УНИВЕРСИТЕТ ИТМО

Проектирование БД

Лекция 8.
Распределенные данные.
Секционирование(партиционирование)

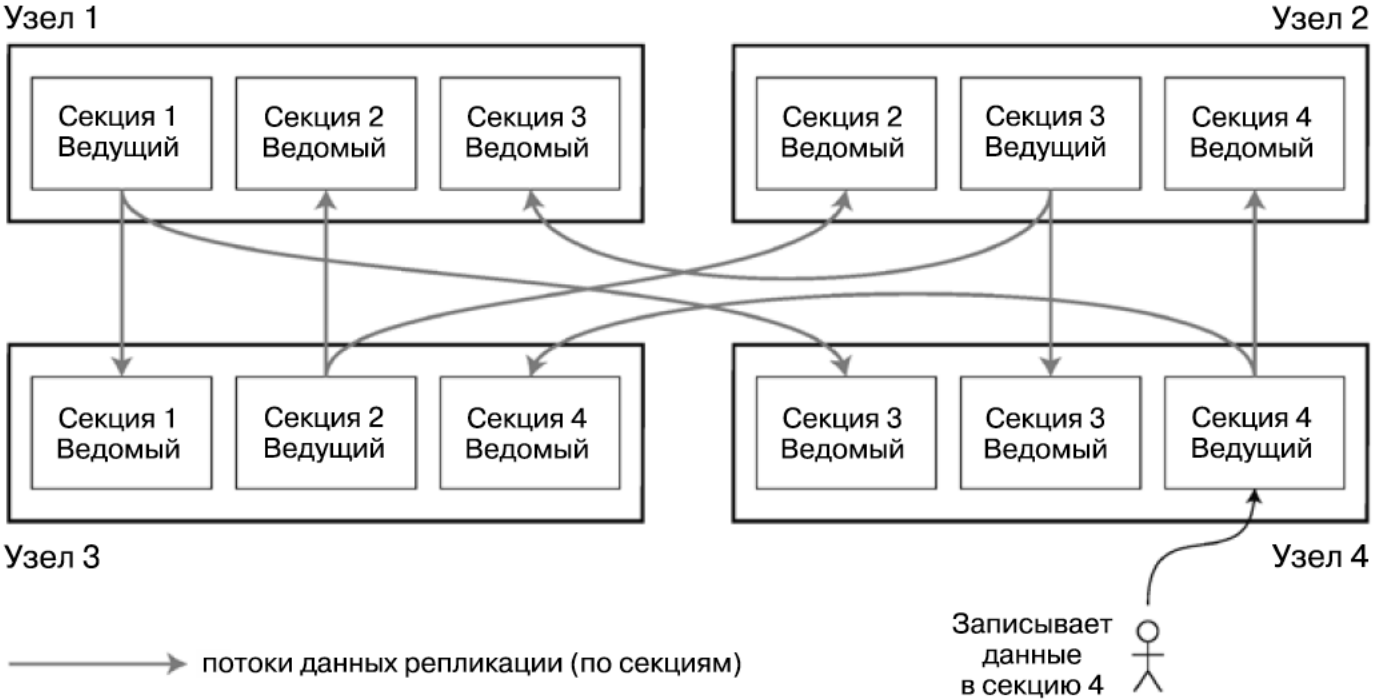
Секционирование (partitioning)

- Это способ умышленного разбиения большого набора данных на меньшие
- Основная цель секционирования данных — масштабируемость.
- появились в 1980-х годах
- “открыта заново” БД NoSQL и БД на основе Hadoop.

Виды секционирования

- Вертикальное секционирование (**Vertical** partitioning)
- Горизонтальное секционирование (**Horizontal** partitioning)

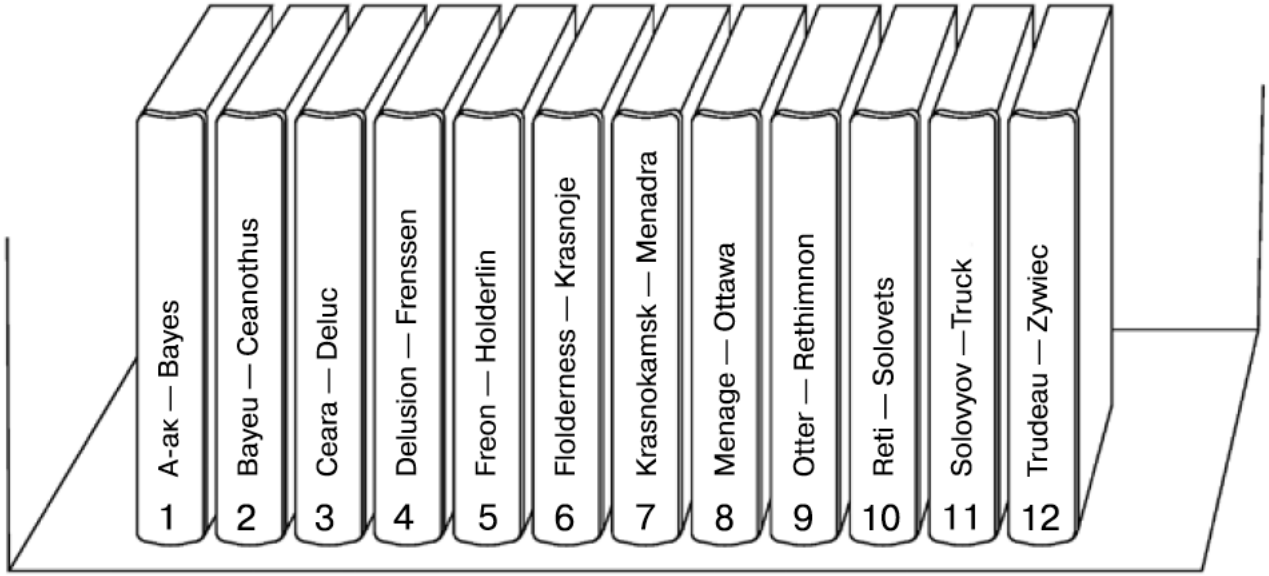
Секционирование и репликация



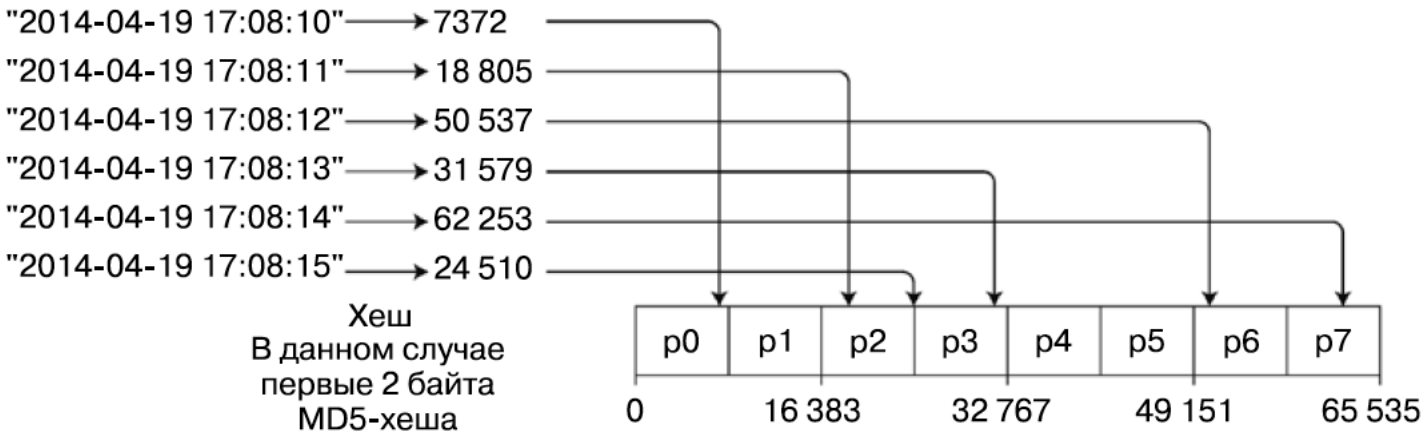
Секционирование данных типа «ключ — значение»

- Цель секционирования — равномерно распределить по узлам данные и загрузку по запросам
- Если же секционирование выполнено неравномерно, то оно называется асимметричным (**skewed**)
- Секция с непропорционально высокой нагрузкой называется горячей точкой (**hot spot**)
- Простейший способ избежать горячих точек — назначать узлы для записей случайным образом.

Секционирование по диапазонам значений ключа



Секционирование по хешу ключа



Асимметричные нагрузки и разгрузка горячих точек

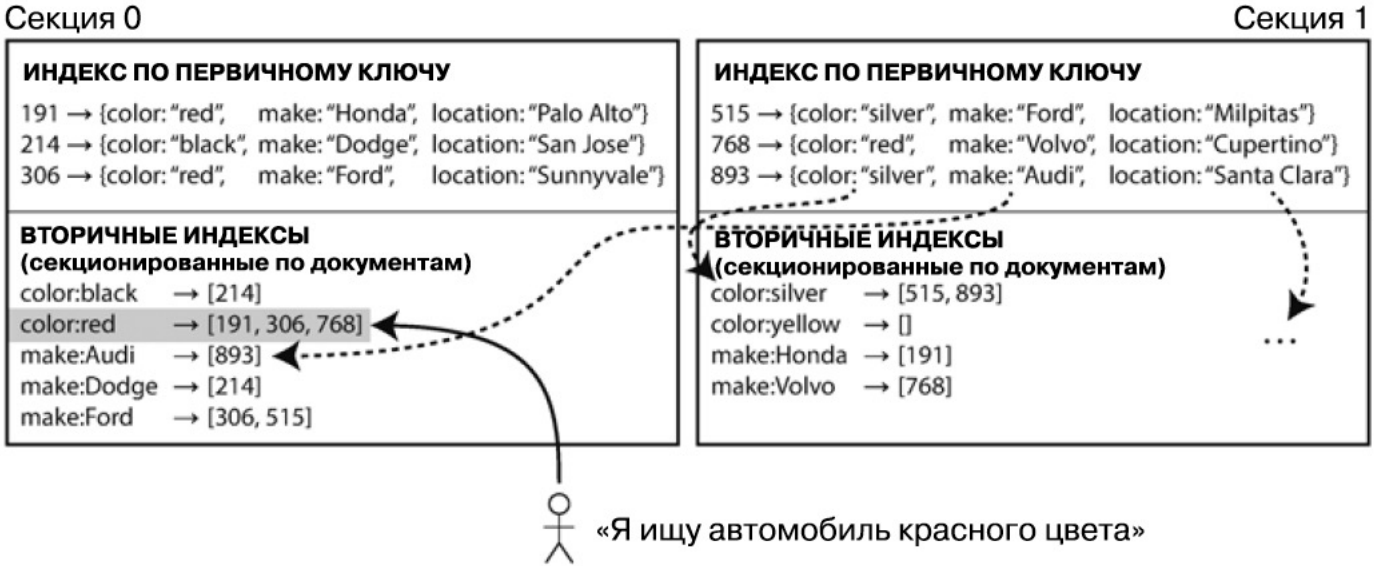
- Если все операции записи и чтения выполняются для одного ключа, все запросы все равно приходятся на одну секцию
- Снижение асимметрии — обязанность приложения:
  - Например, если конкретный ключ — очень горячий, то простейшим решением будет добавление в начало или конец этого ключа случайного числа. Простое двузначное десятичное число приведет к разбиению операций записи для ключа равномерно по 100 различным ключам, что позволит распределить их по разным секциям

Секционирование и вторичные индексы

- Вторичный индекс обычно не идентифицирует запись однозначно
- Основная проблема вторичных индексов — в том, что невозможно поставить их в четкое соответствие секциям
- Существует два основных подхода к секционированию базы данных с вторичными индексами:
  - секционирование по документам (document-based partitioning)
  - секционирование по термам (term-based partitioning).

Секционирование вторичных индексов по документам

Секционирование вторичных индексов по термам



Перебалансировка секций. Причины

- количество обрабатываемых запросов растет, так что для возросшей нагрузки понадобятся дополнительные процессоры;
- размер набора данных растет, поэтому для его хранения понадобятся дополнительные жесткие диски и оперативная память
- некоторые компьютеры испытывают сбои, вследствие чего другим компьютерам приходится брать на себя их обязанности.

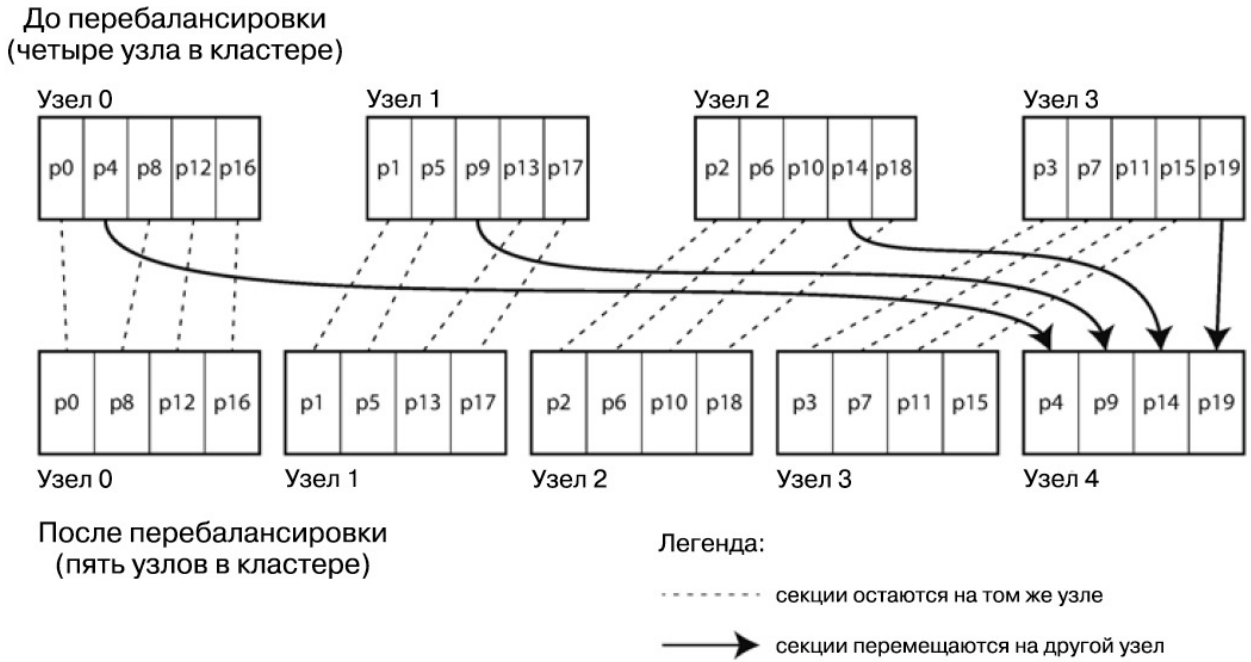
Перебалансировка секций. Требования

- после перебалансировки нагрузка должна быть распределена равномерно по узлам кластера;
- база данных должна продолжать принимать запросы на чтение и запись во время перебалансировки;
- между узлами должно перемещаться ровно то количество данных, которое необходимо

Как делать не следует: хеширование по модулю N

- hash(key) mod 10 возвращает число от 0 до 9
- Если 10 узлов:
  - hash(key) = 123456, 123456 mod 10 = **6**
- Если 11 узлов:
  - hash(key) = 123456, 123456 mod 11 = **3**
- Если 12 узлов:
  - hash(key) = 123456, 123456 mod 12 = **0**

Фиксированное количество секций



Динамическое секционирование

- Когда размер секции перерастает заданный (например, 10 Гбайт), она разбивается на две секции
- Преимуществом динамического секционирования является адаптация количества секций к общему объему данных
- Пока набор данных невелик — до момента разбиения первой секции, — все операции записи обрабатываются одним узлом, в то время как все остальные узлы простаивают.

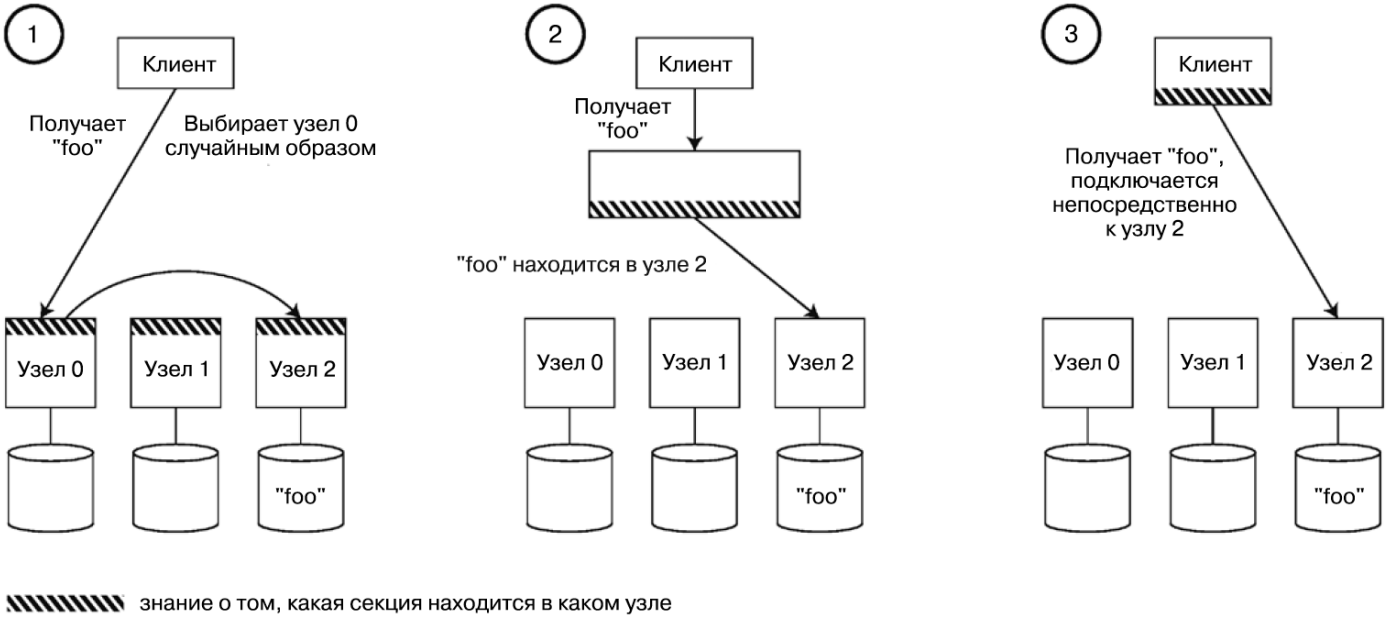
Секционирование пропорционально количеству узлов

- количество секций пропорционально количеству узлов — другими словами, на каждый узел приходится фиксированное количество секций
- такой подход обеспечивает практически постоянные размеры отдельных секций
- выбор границ секций случайным образом требует использования хеш-секционирования

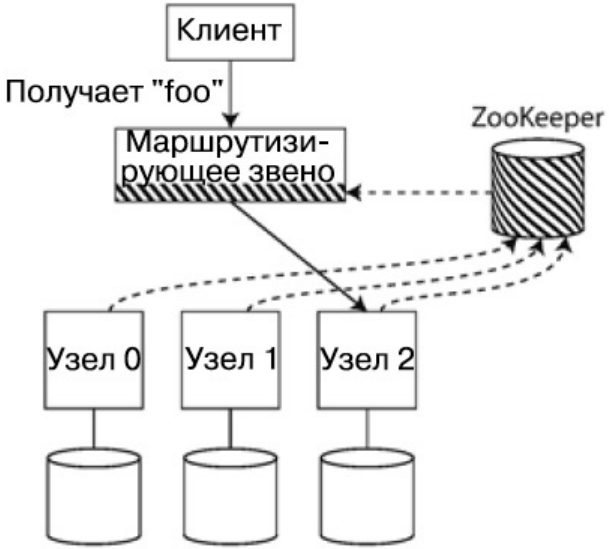
Эксплуатация: автоматическая или ручная перебалансировка

- Перебалансировка — операция с большими накладными расходами, требующая изменения маршрутов запросов и перемещения большого объема данных из одного узла в другой.
- Присутствие человека в цикле перебалансировки может быть хорошей идеей.

Маршрутизация запросов



Маршрутизация запросов



Диапазон ключа	Секция	Узел	IP-Address
A-ak — Bayes	partition 0	node 0	10.20.30.100
Bayeu — Ceanothus	partition 1	node 1	10.20.30.101
Ceara — Deluc	partition 2	node 2	10.20.30.102
Delusion — Frenssen	partition 3	node 0	10.20.30.100
Freon — Holderlin	partition 4	node 1	10.20.30.101
Holderness — Krasnoje	partition 5	node 2	10.20.30.102
Krasnokamsk — Menadra	partition 6	node 0	10.20.30.100
Menage — Ottawa	partition 7	node 1	10.20.30.101
Otter — Rethimnon	partition 8	node 2	10.20.30.102
Reti — Solovets	partition 9	node 0	10.20.30.100
Solovyov — Truck	partition 10	node 1	10.20.30.101
Trudeau — Zywiec	partition 11	node 2	10.20.30.102

знание о том, какая секция находится на каком узле

Ограничения и возможные проблемы секционирования Postgres

- партицируемая таблица должна быть достаточно большого размера
- на партицируемую таблицу нельзя ссылаться через FOREIGN KEYS (**можно, начиная с PostgreSQL 12**)
- может ухудшить производительность на операциях чтения и записи
- в идеале запрос будет выполняться против одной партиции, но в худшем случае — затронет все партиции

Виды секционирования Postgres. Декларативное партицирование

- требует изначально создать таблицу, готовую к партицированию;
- нельзя партицировать уже существующую таблицу через ALTER TABLE;
- при добавлении и удалении партиций будет простой в работе таблицы из-за ACCESS EXCLUSIVE LOCK

Виды секционирования Postgres. Партицирование через наследование

- можно партицировать уже существующую таблицу;
- нет даунтайма при добавлении и удалении партиций;
- можно задать любой произвольный критерий партицирования
- возможно множественное наследование (наследование схем более чем одной таблицы);
- в конце концов партицирование можно безболезненно отменить.

Postgres. Партицирование через наследование

Таблица:

```
create table lap_times
(
id bigserial constraint id_key primary key,
lap_number integer not null,
lap_time interval not null
);
```

```
create index lap_time_idx on lap_times (lap_time);
```

Postgres. Партицирование через наследование

Генерим данные:

```
insert into lap_times (lap_number, lap_time)
select
r.id,
interval '1 minute 30 seconds' + round
(random()::decimal, 3) * interval '1 second'
from
generate_series (1, 100000) as r(id);
```

id	lap_number	lap_time
5	5	00:01:35.192
6	6	00:01:33.661
7	7	00:01:36.164
8	8	00:01:39.343
9	9	00:01:30.498
10	10	00:01:34.908
14	14	00:01:36.423

Postgres. Партицирование через наследование

Создаём таблицы-партиции с использованием ключевого слова INHERITS:

```
create table lap_times_under40 (
check (lap_time >= interval '1 minute 30 seconds' and lap_time < interval '1 minute 40 seconds')
) inherits (lap_times);

create table lap_times_under50 (
check (lap_time >= interval '1 minute 40 seconds' and lap_time < interval '1 minute 50 seconds')
) inherits (lap_times);

create table lap_times_under60 (
check (lap_time >= interval '1 minute 50 seconds' and lap_time < interval '2 minutes 00 seconds')
) inherits (lap_times);
```

Postgres. Партицирование через наследование

Добавляем индексы, такие же, как в мастер-таблице:

```
alter table only lap_times_under40
add constraint lap_times_under40_key primary key (id);

create index lap_times_under40_idx on lap_times_under40 (lap_time);

alter table only lap_times_under50
add constraint lap_times_under50_key primary key (id);

create index lap_times_under50_idx on lap_times_under50 (lap_time);

alter table only lap_times_under60
add constraint lap_times_under60_key primary key (id);

create index lap_times_under60_idx on lap_times_under60 (lap_time);
```

Postgres. Партицирование через наследование

Создаём функцию, обеспечивающую партицирование:

```
create or replace function
lap_times_inherit_trigger()
returns trigger as $$
begin
if (new.lap_time >= interval '1 minute 30 seconds' and new.lap_time < interval '1
minute 40 seconds') then insert into lap_times_under40 values (new.*);
elseif (new.lap_time >= interval '1 minute 40 seconds' and new.lap_time < interval
'1 minute 50 seconds') then insert into lap_times_under50 values (new.*);
elseif (new.lap_time >= interval '1 minute 50 seconds' and new.lap_time < interval
'2 minutes 00 seconds') then insert into lap_times_under60 values (new.*);
else
raise exception 'Time out of range. Fix the lap_times_inherit_trigger() function!';
end if;
return null;
end;
$$
language plpgsql;
```

Postgres. Партицирование через наследование

Подключаем функцию к мастер-таблице:

```
create trigger insert_lap_times
before insert on lap_times
for each row execute function
lap_times_inherit_trigger();
```

Postgres. Партицирование через наследование

Вариант 2 – определяем правила распределения данных при записи:

```
create rule lap_times_under40 as on insert to lap_times
where (lap_time >= interval '1 minute 30 seconds' and lap_time < interval '1
minute 40 seconds')
do instead insert into lap_times_under40 values (new.*);

create rule lap_times_under50 as on insert to lap_times
where (lap_time >= interval '1 minute 40 seconds' and lap_time < interval '1
minute 50 seconds')
do instead insert into lap_times_under50 values (new.*);

create rule lap_times_under60 as on insert to lap_times
where (lap_time >= interval '1 minute 50 seconds' and lap_time < interval '2
minutes 00 seconds')
do instead insert into lap_times_under60 values (new.*);
```

УНИВЕРСИТЕТ ИТМО

Postgres. Партицирование через наследование

Разносим данные из мастер-таблицы по партициям:

```
with x as (
delete from only lap_times
where lap_time between interval '1 minute 30 seconds' and interval '1 minute 40 seconds' returning *)
insert into lap_times_under40
select * from x;

with x as (
delete from only lap_times
where lap_time between interval '1 minute 40 seconds' and interval '1 minute 50 seconds' returning *)
insert into lap_times_under50
select * from x;

with x as (
delete from only lap_times
where lap_time between interval '1 minute 50 seconds' and interval '2 minutes 00 seconds' returning *)
insert into lap_times_under60
select * from x;
```

Postgres. Партицирование через наследование

Очищаем мастер-таблицу:

```
truncate only lap_times;
```

```
306 select * from only lap_times;
307
```

Query History

id	lap_number	lap_time

```
308 select * from lap_times;
309
```

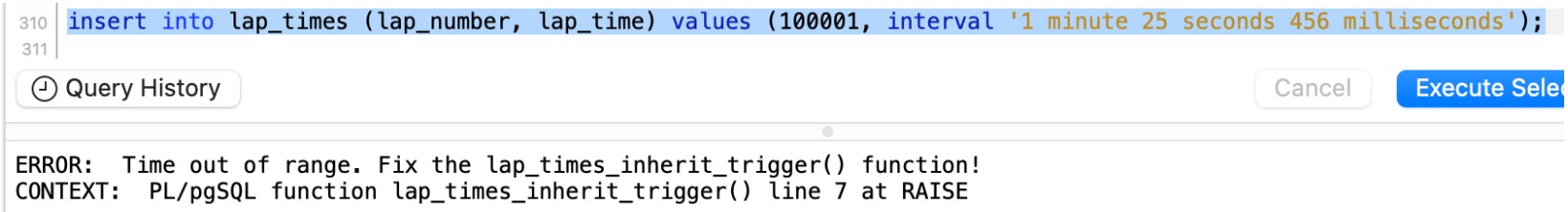
Query History

id	lap_number	lap_time
5	5	00:01:35.192
6	6	00:01:33.661
7	7	00:01:36.164

Postgres. Партицирование через наследование

Вставляем новую запись (не подходящую под условия):

```
insert into lap_times (lap_number, lap_time) values (100001, interval '1 minute 25 seconds
456 milliseconds');
```



Postgres. Партицирование через наследование

Вставляем новую запись (хорошую):

```
insert into lap_times (lap_number, lap_time) values (100001, interval '1 minute 35 seconds
456 milliseconds');
```

```
314 select * from only lap_times;
315
```

Query History

id	lap_number	lap_time

```
316 select * from lap_times times
317  where lap_number = 100001;
```

Query History

id	lap_number	lap_time
100002	100001	00:01:35.456

Postgres. Партицирование через наследование

```
319 explain analyse select * from lap_times;
320
```

Query History Cancel Exe

QUERY PLAN

```
Append  (cost=0.00..2237.01 rows=100001 width=28) (actual time=0.006..19.134 rows=100001 loops=1)
  ->  Seq Scan on lap_times lap_times_1  (cost=0.00..0.00 rows=1 width=28) (actual time=0.003..0.003 rows=0 loops=1)
  ->  Seq Scan on lap_times_under40 lap_times_2  (cost=0.00..577.31 rows=33231 width=28) (actual time=0.003..3.146 rows=33232 loops=1)
  ->  Seq Scan on lap_times_under50 lap_times_3  (cost=0.00..580.01 rows=33401 width=28) (actual time=0.017..3.229 rows=33401 loops=1)
  ->  Seq Scan on lap_times_under60 lap_times_4  (cost=0.00..579.68 rows=33368 width=28) (actual time=0.026..2.755 rows=33368 loops=1)
Planning Time: 0.105 ms
Execution Time: 23.689 ms
```

УНИВЕРСИТЕТ ИТМО

Postgres. Партицирование через наследование

```
321 explain analyse select * from lap_times where lap_time > interval '1 minute 50 seconds';
322
```

Query History Cancel

```
QUERY PLAN
Append  (cost=0.00..829.93 rows=33366 width=28) (actual time=0.008..8.925 rows=33363 loops=1)
  ->  Seq Scan on lap_times lap_times_1  (cost=0.00..0.00 rows=1 width=28) (actual time=0.003..0.003 rows=0 loops=1)
        Filter: (lap_time > '00:01:50'::interval)
  ->  Seq Scan on lap_times_under60 lap_times_2  (cost=0.00..663.10 rows=33365 width=28) (actual time=0.004..5.382 rows=33363 loops=1)
        Filter: (lap_time > '00:01:50'::interval)
        Rows Removed by Filter: 5
Planning Time: 0.217 ms
Execution Time: 10.978 ms
```

Спасибо за внимание!

www.ifmo.ru

ITMOre than a UNIVERSITY