udp Escuela de Informática y Telecomunicaciones

Bases de Datos Avanzada

Actividad 4: Actualización de datos

Autores:
Nicolás Correa
Joaquín Fernández
David Pazán

Profesor:
Juan Ricardo Giadach

mayo 11 del 2021

Índice

1. Introducción:

En el presente informe se desarrollara la entrega numero 4 de la asignatura bases de datos avanzadas, donde tomando en cuanta el desarrollo de las actividades pasadas se procederá a la actualización de datos en las diferentes tablas, esto con la intención de captar tiempos de ejecución para un posterior análisis comparativo.

2. Desarrollo:

2.1. Herramientas y conceptos importantes

El ordenador que se utiliza para esta actividad es el mismo que se utilizo para las anteriores, siendo poseedor de las siguientes especificaciones.

- Acer Aspire E-15-575G-76P4
- Sistema Operativo: Linux, con distribución Elementary Os versión Hera
- Intel core i7-7500U 4 cpus.
- SSD KINGSTON SA400 con capacidad 480gb
- Velocidad para lectura de 550 MB/s y escritura 450 MB/s.

Dicho hardware se mantiene en el desarrollo de los trabajos con la intención de tener datos relacionados entre ellos, ya que el tiempo de las consultas medidas depende del equipo en el cual se ejecuta.

Para el desarrollo de la actividad se debe utilizar distintas herramientas, entre ellas los comandos proporcionadas por Postgres llamados BEGIN y ROLLBACK (comandos desarrollados en profundidad en el informe numero 3, eliminado de datos) los cuales dan la posibilidad de ejecutar actualizaciones dentro de transacciones sin realmente modificarlas, de esta manera realizar distintas ejecuciones de actualización en tablas idénticas.

Y no olvidar DML (declaraciones del lenguaje de manipulación de datos) y su utilidad en la actividad. Ahora se da la definición; DML o también llamado Lenguaje de manipulación de datos es un idioma proporcionado por los sistemas gestores de bases de datos, que permite a los usuarios de la misma llevar a cabo las tareas de consulta o modificación de datos contenidos en la bases de datos. Calificando así dos grandes grupos, primero, lenguajes de consulta procedimentales y en segundo lugar, lenguaje de consulta no procedimentales. El primero incluye instrucciones dadas por el usuario al sistema para que realice una serie de procedimientos u operaciones en la base de datos para calcular un resultado final ($INSERT$, $UPDATE$ y $DELETE$). El segundo describe la información deseada sin un procedimiento específico para obtener esa información.

2.2. Plan de ejecución

La actividad consta de los siguientes puntos:

1. Asegurarse de que las tablas PERSONAS1 y PERSONAS2 cumplan con la no existencia (Personas1) y existencia de índice (Personas2) respectivamente.

2. Actualizar los registros midiendo el tiempo, tanto para PERSONAS 1 como para PERSONAS 2 incrementar en 15 las edades con límite máximo 99, en caso de sobrepasar este valor, reemplazar por 00.

3. Analizar los tiempos obtenidos e indicar si hay alguna forma de optimizar este proceso.

4. Comparar los resultados con los tiempos obtenidos en la actividad 2 y 3.

Teniendo presente los puntos asociados a la actividad, se utilizan los comandos proporcionados por postgres, se revisa los estados de las tablas, a lo que índices concierne.

Luego, se generan las consultas que cumplan con los parámetros establecidos en el punto numero 2, además de medir el tiempo de las actualizaciones en las edades en este caso PERSONAS 1 y PERSONAS 2.

Por prevención se opta por realizar 2 acciones, previo a la ejecución de las querys:

1. Posicionar la consulta en un entorno ideal de transacción con los comandos BEGIN y ROLLBACK. Esto con la intención de simular la instrucción una vez ejecutada la consulta; Y de esta forma no afectar directamente a la bases de datos.

2. Reiniciar el ordenador luego de la aplicación de cada transacción. Debido a que el computador guarda información de esta en la memoria (data cache), lo cual puede interferir en los tiempos de ejecución de consultas posteriores, una de las formas que se tiene para evitar este contratiempo y no afectar el rendimiento de la consulta a la bases de datos es reiniciando el ordenador.

Posterior a lo ya mencionado se construye una tabla comparativa con los tiempos de las diferentes consultas, con la intención de señalar la mas óptima (ver Análisis generales, Cuadro3, página 9).

Por último, la información de tiempos y procesos recolectados en este informe; se comparan con las actividades pasadas 2 y 3 (ver Comparativa de operaciones, cuadro 4, página 10).

2.3. Actualización de datos

Para realizar la actualización de datos, primero se debe cumplir la no existencia y la existencia de índice en Personas 1 y Personas 2 respectivamente. Es posible revisar el estado de las tablas con el comando $\backslash d$, el cual proporciona la descripción de una tabla, siendo esta información compuesta por las columnas, sus tipos, el espacio de tabla (si no es el predeterminado) y cualquier atributo especial como puede ser una variable definida como NO NULO o valores predeterminados. Uno de los valores que proporciona es la existencia de índice para alguno de sus atributos.

```
tablas=# \d personas1
                    Tabla «public.personas1»
  Columna  |      Tipo      | Ordenamiento | Nulable | Por omisión
-----------+----------------+--------------+---------+-------------
 rut       | numeric(10,0)  |              |         |
 nombre    | character(50)  |              |         |
 edad      | numeric(2,0)   |              |         |
 direccion | character(100) |              |         |
```

Figura 1: Comprobación de la NO existencia de índice para Personas 1.

```
tablas=# \d personas2
                    Tabla «public.personas2»
  Columna  |      Tipo      | Ordenamiento | Nulable  | Por omisión
-----------+----------------+--------------+----------+-------------
 rut       | numeric(10,0)  |              | not null |
 nombre    | character(50)  |              |          |
 edad      | numeric(2,0)   |              |          |
 direccion | character(100) |              |          |
Índices:
    "personas2_pkey" PRIMARY KEY, btree (rut)
```

Figura 2: Comprobación de la existencia de llave primaria para Personas 2.

Como se puede apreciar en las figuras 1 y 2, solo Personas 2 posee un índice en cual pertenece al atributo rut.

2.3.1. Actualización en Personas 1

Query 1: Se evalúa la siguiente consulta encargada de actualizar los registros a partir de una suma que se efectúa respecto al atributo edad, como máximo puede sumar 99 y si pasa del límite reemplaza el valor por un 00.

```
1 begin;
2 update personas1 set edad = case when edad < 85 then edad + 15 else 00 end where
    rut in (select rut from ruts);
```

```
3 rollback;
```

Query 2: Se evalúa la siguiente consulta encargada de actualizar los registros a partir de una suma que se efectúa respecto al atributo edad, como máximo puede sumar 99 y si pasa del límite reemplaza el valor por un 00. Esta consulta llega a demorarse un tanto más, puesto a que verifica la existencia de la relación de un atributo común, en este caso rut en ruts y en PERSONAS.

```
1 begin;
2 update personas1 set edad = (case when personas1.edad < 85 then personas1.edad  +
      15 else 00 end) where exists (select from ruts where personas1.rut = ruts.
    rut);
3 rollback;
```

2.3.2. Actualización en Personas 2

Query 1: Se evalúa la siguiente consulta encargada de actualizar los registros a partir de una suma que se efectúa respecto al atributo edad, como máximo puede sumar 99 y si pasa del límite reemplaza el valor por un 00.

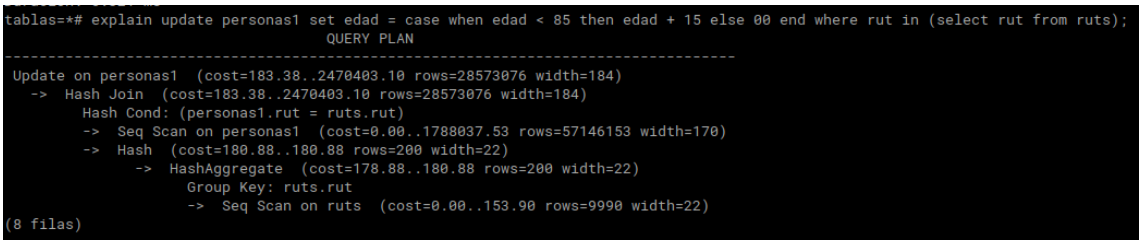
```
1 begin;
2 update personas2 set edad = case when edad < 85 then edad + 15 else 00 end where
    rut in (select rut from ruts);
3 rollback;
```

Query 2: Se evalúa la siguiente consulta encargada de actualizar los registros a partir de una suma que se efectúa respecto al atributo edad, como máximo puede sumar 99 y si pasa del límite reemplaza el valor por un 00. Esta consulta llega a demorarse un tanto más, puesto a que verifica la existencia de la relación de un atributo común, en este caso rut en ruts y en PERSONAS.

```
1 begin;
2 update personas2 set edad = (case when personas1.edad < 85 then personas2.edad  +
      15 else 00 end) where exists (select from ruts where personas2.rut = ruts.
    rut);
3 rollback;
```

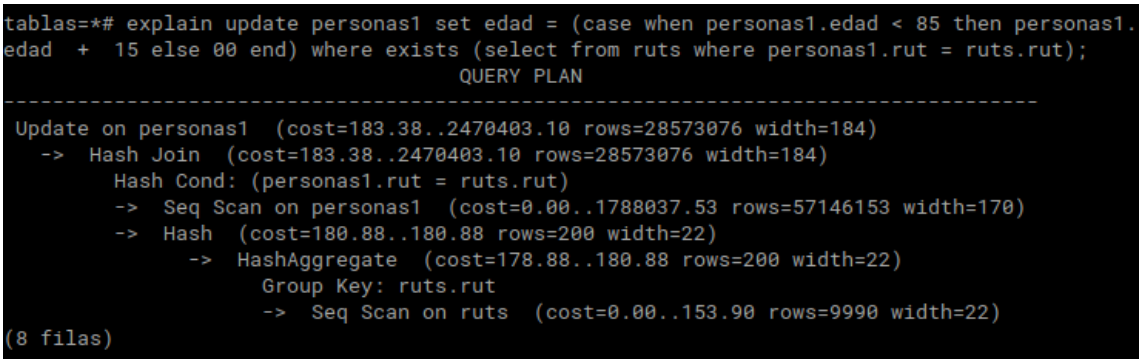
2.4. Plan de ejecución para la consulta utilizando EXPLAIN

A continuación se ilustraran los planes de ejecución resultantes de las consultas utilizadas, esto con la intención de recolectar mayor información para el análisis de resultados.



```
tablas=*# explain update personas1 set edad = case when edad < 85 then edad + 15 else 00 end where rut in (select rut from ruts);
                                QUERY PLAN
---------------------------------------------------------------------------
 Update on personas1  (cost=183.38..2470403.10 rows=28573076 width=184)
   ->  Hash Join  (cost=183.38..2470403.10 rows=28573076 width=184)
         Hash Cond: (personas1.rut = ruts.rut)
         ->  Seq Scan on personas1  (cost=0.00..1788037.53 rows=57146153 width=170)
         ->  Hash  (cost=180.88..180.88 rows=200 width=22)
               ->  HashAggregate  (cost=178.88..180.88 rows=200 width=22)
                     Group Key: ruts.rut
                     ->  Seq Scan on ruts  (cost=0.00..153.90 rows=9990 width=22)
(8 filas)
```

Figura 3: EXPLAIN a consulta numero 1 para Personas 1



```
tablas=*# explain update personas1 set edad = (case when personas1.edad < 85 then personas1.
edad  +  15 else 00 end) where exists (select from ruts where personas1.rut = ruts.rut);
                                  QUERY PLAN
---------------------------------------------------------------------------------
 Update on personas1  (cost=183.38..2470403.10 rows=28573076 width=184)
   ->  Hash Join  (cost=183.38..2470403.10 rows=28573076 width=184)
         Hash Cond: (personas1.rut = ruts.rut)
         ->  Seq Scan on personas1  (cost=0.00..1788037.53 rows=57146153 width=170)
         ->  Hash  (cost=180.88..180.88 rows=200 width=22)
               ->  HashAggregate  (cost=178.88..180.88 rows=200 width=22)
                     Group Key: ruts.rut
                     ->  Seq Scan on ruts  (cost=0.00..153.90 rows=9990 width=22)
(8 filas)
```

Figura 4: EXPLAIN a consulta numero 2 para Personas 1

Tanto Figura 3 y 4 corresponden a los planes de ejecución de las consultas realizadas en la tabla PERSONAS1, la cual no es poseedora de llave primaria. Esta tiene como apartado principal Seq Scan, la cual posee la funcionalidad de disminuir los datos manejados en pasos siguientes de la consulta, recordando que la tabla ruts tiene menor cantidad de filas que PERSONAS1. Posteriormente, el plan ejecuta HashAggregate, el cual se encarga de generar una tabla temporal de hash, alude al momento de realizar la consulta anidada, ordenando los registros, y comprobando que cumple lo solicitado para actualizar.

```
tablas=*# explain update personas2 set edad = case when edad < 85 then edad + 15 else 00 end where rut in (select rut from ruts);
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
 Update on personas2  (cost=179.44..188926.43 rows=24972752 width=184)
   ->  Nested Loop  (cost=179.44..188926.43 rows=24972752 width=184)
         ->  HashAggregate  (cost=178.88..180.88 rows=200 width=22)
               Group Key: ruts.rut
               ->  Seq Scan on ruts  (cost=0.00..153.90 rows=9990 width=22)
         ->  Index Scan using personas2_pkey on personas2  (cost=0.56..8.58 rows=1 width=170)
               Index Cond: (rut = ruts.rut)
(7 filas)
```

Figura 5: EXPLAIN a consulta numero 1 para Personas 2

```
                                    QUERY PLAN

-----------------------------------------------------------------------------------------
--
 Update on personas2  (cost=179.44..188926.43 rows=24972752 width=184)
   ->  Nested Loop  (cost=179.44..188926.43 rows=24972752 width=184)
         ->  HashAggregate  (cost=178.88..180.88 rows=200 width=22)
               Group Key: ruts.rut
               ->  Seq Scan on ruts  (cost=0.00..153.90 rows=9990 width=22)
         ->  Index Scan using personas2_pkey on personas2  (cost=0.56..8.58 rows=1 width=170
)
               Index Cond: (rut = ruts.rut)
(7 filas)
```

Figura 6: EXPLAIN a consulta numero 2 para Personas 2

Tanto Figura 5 como Figura 6 corresponden a los planes de ejecución de las consultas realizadas en la tabla PERSONAS 2. Se identifica que al poseer llave primara en rut, las consultas generan el apartado de HashAggregate en el plan de ejecución. Esta es una tabla de hash temporal que tiene la función de agrupar los registros de la tabla, y al ejecutarse la consulta, actualiza los registros a través de los índices de la tabla.

3. Resultados

A continuación se ilustraran las tablas de resultados con los valores asociados a cada consulta realizada, además de un promedio.

El tiempo asociado a las consultas se trabaja en milisegundos mientras que los promedios en horas, minutos y segundos según corresponda.

Query 1	Primer intento	Segundo intento	Tercer intento	Promedio
Pesonas 1	36162,208 [ms]	36518,264 [ms]	34762,746 [ms]	35,814 [s]
Pesonas 2	11158,690 [ms]	10118,745 [ms]	10018,561[ms]	10,432 [s]

Cuadro 1: Cuadro comparativo de tiempos, Query 1

Query 2	Primer intento	Segundo intento	Tercer intento	Promedio
Pesonas 1	38305,695 [ms]	52192,693 [ms]	55156,088 [ms]	48,551 [s]
Pesonas 2	10422,344[ms]	10010,123[ms]	10960,901[ms]	10,464 [s]

Cuadro 2: Cuadro comparativo de tiempos, Query 2

4. Análisis generales

A continuación se ilustrara un cuadro con los valores promediados de cada consulta tanto para la tabla de Personas 1 como Personas 2, esto con la intención de un posterior análisis comparativo.

	Query 1	Query 2
Pesonas 1	35,8144 [s]	48,551 [s]
Pesonas 2	10,432 [s]	10,464 [s]

Cuadro 3: Cuadro comparativo respecto a los promedios de las Query

El cuadro comparativo logra evidenciar por una parte los tiempos de las diferentes querys (1 y 2) para una tabla sin y con índices, como lo son Personas 1 y Personas 2 respectivamente. Es una constante la diferencia de tiempo, en Personas 1 la eliminación es mucho más lenta que en Personas 2, sin importar la query que se haya utilizado.

Esto es debido a que Personas1 tiene una búsqueda por bloques, mientras que personas2, puede generar búsquedas mucho más rápidas gracias al árbol creado a partir de su índice. Esta es la principal razón por la cual la actualización en Personas2 es más rápida, debido a que encuentra el dato que se actualiza mucho antes.

Ahora bien no se debe olvidar la acción de la operación UPDATE en la base de datos, ya que además de analizar los tiempos de búsqueda se necesita analizar las acciones complementarias que hace el comando ya mencionado. Debido a que el sistema no puede cambiar la edad de forma directa en la fila, puesto que si dentro de la transacción hubiera un rollback se necesita restablecer la tabla; Para cumplir esto, se hace una copia de la fila y se elimina de la tabla para luego insertar la fila modificada nuevamente. Por lo tanto se debe tener en cuenta que las inserciones y eliminaciones pueden afectar de alguna forma al rendimiento de la consulta a la bases de datos.

5. Comparativa de operaciones

Análisis comparativo a las operaciones asociados a las actividades 2, 3 y 4.

Teniendo en consideración que tanto el presente informe como el numero 3, fueron realizados con la comparativa de 3 querys distintas, el siguiente cuadro muestra los valores óptimos asociados a cada operación.

	SELECT	DELETE	UPDATE
Pesonas 1	151981[ms]	34505 [ms]	35814 [ms]
Pesonas 2	159[ms]	9323[ms]	10432[ms]

Cuadro 4: Tabla comparativa entre operaciones

Con estos valores se procede a un análisis comparativo de las sentencias $SELECT$, $DELETE$ y $UPDATE$ con una investigación del comportamiento de estos en la bases de datos.

Una forma de enfocar esta comparación es centrandose en la terminología DML o también conocido como lenguaje de modificación de datos; Donde existe una relación de las consultas $DELETE$, $UPDATE$ e $INSERT$. ya que estas van a actuar de forma distinta dependiendo de la existencia de índices. Esto se debe a que dichos comandos no afectan exclusivamente una tabla sino que a su vez afectan la tabla de índices, puesto que la base de datos debe de mantener la consistencia de la información. Si se realizan cambios en el sistema, estos deben de ser consistentes en toda la base de datos, de esta forma el comando $SELECT$ al no modificar la base de datos, no pertenece a los DML y por lo tanto no se ve afectada por el uso de índices.

El comando $SELECT$ funciona similar que su contraparte $DELETE$, con la diferencia de que el segundo aplica un borrado en la tabla seleccionada y se debe de encargar de mantener una consistencia en los datos, tanto en la tabla donde fue eliminado dicho dato, como en las tablas de índices en las cuales hubiese pertenecido.

El comando $UPDATE$, por otro lado, se puede interpretar como un comando $DELETE$, más un $INSERT$, encargándose primero de borrar el dato y luego de insertar el nuevo dato.

Los 3 comandos son DML, por lo tanto se ven afectados por el número de índices procedentes de la tabla que usa como llave para modificar así la fila.

Comandos de ejemplo del lenguaje de modificación de datos:

- INSERT: Sentencia SQL que agrega uno o más registros a una, y sólo una tabla.
- UPDATE: Sentencia SQL utilizada para modificar los valores de un conjunto de registros existentes en una tabla.
- DELETE: Sentencia SQL la cual borra uno o más registros existentes en una tabla.

Teniendo en cuenta el cuadro comparativo (ver cuadro 4, página 10), además de lo anteriormente descrito, el tiempo más prometedor o conveniente es el $SELECT$ pero al no ser DML no aplica. Pero para este caso particular, surge algo interesante lo cual es que el mismo comando hace el trabajo de un $INSERT$; y ocurre que esta última sentencia es la más demorosa y por ende menos óptima.

Este comando hace efecto en el peor de los casos, ingresos de 50 millones de datos llevando a una extensión n con el valor ya mencionado. Esto implica que el tiempo secuencial es igual a 50 millones de ms, debido al tiempo de escritura del SSD, el cual corresponde a 1 ms. Concluyendo así con un tiempo hipotético de 50 millones de ms.

De esta forma el tiempo más óptimo si se habla de DML como tal, es la sentencia $DELETE$ tras los resultados obtenidos para las distintas actividades.

6. Conclusión

De acuerdo con los datos proporcionados y demostrados, se puede apreciar que las consultas enfocadas a la actualización de datos en Personas 2, son siempre más óptimas en tiempo de ejecución que las enfocadas en actualizar a personas 1, la característica principal por lo que esto sucede es por la existencia de índices en personas 2. Para actualizar cualquier valor en la tabla, primero se debe buscar dicho valor, por lo que este índice proporciona una búsqueda más eficiente que la búsqueda por bloques presente en la actualización de Personas 1.

Además, tomando en cuenta los tiempos de ejecución vistos en las tablas (Cuadro 3), la query que proporciono los tiempos más eficientes fue la numero 1, con esto podemos observar que a pesar de que ambas consultas son de carácter anidado, la existencia del comando "where exists" complica el tiempo de ejecución para la query numero 2.

Uno de los puntos a resaltar es la comparativa a las distintas operaciones realizadas tanto en esta actividad como en la 2 y 3. Donde a través de la denominación DML, se plasmo un enfoque comparativo para las sentencias $DELETE$, $UPDATE$ Y $SELECT$, apoyando este ultimo en la sentencia $INSERT$. Dando como resultado que la sentencia $DELETE$ es más óptima en tiempos de ejecución para este caso particular.

Finalmente teniendo presente que los datos se actualizaron de forma correcta para todos los casos propuestos, se concluye con la realización exitosa de los objetivos de la actividad.

7. Bibliografía

- Comandos postgres, BEGIN y ROLLBACK
  https://www.geeksforgeeks.org/postgresql-rollback/

- Comandos postgres, para información de tablas
  https://www.todopostgresql.com/meta-commands-de-psql-mas-utilizados-en-postgresql/

- Comandos postgres, EXPLAIN
  https://www.postgresql.org/docs/9.1/sql-explain.html

8. Anexado

```
tablas=# begin;
BEGIN
Duración: 0,289 ms
tablas=*# update personas1 set edad = case when edad < 85 then edad + 15 else 00 end where rut in (select rut from ruts);
UPDATE 9999
Duración: 36162,208 ms (00:36,162)
tablas=*# rollback;
ROLLBACK
Duración: 1,202 ms
```

Figura 7: Ejecución consulta número 1 para Personas 1, primer intento

```
tablas=# begin;
BEGIN
Duración: 0,323 ms
tablas=*# update personas1 set edad = case when edad < 85 then edad + 15 else 00 end where rut in (select rut from ruts);
UPDATE 9999
Duración: 36518,264 ms (00:36,518)
tablas=*# rollback;
ROLLBACK
Duración: 0,532 ms
```

Figura 8: Ejecución consulta número 1 para Personas 1, segundo intento

```
tablas=# begin;
BEGIN
Duración: 0,326 ms
tablas=*# update personas1 set edad = case when edad < 85 then edad + 15 else 00 end where rut in (select rut from ruts);
UPDATE 9999
Duración: 34762,746 ms (00:34,763)
tablas=*# rollback;
ROLLBACK
Duración: 0,686 ms
```

Figura 9: Ejecución consulta número 1 para Personas 1, tercer intento

```
tablas=# begin;
BEGIN
Duración: 0,180 ms
tablas=*# update personas2 set edad = case when edad < 85 then edad + 15 else 00 end where rut in (select rut from ruts);
UPDATE 9999
Duración: 11158,690 ms (00:11,159)
tablas=*# rollback;
ROLLBACK
Duración: 0,608 ms
```

Figura 10: Ejecución consulta número 1 para Personas 2, primer intento

```
tablas=# begin;
BEGIN
Duración: 0.405 ms
tablas=*# update personas2 set edad = case when edad < 85 then edad + 15 else 00 end where rut in (select rut from ruts);
UPDATE 9999
Duración: 10118.745 ms (00:10.119)
tablas=*# rollback;
ROLLBACK
Duración: 1.323 ms
```

Figura 11: Ejecución consulta número 1 para Personas 2, segundo intento

```
tablas=# begin;
BEGIN
Duración: 0.286 ms
tablas=*# update personas2 set edad = case when edad < 85 then edad + 15 else 00 end where rut in (select rut from ruts);
UPDATE 9999
Duración: 10018.561 ms (00:10.019)
tablas=*# rollback;
ROLLBACK
Duración: 0.218 ms
```

Figura 12: Ejecución consulta número 1 para Personas 2, tercer intento

```
tablas=# begin;
BEGIN
Duración: 0,289 ms
tablas=*# update personas1 set edad = case when edad < 85 then edad + 15 else 00 end where rut in (select rut from ruts);
UPDATE 9999
Duración: 36162,208 ms (00:36,162)
tablas=*# rollback;
ROLLBACK
Duración: 1,202 ms
```

Figura 13: Ejemplo consulta SELECT, Actividad 1