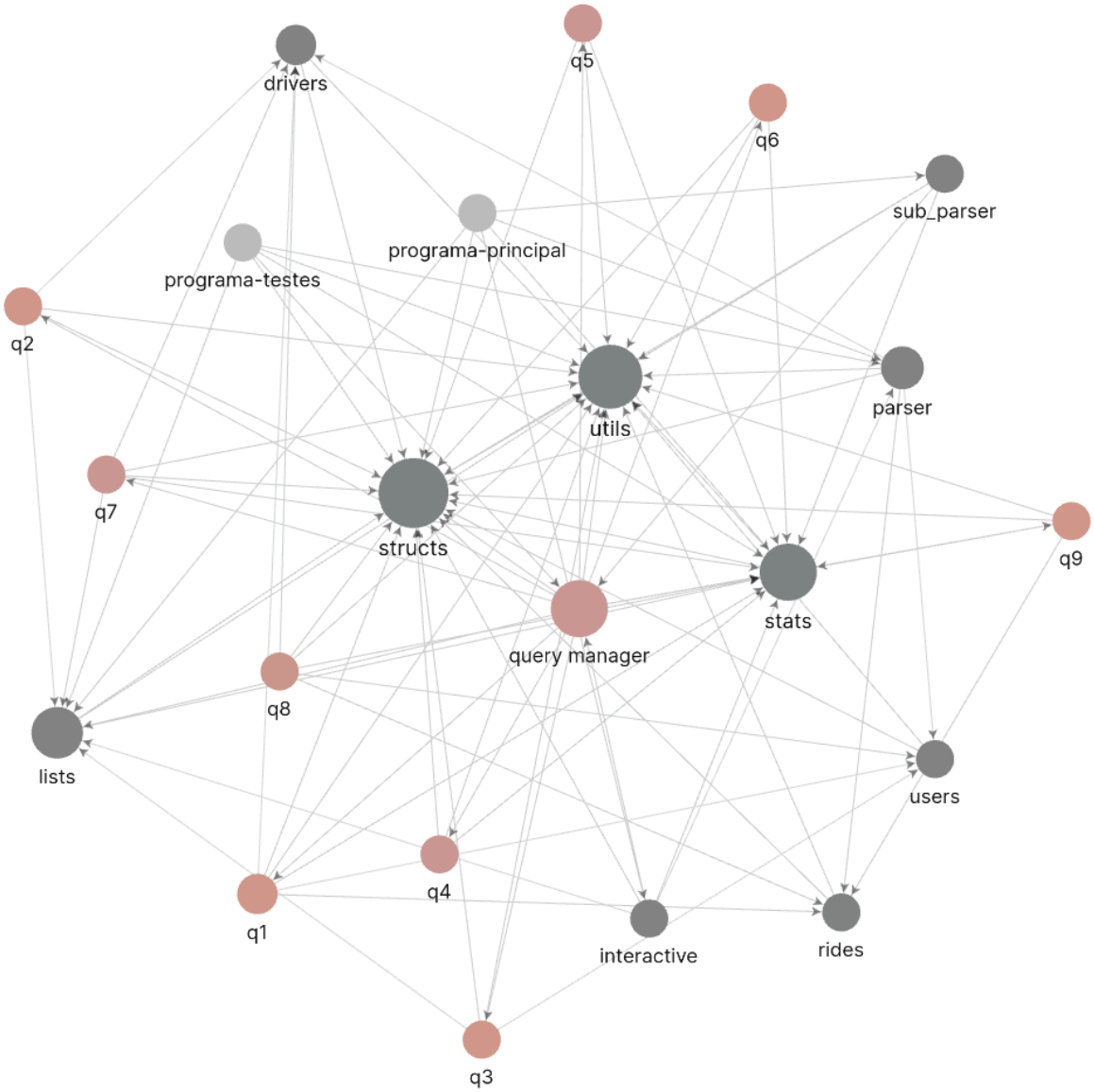
Relatório - 2ª Fase

Fevereiro, 2023.
Grupo 47:

- a100549 - Luís Carlos Fragoso Figueiredo - luiscff;
- a100651 - Miguel Dias Santa Marinha - MiguelMarinha404;
- a100706 - Rodrigo Miguel Eiras Monteiro - rodrigo72.

Arquitetura da aplicação

A maior mudança feita à arquitetura da aplicação consiste na separação do módulo `structs` em três novos módulos: `rides`, `users`, e `rides`, ficando apenas as funções relativas a `HASH` no módulo `structs`. Para além disso, foi adicionado um módulo de testes, um modo interativo, e existem mais queries funcionais.



Abordagens na redução de memória

Em comparação com o *dataset* da 1ª Fase ($\approx 100\,MB$), o *dataset* da 2ª Fase ($\approx 1.5\,GB$) exige uma melhor utilização de memória.
Portanto, decidimos simplificar certos elementos das estruturas usadas ao longo do código.

Nas estruturas *users* e *drivers*, *account_status* passou a ser apenas um `char` : `a` (active) ou `i` (inactive). `char *account_status` -> `char account_stats`

O mesmo foi feito para *car_class* e *pay_method*:
`char *car_class*` -> `char car_class` (`b`, `g`, ou `p`), etc.

O `score_user` e `score_driver` passaram a ser `unsigned short` em vez de `int`.

Para além disso, em vez de as datas serem guardadas como *strings*/ `char *`, decidimos guardá-las como *ints*.
Abordamos esta otimização de duas maneiras diferentes:

Datas como `unsigned short`

As datas são guardadas como `unsigned short` (ou `unsigned short int`), ocupando $2\,bytes$, com uma amplitude de 0 a 65535. O número guardado representa a "distância" em dias entre a data e uma data fixa `09/10/2022`, sendo, por isso, a amplitude do `unsigned short` suficiente ($\frac{65535}{365} \approx 180\ anos$).

São necessárias então as funções: `days_to_date`, `date_to_days`, e `days_to_age`.

> Exemplo:

```
int days_to_age (int days) {
        int age = 0;
        int year = YEAR;
        while (days >= 365) days -= days(year-age) ? 366 : 365, age++;
        return age;
}
```

Datas como `int`

As datas são guardadas como `int`, ocupando $4\,bytes$. O número guardado representa a data da seguinte forma: `"01/01/2020" -> 01012020`.

São necessárias então as funções: `int_to_date`, `date_to_int`, `int_to_age` e `next_date_int` (que calcula o próximo dia com um int como input, sendo útil para *queries* que envolvem datas).

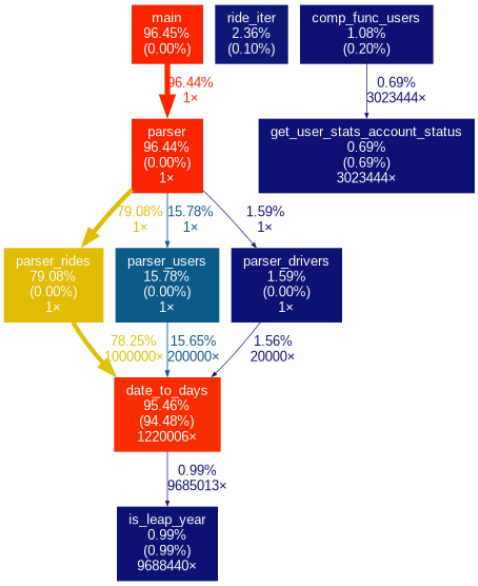
> Exemplo:

```
int date_to_int (char *date) {
        int d, m, y;
        sscanf (date, "%02d/%02d/%04d", &d, &m, &y);
        return (y*10000) + (m*100) + (d*1);
}
```

Abordagem escolhida: Datas como `int`

Escolhemos guardar as datas como `int`, uma vez que, apesar de ocupar mais memória (leve estimativa: $\approx +30\,MB$), é bastante mais rápida do que a abordagem anterior, devido às funções `int_to_date`, `date_to_int`, `int_to_age` possuirem menos `loops`, e serem, em geral, menos complexas do que as funções `days_to_date`, `date_to_days`, e `days_to_age`.

Performance: (Datas como `unsigned shorts`)



> **Note**
>
> Para além das otimizações que foram feitas, poder-se-ia também guardar os `id` como `int` em vez de `char *` / *string*.

Makefile

Modificamos o *Makefile* de modo a colocar todos os *object files* numa pasta própria, e adicionamos um comando que executa o `programa-testes`.

> Obtém o nome dos ficheiros que contêm *mains*

```
IGNORE_PRINCIPAL = $(OBJ_DIR)/$(PRINCIPAL).o
IGNORE_TESTES = $(OBJ_DIR)/$(TESTES).o
```

> Filtra o *object file* da *main* que não está a ser compilada

```
$(PRINCIPAL): $(filter-out $(IGNORE_TESTES), $(OBJ_FILES))
```

Programa-testes e Modo Interativo

Programa-teste

O módulo de testes consiste em duas funções: `compare_files` e `test_query_n`.
A `compare_files` compara os resultados obtidos com os ficheiros guardados que contêm o *output* correto. A `test_query_n` recebe o input, chama a função `compare_files` e mede o tempo de execução da query.

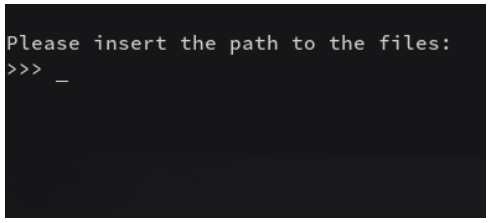
```
int compare_files (int size, int query);
void test_query_n (LIST *lists, STATS *stats, HASH *hash, int query, char **input, int input_len, int
repeat);
// ...
test_query_n(lists, stats, hash, 1, input_q1, 20, 2);
```

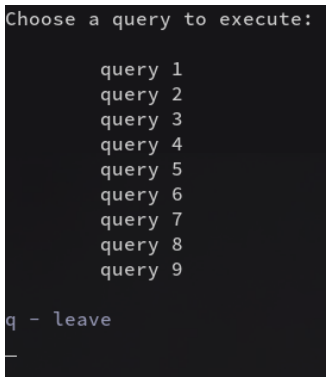
> Exemplo:

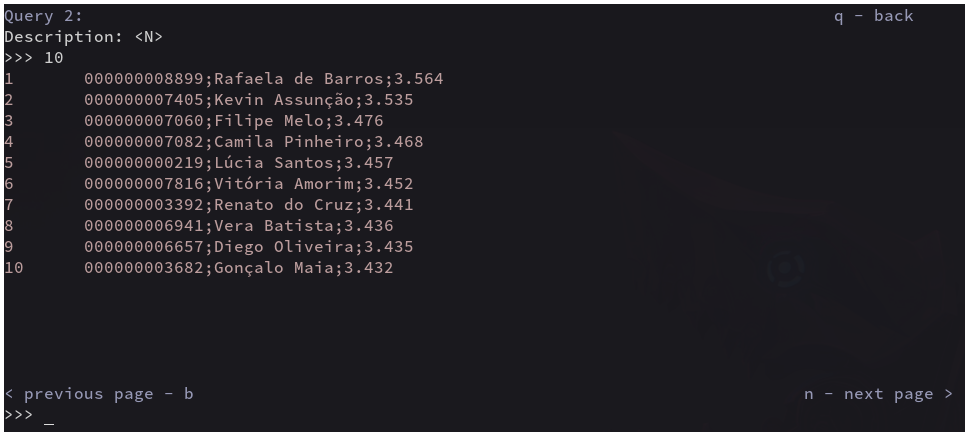
```
[              Loading time: 2.228070 seg.            ]

Média de 40 testes da query 1: 0.000050 seg
A query 1 passou em todos os testes.
```

Modo interativo

> Demonstração:







Queries

> (Queries não discutidas no relatório da 1ª Fase.)

Query 4

> Preço médio das viagens (sem considerar gorjetas) numa determinada cidade.

A query 4 acede a uma *hash table* de estatísticas de cidades:

```
g_hash_table_lookup(get_cities_stats_hash(stats), line);
```

E, a partir dos dados da estrutura, `preco_total` e do `numero_de_viagens`, calcula-se o preço médio.

> `stats.c`

```
typedef struct ct_st {
        int numero_viagens;
        int distancia_viajada;
        double preco_total;

        void *driver_stats_ht;
} CITY_STATS;
```

Antes da execução das queries, as estruturas são atualizadas quando se percorre o *array* de *rides* (`void ride_iter (void *key, void *value, void *data)`).

Query 5

> Preço médio das viagens (sem considerar gorjetas) num dado intervalo de tempo.

Estatísticas de cada data são guardadas numa *hash table*.
As *keys* utilizadas são *ints* (é utilizado o método discutido em Datas como int):
`g_hash_table_new_full(g_direct_hash, g_direct_equal, NULL, destroy_date_stats)`

Num *loop*, a query 5 acede à estrutura de estatística da data e incrementa a data até ser maior ou igual à data mais recente

> q5.c

```
while (d_inf <= d_sup) {
    DATE_STATS *date_stats = g_hash_table_lookup(get_dates_stats_hash(stats), GINT_TO_POINTER(d_inf));
    if (date_stats != NULL) {
        total_de_viagens += get_date_stats_numero_viagens(date_stats);
        total_gasto += get_date_stats_preco_total(date_stats);
    }

    d_inf = next_date_int(d_inf);
}
```

Query 6

> Distância média percorrida numa determinada cidade num dado intervalo de tempo.

Esta query, tal como a query 5, acede a várias estruturas `DATE_STATS` durante o *loop* que percorre o intervalo de datas.

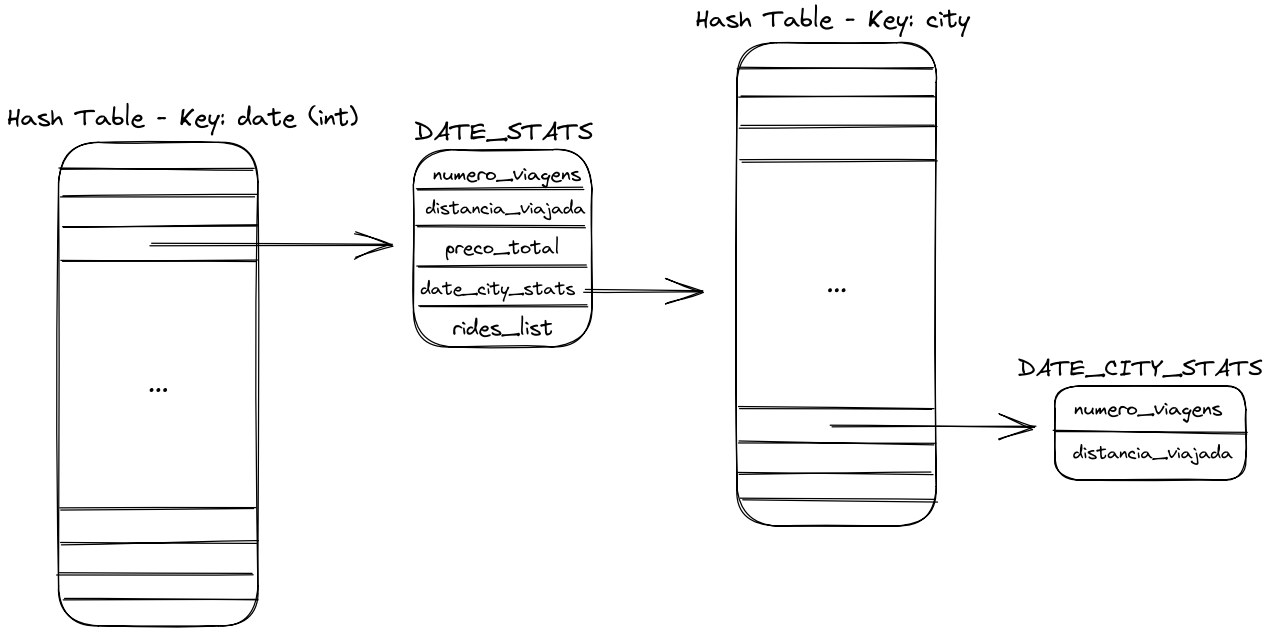
```
typedef struct dt_st {
        int numero_viagens;
        int distancia_viajada;
        double preco_total;
```

```
        void *date_city_stats;
        void *rides_list;
} DATE_STATS;
```

Com essa estrutura, acede à informação da cidade nessa data através da *hash table* `date_city_stats`.

```
typedef struct dt_ct_st {
        int numero_viagens;
        int distancia_viajada;
} DATE_CITY_STATS;
```

Assim, com a distância total, e com o número total de viagens, obtém-se a distância média.

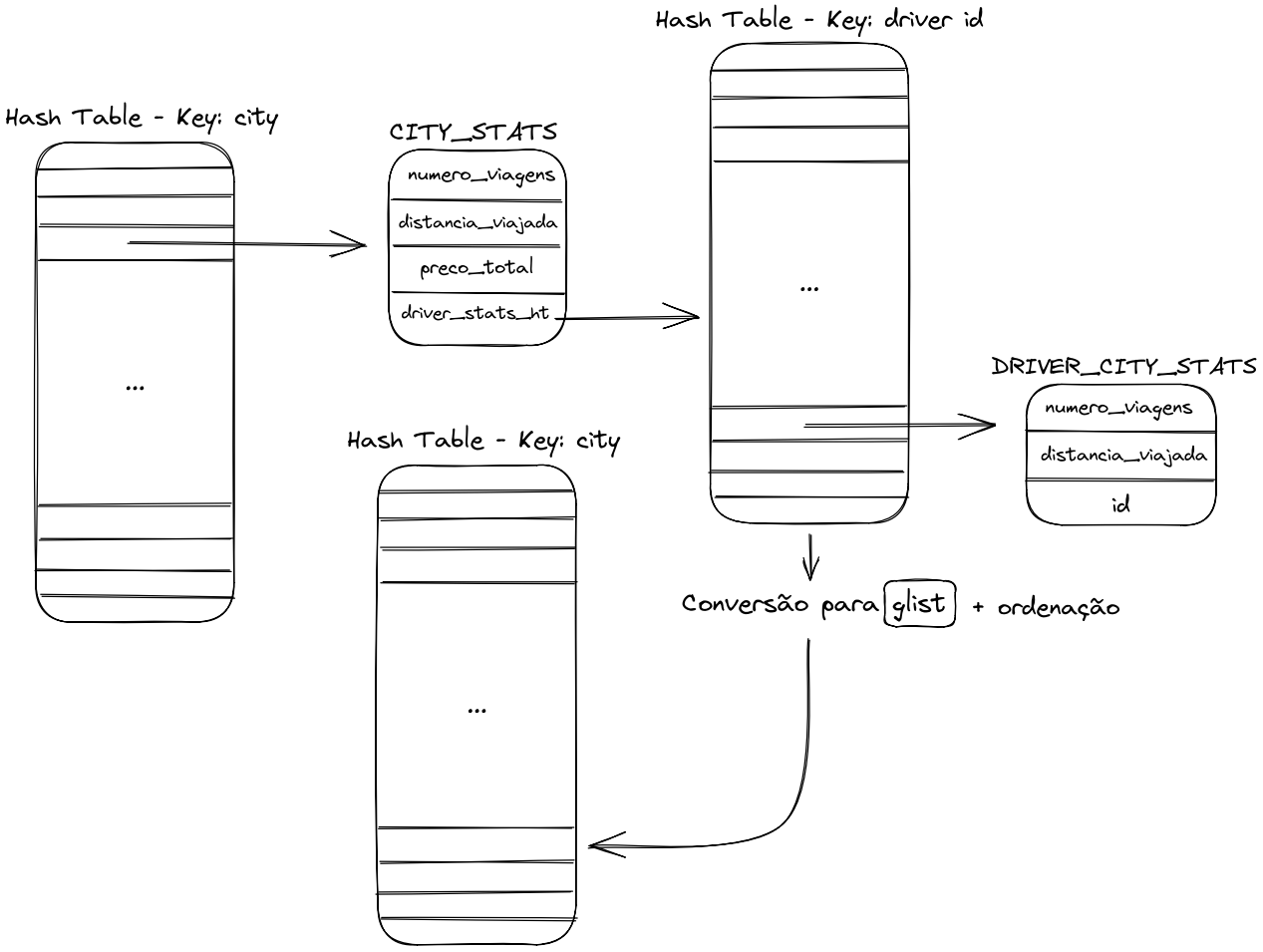


Query 7

> Top N condutores numa determinada cidade ordenado pela avaliação média do condutor.

Primeiro, a query 7 verifica se já existe uma lista ordenada com os condutores da dada cidade no módulo de listas. Se existe, então percorre essa lista. Se não existe, transforma a *hash table* de `driver_city_stats` numa lista, ordena-a, adiciona essa lista na *hash table* do módulo de listas, e depois percorre-a devolvendo os N primeiros elementos.

```
GList *list = g_hash_table_lookup(get_city_driver_stats_lists_ht(lists), city);
    if (list == NULL) {
        CITY_STATS *city_stats = g_hash_table_lookup(get_cities_stats_hash(stats), city);
        if (city_stats != NULL) {
            void *drivers_ht = get_city_stats_driver_stats_ht(city_stats);
            if (drivers_ht != NULL) {
                list = g_hash_table_get_values(drivers_ht);
                list = g_list_sort(list, compare_drivers_av);
                char *city_dup = strdup(city);
                                g_hash_table_insert(get_city_driver_stats_lists_ht(lists), city_dup,
(void *) list);
            }
        }
    }
```



Query 8

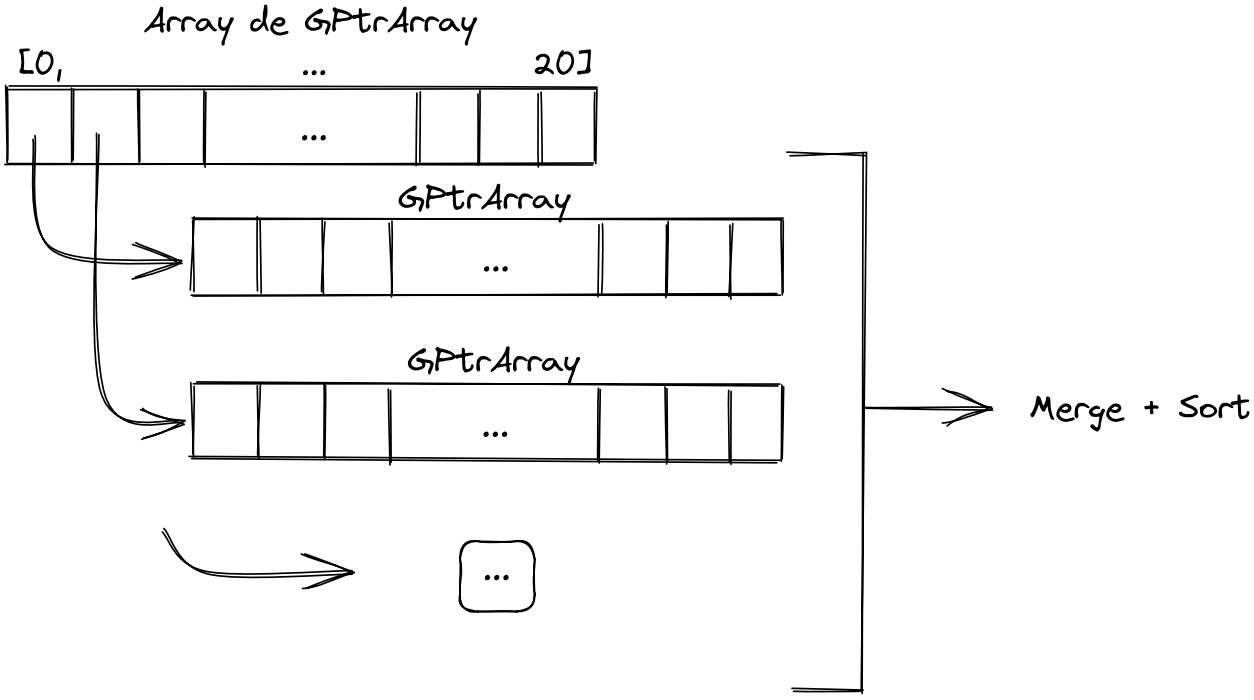
> Listar todas as viagens nas quais o utilizador e o condutor são do género passado como parâmetro, e têm perfis com X ou mais anos.

Primeiramente, pensamos em criar um *g_ptr_array* para cada género, adicionar elementos percorrendo o *array* de *rides* e, no final, ordenar as duas listas.
No entanto, esta abordagem pareceu tornar-se lenta para um *large dataset*.

Portanto, usamos um array de `GPtrArray *`, cada um contendo *rides* em que tanto o *driver* como o *user* têm uma conta com `X` ou mais anos, sendo `X` equivalente à posição do *array*:

```
if ((gender_driver == gender_user) && (d_account_status == 'a') && (u_account_status == 'a')) {

            int key = u_account_age > d_account_age ? d_account_age : u_account_age;
            if (gender_driver == 'F') {

                GPtrArray **main_arr = stats->f_account_status_ht;
                GPtrArray *arr = main_arr[key];

                if (arr == NULL) {
                    GPtrArray *new = g_ptr_array_new();
                    g_ptr_array_add(new, ride);
                    main_arr[key] = new;
                } else {
                    g_ptr_array_add(arr, ride);
                }
    // ...
```

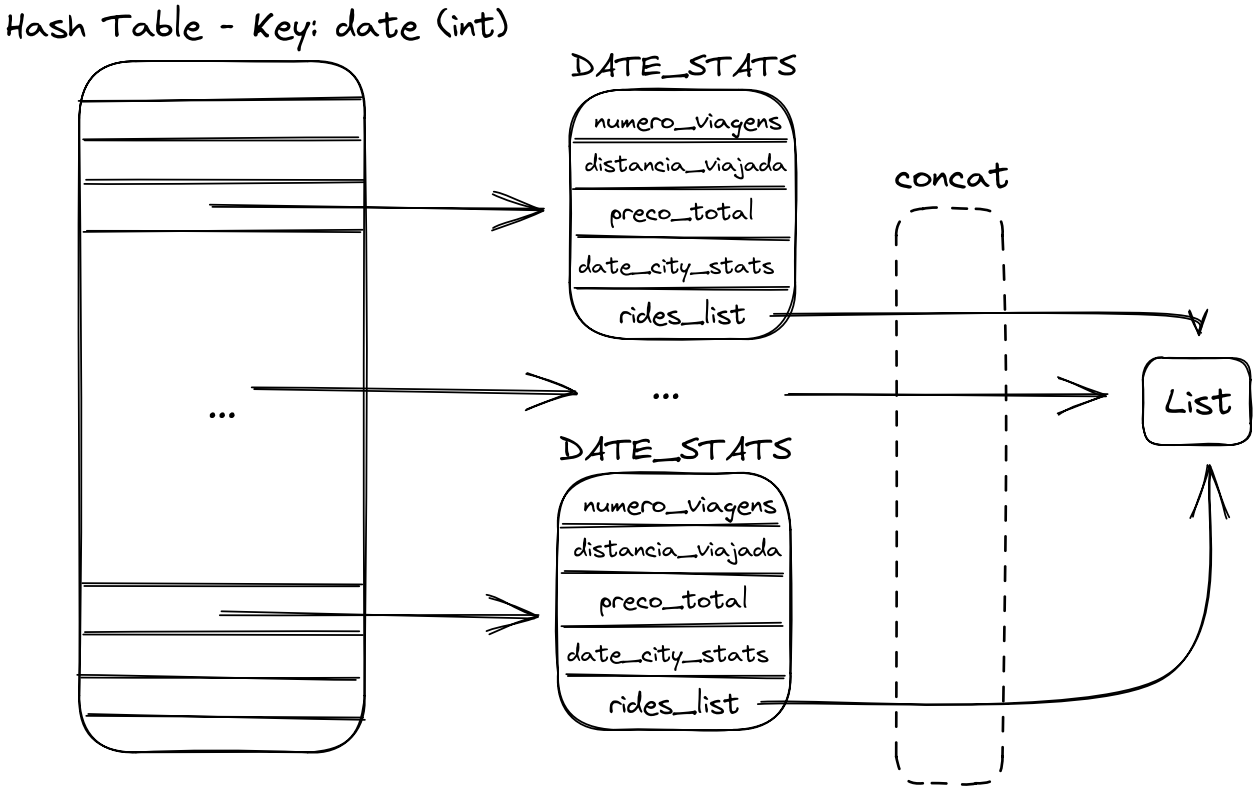
Depois, se necessário é feito *merge* de *arrays*, e é feita a ordenação.



Query 9

> Listar as viagens nas quais o passageiro deu gorjeta, num intervalo de tempo.

A query 9, ao percorrer um intervalo de datas, junta as listas de *rides* dessas datas numa única lista e ordena essa lista.



Performance

Características dos computadores usados.

- `PC-1`

```
OS: Arch Linux x86_64
Host: 82B1 Lenovo Legion 5 15ARH05H
Kernel: 6.1.9-arch1-1
CPU: AMD Ryzen 5 4600H with Radeon Graphics (12) @ 3.000GHz
GPU: AMD ATI 05:00.0 Renoir
GPU: NVIDIA GeForce RTX 2060 Mobile
Memory: 7310MiB
```

- `PC-2`

```
OS: Manjaro Linux x86_64
Host: TravelMate P215-53 V1.42
Kernel: 5.15.91-1-MANJARO
CPU: 11th Gen Intel i7-1165G7 (8) @ 4.700GHz
GPU: Intel TigerLake-LP GT2 [Iris Xe Graphics]
Memory: 15778MiB
```

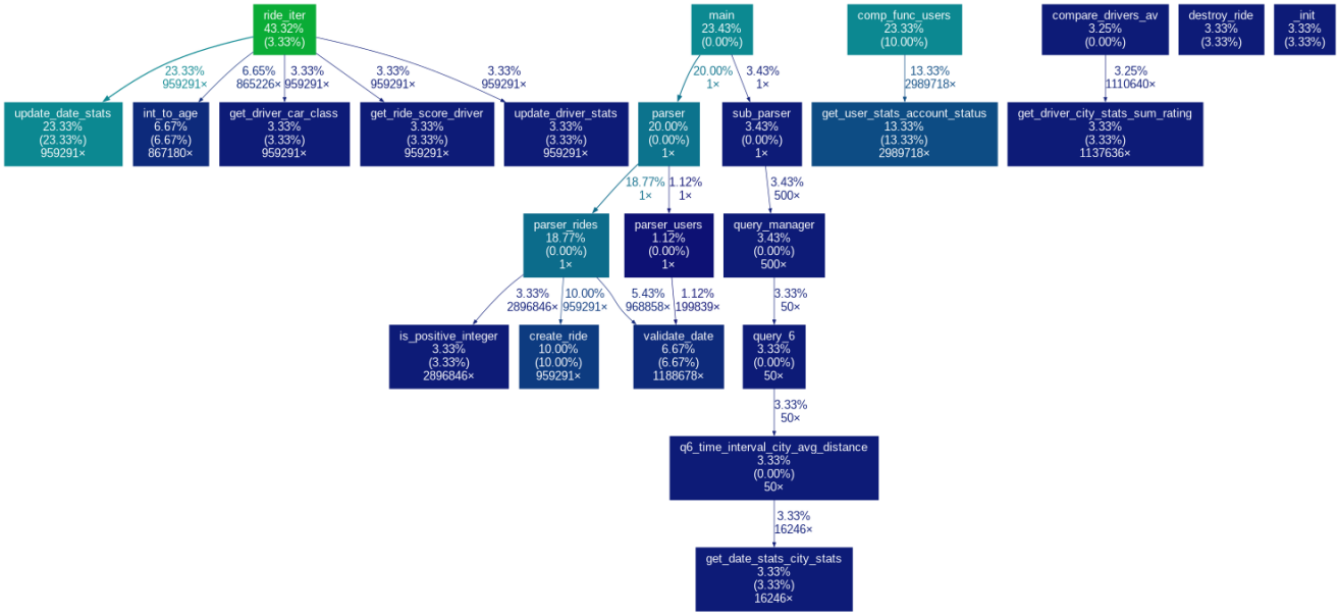
Regular Dataset

Execution Time (`PC-1`)

> Com `-O3`

```
real    0m2.562s
user    0m2.430s
sys         0m0.120s
```

Profiling (`PC-1`)



Memória utilizada (`PC-1`)

```
HEAP SUMMARY:
    in use at exit: 18,804 bytes in 9 blocks
  total heap usage: 9,784,757 allocs, 9,784,748 frees, 202,250,192 bytes allocated

LEAK SUMMARY:
   definitely lost: 0 bytes in 0 blocks
   indirectly lost: 0 bytes in 0 blocks
     possibly lost: 0 bytes in 0 blocks
   still reachable: 192 bytes in 3 blocks
        suppressed: 18,612 bytes in 6 blocks
```

Large Dataset

Tempo de Execução (`PC-1`)

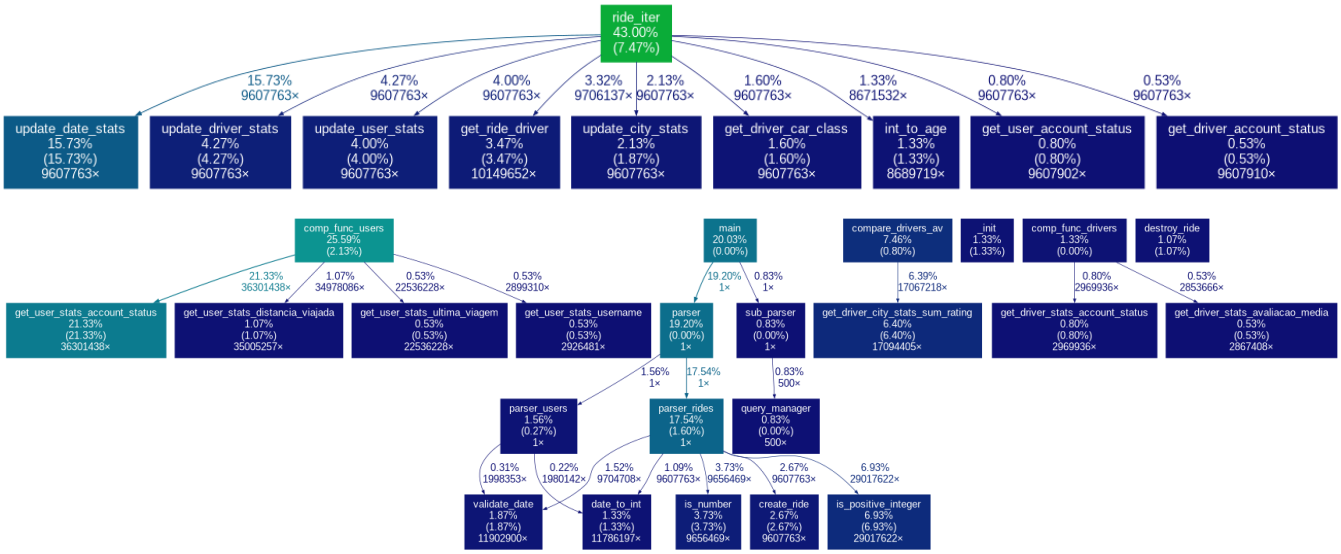
> Com `-O0`

```
real    0m38.173s
user    0m36.101s
sys         0m1.794s
```

> Com `-O3`

```
real    0m32.527s
user    0m31.057s
sys         0m1.307s
```

Profiling (`PC-1`)



Memória utilizada (`PC-2`)

```
HEAP SUMMARY:
    in use at exit: 18,804 bytes in 9 blocks
  total heap usage: 104,093,618 allocs, 104,093,609 frees, 2,344,589,864 bytes allocated
```

```
LEAK SUMMARY:
  definitely lost: 0 bytes in 0 blocks
  indirectly lost: 0 bytes in 0 blocks
    possibly lost: 0 bytes in 0 blocks
  still reachable: 192 bytes in 3 blocks
       suppressed: 18,612 bytes in 6 blocks
```

Observações Finais

Para concluir, achamos que podíamos ter melhorado as queries 1 e 2, no entanto não conseguimos descobrir qual era o problema na ordenação dos resultados.
Apesar disso, as restantes queries são funcionais, têm um bom tempo de execução, juntamente com uma boa gestão de memória, e, para além disso, implementamos conceitos de modularidade e encapsulamento ao longo do projeto.