

**Задание:**

Решить проблему рассогласованности спец. таблицы и отслеживаемой таблицы в случае, если в последнюю была проведена вставка в обход разработанной ранее хранимой процедуры (ХП) новой записи с идентификатором, превышающим предыдущий максимум, или было проведено обновление идентификаторов, приведшее к превышению максимума. Для этого автоматически создавать триггер для таблицы, имя которой передаётся в ХП в качестве параметра для каждой новой уникальной пары 'имя таблицы' + 'имя столбца', который только в описанных выше случаях превышения обновляет соответствующее текущее максимальное значение в спец. таблице. Имя триггера, помимо осмысленной части, должно содержать GUID.

**Реализация:**

Т.к для формирования названия триггера нам понадобится функция для генерации uuid, объявляем функцию `uuid_generate_v1mc()`. Я не стал подключать весь модуль “`uuid-oss`”, а добавил одну определенную `uuid` функцию

```
--функция uuid_generate_v1mc из модуля uuid-oss
```

```
CREATE OR REPLACE FUNCTION public.uuid_generate_v1mc()
```

```
RETURNS uuid
```

```
LANGUAGE 'c'
```

```
COST 1
```

```
VOLATILE STRICT PARALLEL SAFE
```

```
AS '$libdir/uuid-oss', 'uuid_generate_v1mc';
```

```
ALTER FUNCTION public.uuid_generate_v1mc()
```

```
OWNER TO postgres;
```

▼ Functions (1)

{=} uuid\_generate\_v1mc()

И создал триггерную функцию, которая будет обновлять таблицу spec по необходимости. В этой функции будет два параметра: название таблицы(tg\_argv[0]) и название столбца(tg\_argv[1]). В случае, когда в рассматриваемую таблицу в рассматриваемый столбец записали значение, большее чем в столбце cur\_max\_value у таблицы spec, таблица spec будет обновляться автоматически.

```
--создаем триггерную функцию
```

```
CREATE OR REPLACE FUNCTION upd_spec_maxvalue()
```

```
RETURNS trigger AS
```

```
$$
```

```
DECLARE
```

```
    maxValue integer;
```

```
BEGIN
```

```
    EXECUTE format('select max(%s) from %s', tg_argv[1], tg_argv[0]) INTO maxValue;
```

```
    UPDATE spec
```

```
    SET cur_max_value = maxValue
```

```
    WHERE table_name = tg_argv[0] AND column_name = tg_argv[1] AND maxValue >
    cur_max_value;
```

```
    RETURN NULL;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

Так же пришлось немного изменить хранимую процедуру с прошлой л.р., а именно добавить создание триггеров для рассматриваемых таблиц для операций изменения и вставки. В конечном итоге хп стала выглядеть следующим образом

```
--наша хранимая процедура, в которую добавили создание тригера
```

```
CREATE OR REPLACE FUNCTION xp (_table_name text, _column_name text)
```

```
RETURNS integer
```

```
AS $$
```

```
DECLARE
```

```
    maxValue integer := 0;
```

```
BEGIN
```

```
    IF
```

```
        (SELECT COUNT(*)
```

```
FROM spec
```

```
WHERE column_name = _column_name AND table_name = _table_name) > 0
```

```
THEN
```

```
UPDATE spec
```

```
SET cur_max_value = cur_max_value + 1
```

```
WHERE column_name = _column_name AND table_name = _table_name;
```

```
RETURN cur_max_value FROM spec
```

```
WHERE column_name = _column_name AND table_name = _table_name;
```

```
ELSE
```

```
EXECUTE format('SELECT MAX(%s)
```

```
FROM %s ',
```

```
_column_name, _table_name)
```

```
INTO maxValue;
```

```
IF maxValue IS null THEN maxValue := 1; ELSE maxValue := maxValue + 1; END IF;
```

```
EXECUTE format('INSERT INTO spec
```

```
VALUES (%s, "%s", "%s", %s)',
```

```
(SELECT xp('spec', 'id')), _table_name, _column_name, maxValue);
```

```
--создания тригера на вставку
```

```
EXECUTE format('CREATE TRIGGER %I AFTER INSERT ON %s
```

```
FOR EACH STATEMENT
```

```
EXECUTE FUNCTION upd_spec_maxvalue(%s, %s);',
```

```
'spec_curmax_insert_' || _column_name || '_' || (SELECT uuid_generate_v1mc()), _table_name,  
_table_name, _column_name);
```

```
--создание тригера на изменение
```

```
EXECUTE format ('CREATE TRIGGER %I AFTER UPDATE ON %s
```

```
FOR EACH STATEMENT
```

```
EXECUTE FUNCTION upd_spec_maxvalue(%s, %s);',
```

```
'spec_curmax_update_' || _column_name || '_' || (SELECT uuid_generate_v1mc()),  
_table_name, _table_name, _column_name);
```

```

RETURN maxValue;
END IF;
END;
$$ LANGUAGE plpgsql;

```

## Тестирование

--создадим таблицу spec

```

CREATE TABLE spec
(
    id integer NOT NULL,
    table_name character varying(30) NOT NULL,
    column_name character varying(30) NOT NULL,
    cur_max_value integer NOT NULL
);

```

--добавим изначальные значения

```
INSERT INTO spec VALUES (1, 'spec', 'id', 1);
```

--создадим таблицу test

```

CREATE TABLE test
(
    id integer NOT NULL
);

```

--добавим в столбец id таблицы тест значение 30

```
INSERT INTO test VALUES (30);
```

--вызовем хранимую процедуру с параметрами test id

```
SELECT xp('test', 'id')
```

--посмотрим таблицу spec до добавления больших значений в таблицу test

```
SELECT * FROM spec
```

	id integer	table_name character varying (30)	column_name character varying (30)	cur_max_value integer
1	1	spec	id	2
2	2	test	id	31

--добавим значение, больше максимального в id таблицы test

```
INSERT INTO test VALUES (100);
```

--проверим таблицу spec

```
SELECT * FROM spec
```

	id integer	table_name character varying (30)	column_name character varying (30)	cur_max_value integer
1	1	spec	id	2
2	2	test	id	100

--посмотрим таблицу test

```
SELECT * FROM test
```

	id integer
1	30
2	100

--теперь попробуем изменить максимальный id в таблице test

```
UPDATE test
```

```
SET id = 200
```

```
WHERE id = (SELECT min(id) FROM TEST)
```

--таблица spec после изменения

```
SELECT * FROM spec
```

	id integer	table_name character varying (30)	column_name character varying (30)	cur_max_value integer
1	1	spec	id	2
2	2	test	id	200

--таблица test после изменения

```
SELECT * FROM test
```

	id integer
1	100
2	200

--попробуем уменьшить максимальный id в таблице test (200 изменится на 20)

```
UPDATE test
```

```
SET id = 20
```

```
WHERE id = (SELECT min(id) FROM TEST)
```

--проверяем, что cur\_max\_value у test в таблице spec не изменилась

```
SELECT * FROM spec
```

	id integer	table_name character varying (30)	column_name character varying (30)	cur_max_value integer
1	1	spec	id	2
2	2	test	id	200

--потестируем с таблицей, у которой более 1 столбца

--создадим таблицу test2 со столбцами max1 и max2

```
CREATE TABLE test2
```

```
(
```

```
max1 integer NOT NULL,
```

```
max2 integer NOT NULL
```

```
);
```

--добавим в test2 значения 5 и 10

```
INSERT INTO test2 VALUES (5, 10);
```

--вызовем хранимую процедуру с обоими столбцами

--и посмотрим корректно ли создались для них триггеры

```
SELECT xp('test2', 'max1')
```

```
SELECT xp('test2', 'max2')
```

test2

> Columns

> Constraints

> Indexes

> RLS Policies

> Rules

> Triggers (4)

> spec\_curmax\_insert\_max1\_f7112e4c-62df-11ed-919d-df3d2be3330f

> spec\_curmax\_insert\_max2\_f86b603c-62df-11ed-919f-bf7fccc0c289

> spec\_curmax\_update\_max1\_f7115548-62df-11ed-919e-1366ac9e5ab2

> spec\_curmax\_update\_max2\_f86b603d-62df-11ed-91a0-0b7a597d1e49

--для каждого столбца создается 2 триггера на изменения и добавление  
--проверим их работоспособность (для начала добавим новую запись так,  
--чтобы появился новый максимальный max1)

```
INSERT INTO test2 VALUES (50, 5);
```

--проверим таблицу spec

```
SELECT * FROM spec
```

	id integer	table_name character varying (30)	column_name character varying (30)	cur_max_value integer
1	2	test	id	200
2	1	spec	id	4
3	4	test2	max2	11
4	3	test2	max1	50

--теперь обновим максимумы сразу в двух столбиках

```
INSERT INTO test2 VALUES (150, 200);
```

--и проверим таблицу spec

```
SELECT * FROM spec
```

	id integer	table_name character varying (30)	column_name character varying (30)	cur_max_value integer
1	2	test	id	200
2	1	spec	id	4
3	3	test2	max1	150
4	4	test2	max2	200

**Все тесты отрабатывают корректно**