's score and accumulate it. If the date is recent from the one that's in use, update it.

The drivers catalog shall be reached by the driver id.

Catalog	**info.**	**func.**
drivers.csv	id	Primary key
drivers.csv	name	Get

2.1 Algorithm

```
// Gather the ids
drivers_id = drivers_catalog_get_all_ids ( catalog_drivers );

// For each id...
for (int i=0; i < total ( drivers_id ); i++)
{
    // Alloc a new answer
    struct query_2 *q;
    q = (struct query_2 *)malloc ( sizeof (struct query_2) );


    // Get driver's name
    q->name = strdup ( drivers_catalog_get( drivers_id [i] ).name );


    // Get all driver's rides
    slice_rides_driver_id = rides_driver_id_get_all ( drivers_id [i] );

    // Per each ride from the driver...
    for (int j=0; j < total ( slice_rides_driver_id ); j++)
    {
        // Sum score
        q->score += slice_rides_driver_id[j].score;

        // Update the most recent date
        if ( strcmp ( slice_rides_driver_id[j].date, q->date ) < 0 )
            q->date = slice_rides_driver_id[j].date;
    }
    // Average score
    q->score /= total ( slice_rides_driver_id );


    // Add answer to the rank
    add_array ( rank_query_2, q );
}

// Build the rank
sort ( rank_query_2, comparator_query_2 );

// Answear
for (int i=0; i < total_N_rank_desired; i++)
    ... rank_query_2 [i] ...
```

3 Query 3 - Top users distance

The goal is to list a rank of users by travelled distance.

In case of tie the top shall dispose the most recent ride first. If tie occurs, shall be ordered by users username.

Prompt
3 <N>
Output
username;name;total_distance
username;name;total_distance
...

To answer the query we shall address the following structure.

```
structure query_3
{
    char *username;

    float distance;
    char *date;

    char *name;
};
```

For general purposes the structure shall fit in an array with much space as the total of users in the system catalog.

```
Array *rank_query_3;
```

The rides catalog shall be accessed by their users usernames.

Catalog	**info.**	**func.**
rides.csv	user	Get All
rides.csv	distance	Sum
rides.csv	date	Get

For each entry we shall sum user's travelled distance and accumulate it. If the date is recent from the one that's in use, update it.

The users catalog shall be reached by the users username.

Catalog	**info.**	**func.**
users.csv	username	Primary key
users.csv	name	Get

3.1 Algorithm

```
// Gather the usernames
users_usernames = users_catalog_get_all_usernames ( catalog_users );

// For each username...
for (int i=0; i < total ( users_usernames ); i++)
{
    // Alloc a new answer
    struct query_3 *q;
    q = (struct query_3 *)malloc ( sizeof (struct query_3) );


    // Get user's name
    q->name = strdup ( users_catalog_get( users_usernames [i] ).name );

    // Get all user's rides
    slice_rides_user = rides_user_get_all ( users_usernames [i] );

    // Per each ride from the user...
    for (int j=0; j < total ( slice_rides_user ); j++)
    {
        // Sum distance
        q->distance += slice_rides_user[j].distance;

        // Update the most recent date
        if ( strcmp ( slice_rides_user[j].date, q->date ) < 0 )
            q->date = slice_rides_user[j].date;
    }

    // Add answer to the rank
    add_array ( rank_query_3, q );
}

// Build the rank
sort ( rank_query_3, comparator_query_3 );

// Answear
for (int i=0; i < total_N_rank_desired; i++)
    ... rank_query_3 [i] ...
```

4 Query 4 - City average price

The goal is to calculate the city average ride price based on the type of the car class.

Prompt
4 <city>
Output
average_price

The rides catalog shall be ordered by city.

Catalog	info.	func.
rides.csv	city	Get All
rides.csv	distance	Get
rides.csv	driver_id	Get

For each ride we shall get the intended car class from the driver's id and calculate the price of the ride.

The drivers catalog shall be ordered by drivers id.

Catalog	info.	func.
drivers.csv	driver_id	Primary key
drivers.csv	car_class	Convert & Calculate

4.1 Algorithm

```
// The answer
float price = 0.0f;

// Gather the samples
slice_city = rides_catalog_city_get_all ( city, catalog_rides );

// For each ride...
for (int i=0; i < total ( slice_city ); i++)
{
    // Get driver
    struct driver *driver;
    driver = drivers_catalog_id_get ( slice_city[i].driver_id );

    // Get car class fee
    float fee;
    fee = tax_fee ( driver->car_class );

    // Sum price
    price += fee * slice_city[i].distance;
}

// Average price, the answer
price /= total ( slice_city );
```

5 Query 5 - Between dates average price

The goal is to calculate the average ride price between 2 dates.

Prompt
5 <date A> <date B>
Output
average_price

The rides catalog shall be ordered by date.

Catalog	**info.**	**func.**
rides.csv	date	Get All - [Date A, Date B[
rides.csv	distance	Get
rides.csv	driver_id	Get

For each ride we shall get the intended car class from the driver's id and calculate the price of the ride.

The drivers catalog shall be ordered by drivers id.

Catalog	**info.**	**func.**
drivers.csv	driver_id	Primary key
drivers.csv	car_class	Convert & Calculate

5.1 Algorithm

```
// The answer
float price = 0.0f;

// Gather the samples
slice_dates = rides_catalog_between_date_get_all ( date_A,
                                                    date_B,
                                                    catalog_rides );

// For each ride...
for (int i=0; i < total ( slice_dates ); i++)
{
    // Get driver
    struct driver *driver;
    driver = drivers_catalog_id_get ( slice_dates[i].driver_id );

    // Get car class fee
    float fee;
    fee = tax_fee ( driver->car_class );

    // Sum price
    price += fee * slice_dates[i].distance;
}

// Average price, the answer
price /= total ( slice_city );
```

6 Query 6 - City between dates average distance

The goal is to calculate the average distance per ride in a city between 2 dates.

Prompt
5 <city> <date A> <date B>
Output
average_distance

A) The rides catalog shall be ordered by city;
i. After taking the city samples, we shall order it by date and sample it again.
B) The rides catalog shall be ordered by city (and by date inside the city order).

Catalog	**info.**	**func.**
rides.csv	city	Get All
rides.csv	date	Get All between [Date A, Date B[
rides.csv	distance	Get

For each ride we shall accumulate the distance of the ride.

6.1 Algorithm

```
// The answer
float distance = 0.0f;

// Gather the samples
slice_city_between = rides_catalog_city_between_date_get_all ( city,
                                                                date_A,
                                                                date_B,
                                                                catalog_rides );

// For each ride...
for (int i=0; i < total ( slice_city_between ); i++)
{
    // Sum distance
    distance += slice_city_between[i].distance;
}

// Average distance, the answer
distance /= total ( slice_city_between );
```