

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО"

Кафедра програмного забезпечення комп'ютерних систем

Лабораторна робота №3
із дисципліни «Проектування корпоративних інформаційних систем»
на тему «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав:

студент __3__ курсу ФПМ

групи КП-81

Сутковенко Максим Миколайович

Прийняв: Радченко К.О.

КИЇВ 2020

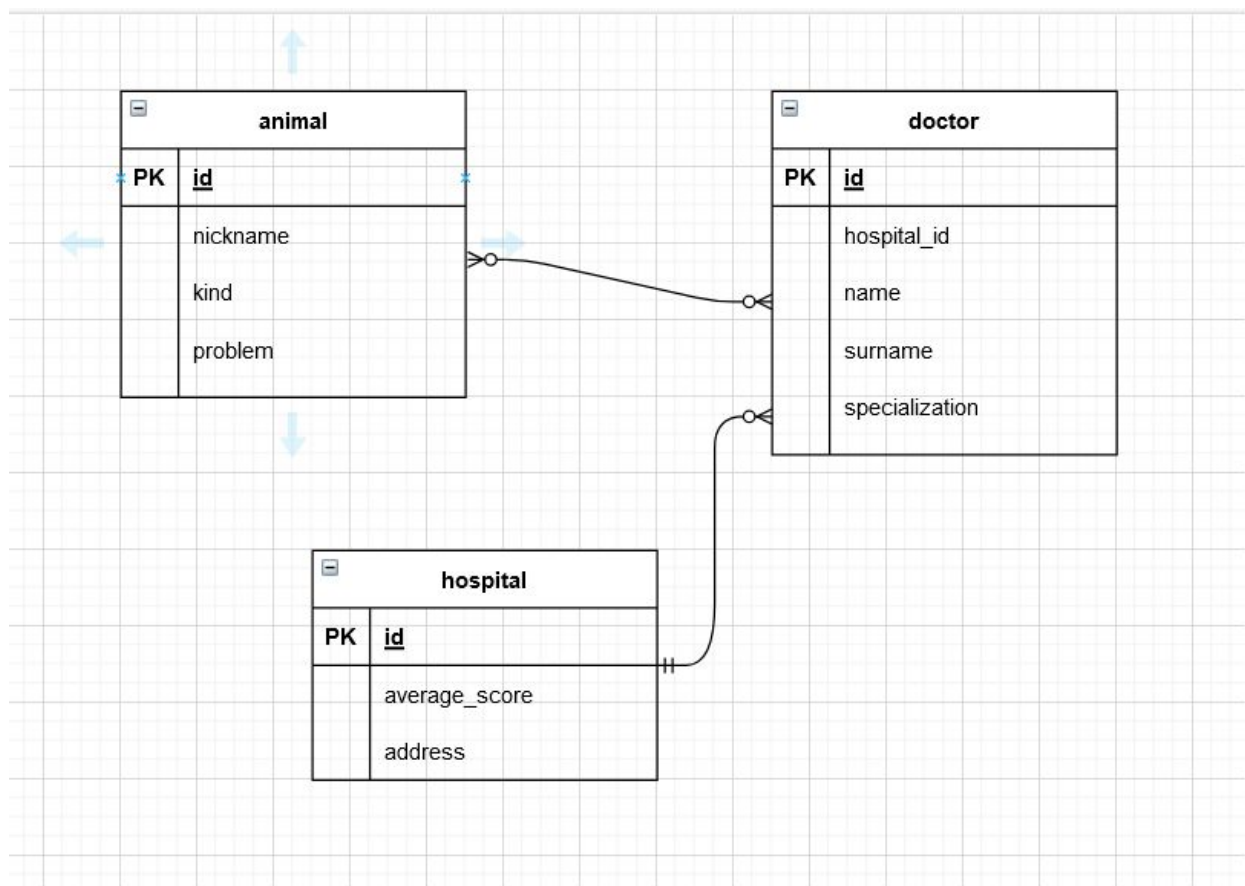


Рис. Схема БД

```

object AnimalTable : IntIdTable( name: "animal") {
    val nickname = varchar( name: "nickname", length: 250)
    val kind = varchar( name: "kind", length: 250)
    val problem = varchar( name: "problem", length: 250)
}

class AnimalDao(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<AnimalDao>(AnimalTable)
    var nickname by AnimalTable.nickname
    var kind by AnimalTable.kind
    var problem by AnimalTable.problem
}
  
```

Рис. 1.1 ORM клас таблиці animal

```

object DoctorTable : IntIdTable( name: "doctor") {
    val hospitalId = reference( name: "hospital_id", HospitalTable.id)
    val name = varchar( name: "name", length: 250)
    val surname = varchar( name: "surname", length: 250)
    val specialization = varchar( name: "specialization", length: 250)
    val birthdate = timestamp( name: "birthdate")
}

class DoctorDao(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<DoctorDao>(DoctorTable)
    var hospitalId by DoctorTable.hospitalId
    var name by DoctorTable.name
    var surname by DoctorTable.surname
    var specialization by DoctorTable.specialization
    var birthdate by DoctorTable.birthdate
}

```

Рис. 1.2 ORM клас таблиці doctor

```

object DoctorHasAnimalTable : Table( name: "doctor_has_animal") {
    val doctorId = reference( name: "doctor_id", DoctorTable.id)
    val animalId = reference( name: "animal_id", AnimalTable.id)
    override val primaryKey: PrimaryKey
        get() = PrimaryKey(DoctorTable.id, AnimalTable.id)
}

```

Рис. 1.3 ORM клас doctor_has_animal

```

object HospitalTable : IntIdTable( name: "hospital") {
    val averageScore = float( name: "average_score")
    val address = varchar( name: "address", length: 250)
}

class HospitalDao(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<HospitalDao>(HospitalTable)
    var averageScore by HospitalTable.averageScore
    var address by HospitalTable.address
}

```

Рис. 1.4 ORM клас таблиці hospital

```

11
12 fun getAllAnimals(): List<Animal> {
13     val animals = mutableListOf<Animal>()
14     transaction { AnimalDao.all().toList().map { animals += it.toAnimal() } }
15     return animals
16 }

```

Рис.1.5 Приклад запиту вилучення всіх даних з таблиці animal

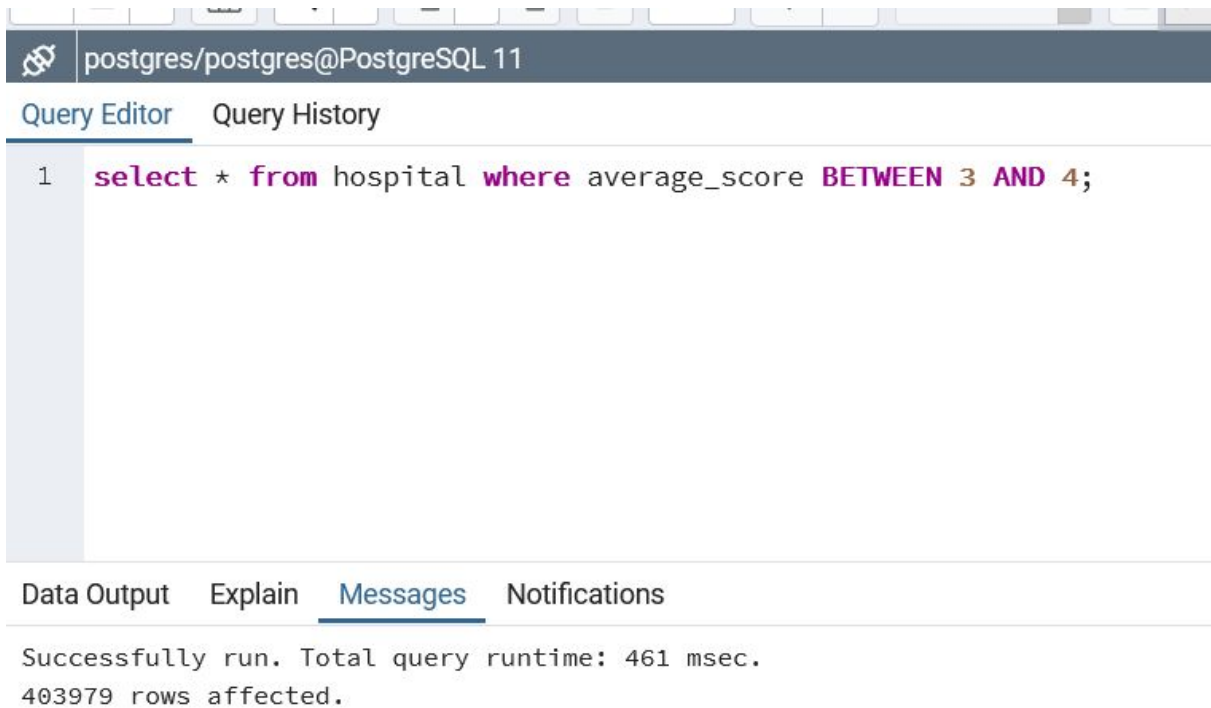
```

fun updateDoctor(
    doctorId: Int,
    hospitalId: Int,
    name: String,
    surname: String,
    specialization: String,
    birthdate: Date
) {
    transaction { this: Transaction
        val hospital = HospitalDao.findById(hospitalId) ?: return@transaction
        DoctorDao.findById(doctorId)?.apply { this: DoctorDao
            this.hospitalId = hospital.id
            this.name = name
            this.surname = surname
            this.specialization = specialization
            this.birthdate = birthdate.toInstant()
        }
    }
}

```

Рис.1.6 Приклад запиту для оновлення даних

BTree Index



The screenshot shows the PostgreSQL Query Editor interface. The top bar indicates the connection is to 'postgres/postgres@PostgreSQL 11'. The 'Query Editor' tab is active, displaying a SQL query: `1 select * from hospital where average_score BETWEEN 3 AND 4;`. Below the query editor, the 'Messages' tab is selected, showing the execution result: 'Successfully run. Total query runtime: 461 msec. 403979 rows affected.'

postgres/postgres@PostgreSQL 11

Query Editor Query History

```
1 select * from hospital where average_score BETWEEN 3 AND 4;
```

Data Output Explain Messages Notifications

Successfully run. Total query runtime: 461 msec.
403979 rows affected.

Рис. 2.1 Швидкість запиту без BTree індексу



The screenshot shows the PostgreSQL Query Editor interface. The 'Query Editor' tab is active, displaying a SQL query: `1 CREATE INDEX average_score_indexed ON hospital using btree (average_score);`. Below the query editor, the 'Messages' tab is selected, showing the execution result: 'CREATE INDEX' and 'Query returned successfully in 828 msec.'

Query Editor Query History

```
1 CREATE INDEX average_score_indexed ON hospital using btree (average_score);
```

Data Output Explain Messages Notifications

CREATE INDEX

Query returned successfully in 828 msec.

Рис. 2.2 Створення BTree індексу

Query Editor

Query History

1

select * from hospital where average_score BETWEEN 3 AND 4;

2

Data Output

Explain

Messages

Notifications

Successfully run. Total query runtime: 217 msec.
403979 rows affected.

Рис. 2.3 Швидкість запиту з BTree індексом

Query Editor

Query History

1

select * from hospital WHERE average_score > 3 order by average_score desc;

Data Output

Explain

Messages

Notifications

Successfully run. Total query runtime: 290 msec.
303874 rows affected.

Рис.2.4 Швидкість запиту без BTree index

```
select * from hospital WHERE average_score > 3 order by average_score desc;
```

Output Explain Messages Notifications

Successfully run. Total query runtime: 186 msec.
374 rows affected.

Рис. 2.5 Швидкість забіту з BTree index

Query Editor Query History

```
1 select * from hospital order by average_score desc;
```

Data Output Explain Messages Notifications

Successfully run. Total query runtime: 661 msec.
1010115 rows affected.

Рис. 2.6 Швидкість забіту без BTree index

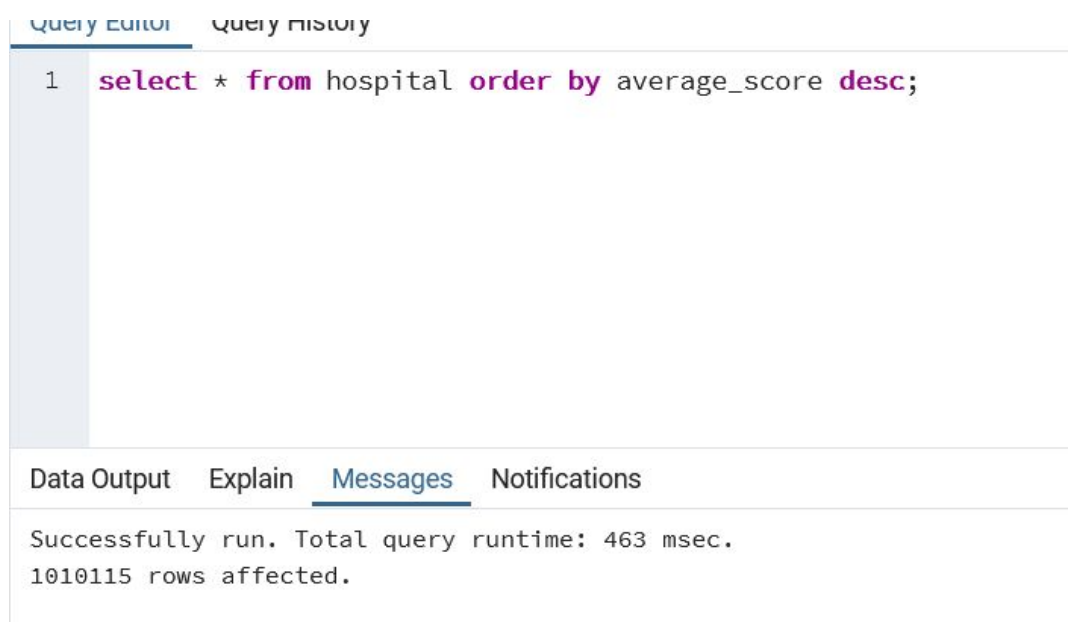


Рис. 2.7 Швидкість забиту BTree index

Висновок: Як бачимо BTree індекс був належно використаний, та дав приріст в швидкості майже в 2 рази у запитах з використанням \geq та \leq та сортуванням. Оптимізації обумовлені використанням такої структури даних, як B-дерева.

Brin Index

The screenshot displays a database interface with two main sections. The top section, titled 'Query Editor', contains a single SQL statement: `1 create index on animal using brin(nickname);`. The bottom section contains tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Messages' tab is selected, showing the output: `CREATE INDEX` and a status message: `Query returned successfully in 934 msec.`

Query Editor Query History

```
1 create index on animal using brin(nickname);
```

Data Output Explain Messages Notifications

CREATE INDEX

Query returned successfully in 934 msec.

Рис. 2.8 Створення Brin Index

Query Editor

Query History

1

`select count(*) from animal where nickname like '%a'`

Data Output

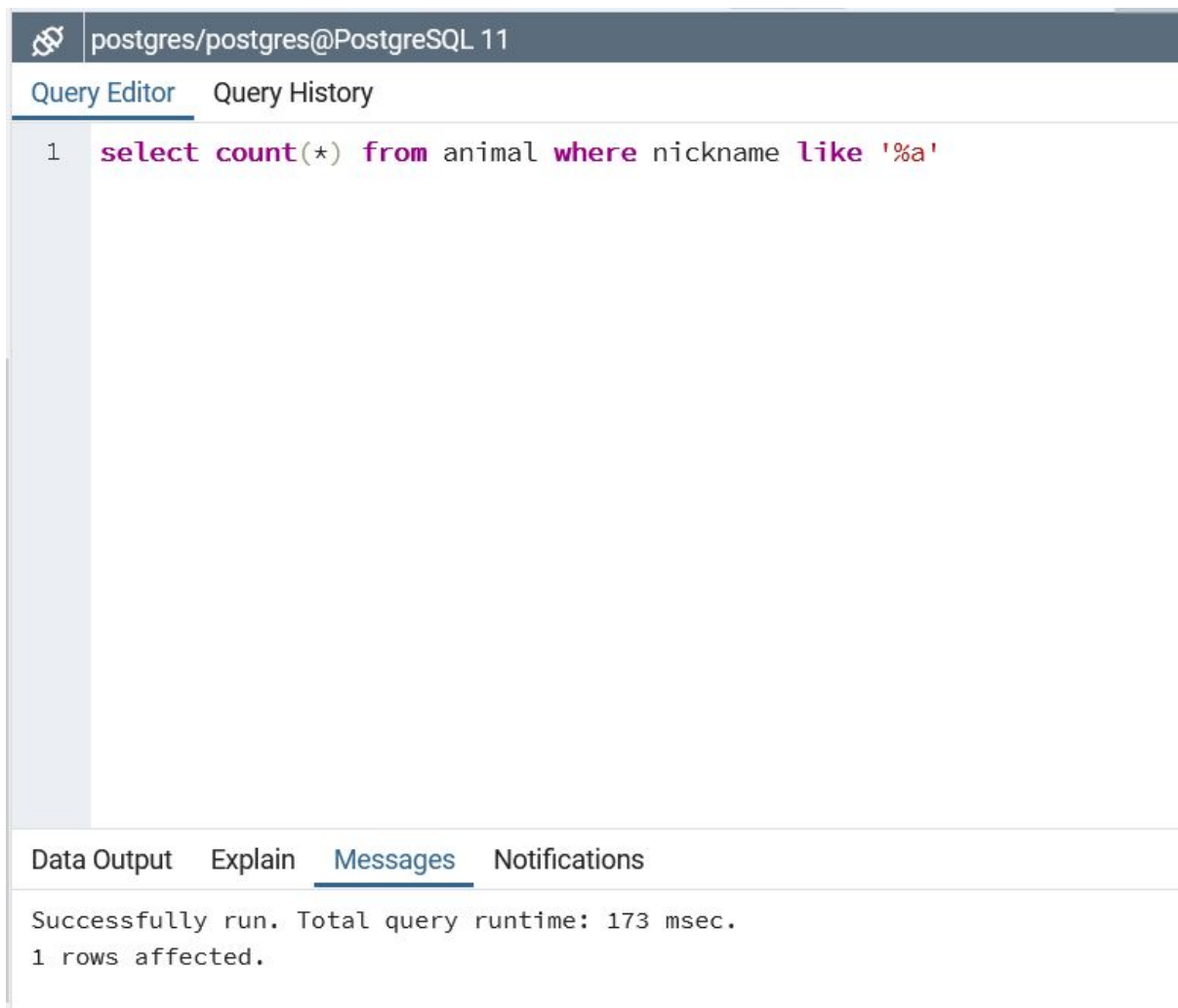
Explain

Messages

Notifications

Successfully run. Total query runtime: 177 msec.
1 rows affected.

Рис. 2.9 Швидкість забиту без Brin index



The screenshot shows a PostgreSQL query editor interface. At the top, a dark blue header bar contains the PostgreSQL logo and the text "postgres/postgres@PostgreSQL 11". Below this, a light blue bar has two tabs: "Query Editor" (which is active) and "Query History". The main area of the "Query Editor" tab contains a single line of SQL code: "1 select count(*) from animal where nickname like '%a'". Below the query editor, there is a horizontal bar with four tabs: "Data Output", "Explain", "Messages" (which is active), and "Notifications". The "Messages" tab displays the execution results: "Successfully run. Total query runtime: 173 msec." and "1 rows affected."

postgres/postgres@PostgreSQL 11

Query Editor Query History

```
1 select count(*) from animal where nickname like '%a'
```

Data Output Explain Messages Notifications

Successfully run. Total query runtime: 173 msec.
1 rows affected.

Рис.2.10 Швидкість забиту з Brin index



Рис. 2.11 Швидкість без Brin Index

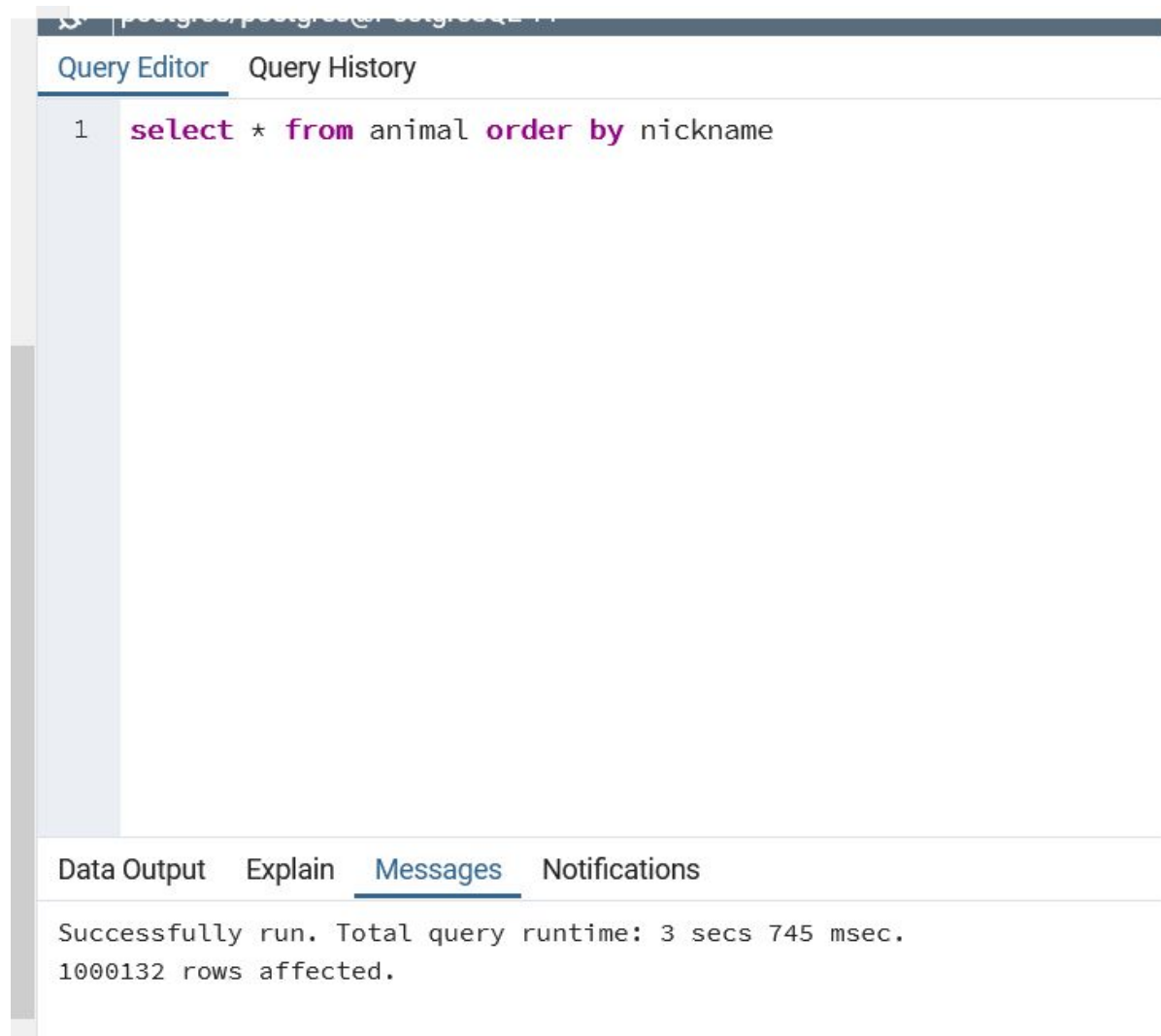


Рис. 2.12 Швидкість з Brin Index

1 `select attname, correlation from pg_stats where tablename='animal' order by correlation desc nulls last`

Data Output

Explain

Messages

Notifications

	attname name	correlation real
1	id	1
2	nickname	0.00303611
3	problem	0.00236794
4	kind	-0.000192222

Рис. 2.13 Дані кореляції полів таблиці animal до id.

Висновок: Як бачимо Brin index на полі nickname не дав ніякого приріст. Очікуваний результат, так як Brin index збільшує швидкість лише на тоді, коли поле корелює зі своїм фізичним розташуванням, тобто в нашому випадку з id.

Before Insert Trigger

```
CREATE OR REPLACE FUNCTION animal_insert_triger_fnc()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS
  $$
BEGIN
  IF NEW.nickname LIKE '%a' THEN
    UPDATE animal SET nickname = 'Trigger Test' WHERE id = 2;
  END IF;

  RETURN NEW;
END;
$$;

CREATE TRIGGER trigger_insert3
  BEFORE INSERT
  ON animal
  FOR EACH ROW
  EXECUTE PROCEDURE animal_insert_triger_fnc();
```

Рис. 3.1 Before insert trigger

```
CREATE OR REPLACE FUNCTION animal_insert_triger_fnc()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS
  $$
BEGIN
  IF NEW.nickname LIKE '%a' THEN
    UPDATE animal SET nickname = 'Trigger Test' WHERE id =
2;
  END IF;

  RETURN NEW;
END;
$$;

CREATE TRIGGER trigger_insert3
  BEFORE INSERT
  ON animal
  FOR EACH ROW
  EXECUTE PROCEDURE animal_insert_triger_fnc();
```

```

1 INSERT INTO animal(nickname, kind, problem) VALUES('TESa', 'TEST', 'TEST');
2 SELECT * FROM animal WHERE id < 5 OR id > 1000155

```

Data Output

	id [PK] integer	nickname character varying (100)	kind character varying (100)	problem character varying (100)
1	3	Bahira	dog	rage
2	1	mg39mf022	dog	hurt legs
3	2	Trigger Test	cat	we23r3unf
4	1000157	98f1370821	4e8e9010f4	3114b04a77
5	1000168	9bf31c7ff0	a4c90653c8	CANT BE DELETED
6	1000174	AEST	TEST	CANT BE DELETED
7	1000175	TaST	TEST	CANT BE DELETED
8	1000176	TESa	TEST	TEST

Рис. 3.2 Запит відповідає умові триггеру

```

1 INSERT INTO animal(nickname, kind, problem) VALUES('TEST', 'TEST', 'TEST');
2 SELECT * FROM animal WHERE id < 5 OR id > 1000155

```

Data Output

	id [PK] integer	nickname character varying (100)	kind character varying (100)	problem character varying (100)
	3	Bahira	dog	rage
	1	mg39mf022	dog	hurt legs
	2	wmf3829mff	cat	we23r3unf
	1000157	98f1370821	4e8e9010f4	3114b04a77
	1000168	9bf31c7ff0	a4c90653c8	CANT BE DELETED
	1000179	AEST	TEST	CANT BE DELETED
	1000180	TEST	TEST	CANT BE DELETED
	1000181	TEST	TEST	TEST

Рис. 3.3 Запит не відповідає умові триггеру

After Delete Trigger

```
1 CREATE OR REPLACE FUNCTION animal_delete_trigger_fnc()
2 RETURNS TRIGGER
3 LANGUAGE PLPGSQL
4 AS
5 $$
6 BEGIN
7     IF MOD(OLD.id, 2) = 0 THEN
8         INSERT INTO animal (nickname, kind, problem) VALUES(OLD.nickname, OLD.kind, 'CANT BE DELETED');
9     END IF;
10
11     RETURN NEW;
12 END;
13 $$;
14
15 CREATE TRIGGER trigger_delete
16 AFTER DELETE
17 ON animal
18 FOR EACH ROW
19 EXECUTE PROCEDURE animal_delete_trigger_fnc();
```

Рис. 3.4 After delete trigger

```
CREATE OR REPLACE FUNCTION
animal_delete_trigger_fnc()
    RETURNS TRIGGER
    LANGUAGE PLPGSQL
    AS
    $$
BEGIN
    IF MOD(OLD.id, 2) = 0 THEN
        INSERT INTO animal (nickname, kind, problem)
VALUES(OLD.nickname, OLD.kind, 'CANT BE DELETED');
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trigger_delete
    AFTER DELETE
    ON animal
    FOR EACH ROW
    EXECUTE PROCEDURE animal_delete_trigger_fnc();
```

```
SELECT * FROM animal ORDER BY id DESC
```

Data Output

	id [PK] integer	nickname character varying (100)	kind character varying (100)	problem character varying (100)
1	1000157	98f1370821	4e8e9010f4	3114b04a77
2	1000155	6f4922f455	6ffc9128ae	dc40ee5596
3	1000152	9bf31c7ff0	a4c90653c8	0a954aeb65
4	1000150	c51ce410c1	5025e6d4ff	d08231d28d
5	1000149	c20ad4d76f	1c09264eda	9a8125e780
6	1000148	6512bd43d9	0d33f3bfe2	75e0e62614
7	1000147	d3d9446802	e01ebf927b	795d9f59e5
8	1000146	45c48cce2e	821d8b9de5	2454947d4f
9	1000145	c9f0f895fb	184efddc24	d86ffe99eb
10	1000144	8f14e45fce	8a306a6aa4	3cf36d97da
11	1000143	1679091c5a	1a07a9d721	a6e664f2f2

Рис. 3.5 Початковий стан таблиці

```
1 DELETE FROM animal where id = 1000152;
2 SELECT * FROM animal ORDER BY id DESC
```

Data Output

	id [PK] integer	nickname character varying (100)	kind character varying (100)	problem character varying (100)
1	1000168	9bf31c7ff0	a4c90653c8	CANT BE DELETED
2	1000157	98f1370821	4e8e9010f4	3114b04a77
3	1000155	6f4922f455	6ffc9128ae	dc40ee5596
4	1000150	c51ce410c1	5025e6d4ff	d08231d28d
5	1000149	c20ad4d76f	1c09264eda	9a8125e780
6	1000148	6512bd43d9	0d33f3bfe2	75e0e62614
7	1000147	d3d9446802	e01ebf927b	795d9f59e5
8	1000146	45c48cce2e	821d8b9de5	2454947d4f
9	1000145	c9f0f895fb	184efddc24	d86ffe99eb
10	1000144	8f14e45fce	8a306a6aa4	3cf36d97da
11	1000143	1679091c5a	1a07a9d721	a6e664f2f2

Рис. 3.6 Запит відповідає умові тригера

1 DELETE FROM animal where id = 1000155;

2 SELECT * FROM animal ORDER BY id DESC

Data Output

	<div>id</div> <div>[PK] integer</div>	<div>nickname</div> <div>character varying (100)</div>	<div>kind</div> <div>character varying (100)</div>	<div>problem</div> <div>character varying (100)</div>	
1	1000168	9bf31c7ff0	a4c90653c8	CANT BE DELETED	
2	1000157	98f1370821	4e8e9010f4	3114b04a77	
3	1000150	c51ce410c1	5025e6d4ff	d08231d28d	
4	1000149	c20ad4d76f	1c09264eda	9a8125e780	
5	1000148	6512bd43d9	0d33f3bfe2	75e0e62614	
6	1000147	d3d9446802	e01ebf927b	795d9f59e5	
7	1000146	45c48cce2e	821d8b9de5	2454947d4f	
8	1000145	c9f0f895fb	184efddc24	d86ffe99eb	
9	1000144	8f14e45fce	8a306a6aa4	3cf36d97da	
10	1000143	1679091c5a	1a07a9d721	a6e664f2f2	
11	1000142	e4da3b7fbb	6866f96892	8dbc8f34ce	
12	1000141	a87ff679a2	7531cd0612	76d23df481	

Рис. 3.7 Запит не відповідає умові тригера