CP 4 advanced techniques

Danil Alshaev

May 2023

1 Transaction

In the following example, I want to rise price for
the teas from Japan that are less expensive then a 100.
I'll use following transaction:

```
--Transanction
BEGIN;

UPDATE tea
SET price = price * 1.05
WHERE country = 'JP' AND price < 100;

COMMIT;
```

If any error occurs during the transaction, the transaction will be rolled back, and the price column will not be updated
Possible conflict
it's possible for multiple updates to the price column to occur simultaneously, leading to conflicts and inconsistent data.

For example, some of the transactions can be executed at the same time, some values are going to be updated twice, but suppose we have a tea that costs 99,but it has to be updated only once.

2 View

I want to get employees with position cashier that have high salary. For that purpose I will create following view:

```
--View
CREATE VIEW cashier_high_salary AS
SELECT employee_id, name, last_name, salary, phone_number, position
FROM employee
WHERE position = 'cashier' AND salary > 10000;

--Querry
SELECT * FROM cashier_high_salary;
```

Now I can use it as a query.

	employee_id integer	name character varying (50)	last_name character varying (50)	salary integer	phone_number character varying (16)	position character varying (50)
1	5	Alex	Brown	11000	+420567890123	cashier

3 Trigger

For a new workers with a position "cleaner" I want to always set their salary to 8500. For that purpose I'll use the following trigger:

```
--Trigger
CREATE OR REPLACE FUNCTION set_cleaner_salary()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW."position" = 'cleaner' THEN
    NEW.salary := 8500;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

--Create Trigger
CREATE TRIGGER cleaner_salary_trigger
BEFORE INSERT ON employee
FOR EACH ROW
EXECUTE FUNCTION set_cleaner_salary();
```

Now I'll insert new employee with a position "cleaner:

```
--Insert new cleaner
INSERT INTO employee (employee_id, name, last_name, phone_number, position)
VALUES (21, 'John', 'Doe', '+420228322723', 'cleaner');
```

4 Index

For the biggest table, which is order, I want to create an index, because it has around 32000 rows and it can be long for some queries:

```
CREATE INDEX order_ordered_time_idx ON public."order" (cost,ordered_time);
```

Let's try to analyze it on a concrete example
I'll run the same query 2 times:

```
EXPLAIN ANALYZE SELECT * FROM public."order" WHERE ordered_time = '2022-06-08' AND cost = 1598;
```

first - without index:

	QUERY PLAN text
1	Seq Scan on "order" (cost=0.00..653.00 rows=1 width=12) (actual time=0.032..7.327 rows=1 loops…
2	Filter: ((ordered_time = '2022-06-08'::date) AND (cost = 1598))
3	Rows Removed by Filter: 31999
4	Planning Time: 0.170 ms
5	Execution Time: 7.367 ms

second time with the index:

	QUERY PLAN text
1	Index Scan using order_ordered_time_idx on "order" (cost=0.29..8.31 rows=1 width=12) (actual time=0.070..0.072 rows=1 loop…
2	Index Cond: ((cost = 1598) AND (ordered_time = '2022-06-08'::date))
3	Planning Time: 0.484 ms
4	Execution Time: 0.116 ms

As we can see, using the index this query was around 63 times faster. Although, table is not very big.