

PostgreSQL Under Pressure

Евгений Фирстов

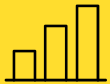
Приложение: биллинг для звонков колл-центра



Звонков – миллионы!



Приложение: биллинг для звонков колл-центра



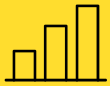
Звонков – миллионы!



Конвейерная обработка



Приложение: биллинг для звонков колл-центра



Звонков – миллионы!



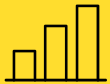
Конвейерная обработка



Нужно промежуточное хранилище



Приложение: биллинг для звонков колл-центра



Звонков – миллионы!



Конвейерная обработка



Нужно промежуточное хранилище





→ Чем нас не устроил Entity Framework?



- Чем нас не устроил Entity Framework?
- Вставка через хранимые процедуры (функции)



- Чем нас не устроил Entity Framework?
- Вставка через хранимые процедуры (функции)
- Лучший вариант импорта



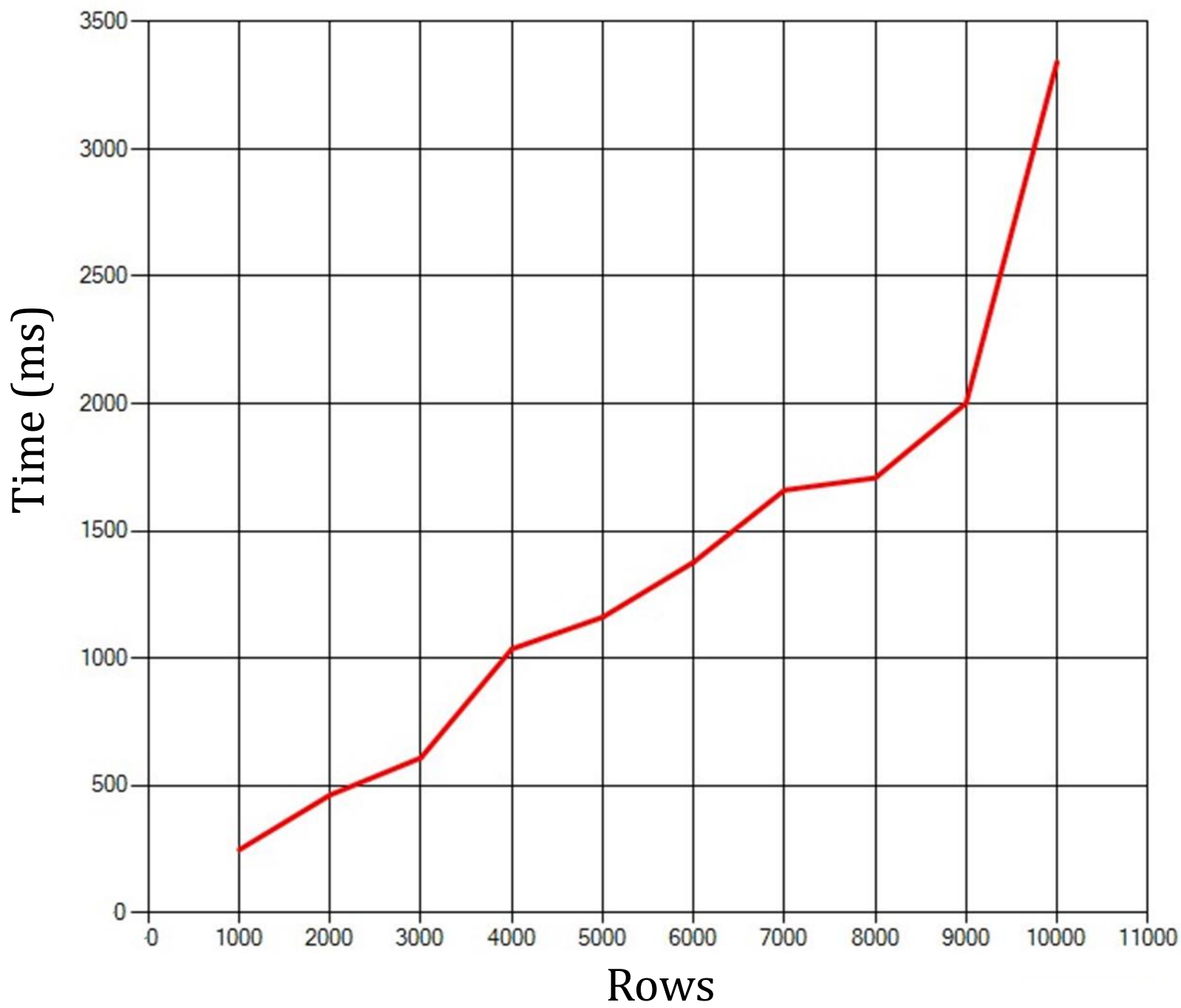
```
[Table( name: "calls")]  
public class Call  
{  
    [Key] public string CallId { get; set; }  
    public DateTime StartTime { get; set; }  
    public DateTime EndTime { get; set; }  
    public string CallingNumber { get; set; }  
    public string CalledNumber { get; set; }  
    public int Duration { get; set; }  
    public CallType CallType { get; set; }  
}
```

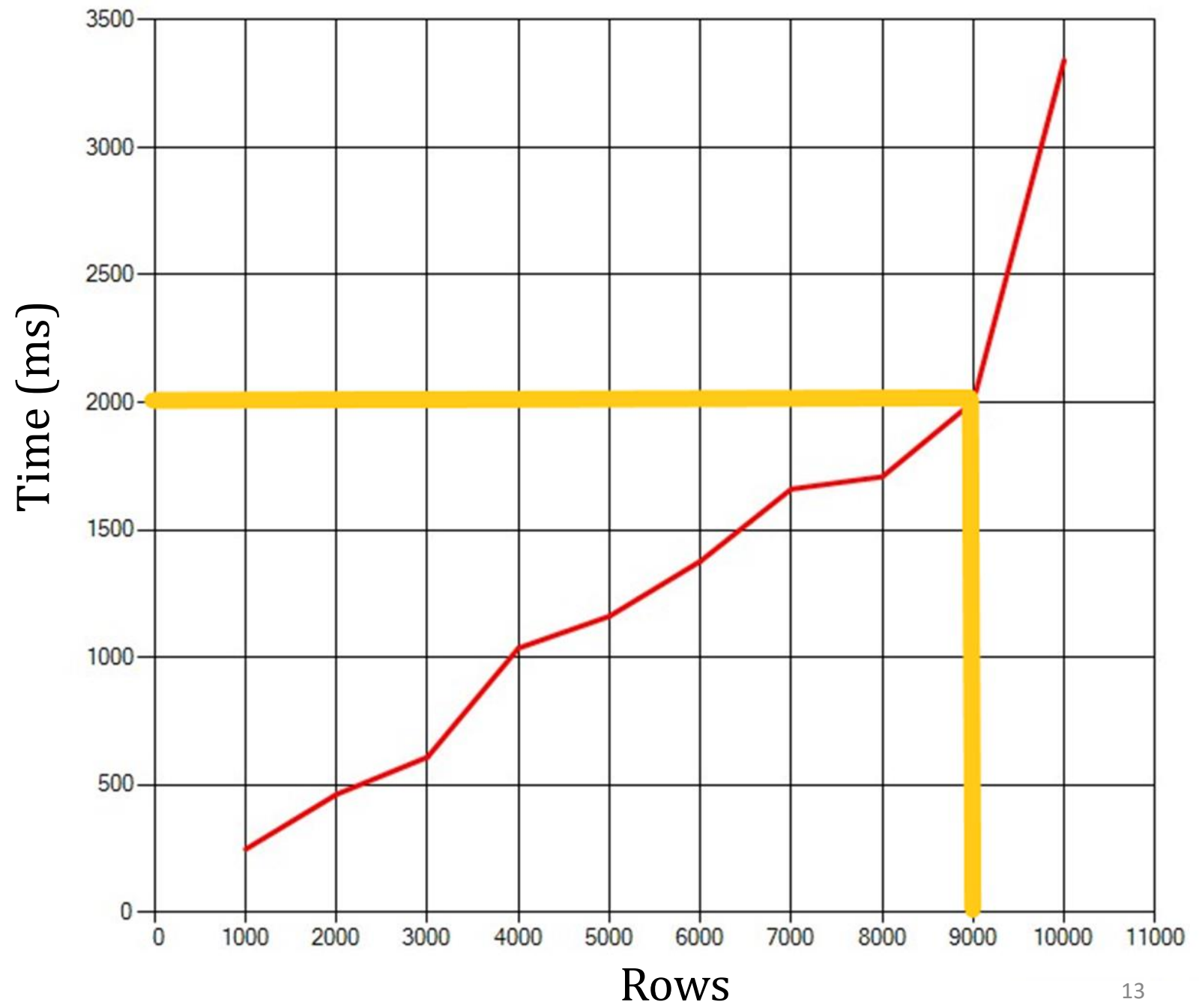


```
create table calls
(
    start_time      timestamp(6),
    end_time        timestamp(6),
    calling_number  varchar(11),
    called_number   varchar(11),
    duration        int,
    call_type       call_type_enum,
    call_id         varchar(11)
);
```



```
public async Task InsertAsync(Call[] calls)
{
    using (var context = new BillingContext())
    {
        context.Calls.AddRange(calls);
        await context.SaveChangesAsync().ConfigureAwait(false);
    }
}
```





Какое самое очевидное
решение?





Summary	
Products Included	
👑 Entity Framework Extensions	
✓ Entity Framework Classic	
✓ Bulk Operations	
✓ C# Eval Expression	
Perpetual License ⓘ	Yes
Royalty Free	Yes
Provider: All Providers	\$ 200
Developer Seats: 15-19 seats	\$ 1399
Support and Upgrades: 1 year	\$ 0
TOTAL	\$1599

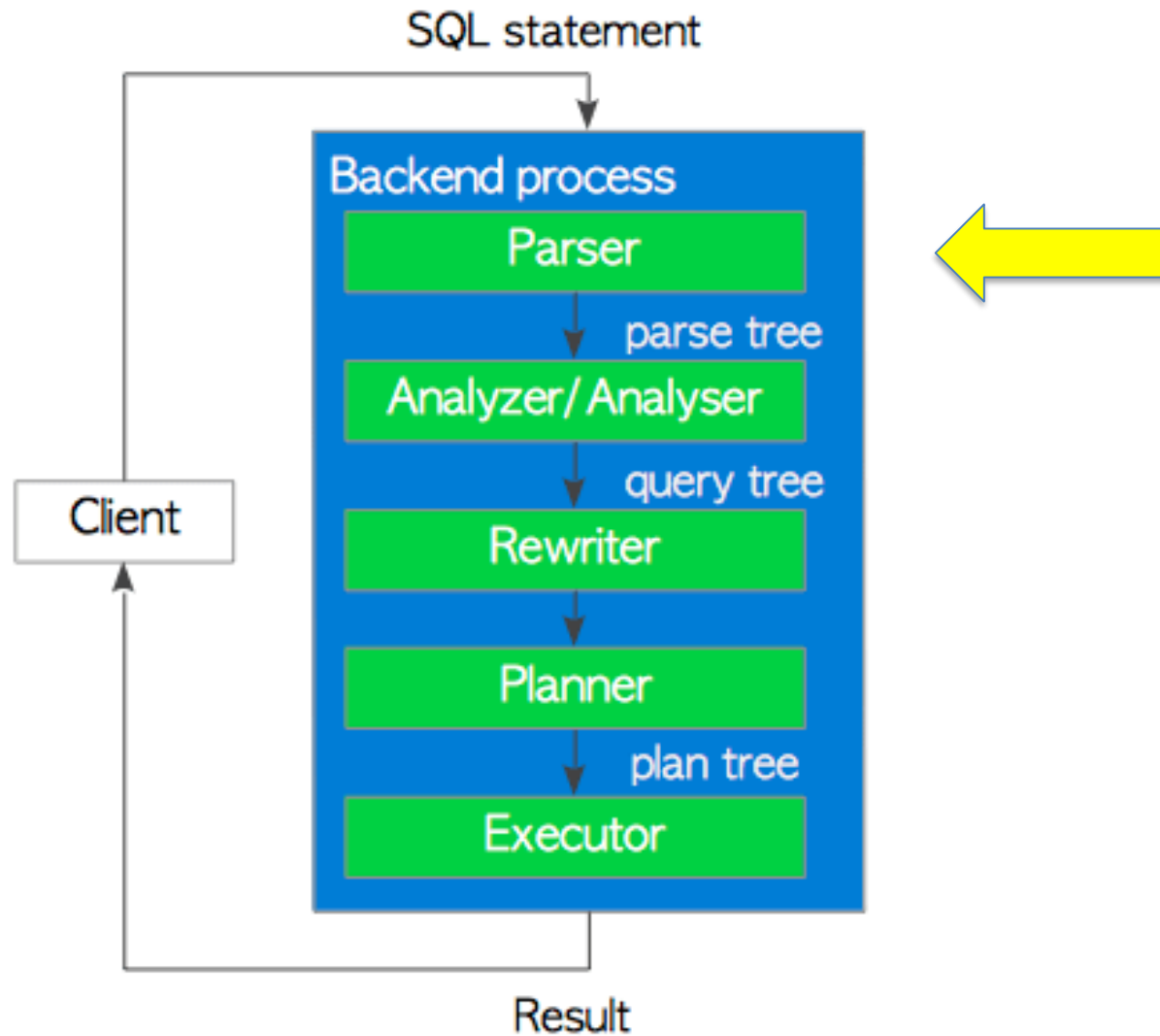


Неоптимальный SQL

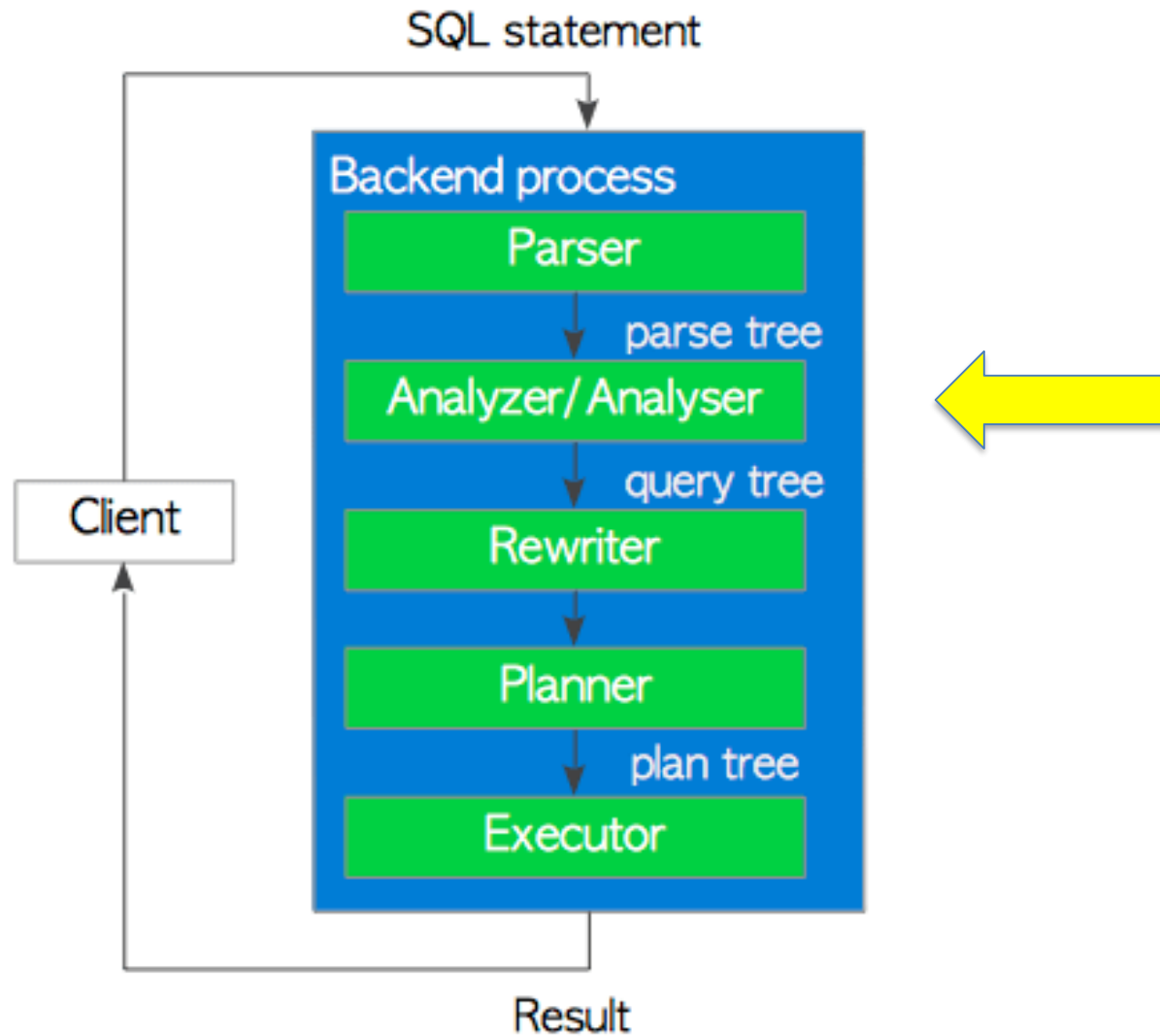
```
INSERT INTO calls
  (call_id, call_type, called_number,
   calling_number, duration, end_time, start_time)
VALUES (@p42, @p43, @p44, @p45, @p46, @p47, @p48);
```

```
INSERT INTO calls
  (call_id, call_type, called_number,
   calling_number, duration, end_time, start_time)
VALUES (@p42, @p43, @p44, @p45, @p46, @p47, @p48);
```

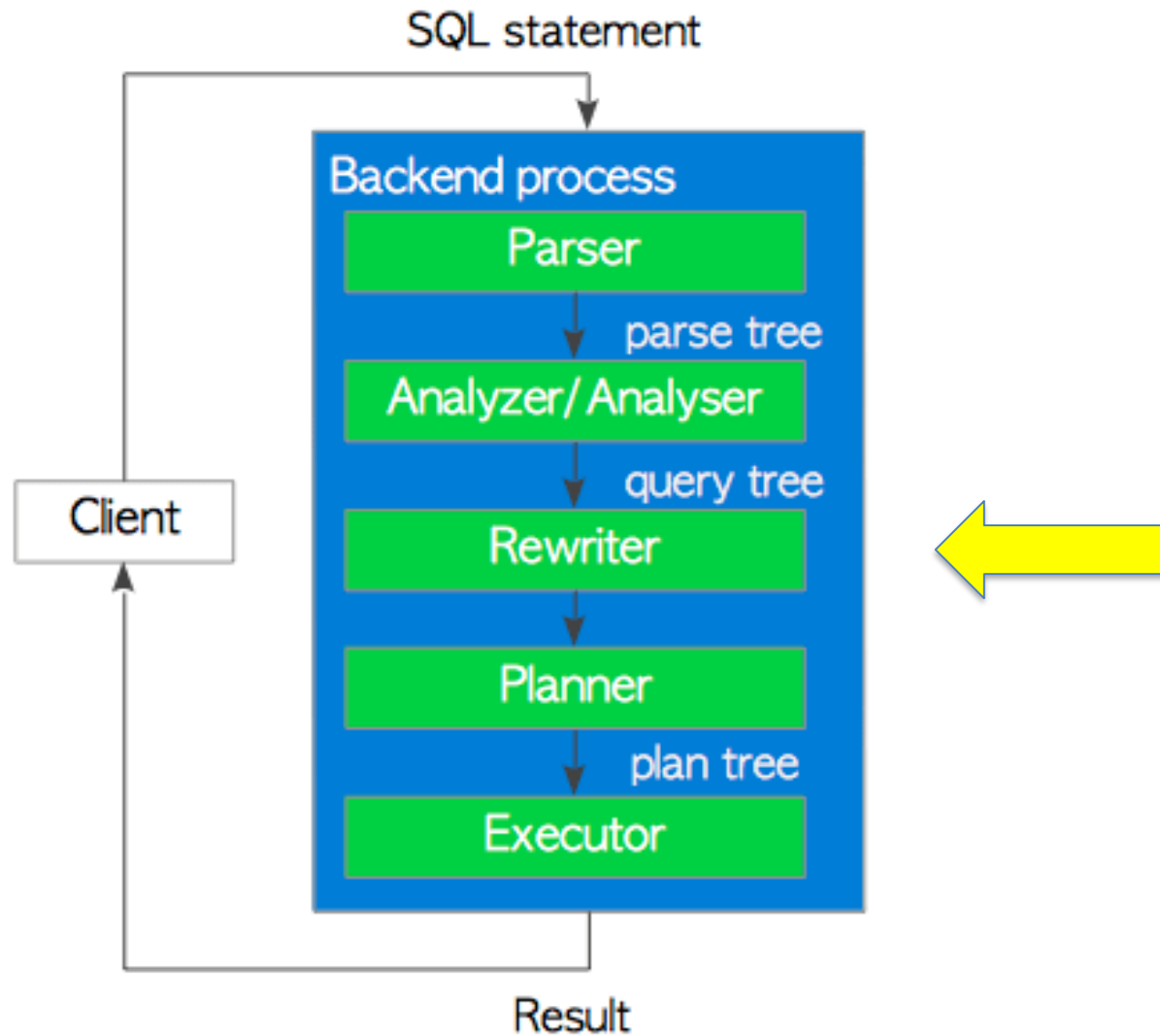

Разбор каждого INSERT`а



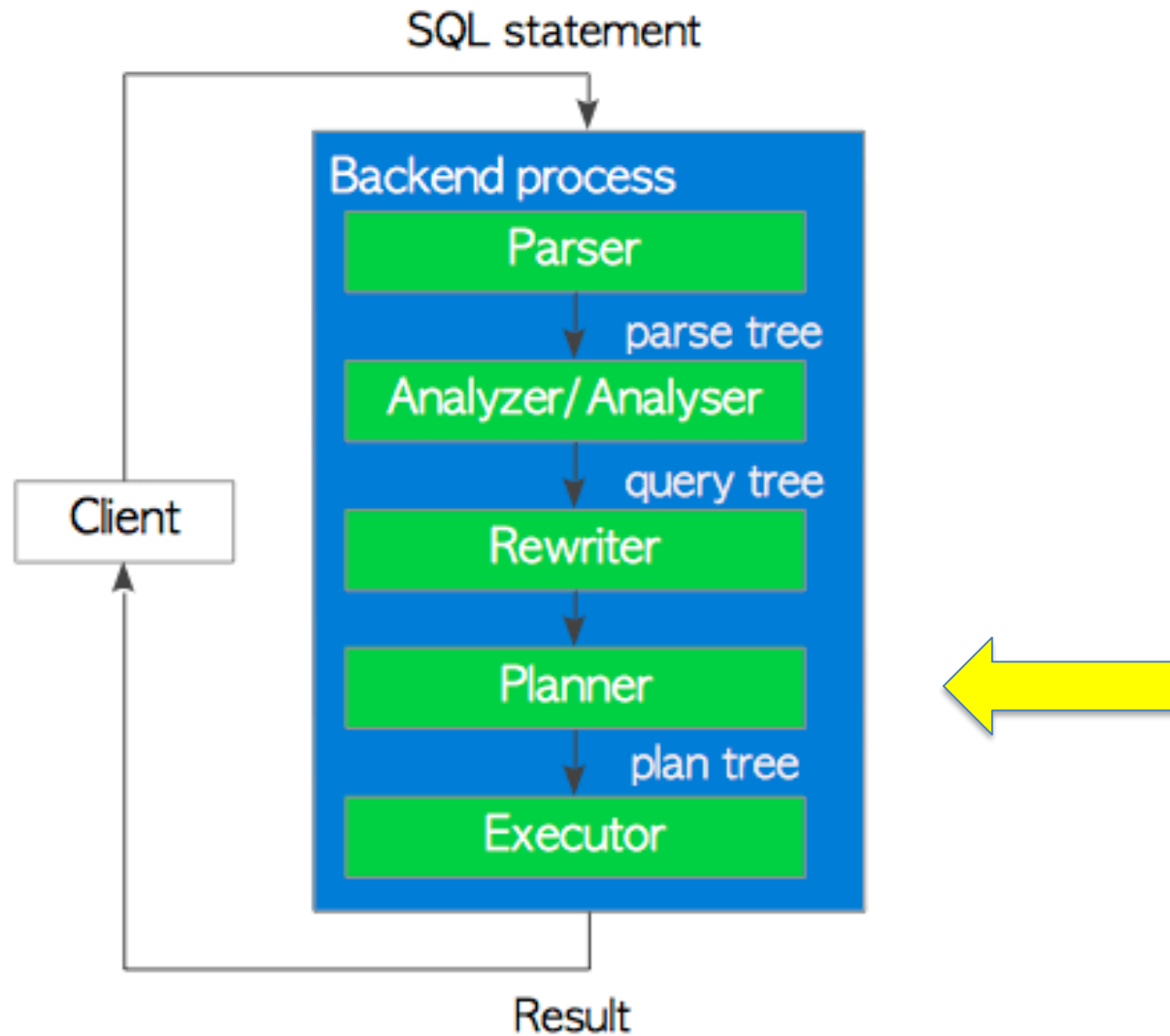
Разбор каждого INSERT`а



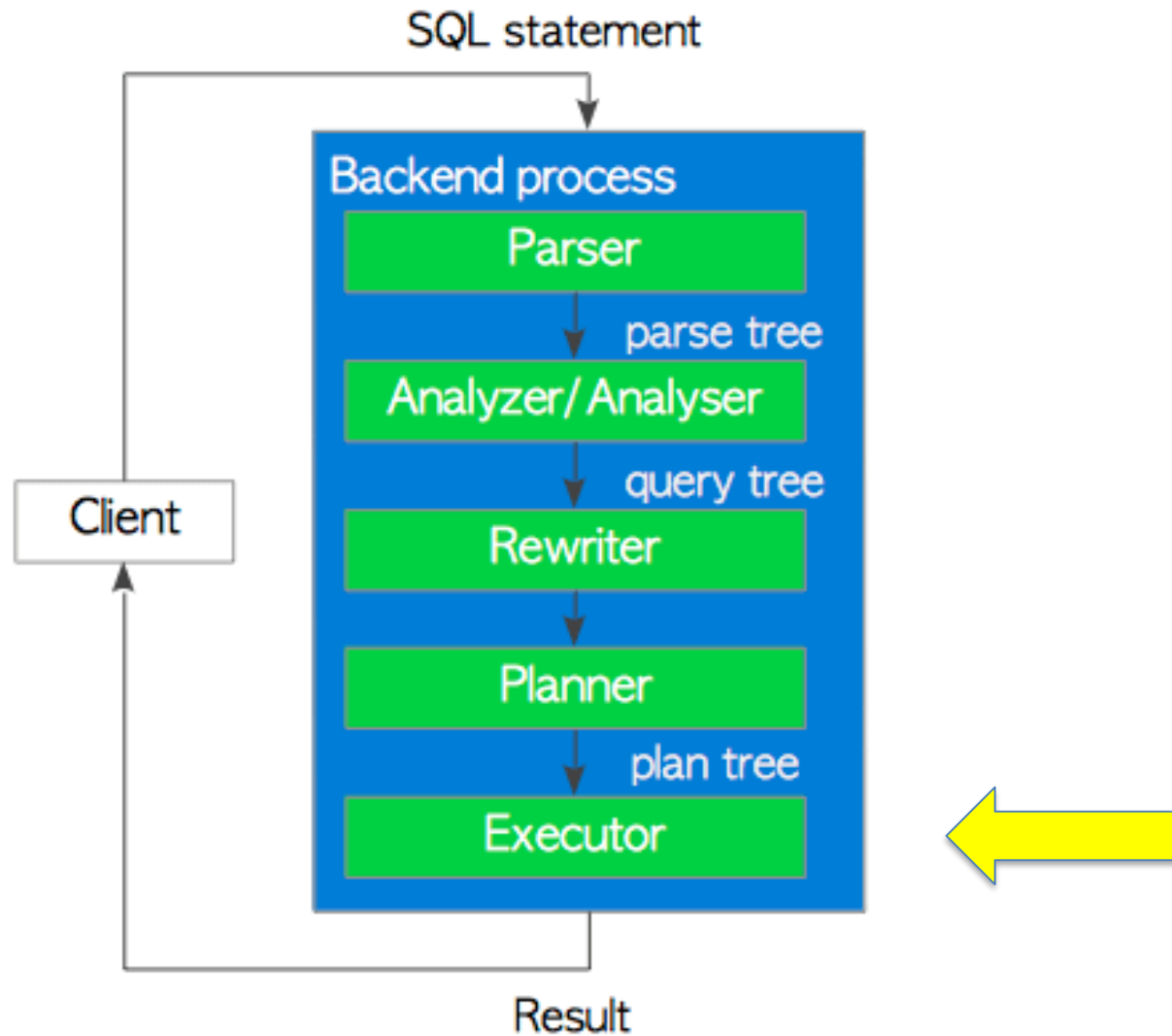
Разбор каждого INSERT`а

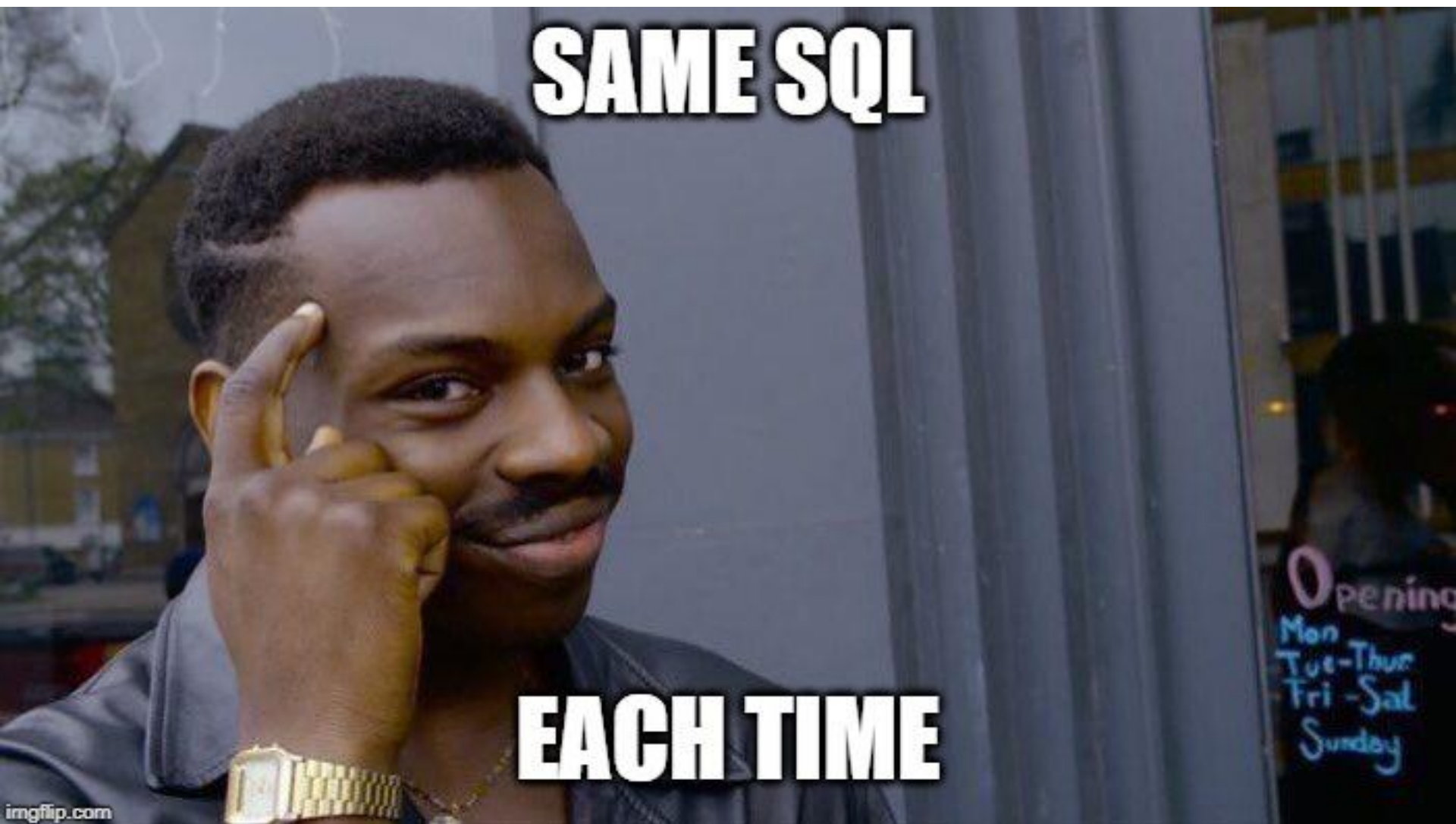


Разбор каждого INSERT`а



Разбор каждого INSERT`а







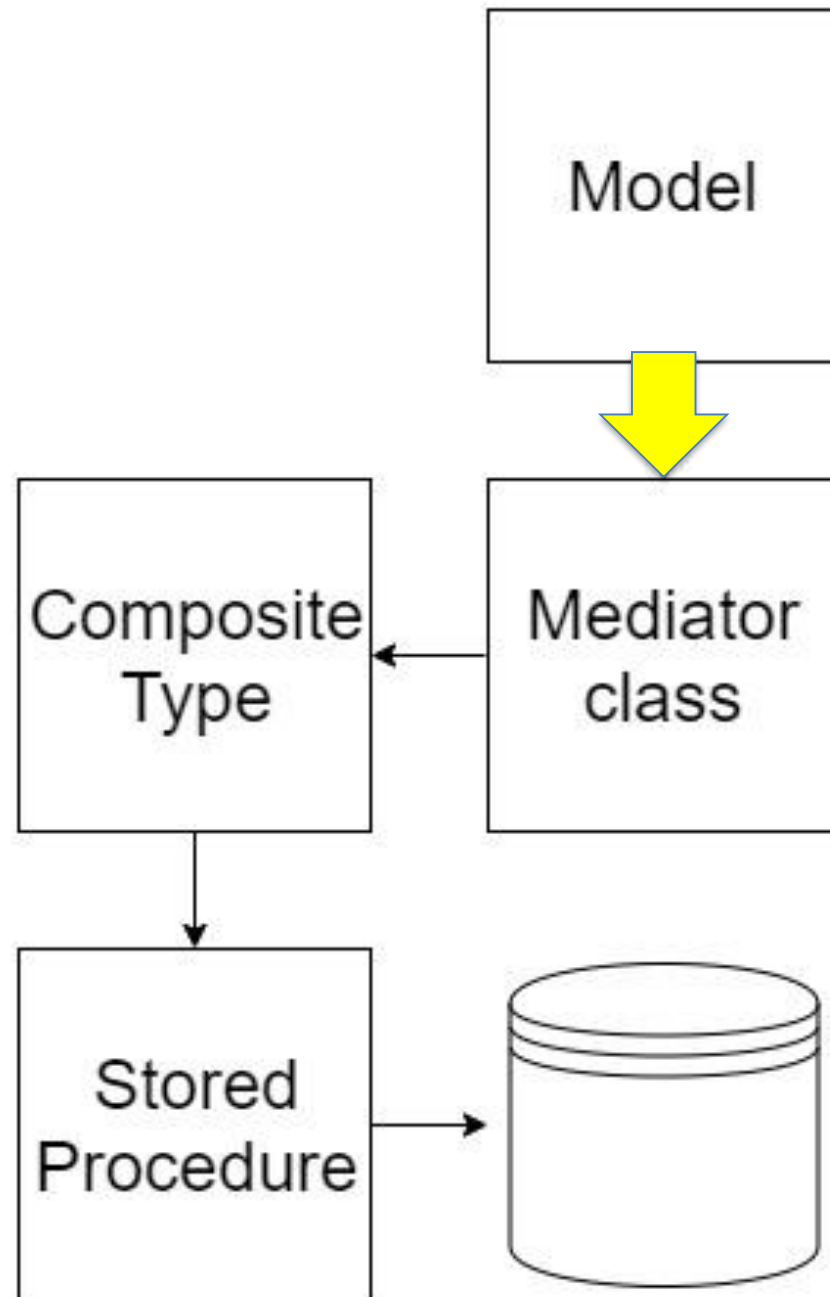


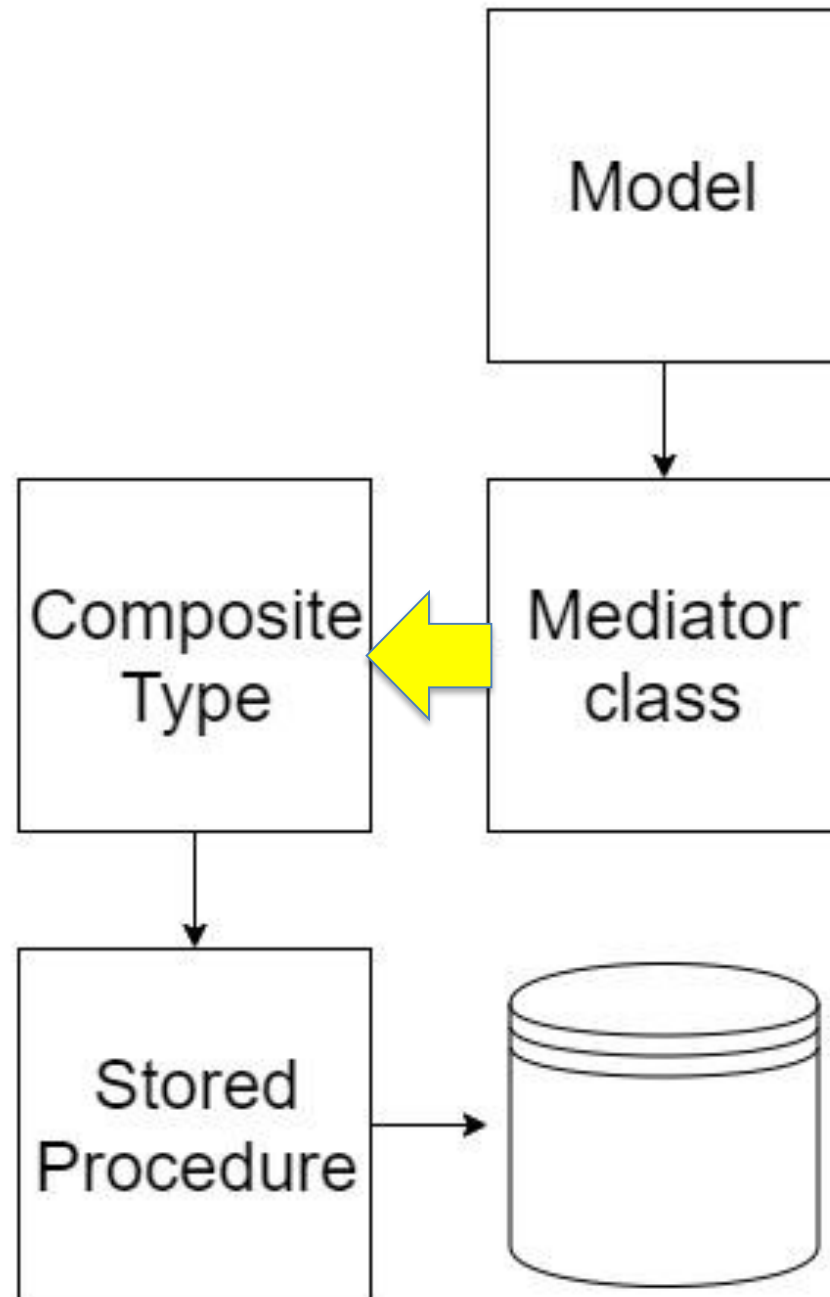
Процедура (функция) в PostgreSQL

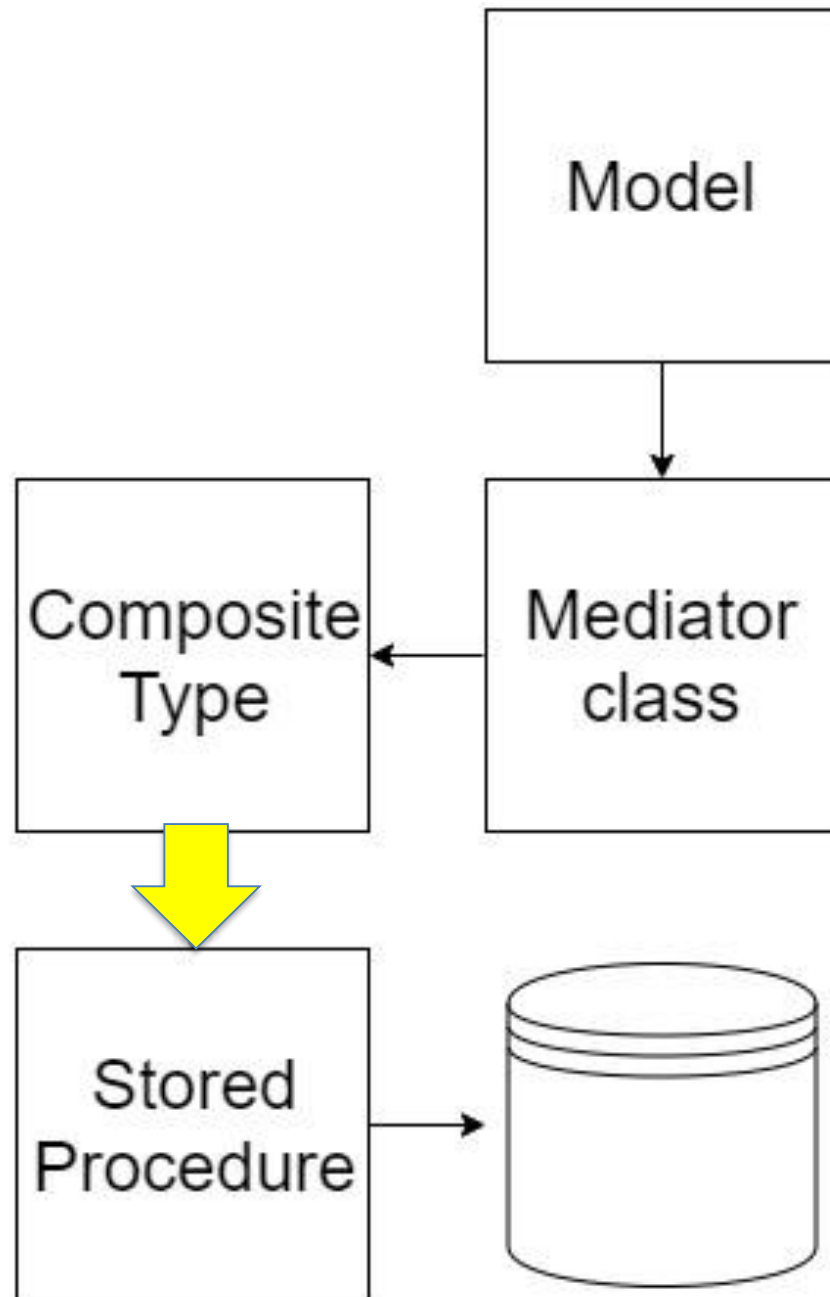
```
foreach call_ct in array call_ct_array
loop
    insert into calls(start_time,
                      end_time,
                      calling_number,
                      called_number,
                      duration,
                      call_type,
                      call_id)
    values (call_ct.start_time,
            call_ct.end_time,
            call_ct.calling_number,
            call_ct.called_number,
            call_ct.duration,
            call_ct.call_type,
            call_ct.call_id);
end loop;
```

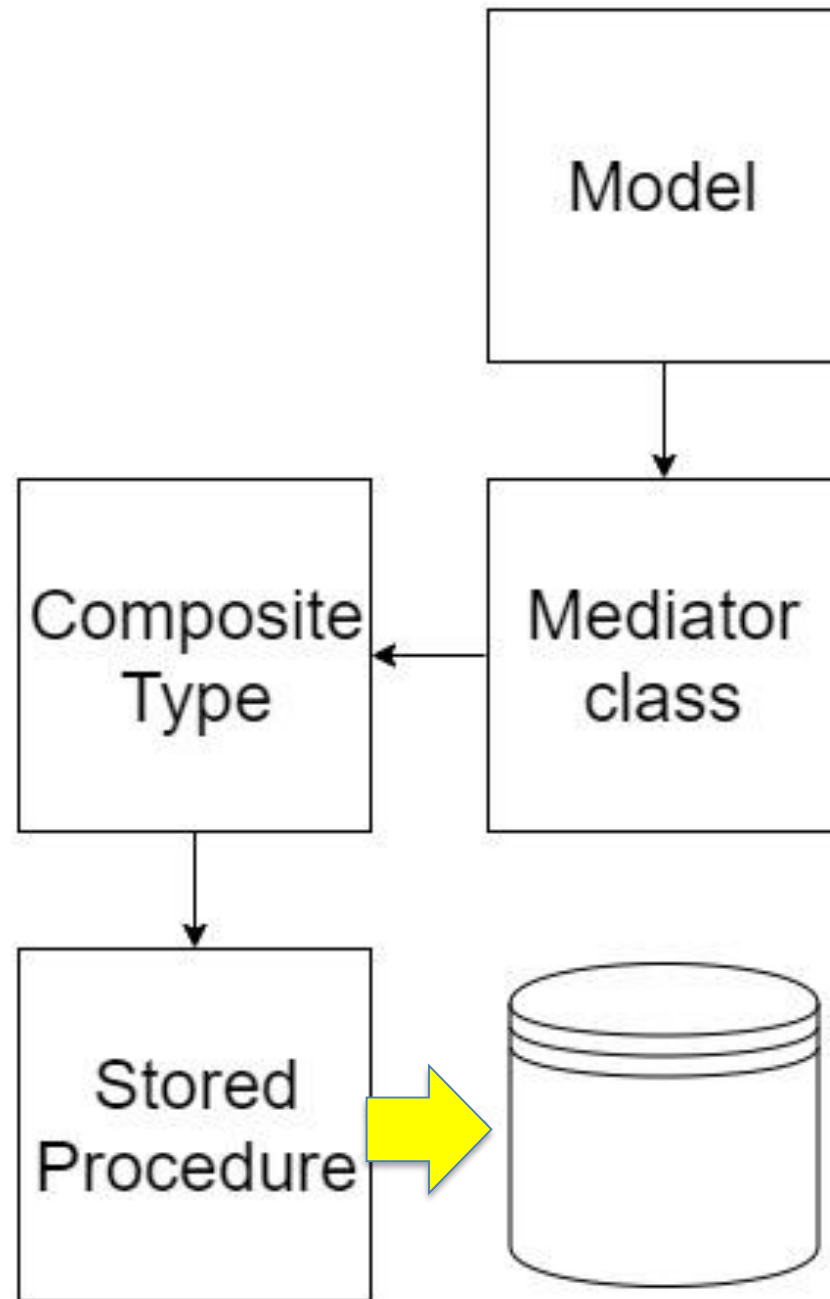


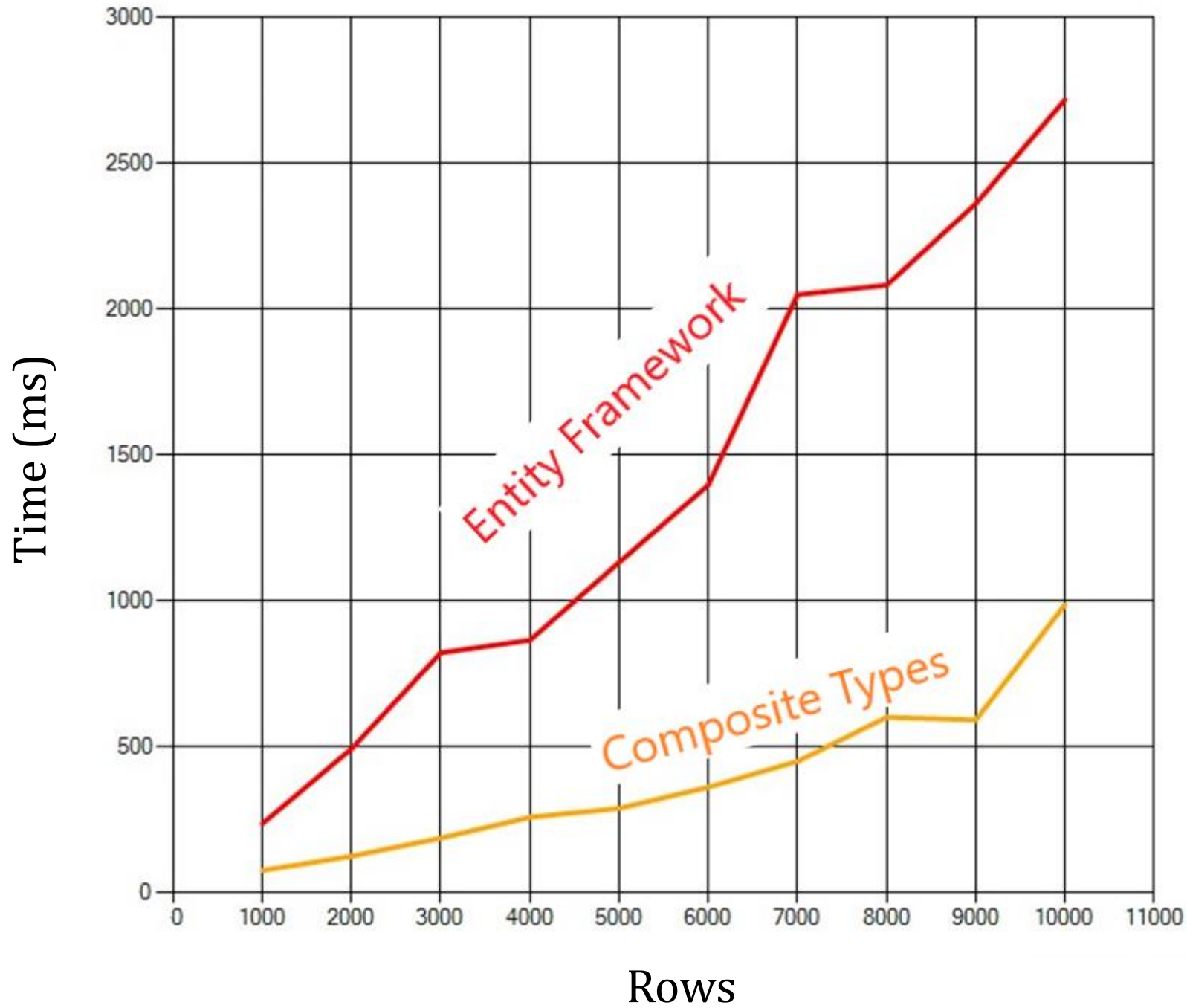

```
create type call_ct as
(
  start_time timestamp(6),
  end_time timestamp(6),
  calling_number varchar(11),
  called_number varchar(11),
  duration int,
  call_type call_type_enum,
  call_id varchar(11)
);
```















Быстрая вставка – проблема решена



 Быстрая вставка – проблема решена

 На других RDBMS можно также

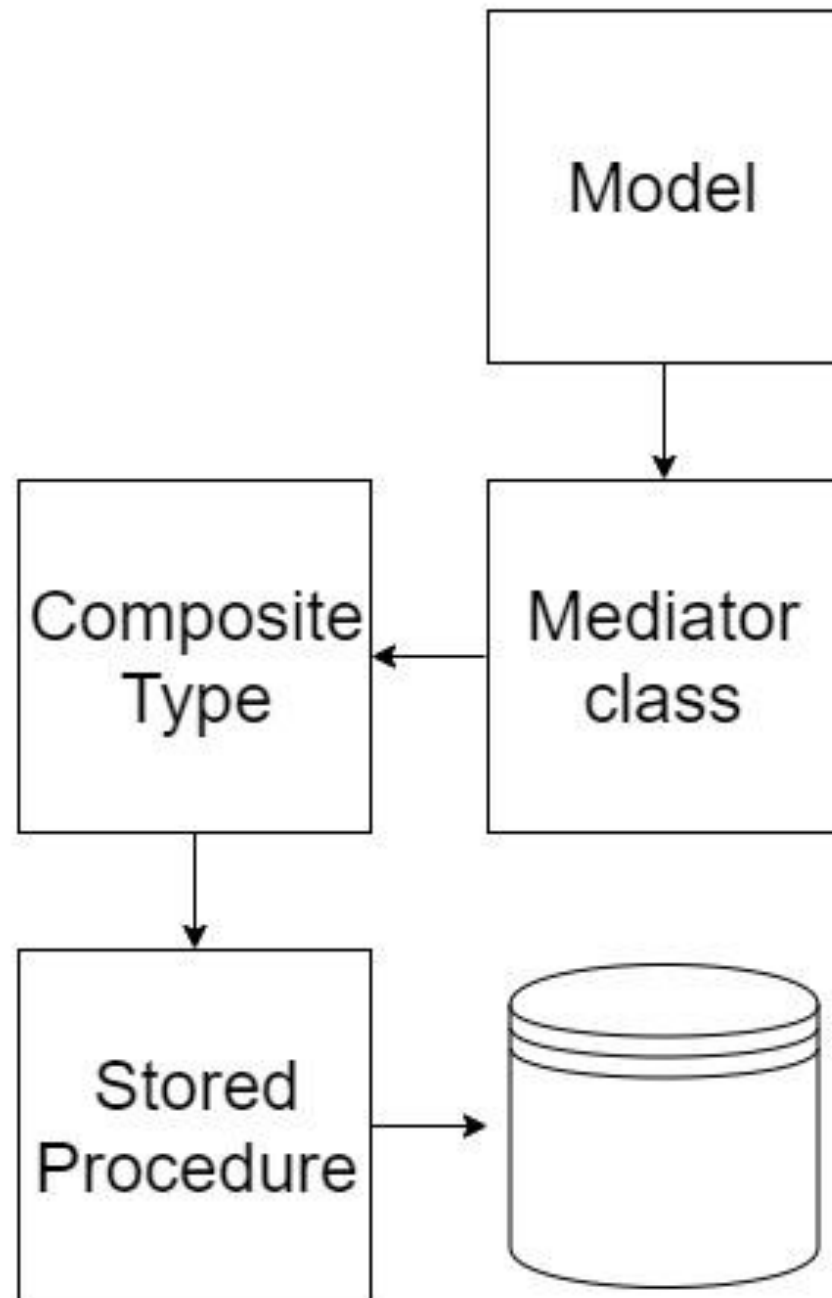


 Быстрая вставка – проблема решена

 На других RDBMS можно также

Но есть и минус:

 Много, много кода



BINARY



IMPORT



```
COPY calls  
  (start_time, end_time, calling_number,  
   called_number, duration, call_type, call_id)  
FROM STDIN (FORMAT BINARY)
```



В коде с использованием Npgsql

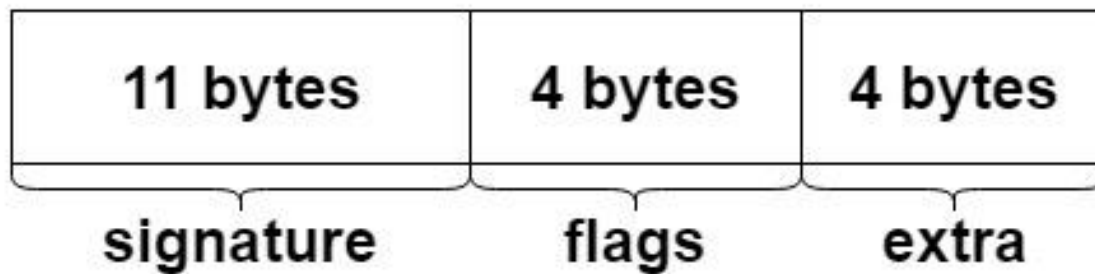
```
using (var conn = new NpgsqlConnection(DbCredentials.ConnectionString))
{
    await conn.OpenAsync().ConfigureAwait(false);
    using (var writer = conn.BeginBinaryImport(
        copyFromCommand: "COPY calls " +
        "(start_time,end_time,calling_number," +
        "called_number,duration,call_type,call_id) " +
        "FROM STDIN (FORMAT BINARY)"))
```



В коде с использованием Npgsql

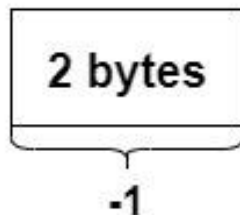
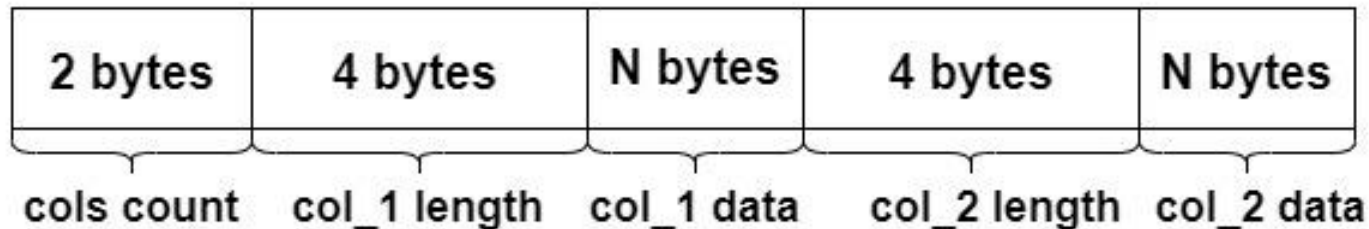
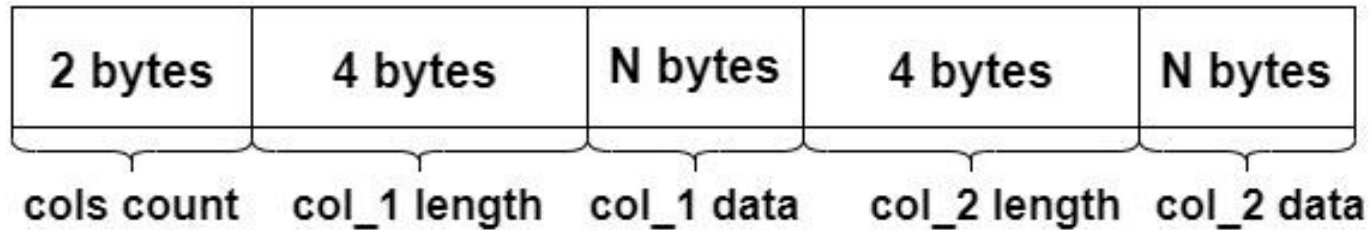
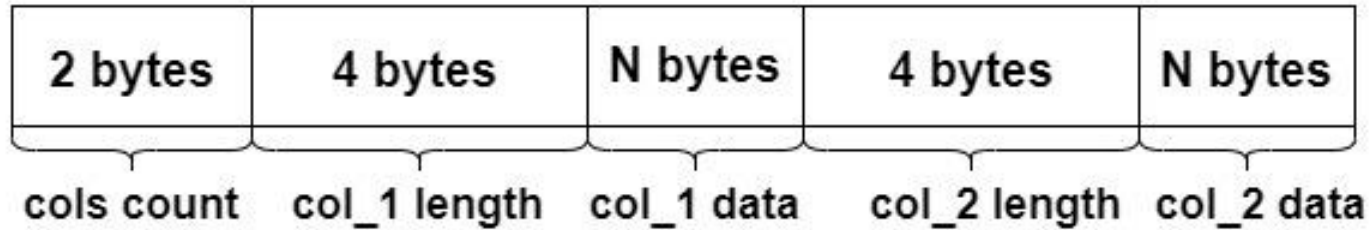
```
foreach (var call in calls)
{
    writer.StartRow();
    writer.Write(call.StartTime, NpgsqlDbType.Timestamp);
    writer.Write(call.EndTime, NpgsqlDbType.Timestamp);
    writer.Write(call.CallingNumber, NpgsqlDbType.Varchar);
    writer.Write(call.CalledNumber, NpgsqlDbType.Varchar);
    writer.Write(call.Duration, NpgsqlDbType.Integer);
    writer.Write(call.CallType);
    writer.Write(call.CallId, NpgsqlDbType.Varchar);
}
await writer.CompleteAsync().ConfigureAwait(continueOnCapturedContext: false);
```

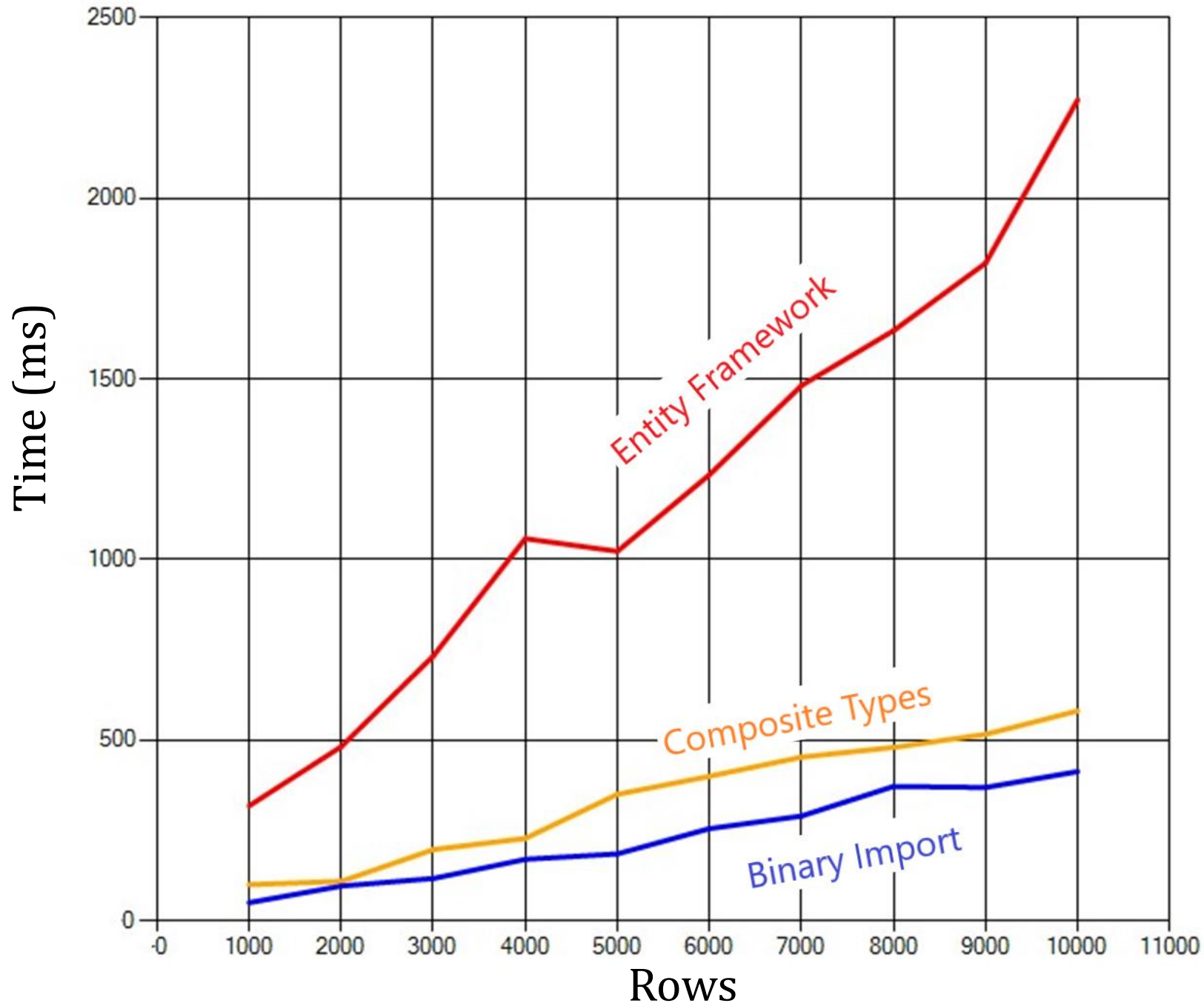
Структура заголовка

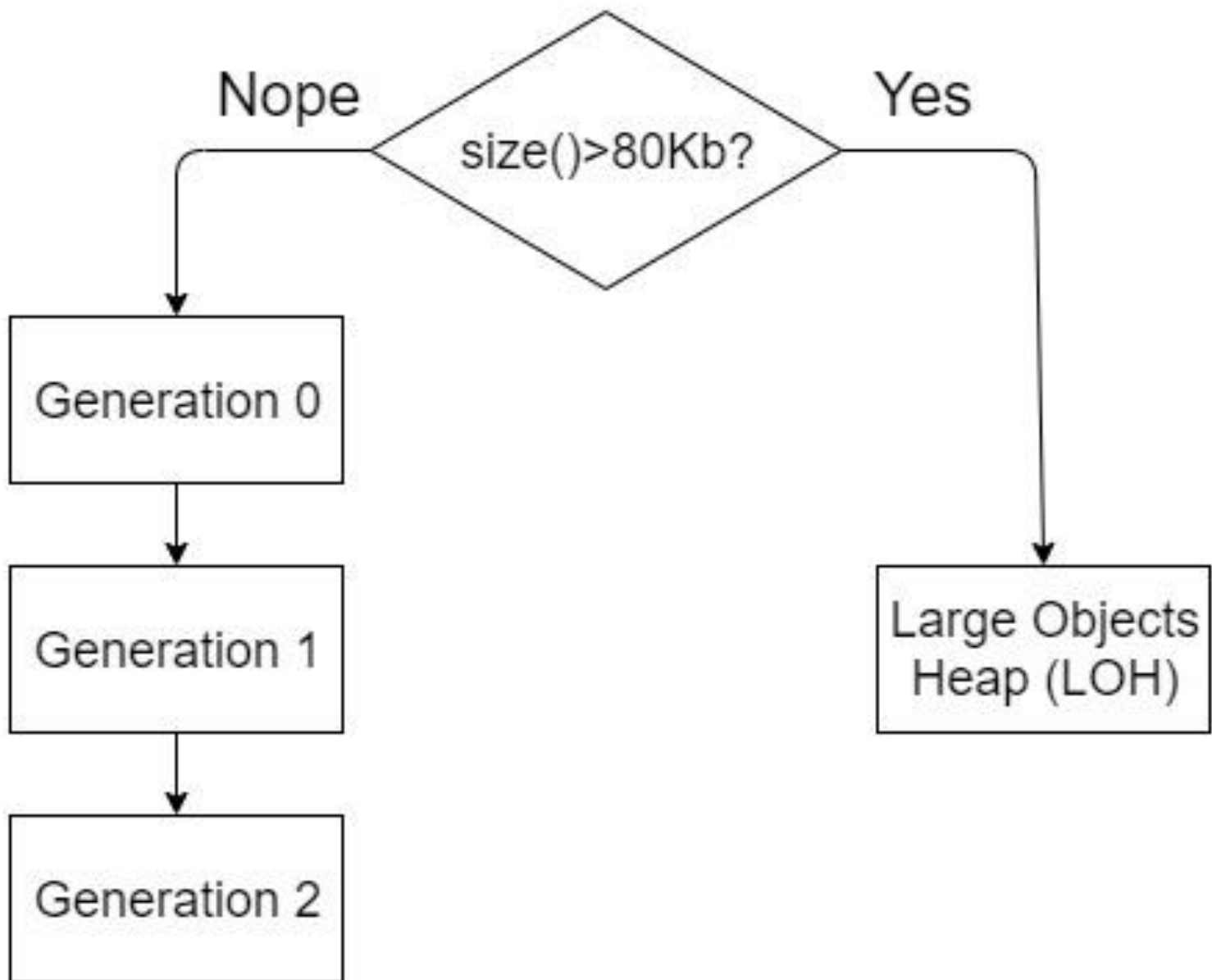




Структура области данных







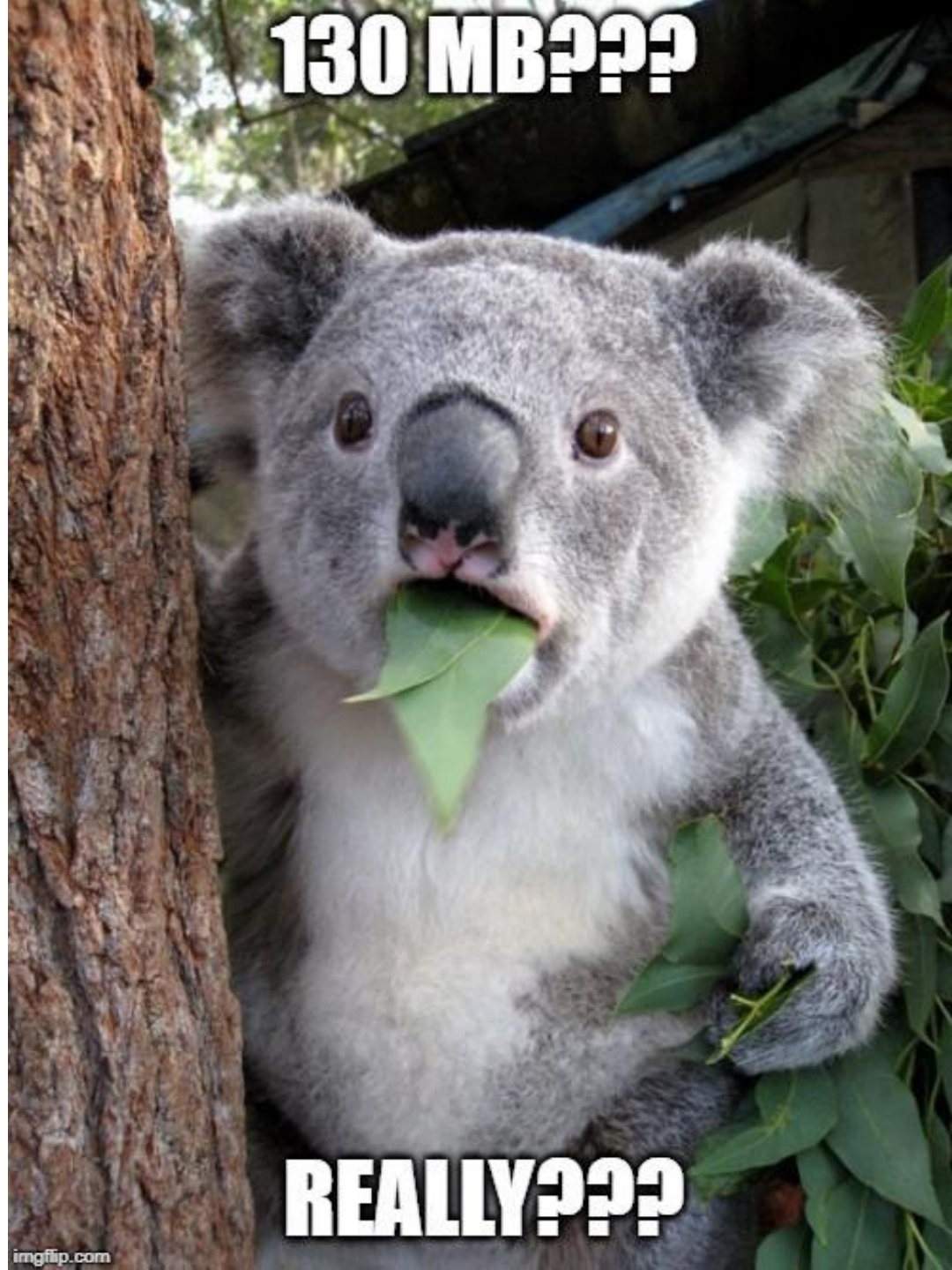


Entity Framework 10K rows	Composite types 10K rows	Binary import 10K rows
	20 Mb	21 Mb



Entity Framework 10K rows	Composite types 10K rows	Binary import 10K rows
130 Mb	20 Mb	21 Mb

130 MB???



REALLY???



Всё – ради генерации SQL

Top Methods

8,1 % ToString • 11 MB / 11 MB • System.Text.StringBuilder.ToString()

7,5 % Stack`1..ctor • 9,8 MB / 9,8 MB • System.Collections.Generic.Stack`1..ctor(Int32)

7,1 % ExpandByABlock • 9,2 MB / 9,2 MB • System.Text.StringBuilder.ExpandByABlock(Int32)

6,5 % Resize • 8,5 MB / 8,5 MB • System.Collections.Generic.Dictionary`2.Resize(Int32, Boolean)

6,1 % set_Capacity • 7,9 MB / 7,9 MB • System.Collections.Generic.List`1.set_Capacity(Int32)

5,2 % GenerateColumnModifications • 6,8 MB / 9,8 MB • Microsoft.EntityFrameworkCore.Update.ModificationCommand

5,0 % CreateDbParameter • 6,5 MB / 9,3 MB • Npgsql.NpgsqlCommand.CreateDbParameter()

4,8 % SortedSet`1+Enumerator..ctor • 6,2 MB / 16 MB • System.Collections.Generic.SortedSet`1+Enumerator..ctor(Sorted

4,5 % GetEnumerator • 5,9 MB / 18 MB • System.Collections.Generic.SortedDictionary`2+ValueCollection.GetEnumerator

2,8 % AddParameter • 3,7 MB / 5,5 MB • Microsoft.EntityFrameworkCore.Storage.RelationalCommandBuilderExtensions.

2,4 % GetClrValue • 3,2 MB / 3,2 MB • Microsoft.EntityFrameworkCore.Metadata.Internal.ClrPropertyGetter`2.GetClrValue

2,4 % ChangeCaseCommon • 3,2 MB / 3,2 MB • System.Globalization.TextInfo.ChangeCaseCommon(String)

2,3 % WriteWithLengthInternal • 3,0 MB / 3,0 MB • Npgsql.TypeHandling.NpgsqlSimpleTypeHandler`1.WriteWithLengthI

2,1 % CreateParameter • 2,8 MB / 2,8 MB • Npgsql.NpgsqlCommand.CreateParameter()

1,8 % Concat • 2,3 MB / 2,3 MB • System.String.Concat(String, String)

1,7 % Ctor • 2,3 MB / 2,3 MB • System.String.Ctor(ReadOnlySpan)

1,5 % Initialize • 1,9 MB / 1,9 MB • System.Collections.Generic.Dictionary`2.Initialize(Int32)

1,4 % List`1..ctor • 1,9 MB / 1,9 MB • System.Collections.Generic.List`1..ctor(Int32)



Large objects heap (!)

Top Methods

38,0 % ToString • **6,4 MB** / 6,4 MB • System.Text.**StringBuilder**.ToString()
29,7 % Resize • **5,0 MB** / 5,0 MB • System.Collections.Generic.**Dictionary`2**.Resize(Int32, Boolean)
19,0 % ExpandByABlock • **3,2 MB** / 3,2 MB • System.Text.**StringBuilder**.ExpandByABlock(Int32)
9,5 % Initialize • **1,6 MB** / 1,6 MB • System.Collections.Generic.**Dictionary`2**.Initialize(Int32)
2,3 % SetCapacity • **0,4 MB** / 0,4 MB • System.Collections.Generic.**HashSet`1**.SetCapacity(Int32)
1,5 % set_Capacity • **0,3 MB** / 0,3 MB • System.Collections.Generic.**List`1**.set_Capacity(Int32)



	Entity Framework	Composite types	Binary import
Скорость	-	+	+
Память	-	+	+
Простота кода	+	-	+ -



	Entity Framework	Composite types	Binary import
Скорость	-	+	+
Память	-	+	+
Простота кода	+	-	+ -



Почему вставка в большие таблицы – медленная?





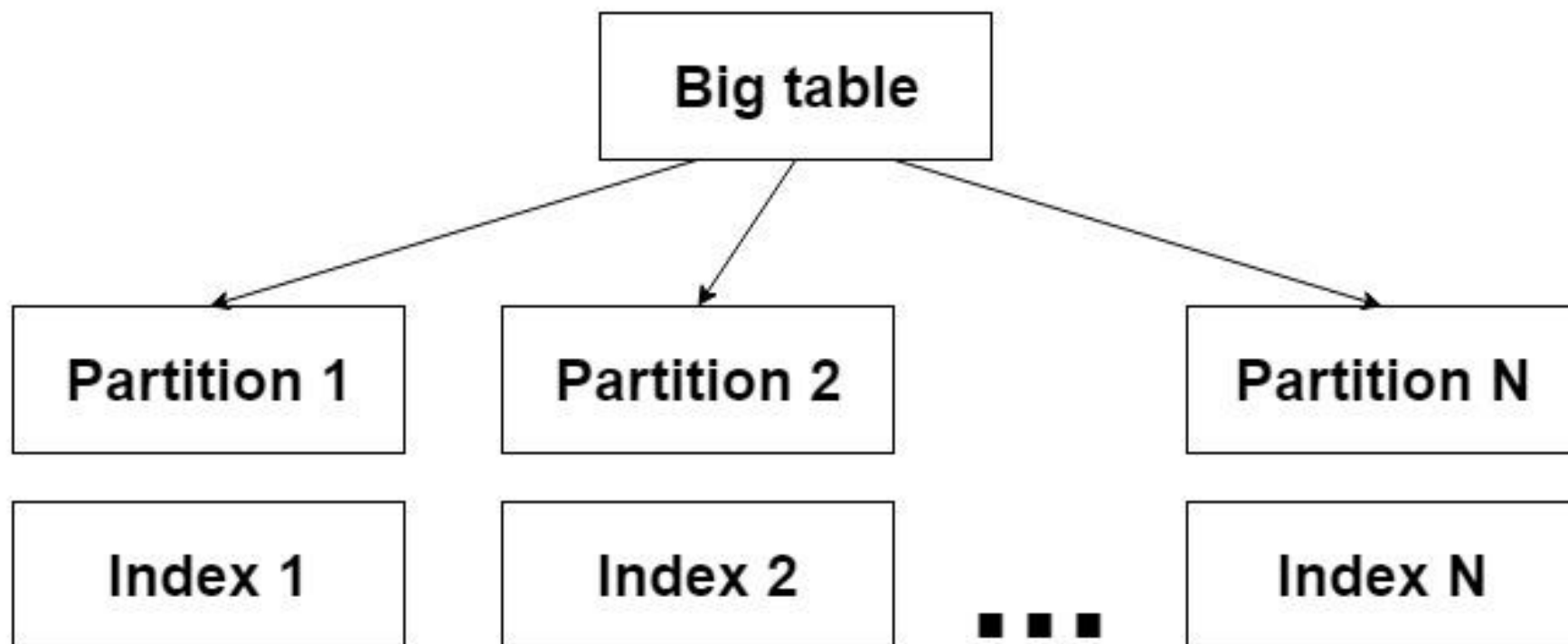
→ CPU full-time выполняет задачи высокого приоритета



- CPU full-time выполняет задачи высокого приоритета
- CPU активен 90-100% времени



- CPU full-time выполняет задачи высокого приоритета
- CPU активен 90-100% времени
- Износ систем охлаждения





Партиционирование – удобно!

```
create table logs  
(  
  time      timestamp,  
  level     text,  
  message   text  
) partition by list (level);
```



Партиционирование – удобно!

```
create table logs  
(  
    time      timestamp,  
    level     text,  
    message   text  
) partition by list (level);
```

```
create table logs_info  
    partition of logs  
    for values in ('INFO');
```




Партиционирование – удобно!

```
create table logs  
(  
    time      timestamp,  
    level     text,  
    message   text  
) partition by list (level);
```

```
create table logs_info  
partition of logs  
for values in ('INFO');
```

```
select * from logs;
```

```
insert into logs (time, level, message)  
values ('2019-11-28 20:45:00', 'INFO', 'Hello partition!')
```

Главное – правильно выбрать ключ!



→ По значению столбца



Главное – правильно выбрать ключ!

- По значению столбца
- По диапазону значений



Главное – правильно выбрать ключ!

- По значению столбца
- По диапазону значений
- По хешу



Главное – правильно выбрать ключ!

- По значению столбца
- По диапазону значений
- По хешу

pgxn 4.2.2

PG Partition Manager

С COPY - проблемка 😊



→ Партицию нужно указать явно

```
copy logs_info (time, level, message)  
from stdin (format binary )
```



- B-Tree занимает меньше памяти (<40%)
- Таблицы с тысячами партиций работают быстрее
- COPY можно нацелить в основную таблицу:

```
copy logs (time, level, message)  
from stdin (format binary )
```



→ Решает проблему громоздкого кода

→ <https://github.com/UGeneF/Copy>





Было много кода

```
using (var conn = new NpgsqlConnection("your connection string"))
{
    await conn.OpenAsync().ConfigureAwait(false);
    using (var writer = conn.BeginBinaryImport(
        copyFromCommand: "COPY calls " +
        "(start_time,end_time,calling_number," +
        "called_number,duration,call_type,call_id) " +
        "FROM STDIN (FORMAT BINARY)"))
    {
        foreach (var call in calls)
        {
            writer.StartRow();
            writer.Write(call.StartTime, NpgsqlDbType.Timestamp);
            writer.Write(call.EndTime);
            writer.Write(call.CallingNumber);
            writer.Write(call.CalledNumber);
            writer.Write(call.Duration, NpgsqlDbType.Integer);
            writer.Write(call.CallType);
            writer.Write(call.CallId);
        }

        await writer.CompleteAsync().ConfigureAwait(continueOnCapturedContext: false);
    }
}
```

Сделали лаконично и удобно



```
using (var context = new BillingContext())  
{  
    await context.BulkCopyAsync(calls).ConfigureAwait(false);  
}
```



Copy 1.0.0

Npgsql extension for simple binary copy

Package Manager

.NET CLI

PackageReference

Paket CLI

PM> Install-Package Copy -Version 1.0.0



> Dependencies

> GitHub Usage



Если что – пишите:



@ugenef

