Consulta 1:

**select** *p*.nombre, *p*.apellido

**from** personas *p*

**join** cuentas *c* **on** *p*.id\_pers = *c*.id\_pers\_titular

**join** (**select** nro\_cuenta

**from** pasadas

**where** fecha\_hora > **now**() - ***interval*** **'1 month'**

**group** **by** nro\_cuenta

**having** **count**(\*) > 10) *p2* **on** *p2*.nro\_cuenta = *c*.nro\_cuenta

Planteo de la consulta segun costos por DBeaver

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

Nuestra consulta está planteada siguiendo reglas heurísticas. Se hace el select antes del join, dado que todas las restricciones (más de 10 pasadas en el último mes) esta solamente relacionado con la tabla pasadas, depende de la fecha y cantidad de pasadas (count) de esta forma se reduce la cantidad de tuplas con las que se va a tener que trabajar en los pasos intermedios.

No se puede optimizar haciendo selects con mas selectividad dado que solo se hace 1 select, dado que para el count, que se precisa previamente hacer el select.

Como se puede ver en planteo de DBeaver, esto reduce en gran manera la cantidad de filas comparándolo con el uso del select después del join, dado que en el segundo caso se precisa trabajar sobre un número mayor de tuplas.

**select** *p*.nombre, *p*.apellido

**from** personas *p*

**join** cuentas *c* **on** *p*.id\_pers = *c*.id\_pers\_titular

**join** pasadas *p2* **on** *p2*.nro\_cuenta = *c*.nro\_cuenta

**where** fecha\_hora > **now**() - ***interval*** **'1 month'**

**group** **by** *p*.nombre, *p*.apellido

**having** **count**(\*) > 10;

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Consulta 2:

**select** *p*.nombre, *p*.ruta

**from** peajes *p* **join** (**select** *p2*.id\_peaje

**from** pasadas *p2*

**where** fecha\_hora > **now**() - ***interval*** **'1 year'**

**group** **by** *p2*.id\_peaje

**order** **by** **count** (matricula || fecha\_hora) **DESC**

**limit** 1) **as** *mas\_pasadas* **on** *mas\_pasadas*.id\_peaje = *p*.id\_peaje;

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Empezando con esta consulta, otra vez se hace el select antes del join. Todas las pasadas (500) son en el ultimo año, así que no se restringe nada, pero se pone por el caso que se tenga una historia de pasadas mas antiguas, que se puede esperar una mayor selectividad.

Se ordenan en función de los peajes, reduciendo a 15 tuplas, ordenándolos por el count de matricula, fecha\_hora. De esta forma pudiendo contar la cantidad de pasadas por cada peaje. Después del ordenamiento se limita a 1 peaje, el de mayor cantidad de pasadas, que es el resultado buscado.

Se puede nuevamente ver la ventaja del select antes del join.

**SELECT** *p*.nombre, *p*.ruta

**FROM** peajes *p*

**JOIN** pasadas *p2* **ON** *p2*.id\_peaje = *p*.id\_peaje

**WHERE** *p2*.fecha\_hora > **now**() - ***interval*** **'1 year'**

**GROUP** **BY** *p*.nombre, *p*.ruta

**ORDER** **BY** **COUNT**(*p2*.matricula || *p2*.fecha\_hora) **desc**

**LIMIT** 1;

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

Consulta 3:

**select** *p*.nombre, *p*.apellido

**from** personas *p*

**join** cuentas *c* **on** *p*.id\_pers = *c*.id\_pers\_titular

**join** (**select** nro\_cuenta

**from** pasadas

**where** fecha\_hora > **now**() - ***interval*** **'1 month'** **and** pasadas.id\_peaje = 1

**group** **by** nro\_cuenta

**having** **count**(\*) > 10) *p2* **on** *p2*.nro\_cuenta = *c*.nro\_cuenta

**group** **by** *p*.id\_pers ;

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

Descripción generada automáticamente

Se hace una implementación similar al 1, añadiendo una condición más. En este caso hay un posible tema de selectividad, al trabajar con el where. Es conveniente primero hacer la comparación con id\_peaje, que reduce a 1/15 la cantidad de tuplas, mientras que la restricción por fechas reduce en aproximadamente un 1/5.

Nuevamente por comparación se puede ver la mejora por hacer el select primero.

**select** *p*.nombre, *p*.apellido

**from** personas *p*

**join** cuentas *c* **on** *p*.id\_pers = *c*.id\_pers\_titular

**join** pasadas *p2* **on** *p2*.nro\_cuenta = *c*.nro\_cuenta

**where** fecha\_hora > **now**() - ***interval*** **'1 month'** **and** p2.id\_peaje = 1

**group** **by** *p*.nombre, *p*.apellido

**having** **count**(\*) > 10

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Consulta 4:

**SELECT** *p*.nombre, *p*.telefono, **COALESCE**(**SUM**(*p2*.*importe*), 0) **AS** *total\_importe*

**FROM** peajes *p* **JOIN** (**SELECT** id\_peaje, **SUM**(importe) **AS** *importe*

**FROM** pasadas

**GROUP** **BY** id\_peaje) *p2* **ON** *p*.id\_peaje = *p2*.id\_peaje

**GROUP** **BY** *p*.id\_peaje, *p*.nombre, *p*.telefono

**ORDER** **BY** **COALESCE**(**SUM**(*p2*.*importe*), 0) **DESC**

**limit** 1;

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

En esta consulta primero se reduce la tabla de pasadas a una mas pequeña donde tenemos lo que necesitamos, el importe total por peaje y el peaje, reduciendo en gran medida la cantidad de tuplas con las que se trabajan. Después se une con peajes para poder obtener la información necesaria, resultando en una tabla igual a la de peajes, pero con el total de importes añadido. Después de esto se agrupa y se ordena para finalmente limitarlo a 1.

Comparación con el select afuera del join.

**SELECT** *p*.nombre, *p*.telefono, **COALESCE**(**SUM**(*p2*.importe), 0) **AS** *total\_importe*

**FROM** peajes *p* **JOIN** pasadas *p2* **ON** *p*.id\_peaje = *p2*.id\_peaje

**GROUP** **BY** *p*.id\_peaje, *p*.nombre, *p*.telefono

**ORDER** **BY** **COALESCE**(**SUM**(*p2*.importe), 0) **DESC**

**limit** 1;

Tabla

Descripción generada automáticamente

Otra razón de esta implementación es la mayor eficiencia cuando se menciona explícitamente el join, haciendo la consulta más eficiente que si no se hace. Usaremos de ejemplo la consulta 4. Se puede ver que el costo es mas de 1000 veces mayor a la primera implementación, y se llega a tener que trabajar con 160000 filas, en vez de un máximo de 500.

**SELECT** *p*.nombre, *p*.telefono, **COALESCE**(**SUM**(*p2*.importe), 0) **AS** *total\_importe*

**FROM** peajes *p*, pasadas *p2*

**GROUP** **BY** *p*.id\_peaje, *p*.nombre, *p*.telefono

**ORDER** **BY** **COALESCE**(**SUM**(*p2*.importe), 0) **DESC**

**limit** 1;

Tabla

Descripción generada automáticamente

Además, se puede ver como trabaja de forma diferente en las distintas implementaciones. Haciendo un hash loop en la primera implementación, pudiéndose trabajar con los registros haciendo una pasada por cada tabla, e intentándolos emparejar. Mientras que trabaja con nested loops en las ultimas 2, teniendo que hacer una pasada por el segundo registro por cada tupla del primero.

Otra posible forma de optimización nacería con la creación de índices, para de esta forma hacer posible la concatenación de ordenación-mezcla. La misma puede resultar beneficiosa en, por ejemplo, joins entre las tablas persona y cuentas. Que pueden ocurrir en varias ocasiones, de existir una mayor cantidad de registros que los presentes en este ejemplo se podría ver una mayor velocidad, siendo optimizado comparado al método hash usado actualmente.