INF3710- Bases de données

Groupe 02

Travail pratique #6

Index et déclencheurs

Présenté à Manel Grichi

David TREMBLAY 1748125

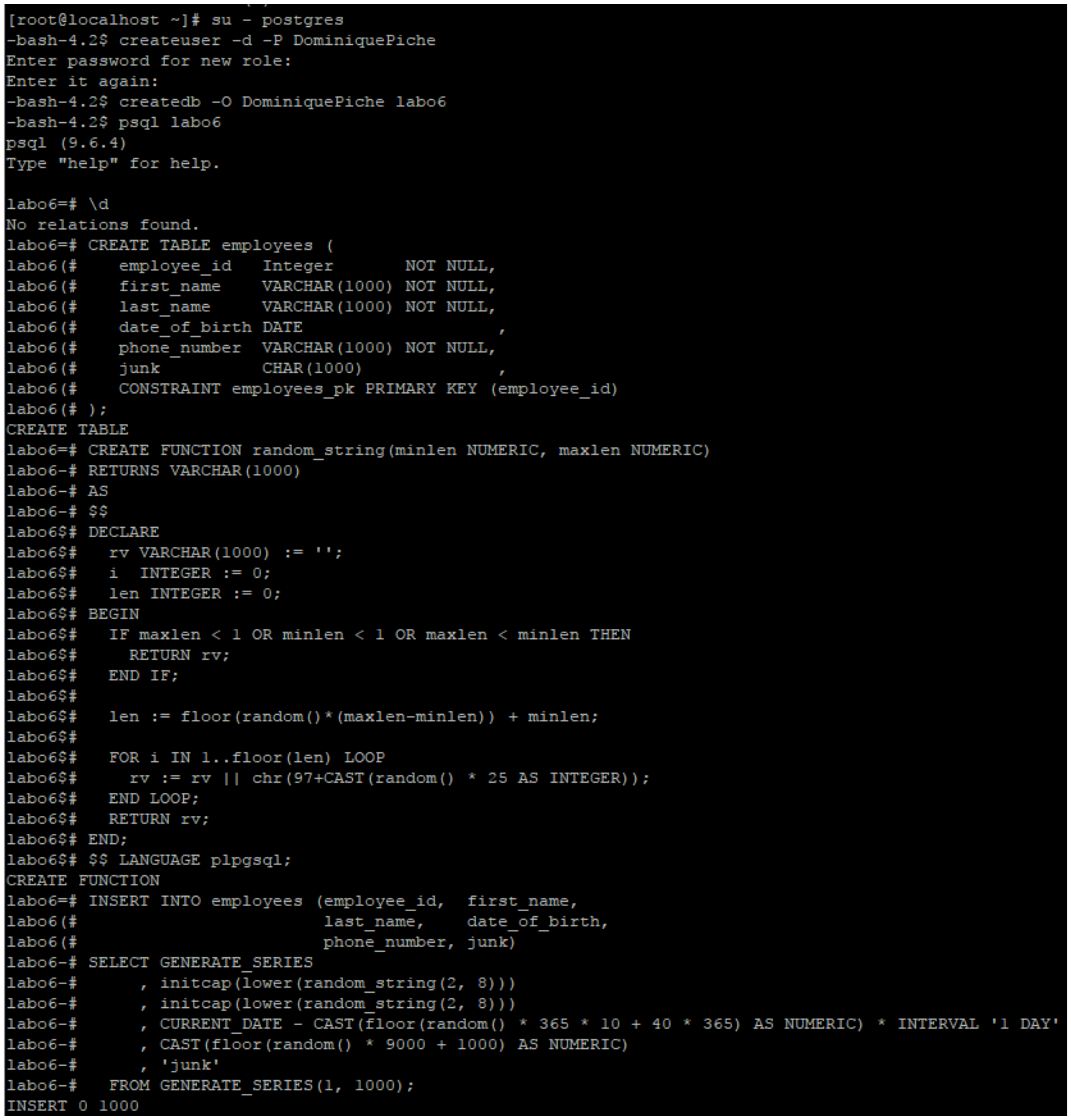
Dominique PICHÉ 1766981

Département de génie informatique et génie logiciel

Le 27novembre 2017
École polytechnique de Montréal

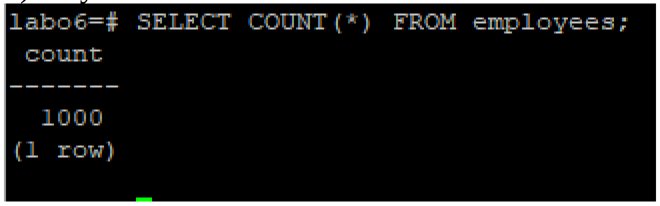
Les index

Voici une capture d'écran de la création et du peuplement de la table Employees:



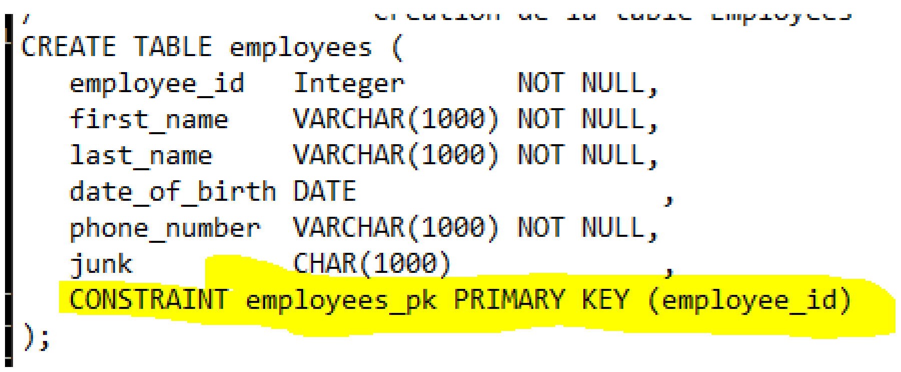
```
[root@localhost ~]# su - postgres
-bash-4.2$ createuser -d -P DominiquePiche
Enter password for new role:
Enter it again:
-bash-4.2$ createdb -O DominiquePiche labo6
-bash-4.2$ psql labo6
psql (9.6.4)
Type "help" for help.

labo6=# \d
No relations found.
labo6=# CREATE TABLE employees (
labo6(#    employee_id   Integer       NOT NULL,
labo6(#    first_name    VARCHAR(1000) NOT NULL,
labo6(#    last_name     VARCHAR(1000) NOT NULL,
labo6(#    date_of_birth DATE                   ,
labo6(#    phone_number  VARCHAR(1000) NOT NULL,
labo6(#    junk          CHAR(1000)             ,
labo6(#    CONSTRAINT employees_pk PRIMARY KEY (employee_id)
labo6(# );
CREATE TABLE
labo6=# CREATE FUNCTION random_string(minlen NUMERIC, maxlen NUMERIC)
labo6-# RETURNS VARCHAR(1000)
labo6-# AS
labo6-# $$
labo6$# DECLARE
labo6$#   rv VARCHAR(1000) := '';
labo6$#   i  INTEGER := 0;
labo6$#   len INTEGER := 0;
labo6$# BEGIN
labo6$#   IF maxlen < 1 OR minlen < 1 OR maxlen < minlen THEN
labo6$#     RETURN rv;
labo6$#   END IF;
labo6$#
labo6$#   len := floor(random()*(maxlen-minlen)) + minlen;
labo6$#
labo6$#   FOR i IN 1..floor(len) LOOP
labo6$#     rv := rv || chr(97+CAST(random() * 25 AS INTEGER));
labo6$#   END LOOP;
labo6$#   RETURN rv;
labo6$# END;
labo6$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
labo6=# INSERT INTO employees (employee_id,  first_name,
labo6(#                        last_name,    date_of_birth,
labo6(#                        phone_number, junk)
labo6-# SELECT GENERATE_SERIES
labo6-#      , initcap(lower(random_string(2, 8)))
labo6-#      , initcap(lower(random_string(2, 8)))
labo6-#      , CURRENT_DATE - CAST(floor(random() * 365 * 10 + 40 * 365) AS NUMERIC) * INTERVAL '1 DAY'
labo6-#      , CAST(floor(random() * 9000 + 1000) AS NUMERIC)
labo6-#      , 'junk'
labo6-#   FROM GENERATE_SERIES(1, 1000);
INSERT 0 1000
```

1) Il y a 1000 entrées insérées:

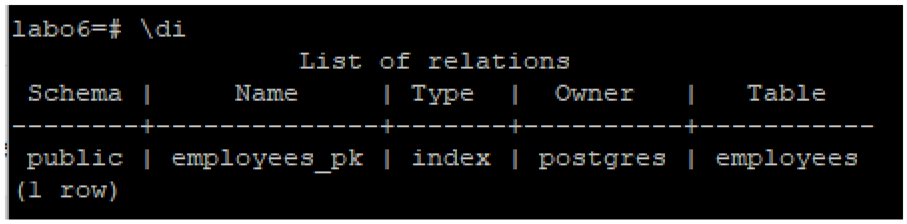


```
labo6=# SELECT COUNT(*) FROM employees;
 count
-------
  1000
(1 row)
```

2) Un index est créé automatiquement sur la CONSTRAINT employees_pk qui spécifie que l'attribut employee_id est la clé primaire. Pour chaque contrainte impliquant qu'un attribut est unique, un index est créé automatiquement.

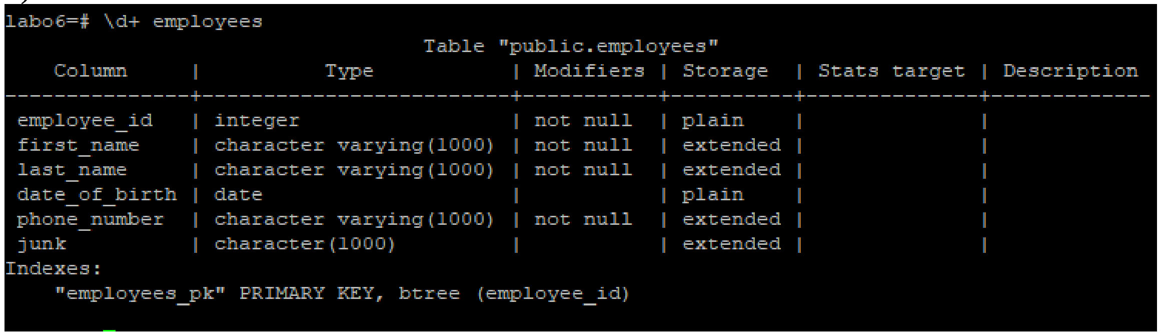


```
CREATE TABLE employees (
   employee_id   Integer       NOT NULL,
   first_name    VARCHAR(1000) NOT NULL,
   last_name     VARCHAR(1000) NOT NULL,
   date_of_birth DATE                  ,
   phone_number  VARCHAR(1000) NOT NULL,
   junk          CHAR(1000)            ,
   CONSTRAINT employees_pk PRIMARY KEY (employee_id)
);
```



```
labo6=# \di
                   List of relations
 Schema |     Name     | Type  |  Owner   |   Table
--------+--------------+-------+----------+-----------
 public | employees_pk | index | postgres | employees
(1 row)
```

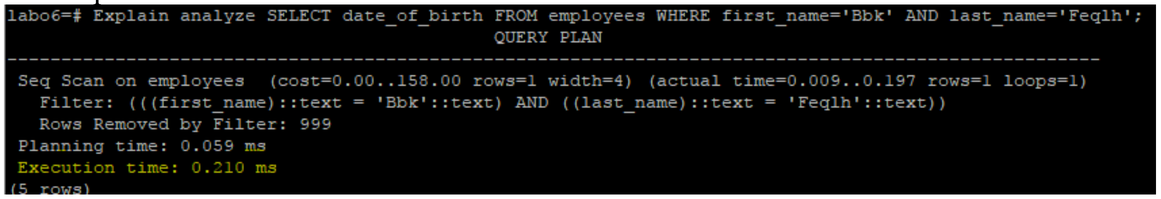
3) Voici la structure de la table:



```
labo6=# \d+ employees
                                  Table "public.employees"
    Column     |          Type           | Modifiers | Storage  | Stats target | Description
---------------+-------------------------+-----------+----------+--------------+-------------
 employee_id   | integer                 | not null  | plain    |              |
 first_name    | character varying(1000) | not null  | extended |              |
 last_name     | character varying(1000) | not null  | extended |              |
 date_of_birth | date                    |           | plain    |              |
 phone_number  | character varying(1000) | not null  | extended |              |
 junk          | character(1000)         |           | extended |              |
Indexes:
    "employees_pk" PRIMARY KEY, btree (employee_id)
```

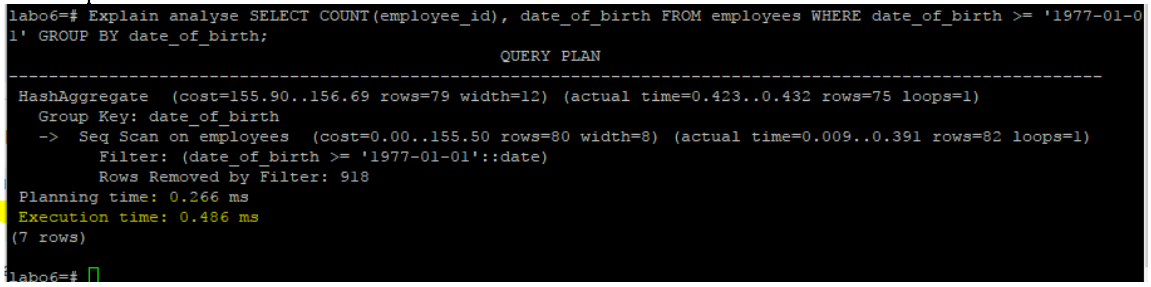
4)

a) Le temps d'exécution est de 0.210 ms:



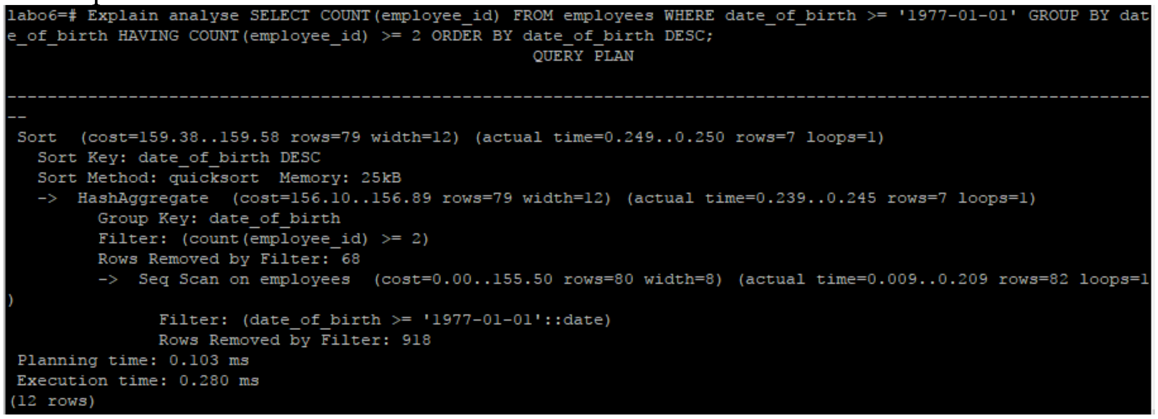
```
labo6=# Explain analyze SELECT date_of_birth FROM employees WHERE first_name='Bbk' AND last_name='Feqlh';
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..158.00 rows=1 width=4) (actual time=0.009..0.197 rows=1 loops=1)
   Filter: (((first_name)::text = 'Bbk'::text) AND ((last_name)::text = 'Feqlh'::text))
   Rows Removed by Filter: 999
 Planning time: 0.059 ms
 Execution time: 0.210 ms
(5 rows)
```

b) Le temps d’exécution est de 0.486 ms:



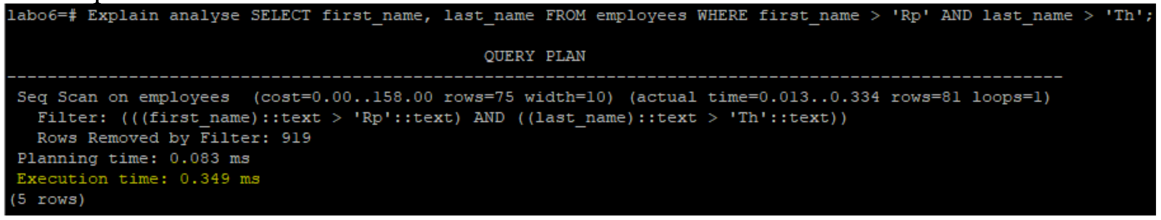
```
labo6=# Explain analyse SELECT COUNT(employee_id), date_of_birth FROM employees WHERE date_of_birth >= '1977-01-0
1' GROUP BY date_of_birth;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=155.90..156.69 rows=79 width=12) (actual time=0.423..0.432 rows=75 loops=1)
   Group Key: date_of_birth
   ->  Seq Scan on employees  (cost=0.00..155.50 rows=80 width=8) (actual time=0.009..0.391 rows=82 loops=1)
         Filter: (date_of_birth >= '1977-01-01'::date)
         Rows Removed by Filter: 918
 Planning time: 0.266 ms
 Execution time: 0.486 ms
(7 rows)

labo6=#
```

c) Le temps d’exécution est de 0.280 ms:



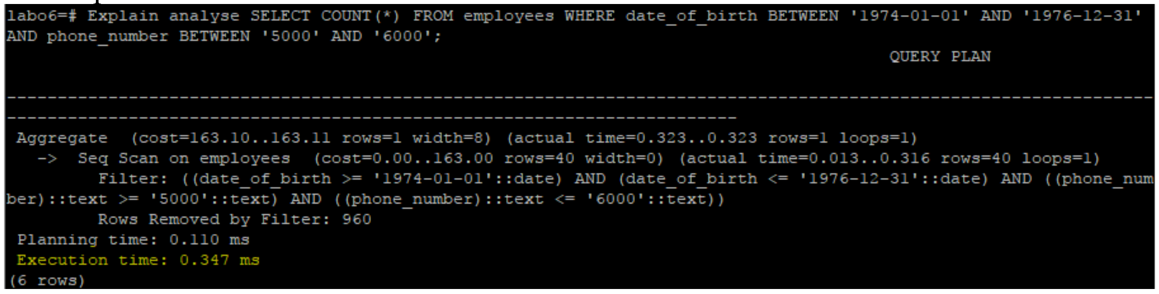
```
labo6=# Explain analyse SELECT COUNT(employee_id) FROM employees WHERE date_of_birth >= '1977-01-01' GROUP BY dat
e_of_birth HAVING COUNT(employee_id) >= 2 ORDER BY date_of_birth DESC;
                                                      QUERY PLAN

-----------------------------------------------------------------------------------------------------------------
--
 Sort  (cost=159.38..159.58 rows=79 width=12) (actual time=0.249..0.250 rows=7 loops=1)
   Sort Key: date_of_birth DESC
   Sort Method: quicksort  Memory: 25kB
   ->  HashAggregate  (cost=156.10..156.89 rows=79 width=12) (actual time=0.239..0.245 rows=7 loops=1)
         Group Key: date_of_birth
         Filter: (count(employee_id) >= 2)
         Rows Removed by Filter: 68
         ->  Seq Scan on employees  (cost=0.00..155.50 rows=80 width=8) (actual time=0.009..0.209 rows=82 loops=1
)
               Filter: (date_of_birth >= '1977-01-01'::date)
               Rows Removed by Filter: 918
 Planning time: 0.103 ms
 Execution time: 0.280 ms
(12 rows)
```

e) Le temps d’exécution est de 0.349 ms:



```
labo6=# Explain analyse SELECT first_name, last_name FROM employees WHERE first_name > 'Rp' AND last_name > 'Th';

                                                QUERY PLAN
---------------------------------------------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..158.00 rows=75 width=10) (actual time=0.013..0.334 rows=81 loops=1)
   Filter: (((first_name)::text > 'Rp'::text) AND ((last_name)::text > 'Th'::text))
   Rows Removed by Filter: 919
 Planning time: 0.083 ms
 Execution time: 0.349 ms
(5 rows)
```

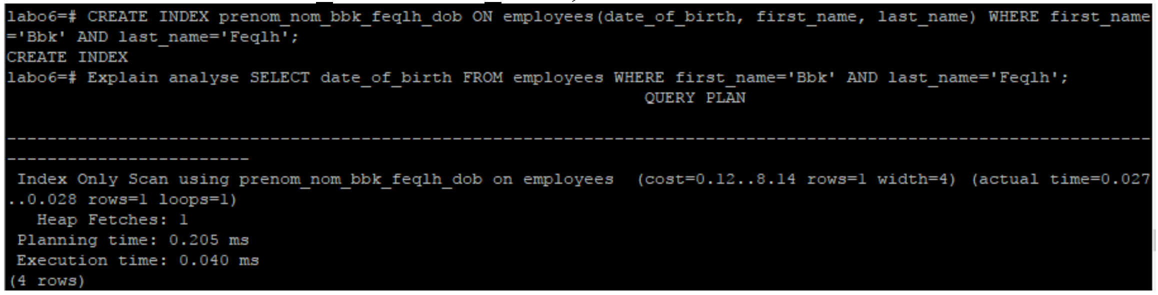
f) Le temps d’exécution est de 0.347 ms:



```
labo6=# Explain analyse SELECT COUNT(*) FROM employees WHERE date_of_birth BETWEEN '1974-01-01' AND '1976-12-31'
AND phone_number BETWEEN '5000' AND '6000';
                                                                                      QUERY PLAN

-----------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------
 Aggregate  (cost=163.10..163.11 rows=1 width=8) (actual time=0.323..0.323 rows=1 loops=1)
   ->  Seq Scan on employees  (cost=0.00..163.00 rows=40 width=0) (actual time=0.013..0.316 rows=40 loops=1)
         Filter: ((date_of_birth >= '1974-01-01'::date) AND (date_of_birth <= '1976-12-31'::date) AND ((phone_num
ber)::text >= '5000'::text) AND ((phone_number)::text <= '6000'::text))
         Rows Removed by Filter: 960
 Planning time: 0.110 ms
 Execution time: 0.347 ms
(6 rows)
```

5)

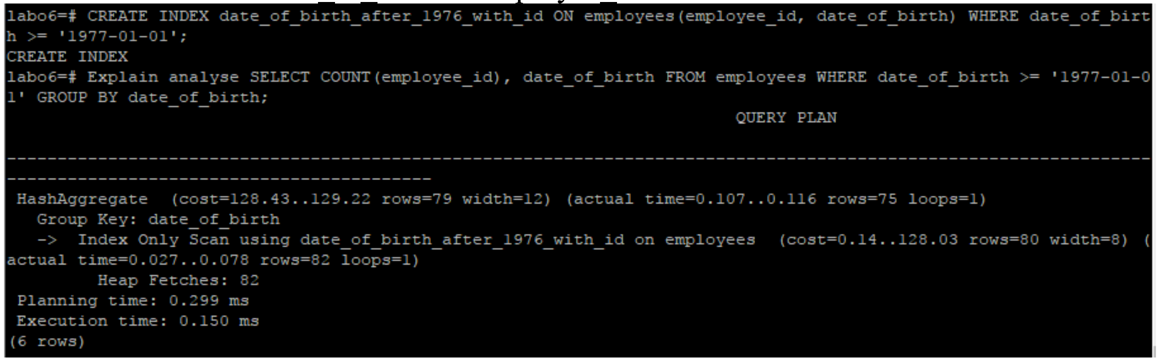
a) L’index est créé sur first_name et last_name, avec une clause WHERE:



```
labo6=# CREATE INDEX prenom_nom_bbk_feqlh_dob ON employees(date_of_birth, first_name, last_name) WHERE first_name
='Bbk' AND last_name='Feqlh';
CREATE INDEX
labo6=# Explain analyse SELECT date_of_birth FROM employees WHERE first_name='Bbk' AND last_name='Feqlh';
                                                               QUERY PLAN

-----------------------------------------------------------------------------------------------------------------
------------------------
 Index Only Scan using prenom_nom_bbk_feqlh_dob on employees  (cost=0.12..8.14 rows=1 width=4) (actual time=0.027
..0.028 rows=1 loops=1)
   Heap Fetches: 1
 Planning time: 0.205 ms
 Execution time: 0.040 ms
(4 rows)
```

Le temps d’exécution est de 0.04 ms.

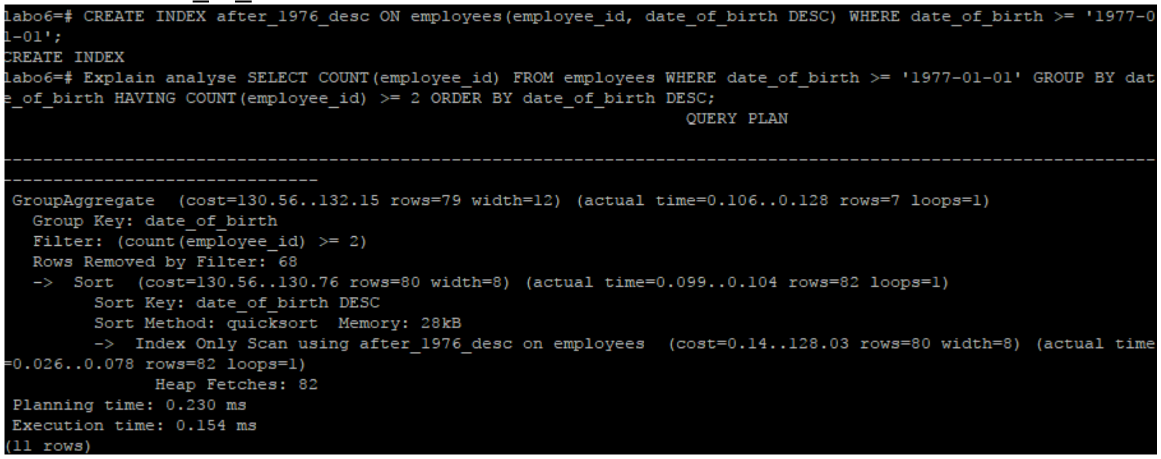
b) L’index est créé sur date_of_birth et employee_id avec un WHERE:



```
labo6=# CREATE INDEX date_of_birth_after_1976_with_id ON employees(employee_id, date_of_birth) WHERE date_of_birt
h >= '1977-01-01';
CREATE INDEX
labo6=# Explain analyse SELECT COUNT(employee_id), date_of_birth FROM employees WHERE date_of_birth >= '1977-01-0
1' GROUP BY date_of_birth;
                                                                        QUERY PLAN

-----------------------------------------------------------------------------------------------------------------
------------------------------------------
 HashAggregate  (cost=128.43..129.22 rows=79 width=12) (actual time=0.107..0.116 rows=75 loops=1)
   Group Key: date_of_birth
   ->  Index Only Scan using date_of_birth_after_1976_with_id on employees  (cost=0.14..128.03 rows=80 width=8) (
actual time=0.027..0.078 rows=82 loops=1)
         Heap Fetches: 82
 Planning time: 0.299 ms
 Execution time: 0.150 ms
(6 rows)
```

Le temps d’exécution est de 0.150 ms.

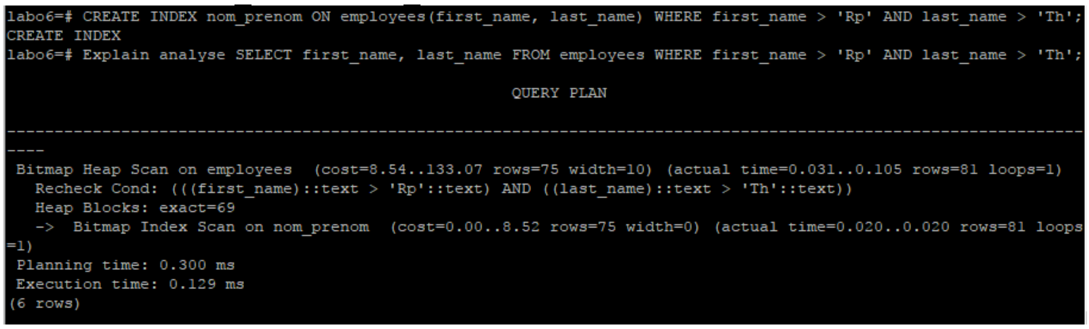
c) L’index est sur date_of_birth et employee_id, avec le même WHERE qu’en b) mais avec date_of_birth DESC:



```
labo6=# CREATE INDEX after_1976_desc ON employees(employee_id, date_of_birth DESC) WHERE date_of_birth >= '1977-0
1-01';
CREATE INDEX
labo6=# Explain analyse SELECT COUNT(employee_id) FROM employees WHERE date_of_birth >= '1977-01-01' GROUP BY dat
e_of_birth HAVING COUNT(employee_id) >= 2 ORDER BY date_of_birth DESC;
                                                                   QUERY PLAN

-----------------------------------------------------------------------------------------------------------------
-------------------------------
 GroupAggregate  (cost=130.56..132.15 rows=79 width=12) (actual time=0.106..0.128 rows=7 loops=1)
   Group Key: date_of_birth
   Filter: (count(employee_id) >= 2)
   Rows Removed by Filter: 68
   ->  Sort  (cost=130.56..130.76 rows=80 width=8) (actual time=0.099..0.104 rows=82 loops=1)
         Sort Key: date_of_birth DESC
         Sort Method: quicksort  Memory: 28kB
         ->  Index Only Scan using after_1976_desc on employees  (cost=0.14..128.03 rows=80 width=8) (actual time
=0.026..0.078 rows=82 loops=1)
               Heap Fetches: 82
 Planning time: 0.230 ms
 Execution time: 0.154 ms
(11 rows)
```

Le temps d’exécution est 0.154 ms.

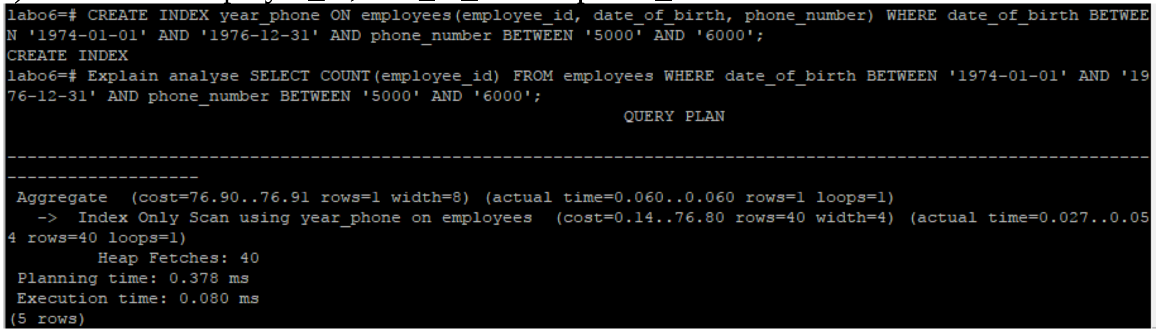
e) L’index est sur first_name et last_name:



```
labo6=# CREATE INDEX nom_prenom ON employees(first_name, last_name) WHERE first_name > 'Rp' AND last_name > 'Th';
CREATE INDEX
labo6=# Explain analyse SELECT first_name, last_name FROM employees WHERE first_name > 'Rp' AND last_name > 'Th';

                                                    QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------
----
 Bitmap Heap Scan on employees  (cost=8.54..133.07 rows=75 width=10) (actual time=0.031..0.105 rows=81 loops=1)
   Recheck Cond: (((first_name)::text > 'Rp'::text) AND ((last_name)::text > 'Th'::text))
   Heap Blocks: exact=69
   ->  Bitmap Index Scan on nom_prenom  (cost=0.00..8.52 rows=75 width=0) (actual time=0.020..0.020 rows=81 loops
=1)
 Planning time: 0.300 ms
 Execution time: 0.129 ms
(6 rows)
```

Le temps d’exécution est de 0.129 ms.

f) L'index sur employee_id, date_of_birth et phone_number avec WHERE:



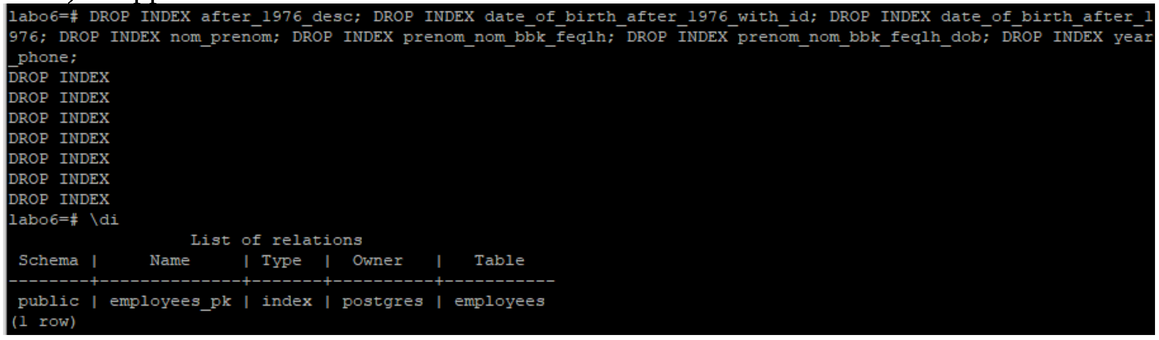
```
labo6=# CREATE INDEX year_phone ON employees(employee_id, date_of_birth, phone_number) WHERE date_of_birth BETWEE
N '1974-01-01' AND '1976-12-31' AND phone_number BETWEEN '5000' AND '6000';
CREATE INDEX
labo6=# Explain analyse SELECT COUNT(employee_id) FROM employees WHERE date_of_birth BETWEEN '1974-01-01' AND '19
76-12-31' AND phone_number BETWEEN '5000' AND '6000';
                                                    QUERY PLAN

-----------------------------------------------------------------------------------------------------------------
-------------------
 Aggregate  (cost=76.90..76.91 rows=1 width=8) (actual time=0.060..0.060 rows=1 loops=1)
   ->  Index Only Scan using year_phone on employees  (cost=0.14..76.80 rows=40 width=4) (actual time=0.027..0.05
4 rows=40 loops=1)
         Heap Fetches: 40
 Planning time: 0.378 ms
 Execution time: 0.080 ms
(5 rows)
```

Le temps d’exécution est de 0.080 ms.

En comparant la question 4 et la question 5, on remarque que les requête SQL sont bien plus rapides avec l'ajout de nos index. Ceci n'est pas surprenant puisque l'ajout d'index est le meilleur outil pour optimiser des requêtes SQL. On remarque une optimisation de plus de 75% pour certaines des requêtes à la question 5.

Question B

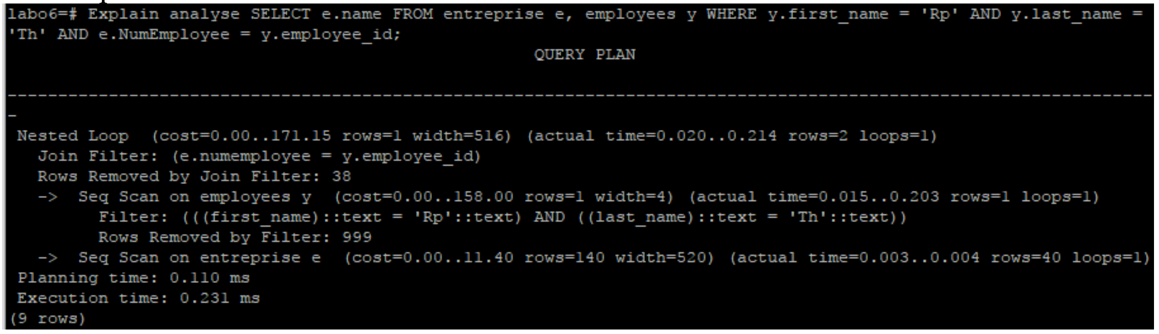
1) Suppression des index:



```
labo6=# DROP INDEX after_1976_desc; DROP INDEX date_of_birth_after_1976_with_id; DROP INDEX date_of_birth_after_1
976; DROP INDEX nom_prenom; DROP INDEX prenom_nom_bbk_feqlh; DROP INDEX prenom_nom_bbk_feqlh_dob; DROP INDEX year
_phone;
DROP INDEX
DROP INDEX
DROP INDEX
DROP INDEX
DROP INDEX
DROP INDEX
DROP INDEX
labo6=# \di
                  List of relations
 Schema |     Name      | Type  |  Owner   |   Table
--------+---------------+-------+----------+-----------
 public | employees_pk | index | postgres | employees
(1 row)
```

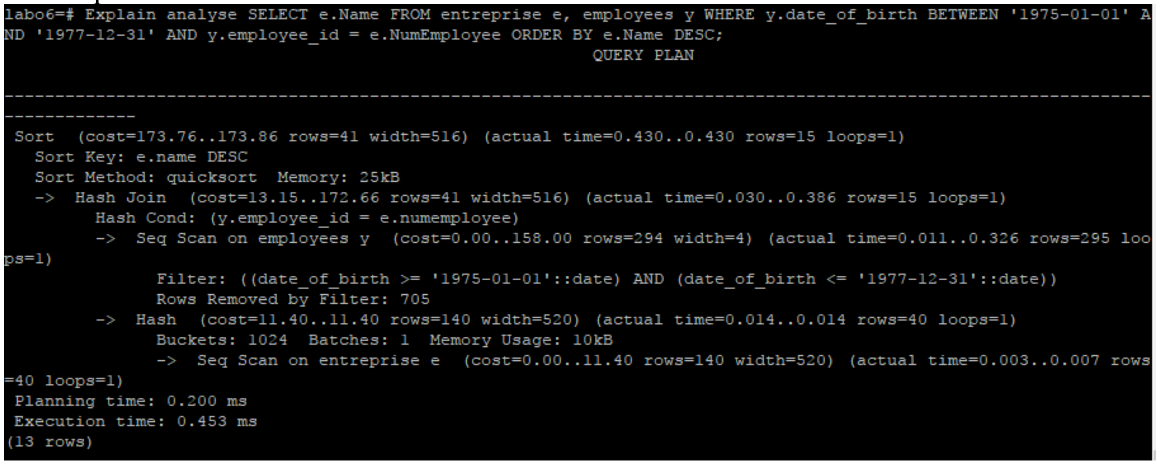
2)

a) Le temps d'exécution est de 0.231 ms.



```
labo6=# Explain analyse SELECT e.name FROM entreprise e, employees y WHERE y.first_name = 'Rp' AND y.last_name =
'Th' AND e.NumEmployee = y.employee_id;
                                                   QUERY PLAN

-------------------------------------------------------------------------------------------------------------------
-
 Nested Loop  (cost=0.00..171.15 rows=1 width=516) (actual time=0.020..0.214 rows=2 loops=1)
   Join Filter: (e.numemployee = y.employee_id)
   Rows Removed by Join Filter: 38
   ->  Seq Scan on employees y  (cost=0.00..158.00 rows=1 width=4) (actual time=0.015..0.203 rows=1 loops=1)
         Filter: (((first_name)::text = 'Rp'::text) AND ((last_name)::text = 'Th'::text))
         Rows Removed by Filter: 999
   ->  Seq Scan on entreprise e  (cost=0.00..11.40 rows=140 width=520) (actual time=0.003..0.004 rows=40 loops=1)
 Planning time: 0.110 ms
 Execution time: 0.231 ms
(9 rows)
```

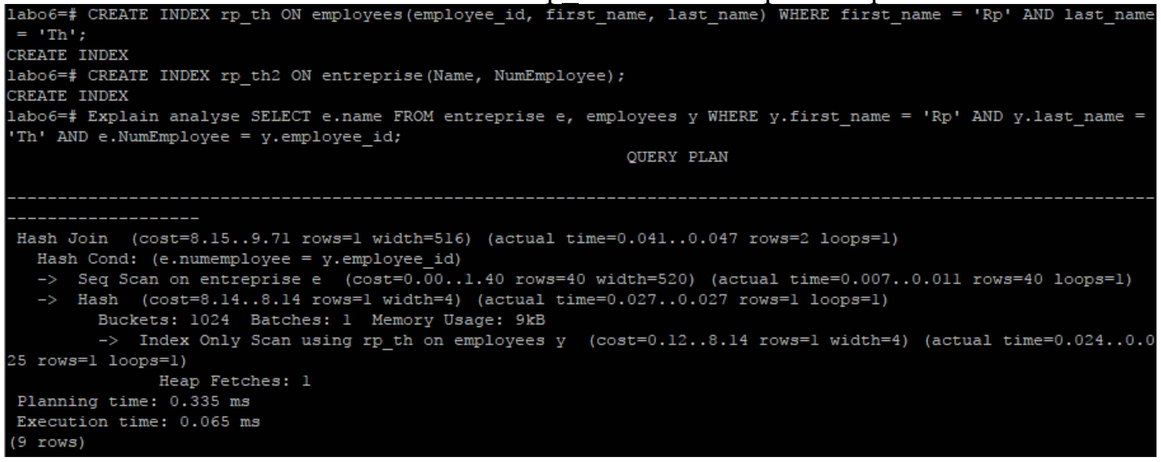
b) Le temps d'exécution est de 0.453 ms.



```
labo6=# Explain analyse SELECT e.Name FROM entreprise e, employees y WHERE y.date_of_birth BETWEEN '1975-01-01' A
ND '1977-12-31' AND y.employee_id = e.NumEmployee ORDER BY e.Name DESC;
                                                         QUERY PLAN

-------------------------------------------------------------------------------------------------------------------
-------------
 Sort  (cost=173.76..173.86 rows=41 width=516) (actual time=0.430..0.430 rows=15 loops=1)
   Sort Key: e.name DESC
   Sort Method: quicksort  Memory: 25kB
   ->  Hash Join  (cost=13.15..172.66 rows=41 width=516) (actual time=0.030..0.386 rows=15 loops=1)
         Hash Cond: (y.employee_id = e.numemployee)
         ->  Seq Scan on employees y  (cost=0.00..158.00 rows=294 width=4) (actual time=0.011..0.326 rows=295 loo
ps=1)
               Filter: ((date_of_birth >= '1975-01-01'::date) AND (date_of_birth <= '1977-12-31'::date))
               Rows Removed by Filter: 705
         ->  Hash  (cost=11.40..11.40 rows=140 width=520) (actual time=0.014..0.014 rows=40 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 10kB
               ->  Seq Scan on entreprise e  (cost=0.00..11.40 rows=140 width=520) (actual time=0.003..0.007 rows
=40 loops=1)
 Planning time: 0.200 ms
 Execution time: 0.453 ms
(13 rows)
```

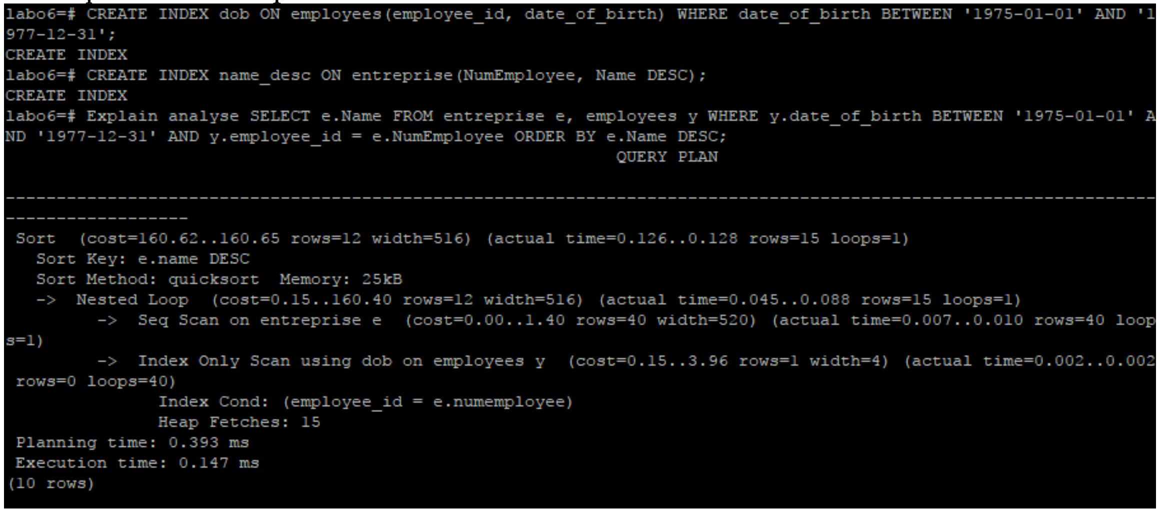
3)

a) Avec un index sur employee_id, first_name et last_name, le temps d'exécution est de 0.065 ms. L'index rp_th2 ne semble pas être pertinent.



```
labo6=# CREATE INDEX rp_th ON employees(employee_id, first_name, last_name) WHERE first_name = 'Rp' AND last_name
 = 'Th';
CREATE INDEX
labo6=# CREATE INDEX rp_th2 ON entreprise(Name, NumEmployee);
CREATE INDEX
labo6=# Explain analyse SELECT e.name FROM entreprise e, employees y WHERE y.first_name = 'Rp' AND y.last_name =
'Th' AND e.NumEmployee = y.employee_id;
                                                         QUERY PLAN

-------------------------------------------------------------------------------------------------------------------
-------------------
 Hash Join  (cost=8.15..9.71 rows=1 width=516) (actual time=0.041..0.047 rows=2 loops=1)
   Hash Cond: (e.numemployee = y.employee_id)
   ->  Seq Scan on entreprise e  (cost=0.00..1.40 rows=40 width=520) (actual time=0.007..0.011 rows=40 loops=1)
   ->  Hash  (cost=8.14..8.14 rows=1 width=4) (actual time=0.027..0.027 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Index Only Scan using rp_th on employees y  (cost=0.12..8.14 rows=1 width=4) (actual time=0.024..0.0
25 rows=1 loops=1)
               Heap Fetches: 1
 Planning time: 0.335 ms
 Execution time: 0.065 ms
(9 rows)
```

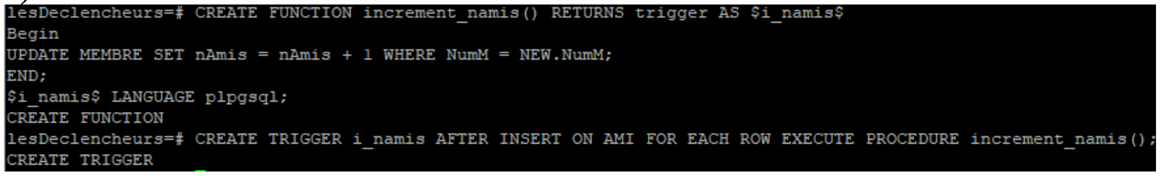
b) Index sur les date de naissances respectant la condition du WHERE de la requête et un temps d'exécution de 0.147 ms.



```
labo6=# CREATE INDEX dob ON employees(employee_id, date_of_birth) WHERE date_of_birth BETWEEN '1975-01-01' AND '1
977-12-31';
CREATE INDEX
labo6=# CREATE INDEX name_desc ON entreprise(NumEmployee, Name DESC);
CREATE INDEX
labo6=# Explain analyse SELECT e.Name FROM entreprise e, employees y WHERE y.date_of_birth BETWEEN '1975-01-01' A
ND '1977-12-31' AND y.employee_id = e.NumEmployee ORDER BY e.Name DESC;
                                                            QUERY PLAN

--------------------------------------------------------------------------------------------------------------------
------------------
 Sort  (cost=160.62..160.65 rows=12 width=516) (actual time=0.126..0.128 rows=15 loops=1)
   Sort Key: e.name DESC
   Sort Method: quicksort  Memory: 25kB
   ->  Nested Loop  (cost=0.15..160.40 rows=12 width=516) (actual time=0.045..0.088 rows=15 loops=1)
         ->  Seq Scan on entreprise e  (cost=0.00..1.40 rows=40 width=520) (actual time=0.007..0.010 rows=40 loop
s=1)
         ->  Index Only Scan using dob on employees y  (cost=0.15..3.96 rows=1 width=4) (actual time=0.002..0.002
 rows=0 loops=40)
               Index Cond: (employee_id = e.numemployee)
               Heap Fetches: 15
 Planning time: 0.393 ms
 Execution time: 0.147 ms
(10 rows)
```

Les Déclencheurs

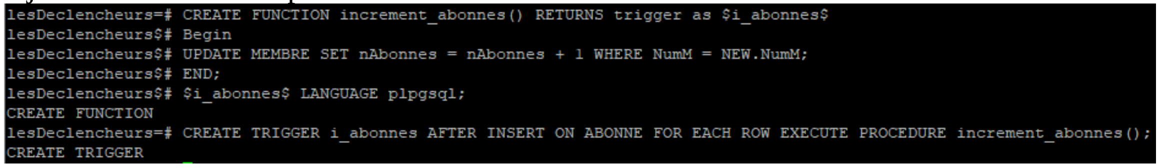
1)



```
lesDeclencheurs=# CREATE FUNCTION increment_namis() RETURNS trigger AS $i_namis$
Begin
UPDATE MEMBRE SET nAmis = nAmis + 1 WHERE NumM = NEW.NumM;
END;
$i_namis$ LANGUAGE plpgsql;
CREATE FUNCTION
lesDeclencheurs=# CREATE TRIGGER i_namis AFTER INSERT ON AMI FOR EACH ROW EXECUTE PROCEDURE increment_namis();
CREATE TRIGGER
```

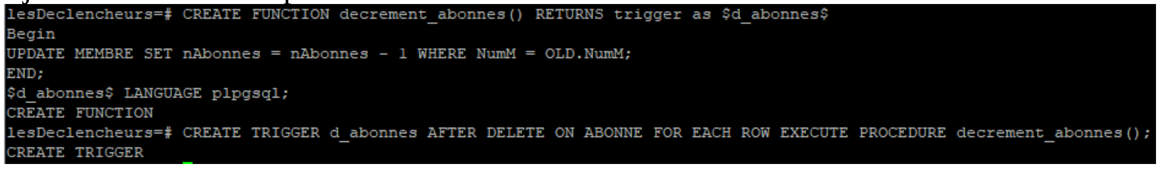
2)

Ajout du déclencheur pour incrémenter les abonnés:



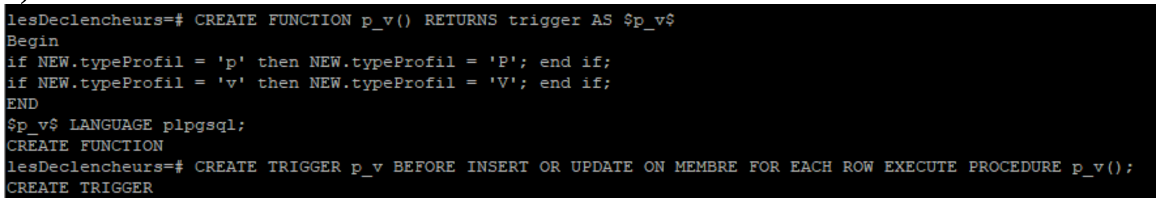
```
lesDeclencheurs=# CREATE FUNCTION increment_abonnes() RETURNS trigger as $i_abonnes$
lesDeclencheurs$# Begin
lesDeclencheurs$# UPDATE MEMBRE SET nAbonnes = nAbonnes + 1 WHERE NumM = NEW.NumM;
lesDeclencheurs$# END;
lesDeclencheurs$# $i_abonnes$ LANGUAGE plpgsql;
CREATE FUNCTION
lesDeclencheurs=# CREATE TRIGGER i_abonnes AFTER INSERT ON ABONNE FOR EACH ROW EXECUTE PROCEDURE increment_abonnes();
CREATE TRIGGER
```

Ajout du déclencheur pour décrémenter les abonnés:



```
lesDeclencheurs=# CREATE FUNCTION decrement_abonnes() RETURNS trigger as $d_abonnes$
Begin
UPDATE MEMBRE SET nAbonnes = nAbonnes - 1 WHERE NumM = OLD.NumM;
END;
$d_abonnes$ LANGUAGE plpgsql;
CREATE FUNCTION
lesDeclencheurs=# CREATE TRIGGER d_abonnes AFTER DELETE ON ABONNE FOR EACH ROW EXECUTE PROCEDURE decrement_abonnes();
CREATE TRIGGER
```

3)



```
lesDeclencheurs=# CREATE FUNCTION p_v() RETURNS trigger AS $p_v$
Begin
if NEW.typeProfil = 'p' then NEW.typeProfil = 'P'; end if;
if NEW.typeProfil = 'v' then NEW.typeProfil = 'V'; end if;
END
$p_v$ LANGUAGE plpgsql;
CREATE FUNCTION
lesDeclencheurs=# CREATE TRIGGER p_v BEFORE INSERT OR UPDATE ON MEMBRE FOR EACH ROW EXECUTE PROCEDURE p_v();
CREATE TRIGGER
```