Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2523 — Sistemas Distribuídos 2020-2

NOMBRE: Matías Duhalde
SECCIÓN: 1

# Informe Tarea 1

## 1 Ejecución del programa

Para ejecutar el programa, se puede ejecutar el código base con:

```
go run main.go N
```

O se puede correr el ejecutable compilado (tal como pide el enunciado) usando:

```
./program.go N
```

Si es necesario volver a compilar el ejecutable, esto se puede hacer con:

```
go build -o program
```

## 2 Aspectos generales

En el diseño de este programa, se implementaron funciones `Map`, `Shuffler`, y `Reduce`, que son comunes para todas las operaciones y tienen el mismo funcionamiento (con respecto a la manera que manejan los threads y la comunicación). Sin embargo, dentro de `Map` y `Reduce`, se llaman a las funciones/métodos `Mapper` y `Reducer` respectivamente, que dependen y son específicas de cada operación: `Select`, `Projection`, y `GroupAggregate`.

### 2.1 Map

Primero, se genera un **thread** que se encarga de ir recibiendo los resultados (tuplas) desde los `mappers` (mediante un `channel`). Luego, se separa la array de filas obtenidas desde el archivo en $n-1$ partes, donde $n$ corresponde a la **cantidad de threads**. Posteriormente se inician los **threads** encargados de realizar el **"mapeo"** con su subarray correspondiente. En otras palabras, los threads llaman a las funciones `mapper` paralelamente.

Como se especifica en la primera parte del enunciado, todos los threads deben terminar antes de empezar la parte de reduce, por lo que se usan `WaitGroups` para esperar, antes de continuar con la siguiente fase.

### 2.2 Reduce

Antes de empezar esta fase, se inicia un **thread** encargado de recibir el output de los `reducers` mediante un `channel`, muy similar a la fase anterior. Este thread también se encarga de **escribir** en el archivo de output. Posteriormente, se inicia un **thread** del `Shuffler`, el cual se encarga de enviar las tuplas al thread de `reducer` correspondiente (según la **key** de la tupla). Esto se hace enviando un `channel` a traves de un `channel`, por el que posteriormente se enviaran las tuplas.

En paralelo a lo anterior también se inician los **threads** que ejecutan la función/método `reducer` paralelamente, a medida que se reciben los `channel` de tuplas desde el `Shuffler`.

Cuando el `Shuffler` termina de enviar todas las tuplas, se cierran todos los canales creados, y se espera mediante un `WaitGroup` a que terminen todos los threads de `reducers` restantes antes de terminar la ejecución de la instrucción.

# 3 Select

## 3.1 Map

El `mapper` revisa para cada fila de su subarray asignada la condición del `Select`, la cual depende de los parámetros de la consulta entregada (una condición de ejemplo es `Region == Magallanes`). Si la cumple, se manda una tupla (`struct Tuple`) por el `channel`.

El **valor** de la tupla corresponde a una **estructura de datos representando la fila**, y la **key** corresponde a un string representando el **hash** único de la fila. Este hash se obtiene mediante la función criptográfica **SHA-1**, por lo tanto, si las filas son diferentes los hash deben ser distintos, y si son iguales, los hash deben ser iguales.

## 3.2 Reduce

Como se especifica en los recursos adjuntados al enunciado, no es necesario la fase `Reduce` en esta operación, por lo que el `reducer` en esta fase es **trivial**. Esta envía por el canal la misma tupla que recibe, sin realizar ningún cambio.

# 4 Projection

## 4.1 Map

El `mapper` se encarga de revisar cada fila asignada en su subarray, y mantener las columnas que se especifican en el input de la operación `Projection`, y eliminar aquellas que no. Se envía por el canal de output una **tupla** que contiene los campos pedidos.

Las tuplas en este caso son similares a las de `Select`. El **valor** de la tupla corresponde a una **estructura de datos representando la fila ya modificada (sólo con los campos especificados)**, y la **key** corresponde a un string representando el **hash** único de la fila modificada. Este hash también se obtiene mediante la función criptográfica **SHA-1**.

## 4.2 Reduce

Como se especifico en el principio, se genera una instancia de `reducer` para cada **key única**. Dado que luego de una operación de proyección las filas (y las keys) pueden repetirse, es trabajo del `reducer` evitar esto. Cada instancia de `reducer` puede recibir múltiples tuplas (iguales entre sí), pero este se encarga de enviar por el canal de output **sólo la primera que recibe**. Esta tupla la envía tal y como la recibió, sin modificarla. De esta manera se eliminan las filas duplicadas.

# 5 GroupAggregate

## 5.1 Map

El `mapper` revisa cada fila asignada en su subarray, y genera una tupla según los valores para `Group` y `Aggregate` que se hayan introducido, la cual posteriormente es enviada por el `channel` de output.

Las tuplas tienen como **key** al valor de la fila para la columna de `Group`, y como **valor** al valor de la fila para la columna de `Aggregate`. Por ejemplo, si la consulta es del tipo `GROUP Region AGGREGATE Fecha`, una tupla resultante puede ser: `(Metropolitana, 2020-07-27)`

## 5.2 Reduce

El `reducer` de esta operación es más complejo que las anteriores, debido a que depende de la función especificada en el input. En cada instancia de `reducer`, se reciben todas las **tuplas** con una misma **key**. Luego de haber recibido todas, se envía por el canal de output una tupla de la forma `(ColumnaGroup, Resultado)`. Se sabe que se recibieron todas las tuplas porque se cierra el canal por el cual se reciben. El

resultado se calcula progresivamente. En `MIN` se mantiene el menor valor recibido, en `MAX` se mantiene el mayor valor recibido, en `SUM` se van sumando los valores recibidos a un contador, y en `AVG` se hace lo mismo que en `SUM`, pero al final se divide por la cantidad de tuplas recibidas.

# 6 Consideraciones

- En la explicación anterior, se usan los términos "thread" y "goroutine" intercambiablemente.
- El output de la consulta se guarda por defecto en el archivo `out.csv`. Este se puede modificar cambiando el valor de la variable `outputPath` en el archivo `operations.go`
- En el enunciado, se dejó a criterio del estudiante qué hacer con las filas que tienen valores nulos. En el caso de mi implementación se optó por no incluirlas en las consultas.
- El programa se implementó en un loop, por lo tanto, se pueden ejecutar múltiples consultas dentro de una misma ejecución.
- El programa no permite encadenar consultas.
- El programa no aceptará consultas mal escritas. Si el parser detecta un error, le entregará feedback al usuario en la consola y permitirá que se ingrese la consulta nuevamente.
- Al ingresar las consultas, para el caso de las keywords y nombres de funciones (e.g. Select, Aggregate, AVG), el input es case-insensitive. Para el caso de las columnas, estas deben ser ingresadas tal como aparecen en el csv.
- La comparación entre fechas y strings funciona correctamente. Para el caso de strings, se sigue un orden alfabético.
- En `GroupAggregate`, si se ingresa como segunda columna una que contiene datos no-numéricos junto con la función `SUM` o `AVG`, el output no hará mucho sentido. Sin embargo, en issue se respondió que no era necesario manejar estos casos.