Annexe A : Bases relationnelles

Création des tables, index, contraintes référentielles (requêtes standard SQL, compatible PostgreSQL et MySQL) :

Table médecin	`CREATE TABLE t_medecin(id int, nom varchar(50), prenom varchar(50), mail varchar(50), specialite int);` `ALTER TABLE t_medecin ADD CONSTRAINT pk_medecin PRIMARY KEY (id);`
Table patient	`CREATE TABLE t_patient(id int, nom varchar(50), prenom varchar(50), mail varchar(50));` `ALTER TABLE t_patient ADD CONSTRAINT pk_patient PRIMARY KEY (id);`

Table charnière médecin/patient	`CREATE TABLE t_charniere_medecin_patient(id_medecin int, id_patient int);`
	`ALTER TABLE t_charniere_medecin_patient ADD CONSTRAINT fk_id_medecin FOREIGN KEY (id_medecin) REFERENCES t_medecin(id);`
	`ALTER TABLE t_charniere_medecin_patient ADD CONSTRAINT fk_id_patient FOREIGN KEY (id_patient) REFERENCES t_patient(id);`

Fonction permettant de générer un nombre aléatoire entre 2 valeurs :

PostgreSQL 9.3.5	MySQL 5.5.40

`-- start_range  inclusive | end_range exclusive`
`CREATE FUNCTION random_range_int(start_range int, end_range int)`
`RETURNS INTEGER AS $$`
`BEGIN`
`  RETURN trunc(random() * (end_range - start_range) + start_range);`
`END;`
`$$ LANGUAGE plpgsql;`
	`-- start_range  inclusive | end_range inclusive`
	`DELIMITER $$`
	`CREATE FUNCTION random_range_int(start_range INTEGER, end_range INTEGER)`
	`RETURNS int`
	`BEGIN`
	`  RETURN round(rand() * (end_range - start_range) + start_range);`
	`END $$`
	`DELIMITER ;`

Requête des tests :

Test	PostgreSQL 9.3.5	MySQL 5.5.40
Note		`SET profiling = 1;`

A1	`CREATE FUNCTION my_proc1()`
	`RETURNS void`
	`AS $$`
	`  FOR i IN 1..4000 LOOP`
	`  INSERT INTO t_medecin(id, nom, prenom, mail, specialite)`
	`  VALUES(i, 'Nom ' \|\| i, 'Prenom ' \|\| i, '@mail ' \|\| i, random_range_int(1, 9));`
	`  END LOOP;`
	`END;`
	`$$ LANGUAGE plpgsql;`
		`DELIMITER $$`
		`CREATE PROCEDURE my_proc1()`
		`BEGIN`
		`  DECLARE i INT DEFAULT 1;`
		`  WHILE i <= 4000 DO`
		`  INSERT INTO t_medecin(id, nom, prenom, mail, specialite)`
		`  VALUES(i, CONCAT('Nom ', i), CONCAT('Prenom ', i), CONCAT('@mail ', i), random_range_int(1, 8));`
		`  SET i = i + 1;`
		`  END WHILE;`
		`END $$`
		`DELIMITER ;`

A2	`CREATE FUNCTION my_proc2()`
	`RETURNS void`
	`AS $$`
	`  FOR i IN 1..1000000 LOOP`
	`  INSERT INTO t_patient(id, nom, prenom, mail)`
	`  VALUES(i, 'Nom ' \|\| i, 'Prenom ' \|\| i, '@mail ' \|\| i);`
	`  END LOOP;`
	`END;`
	`$$ LANGUAGE plpgsql;`
		`DELIMITER $$`
		`CREATE PROCEDURE my_proc2()`
		`BEGIN`
		`  DECLARE i INT DEFAULT 1;`
		`  WHILE i <= 1000000 DO`
		`  INSERT INTO t_patient(id, nom, prenom, mail)`
		`  VALUES(i, CONCAT('Nom ', i), CONCAT('Prenom ', i), CONCAT('@mail ', i));`
		`  SET i = i + 1;`
		`  END WHILE;`
		`END $$`
		`DELIMITER ;`

A3	`CREATE FUNCTION my_proc3()`
	`RETURNS void`
	`AS $$`
	`  FOR i IN 1..1000000 LOOP`
	`  INSERT INTO t_charniere_medecin_patient(id_medecin, id_patient)`
	` VALUES(random_range_int(1,
		`DELIMITER $$`
		`CREATE PROCEDURE my_proc3()`
		`BEGIN`
		`  DECLARE i INT DEFAULT 1;`
		
		`  WHILE i <= 1000000 DO`
		`  INSERT INTO t_charniere_medecin_patient(id_medecin, id_patient)`

	4001), i); END LOOP; END; $$ LANGUAGE plpgsql;	VALUES(random_range_int(1, 4000), i); SET i = i + 1; END WHILE; END $$ DELIMITER ;
A4	EXPLAIN ANALYZE SELECT P.*, M.nom AS nom_medecin, M.prenom AS prenom_medecin FROM t_medecin M JOIN t_charniere_medecin_patient MP ON M.id = MP.id_medecin JOIN t_patient P ON P.id = MP.id_patient;	SELECT P.*, M.nom AS nom_medecin, M.prenom AS prenom_medecin FROM t_medecin M JOIN t_charniere_medecin_patient MP ON M.id = MP.id_medecin JOIN t_patient P ON P.id = MP.id_patient;
A5	EXPLAIN ANALYZE SELECT P.*, M.nom AS nom_medecin, M.prenom AS prenom_medecin FROM t_medecin M JOIN t_charniere_medecin_patient MP ON M.id = MP.id_medecin JOIN t_patient P ON P.id = MP.id_patient WHERE M.id = 40;	SELECT P.*, M.nom AS nom_medecin, M.prenom AS prenom_medecin FROM t_medecin M JOIN t_charniere_medecin_patient MP ON M.id = MP.id_medecin JOIN t_patient P ON P.id = MP.id_patient WHERE M.id = 40;
A6	EXPLAIN ANALYZE SELECT MP.id_medecin, COUNT(*) AS c FROM t_charniere_medecin_patient MP GROUP BY MP.id_medecin ORDER BY c DESC LIMIT 10;	SELECT MP.id_medecin, COUNT(*) AS c FROM t_charniere_medecin_patient MP GROUP BY MP.id_medecin ORDER BY c DESC LIMIT 10;

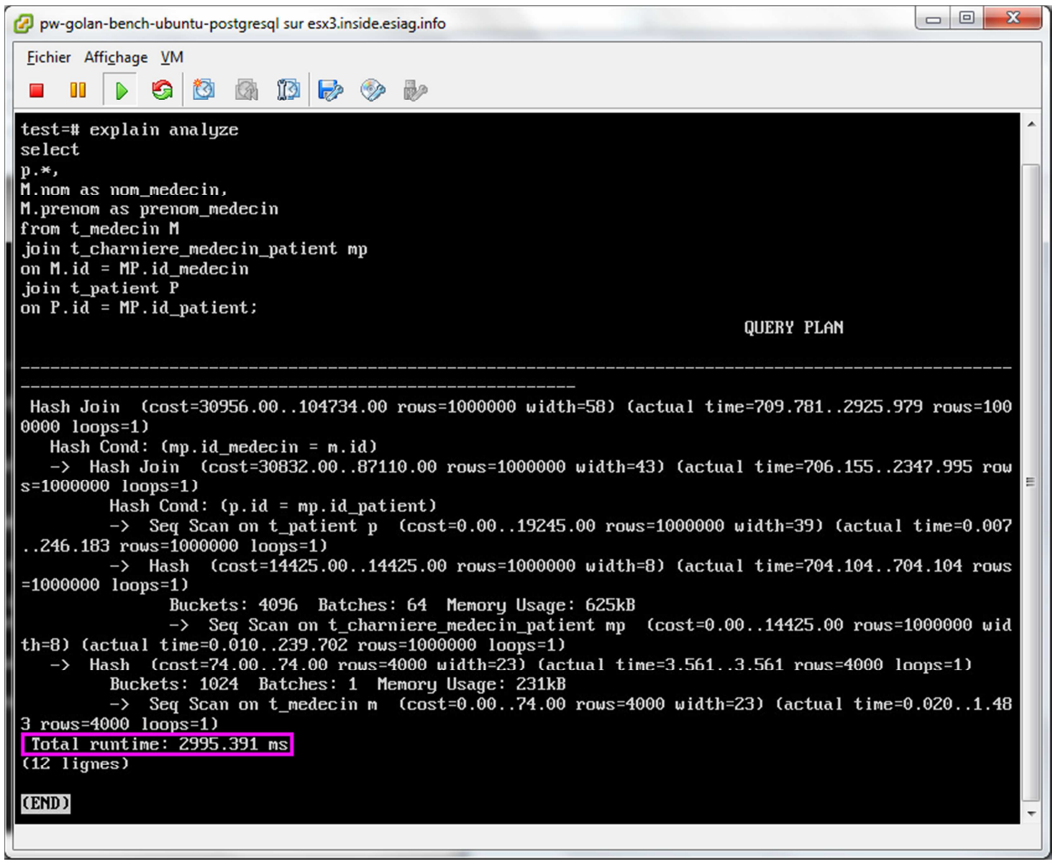

*Schéma : **ANALYZE EXPLAIN** PostgreSQL (A4)*

```
pw-golan-bench-ubuntu-postgresql sur esx3.inside.esiag.info
Fichier  Affichage  VM

test=#
test=#
test=#
test=#
test=# explain analyze select
test-# p.*,
test-# m.nom as nom_medecin,
test-# m.prenom as prenom_medecin
test-# from t_medecin m
test-# join t_charniere_medecin_patient mp
test-# on m.id = mp.id_medecin
test-# join t_patient p
test-# on p.id = mp.id_patient
test-# where m.id = 40;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------
-----------------------------------------
 Nested Loop  (cost=0.71..18946.76 rows=244 width=58) (actual time=0.076..222.404 rows=250 loops=1)
   ->  Nested Loop  (cost=0.28..16935.74 rows=244 width=23) (actual time=0.065..216.608 rows=250 loo
ps=1)
         ->  Index Scan using pk_medecin on t_medecin m  (cost=0.28..8.30 rows=1 width=23) (actual t
ime=0.013..0.015 rows=1 loops=1)
               Index Cond: (id = 40)
         ->  Seq Scan on t_charniere_medecin_patient mp  (cost=0.00..16925.00 rows=244 width=8) (act
ual time=0.044..216.369 rows=250 loops=1)
               Filter: (id_medecin = 40)
               Rows Removed by Filter: 999750
   ->  Index Scan using pk_patient on t_patient p  (cost=0.42..8.23 rows=1 width=39) (actual time=0.
019..0.020 rows=1 loops=250)
         Index Cond: (id = mp.id_patient)
 Total runtime: 222.573 ms
(10 lignes)

(END)
```

*Schéma : **ANALYZE EXPLAIN** PostgreSQL (A5)*

```
test=# explain analyze
test-# select
test-# mp.id_medecin,
test-# count(*) as c
test-# from t_charniere_medecin_patient mp
test-# group by mp.id_medecin
test-# order by c
test-# desc limit 10;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------
-----------------------------------------------------
 Limit  (cost=19551.44..19551.46 rows=10 width=4) (actual time=404.317..404.320 rows=10 loops=1)
   ->  Sort  (cost=19551.44..19561.44 rows=4000 width=4) (actual time=404.315..404.315 rows=10 loops
=1)
         Sort Key: (count(*))
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=19425.00..19465.00 rows=4000 width=4) (actual time=403.015..403.68
3 rows=4000 loops=1)
               ->  Seq Scan on t_charniere_medecin_patient mp  (cost=0.00..14425.00 rows=1000000 wid
th=4) (actual time=0.013..112.602 rows=1000000 loops=1)
 Total runtime: 404.421 ms
(7 lignes)

(END)
```

*Schéma : **ANALYZE EXPLAIN** PostgreSQL (A6)*

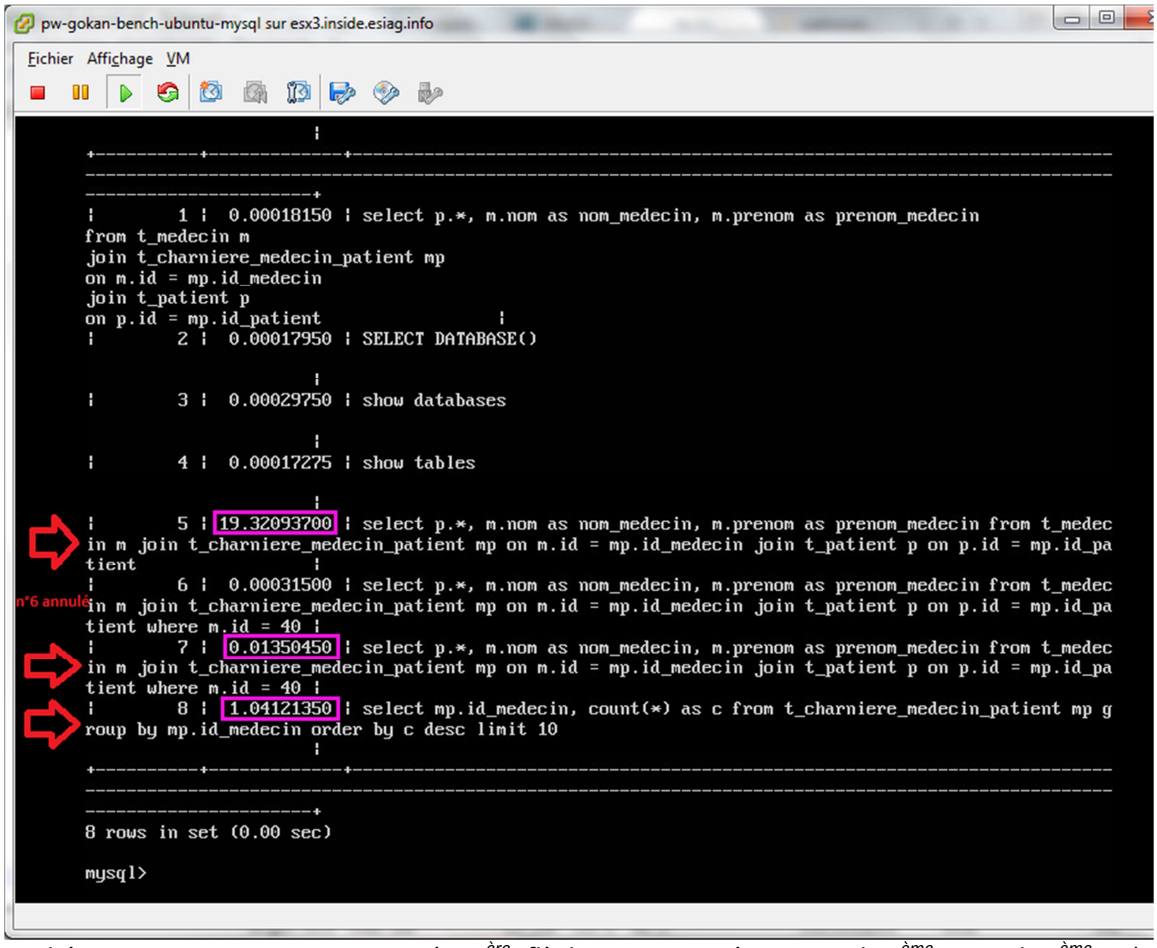

*Schéma : **SHOW PROFILES** MySQL (La 1ère flèche rouge représente A4, la 2ème A5, et la 3ème A6)*