



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря  
Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих  
комп’ютерних систем**

**Лабораторна робота №3**

з дисципліни  
**«Бази даних і засоби управління»**

**Тема: «Засоби оптимізації роботи СУБД  
PostgreSQL»**

Виконав: студент III курсу  
ФПМ групи КВ-84

Нігматшаєв М. А.

Перевірив:

Київ – 2020

**Завдання роботи:**

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

**Посилання на репозиторій**

<https://github.com/mnihmatshaiev/db-and-ms>

**Вимоги до оформлення лабораторної роботи у електронному вигляді**

Опис та вміст репозиторію лабораторної роботи у **репозиторії GitHub** включає: оновлені файли додатку, назву лабораторної роботи та варіант студента.

Звіт лабораторної роботи має містити:

- a. титульний аркуш затвердженого зразка;
- b. завдання на лабораторну роботу з обов’язковим наведенням варіанту студента;
- c. копії екрану (скріншоти), що **підтверджують вимоги 1-3 завдання**, а також:
  - для завдання №1: схему бази даних у вигляді таблиць і зв’язків між ними, а також класи ORM і зв’язки між ними, що відповідають таблицям бази даних. Навести приклади запитів у вигляді ORM.
  - для завдання №2: команди створення індексів, тексти, результати і час виконання запитів SQL, пояснити чому індекси прискорюють (або не прискорюють) швидкість виконання запитів.
  - для завдання №3: команди, що ініціюють виконання тригера, текст тригера та скріншоти зі змінами у таблицях бази даних;

**Варіант 20:**

20	GIN, BRIN	after insert, update
----	-----------	----------------------

## *Пункт №1 завдання:*

Класи ORM і зв'язки між ними:

```
class Film(Base):
    __tablename__ = 'film'
    film_id = Column(Integer, primary_key=True)
    film_name = Column(String(32))
    show = relationship('Show')

    def __repr__(self):
        return "<(film_id='{}', name='{}')>" \
            .format(self.film_id, self.film_name)

class Hall(Base):
    __tablename__ = 'hall'
    hall_name = Column(String(8))
    hall_id = Column(Integer, primary_key=True)
    capacity = Column(Integer)
    show = relationship("Show")

    def __repr__(self):
        return "<(hall_id='{}', hall_name='{}', capacity='{}')>" \
            .format(self.hall_id, self.hall_name, self.capacity)
```

```

class Show(Base):
    __tablename__ = 'show'
    show_id = Column(Integer, primary_key=True)
    hall_id = Column(Integer, ForeignKey('hall.hall_id'))
    film_id = Column(Integer, ForeignKey('film.film_id'))
    start_date = Column(Date)

    def __repr__(self):
        return "<(show_id='{}', hall_id='{}', film_id='{}', start_date='{}')>".format(self.show_id, self.hall_id, self.film_id, self.start_date)

class Ticket(Base):
    __tablename__ = 'ticket'
    show_id = Column(Integer, ForeignKey('show.show_id'), primary_key=True)
    row = Column(Integer, primary_key=True)
    seat = Column(Integer, primary_key=True)

    def __repr__(self):
        return "<(show_id='{}', row='{}', seat='{}')>".format(self.show_id, self.row, self.seat)

```

Приклади запитів у вигляді ORM:

Виконання запитів:

Додавання:

```

Hall - press 2
Show - press 3
Ticket - press 4
1
Enter film name
exmpl4
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit

4
Choose table: Film - press 1 Hall - press 2 Show - press 3 Ticket - press 4
1
<(film_id='100005', name='Exmpl1'>
<(film_id='100006', name='Exmpl2'>
<(film_id='100007', name='exmpl3'>
<(film_id='100008', name='exmpl4'>
Enter:

```

Редагування:

```
Choose table: Film - press 1 Hall - press 2 Show - press 3 Ticket - press 4
2
<(hall_id='100003', hall_name='red', capacity='60'>
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit

3
Choose table: I
Film - press 1
Hall - press 2
Show - press 3
Ticket - press 4
2
Enter hall name
blue
Enter capacity
70
```

```
Choose table: I
Ticket - press 4
2
Enter hall name
blue
Enter capacity
70
Enter the id of the hall you want to edit
100003
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit

4
Choose table: Film - press 1 Hall - press 2 Show - press 3 Ticket - press 4
2
<(hall_id='100003', hall_name='blue', capacity='70'>
Enter:
1 - create
```

## Видведення:

```
Choose table: Film - press 1 Hall - press 2 Show - press 3 Ticket - press 4
4
<(show_id='1099523', row='2', seat='26'>
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit

2
Choose table: I
Film - press 1
Hall - press 2
Show - press 3
Ticket - press 4
4
Enter the id of the show, row and set you want to delete
1099523
2
26
```

```

1099523
2
26
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit

4
Choose table: Film - press 1 Hall - press 2 Show - press 3 Ticket - press 4
4
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit

```

Обробка помилок(спроба додати film\_id, hall\_id яких немає в первісних таблицях):

```

4
Choose table:
Film - press 1
Hall - press 2
Show - press 3
Ticket - press 4
3
Enter film id
1
Enter hall id
1
Enter start date
2020-09-08
Error
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit

```

## Пункт №2 завдання:

*Команди створення індексів, тексти, результати і час виконання запитів SQL:*

*GIN index:*

Таблиця для тестування індексу

Data Output			
	<b>Id</b> [PK] integer	<b>text</b> text	<b>vector</b> tsvector
1	1	RLY	'rlf':1
2	2	MSR	'msr':1
3	3	LAM	'lam':1
4	4	VXY	'vxf':1
5	5	CUF	'cuf':1
6	6	EDB	'edb':1
7	7	OKD	'okd':1

Successfully run. Total query runtime: 474 msec. 100000 rows affected.

Результати для запитів без використання індексу:

Query Editor

```
1 explain analyze select * from task2_for_gin where vector @@ to_tsquery('rop:*');
```

Data Output

Step	Operation	Cost	Time	Rows	Width	Loops
1	Gather	1000.00..17619.88	actual time=86.735..119.953	7	24	1
2	Workers Planned:	1				
3	Workers Launched:	1				
4	-> Parallel Seq Scan on task2_for_gin	(cost=0.00..16619.18)	actual time=85.351..112.156	4	24	2
5	Filter: (vector @@ to_tsquery('rop.*::text'))					
6	Rows Removed by Filter:	49998				
7	Planning Time:	0.099 ms				
8	Execution Time:	119.972 ms				

Query Editor

```
1 explain analyze select count(vector) from task2_for_gin where vector @@ to_tsquery('rop:*');
```

Data Output

Step	Operation	Cost	Time	Rows	Width	Loops
1	Finalize Aggregate	17619.30..17619.31	actual time=74.762..76.608	1	8	1
2	-> Gather	(cost=17619.19..17619.30)	actual time=74.605..76.602	1	8	2
3	Workers Planned:	1				
4	Workers Launched:	1				
5	-> Partial Aggregate	(cost=16619.19..16619.20)	actual time=70.594..70.594	1	8	2
6	-> Parallel Seq Scan on task2_for_gin	(cost=0.00..16619.18)	actual time=51.820..70.569	4	16	2
7	Filter: (vector @@ to_tsquery('rop.*::text'))					
8	Rows Removed by Filter:	49998				
9	Planning Time:	0.155 ms				
10	Execution Time:	76.650 ms				

Query Editor

```
1 explain analyze select * from task2_for_gin where vector @@ to_tsquery('rop:*') order by vector;
```

Data Output

Step	Operation	Cost	Time	Rows	Width	Loops
1	Gather Merge	17619.23..17619.69	actual time=86.072..88.618	4	24	1
2	Workers Planned:	1				
3	Workers Launched:	1				
4	-> Sort	(cost=16619.22..16619.23)	actual time=82.817..82.817	4	24	2
5	Sort Key: vector					
6	Sort Method: quicksort Memory: 25kB					
7	Worker 0: Sort Method: quicksort Memory: 25kB					
8	-> Parallel Seq Scan on task2_for_gin	(cost=0.00..16619.18)	actual time=66.771..82.714	4	24	2
9	Filter: (vector @@ to_tsquery('rop.*::text'))					
10	Rows Removed by Filter:	49998				
11	Planning Time:	194.268 ms				

Successfully run. Total query runtime: 567 msec. 12 rows affected.

```

1 explain analyze select sum(id) from task2_for_gin where vector @@ to_tsquery('rop:*')

```

**Data Output**

QUERY PLAN  
text

- Finalize Aggregate (cost=17619.30..17619.31 rows=1 width=8) (actual time=80.305..82.187 rows=1 loops=1)
 -> Gather (cost=17619.19..17619.30 rows=1 width=8) (actual time=80.189..82.182 rows=2 loops=1)
 Workers Planned: 1
 Workers Launched: 1
 -> Partial Aggregate (cost=16619.19..16619.20 rows=1 width=8) (actual time=78.212..78.213 rows=1 loops=2)
 -> Parallel Seq Scan on task2\_for\_gin (cost=0.00..16619.18 rows=4 width=4) (actual time=61.781..78.202 rows=2 loops=2)
 Filter: (vector @@ to\_tsquery('rop:\*'::text))
 Rows Removed by Filter: 49998
 Planning Time: 0.265 ms
 Execution Time: 82.210 ms

## Створення індексу:

```

1 create index on task2_for_gin using gin(vector)

```

**Messages**

CREATE INDEX

Query returned successfully in 455 msec.

✓ Query returned successfully in 455 msec.

## Запити з використанням індексу

```

1 explain analyze select * from task2_for_gin where vector @@ to_tsquery('rop:*');

```

**Data Output**

QUERY PLAN  
text

- Bitmap Heap Scan on task2\_for\_gin (cost=20.30..48.52 rows=7 width=24) (actual time=0.057..0.062 rows=5 loops=1)
 -> Recheck Cond: (vector @@ to\_tsquery('rop'\*::text))
 -> Heap Blocks: exact=5
 -> Bitmap Index Scan on task2\_for\_gin\_vector\_idx (cost=0.00..20.30 rows=7 width=0) (actual time=0.054..0.054 rows=5 loops=1)
 Index Cond: (vector @@ to\_tsquery('rop'\*::text))
 Planning Time: 0.314 ms
 Execution Time: 0.238 ms

Query Editor

```
1 explain analyze select count(vector) from task2_for_gin where vector @@ to_tsquery('rop:*');
```

Data Output

QUERY PLAN

```
text
1 Aggregate (cost=48.54..48.55 rows=1 width=8) (actual time=0.025..0.025 rows=1 loops=1)
2   -> Bitmap Heap Scan on task2_for_gin (cost=20.30..48.52 rows=7 width=16) (actual time=0.017..0.022 rows=5 loops=1)
3     Recheck Cond: (vector @@ to_tsquery('rop*::text'))
4     Heap Blocks: exact=5
5     -> Bitmap Index Scan on task2_for_gin_vector_idx (cost=0.00..20.30 rows=7 width=8) (actual time=0.013..0.013 rows=5 loops=1)
6       Index Cond: (vector @@ to_tsquery('rop*::text'))
7   Planning Time: 0.103 ms
8   Execution Time: 0.043 ms
```

Query Editor

```
1 explain analyze select * from task2_for_gin where vector @@ to_tsquery('rop:*') order by vector;
```

Data Output

QUERY PLAN

```
text
1 Sort (cost=48.62..48.64 rows=7 width=24) (actual time=0.028..0.029 rows=5 loops=1)
2 Sort Key: vector
3 Sort Method: quicksort Memory: 25kB
4   -> Bitmap Heap Scan on task2_for_gin (cost=20.30..48.52 rows=7 width=24) (actual time=0.018..0.023 rows=5 loops=1)
5     Recheck Cond: (vector @@ to_tsquery('rop*::text'))
6     Heap Blocks: exact=5
7     -> Bitmap Index Scan on task2_for_gin_vector_idx (cost=0.00..20.30 rows=7 width=0) (actual time=0.014..0.014 rows=5 loops=1)
8       Index Cond: (vector @@ to_tsquery('rop*::text'))
9   Planning Time: 0.127 ms
10  Execution Time: 0.045 ms
```

Query Editor

```
1 explain analyze select sum(id) from task2_for_gin where vector @@ to_tsquery('rop:*)
```

Data Output

QUERY PLAN

```
text
1 Aggregate (cost=48.54..48.55 rows=1 width=8) (actual time=0.026..0.027 rows=1 loops=1)
2   -> Bitmap Heap Scan on task2_for_gin (cost=20.30..48.52 rows=7 width=4) (actual time=0.018..0.023 rows=5 loops=1)
3     Recheck Cond: (vector @@ to_tsquery('rop*::text'))
4     Heap Blocks: exact=5
5     -> Bitmap Index Scan on task2_for_gin_vector_idx (cost=0.00..20.30 rows=7 width=0) (actual time=0.014..0.014 rows=5 loops=1)
6       Index Cond: (vector @@ to_tsquery('rop*::text'))
7   Planning Time: 0.105 ms
8   Execution Time: 0.044 ms
```

## BRIN index:

Таблиця для тестування індексу

Data Output

	<b>id</b> [PK] integer	<b>time</b> timestamp without time zone
6	10008768	2020-01-01 00:05:00
7	10008769	2020-01-01 00:06:00
8	10008770	2020-01-01 00:07:00
9	10008771	2020-01-01 00:08:00
10	10008772	2020-01-01 00:09:00
11	10008773	2020-01-01 00:10:00
12	10008774	2020-01-01 00:11:00

Messages

```
Successfully run. Total query runtime: 26 secs 111 msec.
525601 rows affected.
```

## Приклади запитів без використання індексу:

laba3/postgres@LocalServer

Query Editor

```
1 explain analyze select * from task2 where time = '2020-08-30'
```

Data Output

QUERY PLAN

- Seq Scan on task2 (cost=0.00..56943.00 rows=1 width=12) (actual time=352.343..614.104 rows=1 loops=1)
  - Filter: ("time" = '2020-08-30 00:00:00'::timestamp without time zone)
  - Rows Removed by Filter: 525600
- Planning Time: 0.070 ms
- Execution Time: 614.124 ms

laba3/postgres@LocalServer

Query Editor

```
1 explain analyze select count(id) from task2 where time > '2020-08-29'
```

Data Output

QUERY PLAN

- Finalize Aggregate (cost=60901.41..60901.42 rows=1 width=8) (actual time=157.257..164.636 rows=1 loops=1)
  - >- Gather (cost=60901.20..60901.41 rows=2 width=8) (actual time=156.901..164.607 rows=3 loops=1)
    - Workers Planned: 2
    - Workers Launched: 2
  - >- Partial Aggregate (cost=59901.20..59901.21 rows=1 width=8) (actual time=128.850..128.854 rows=1 loops=3)
    - >- Parallel Seq Scan on task2 (cost=0.00..59722.93 rows=71309 width=4) (actual time=17.803..122.996 rows=59520 loops=3)
      - Filter: ("time" > '2020-08-29 00:00:00'::timestamp without time zone)
      - Rows Removed by Filter: 115680
  - Planning Time: 0.116 ms
  - Execution Time: 164.693 ms

Query Editor

```
1 explain analyze select id from task2 where time > '2020-08-29' order by id desc
```

Data Output

QUERY PLAN

- Sort (cost=80832.61..81260.46 rows=171141 width=4) (actual time=236.017..251.262 rows=178560 loops=1)
  - Sort Key: id DESC
  - Sort Method: external merge Disk: 2464kB
  - >- Seq Scan on task2 (cost=0.00..63614.82 rows=171141 width=4) (actual time=34.383..190.357 rows=178560 loops=1)
    - Filter: ("time" > '2020-08-29 00:00:00'::timestamp without time zone)
    - Rows Removed by Filter: 347041
  - Planning Time: 0.072 ms
  - Execution Time: 256.379 ms

Query Editor

```
1 explain analyze select sum(id) from task2 where time > '2020-08-29'
```

Data Output

QUERY PLAN  
text

```
1 Finalize Aggregate (cost=60901.41..60901.42 rows=1 width=8) (actual time=132.607..136.697 rows=1 loops=1)
2   -> Gather (cost=60901.20..60901.41 rows=2 width=8) (actual time=132.459..136.684 rows=3 loops=1)
3     Workers Planned: 2
4     Workers Launched: 2
5       -> Partial Aggregate (cost=59901.20..59901.21 rows=1 width=8) (actual time=112.755..112.756 rows=1 loops=3)
6         -> Parallel Seq Scan on task2 (cost=0.00..59722.93 rows=71309 width=4) (actual time=14.220..109.247 rows=59520 loops=3)
7           Filter: ("time" > '2020-08-29 00:00:00'::timestamp without time zone)
8           Rows Removed by Filter: 115680
9     Planning Time: 5.347 ms
10    Execution Time: 136.723 ms
```

## Створення індексу:

Dashboard Properties Statistics SQL Dependencies Dependents laba3/postgres... public.table1/l... laba3/postgres... { < > x

laba3/postgres@LocalServer

Query Editor

```
1 CREATE INDEX brin_index ON task2 USING BRIN (time) WITH (pages_per_range = 128);
```

Messages

```
CREATE INDEX
```

Query returned successfully in 417 msec.

## Результати запитів з використанням індексу:

laba3/postgres@LocalServer

Query Editor

```
1 explain analyze select * from task2 where time = '2020-08-29'
```

Data Output

QUERY PLAN  
text

```
1 Bitmap Heap Scan on task2 (cost=12.03..4200.31 rows=1 width=12) (actual time=0.832..2.555 rows=1 loops=1)
2   Recheck Cond: ("time" = '2020-08-29 00:00:00'::timestamp without time zone)
3   Rows Removed by Index Recheck: 23679
4   Heap Blocks: lossy=128
5   -> Bitmap Index Scan on brin_index (cost=0.00..12.03 rows=1181 width=0) (actual time=0.060..0.060 rows=1280 loops=1)
6     Index Cond: ("time" = '2020-08-29 00:00:00'::timestamp without time zone)
7   Planning Time: 0.045 ms
8   Execution Time: 2.573 ms
```

Query Editor

```
1 explain analyze select count(id) from task2 where time > '2020-08-29'
```

Data Output

	QUERY PLAN
1	Aggregate (cost=59534.29..59534.30 rows=1 width=8) (actual time=25.968..25.970 rows=1 loops=1)
2	-> Bitmap Heap Scan on task2 (cost=58.71..59112.97 rows=168529 width=4) (actual time=0.701..18.134 rows=178560 loops=1)
3	Recheck Cond: ("time" > '2020-08-29 00:00:00'::timestamp without time zone)
4	Rows Removed by Index Recheck: 7642
5	Heap Blocks: lossy=1007
6	-> Bitmap Index Scan on brin_Index (cost=0.00..16.58 rows=168901 width=0) (actual time=0.072..0.072 rows=10240 loops=1)
7	Index Cond: ("time" > '2020-08-29 00:00:00'::timestamp without time zone)
8	Planning Time: 0.069 ms
9	Execution Time: 25.992 ms

Successfully run. Total query runtime: 860 ms. 0 rows affected.

Query Editor

```
1 explain analyze select id from task2 where time > '2020-08-29' order by id desc
```

Data Output

	QUERY PLAN
1	Sort (cost=76050.01..76471.33 rows=168529 width=4) (actual time=69.450..84.500 rows=178560 loops=1)
2	Sort Key: id DESC
3	Sort Method: external merge Disk: 2464kB
4	-> Bitmap Heap Scan on task2 (cost=58.71..59112.97 rows=168529 width=4) (actual time=0.710..22.502 rows=178560 loops=1)
5	Recheck Cond: ("time" > '2020-08-29 00:00:00'::timestamp without time zone)
6	Rows Removed by Index Recheck: 7642
7	Heap Blocks: lossy=1007
8	-> Bitmap Index Scan on brin_Index (cost=0.00..16.58 rows=168901 width=0) (actual time=0.072..0.073 rows=10240 loops=1)
9	Index Cond: ("time" > '2020-08-29 00:00:00'::timestamp without time zone)
10	Planning Time: 0.057 ms
11	Execution Time: 89.794 ms

Query Editor

```
1 explain analyze select sum(id) from task2 where time > '2020-08-29'
```

Data Output

	QUERY PLAN
1	Aggregate (cost=59534.29..59534.30 rows=1 width=8) (actual time=28.892..28.893 rows=1 loops=1)
2	-> Bitmap Heap Scan on task2 (cost=58.71..59112.97 rows=168529 width=4) (actual time=1.065..20.067 rows=178560 loops=1)
3	Recheck Cond: ("time" > '2020-08-29 00:00:00'::timestamp without time zone)
4	Rows Removed by Index Recheck: 7642
5	Heap Blocks: lossy=1007
6	-> Bitmap Index Scan on brin_Index (cost=0.00..16.58 rows=168901 width=0) (actual time=0.118..0.118 rows=10240 loops=1)
7	Index Cond: ("time" > '2020-08-29 00:00:00'::timestamp without time zone)
8	Planning Time: 0.084 ms
9	Execution Time: 28.924 ms

## Висновки:

В обох випадка використання індексів пришвидшує виконання запитів.

Використання Brin виявилося ефективним адже в таблиці понад 500000 рядків і всі проіндексовані значення є відсортованими. Цей індекс використовується для великих таблиць і прискорює пошук лише в випадку відсортованості або часткової відсортованості даних. Індекс

дозволяє пропускати великі розділи таблиці, які не можуть містити збігаються значення і характеризується не великими витратами пам'яті.

Для індексу GIN створено таблицю на 100000 рядків. Gin доцільно використовувати для прискорення повнотекстового пошуку, при цьому індексуються не власними значення, а окремі елементи; кожен елемент посилається на ті значення, в яких він зустрічається, що можна порівняти з алфавітним покажчиком.

### **Пункт №3 завдання:**

#### **Текст тригера:**

```
Query Editor
1 CREATE OR REPLACE FUNCTION function() RETURNS trigger AS $$ 
2 BEGIN
3     IF (NEW.value > 100000) THEN
4         RAISE EXCEPTION 'VALUE TOO MUCH';
5     END IF;
6     IF (TG_OP = 'INSERT') THEN
7         INSERT INTO table2 VALUES(NEW.id, NEW.value, 'INSERT', now());
8         RETURN NEW;
9     ELSIF (TG_OP = 'UPDATE') THEN
10        declare
11            change_row record;
12            begin
13                FOR change_row IN (SELECT * FROM table2) LOOP
14                    IF(change_row.id = OLD.id) THEN
15                        UPDATE table2 SET value = NEW.value, request = 'UPDATE', date = now() WHERE id = OLD.id;
16                    END IF;
17                END LOOP;
18            end;
19            RETURN NEW;
20        END IF;
21        RETURN NULL;
22    END; $$ 
23 LANGUAGE plpgsql;
```

Після додавання даних до таблиці table1 тригер записує данні в таблицю table2, а також додає дату і тип виконання запиту. При редагуванні оновлюються данні з відповідним id в таблиці table2, а також дата і відмітка про виконання запиту(UPDATE). Якщо значення більше 100000, видається повідомлення про помилку.

#### **Команди, що ініціюють виконання тригера та зміни в таблицях після їх виконання:**

вставка:

```
24 --CREATE TRIGGER trigger_for_insert_update
25 --AFTER INSERT OR UPDATE ON table1 FOR EACH ROW
26 --EXECUTE PROCEDURE function();
27 --INSERT INTO table1 (id, value) VALUES (7, 66);
28 SELECT * FROM table2
```

Data Output				
	id [PK] integer	value integer	request character varying (30)	date date
1	7	66	INSERT	2020-12-28

редагування:

```
25 --drop trigger trigger_for_insert_update on table1;
26 --drop function function();
27 --CREATE TRIGGER trigger_for_insert_update
28 --AFTER INSERT OR UPDATE ON table1 FOR EACH ROW
29 --EXECUTE PROCEDURE function();
30 UPDATE table1 SET value = 0 WHERE id = 7;
31 SELECT * FROM table2
```

Messages

Successfully run. Total query runtime: 861 msec.  
1 rows affected.

Data Output				
	id [PK] integer	value integer	request character varying (30)	date date
1	7	0	UPDATE	2020-12...

## RAISE EXCEPTION:

```
--drop trigger trigger_for_insert_update on table1;
--drop function function();
--CREATE TRIGGER trigger_for_insert_update
--AFTER INSERT OR UPDATE ON table1 FOR EACH ROW
--EXECUTE PROCEDURE function();
UPDATE table1 SET value = 100001 WHERE id = 7;
SELECT * FROM table2
```

Messages

ERROR: VALUE TOO MUCH  
KQHTEKCT: PL/pgSQL function function() line 4 at RAISE  
SQL state: P0001