CC BY
Shisma
Wikimedia Commons

Wait! What’s going on inside my database?

PostgreSQL and Optimizing Database Performance

Jeremy Schneider

SCaLE 21x
3-14-2024

About PostgreSQL

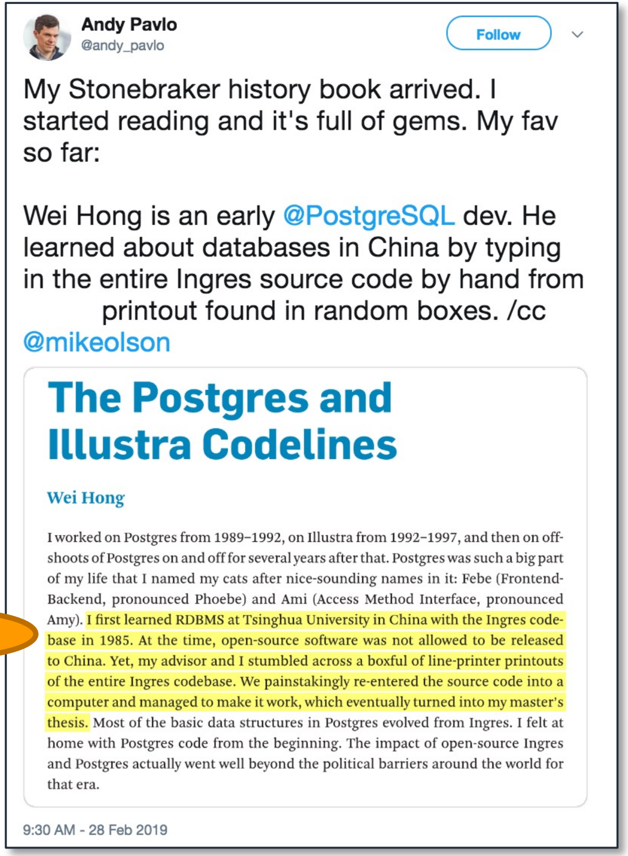
1970: Mathematician Edgar F. Codd, working as researcher for IBM, publishes “A Relational Model of Data for Large Shared Data Banks”

1973: Michael Stonebraker and Eugene Wong at University of California Berkeley seek funding and begin development of a relational database called INGRES

1986: Michael Stonebraker and Lawrence A. Rowe at University of California Berkeley publish “The Design of POSTGRES” – a new database that is the successor to INGRES

1994: Andrew Yu and Jolly Chen at University of California Berkeley add support for the SQL language

1996: Transition to non-university core team of volunteers, official release under new name POSTGRESQL



Andy Pavlo
@andy_pavlo

My Stonebraker history book arrived. I started reading and it's full of gems. My fav so far:

Wei Hong is an early @PostgreSQL dev. He learned about databases in China by typing in the entire Ingres source code by hand from printout found in random boxes. /cc @mikeolson

The Postgres and Illustra Codelines

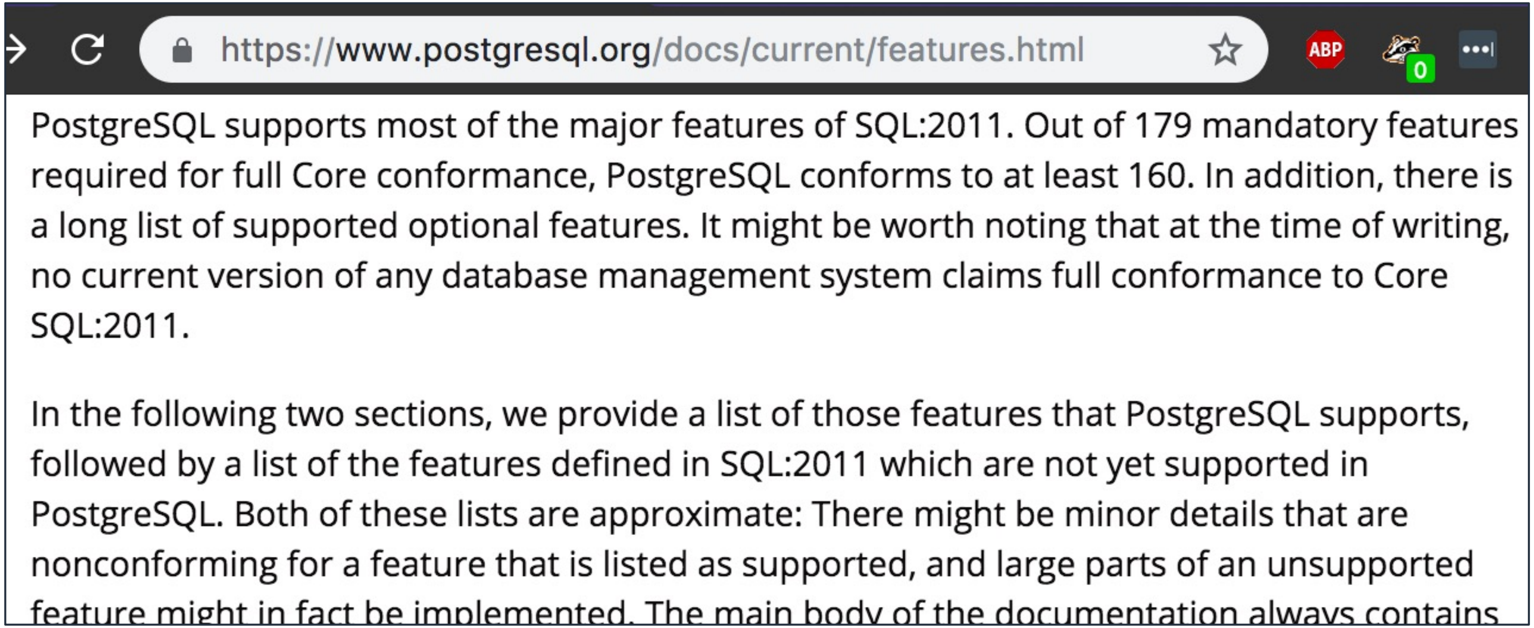
Wei Hong

I worked on Postgres from 1989–1992, on Illustra from 1992–1997, and then on offshoots of Postgres on and off for several years after that. Postgres was such a big part of my life that I named my cats after nice-sounding names in it: Febe (Frontend-Backend, pronounced Phoebe) and Ami (Access Method Interface, pronounced Amy). I first learned RDBMS at Tsinghua University in China with the Ingres codebase in 1985. At the time, open-source software was not allowed to be released to China. Yet, my advisor and I stumbled across a boxful of line-printer printouts of the entire Ingres codebase. We painstakingly re-entered the source code into a computer and managed to make it work, which eventually turned into my master's thesis. Most of the basic data structures in Postgres evolved from Ingres. I felt at home with Postgres code from the beginning. The impact of open-source Ingres and Postgres actually went well beyond the political barriers around the world for that era.

9:30 AM - 28 Feb 2019

1985

About PostgreSQL

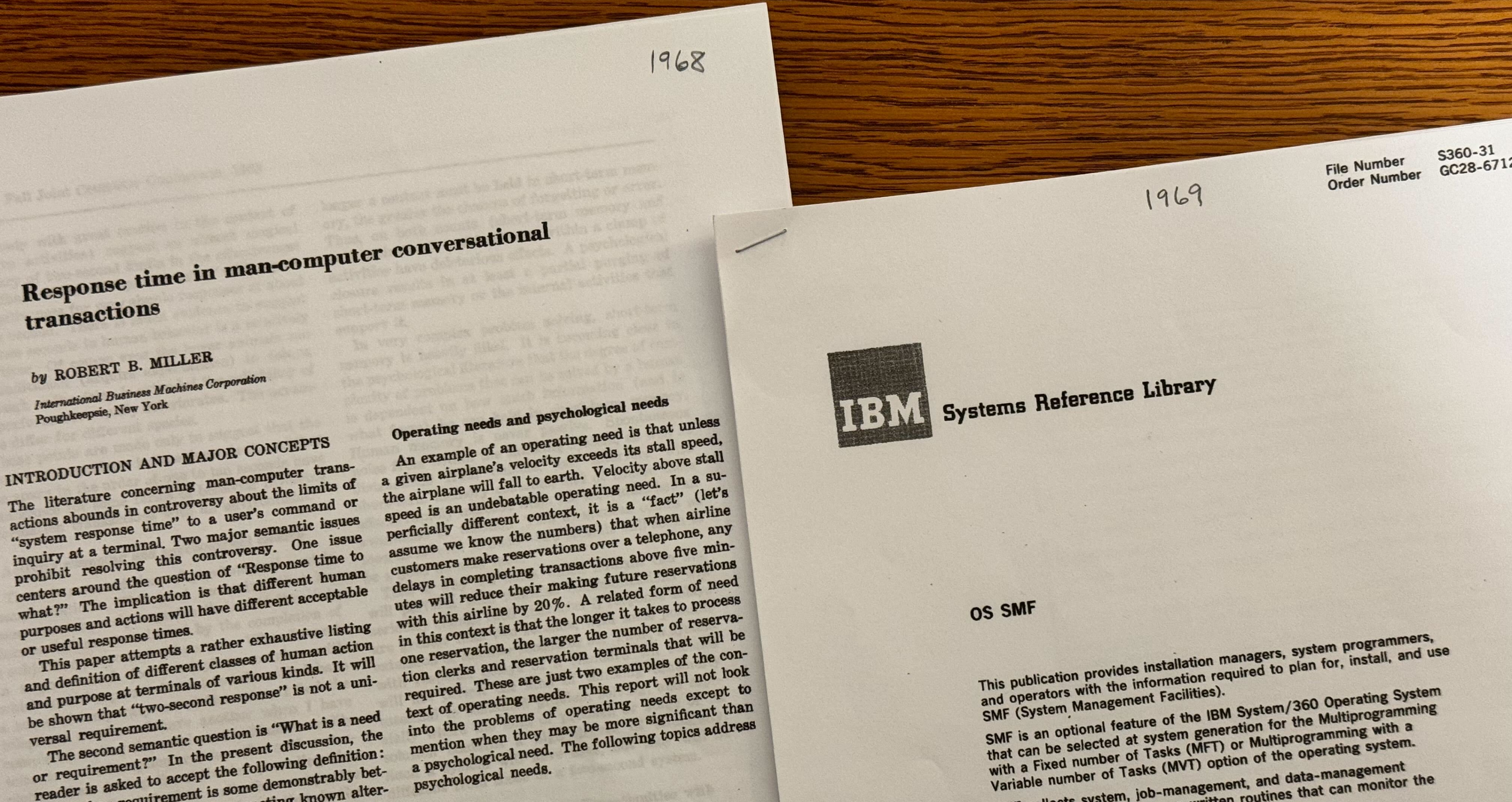


PostgreSQL supports most of the major features of SQL:2011. Out of 179 mandatory features required for full Core conformance, PostgreSQL conforms to at least 160. In addition, there is a long list of supported optional features. It might be worth noting that at the time of writing, no current version of any database management system claims full conformance to Core SQL:2011.

In the following two sections, we provide a list of those features that PostgreSQL supports, followed by a list of the features defined in SQL:2011 which are not yet supported in PostgreSQL. Both of these lists are approximate: There might be minor details that are nonconforming for a feature that is listed as supported, and large parts of an unsupported feature might in fact be implemented. The main body of the documentation always contains

About Database Performance

Wait-Events
Logs
Utilization
USE-Method
Method-R
RED-Method
Perf
Observability
Flame-Graphs
Dashboards
Tracing
YAPP
AAS
TSA-Method
eBPF
Monitoring
GDB



1968

Response time in man-computer conversational transactions

by ROBERT B. MILLER

International Business Machines Corporation
Poughkeepsie, New York

INTRODUCTION AND MAJOR CONCEPTS

The literature concerning man-computer transactions abounds in controversy about the limits of "system response time" to a user's command or inquiry at a terminal. Two major semantic issues prohibit resolving this controversy. One issue centers around the question of "Response time to what?" The implication is that different human purposes and actions will have different acceptable or useful response times.

This paper attempts a rather exhaustive listing and definition of different classes of human action and purpose at terminals of various kinds. It will be shown that "two-second response" is not a universal requirement.

The second semantic question is "What is a need or requirement?" In the present discussion, the reader is asked to accept the following definition:
requirement is some demonstrably bet-
ting known alter-

Operating needs and psychological needs

An example of an operating need is that unless a given airplane's velocity exceeds its stall speed, the airplane will fall to earth. Velocity above stall speed is an undebatable operating need. In a superficially different context, it is a "fact" (let's assume we know the numbers) that when airline customers make reservations over a telephone, any delays in completing transactions above five minutes will reduce their making future reservations with this airline by 20%. A related form of need in this context is that the longer it takes to process one reservation, the larger the number of reservation clerks and reservation terminals that will be required. These are just two examples of the context of operating needs. This report will not look into the problems of operating needs except to mention when they may be more significant than a psychological need. The following topics address psychological needs.

1969

File Number S360-31
Order Number GC28-6712

IBM Systems Reference Library

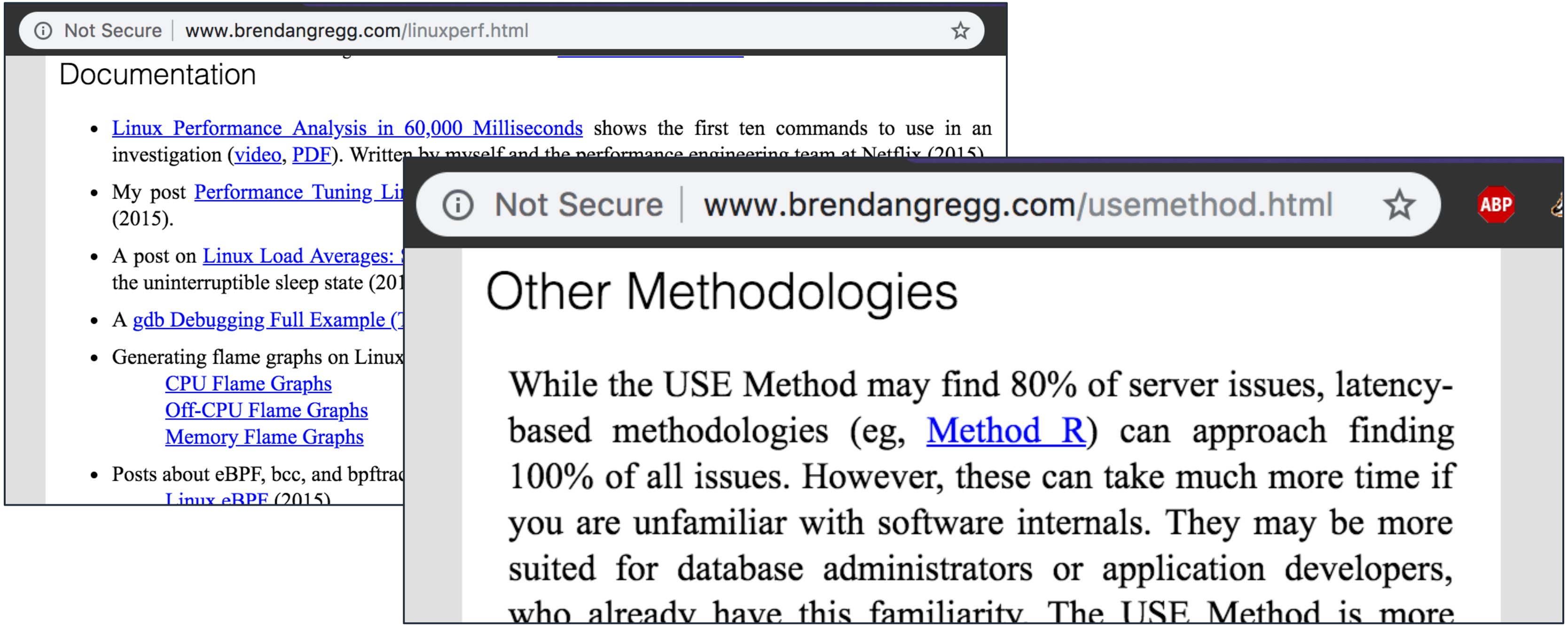
OS SMF

This publication provides installation managers, system programmers, and operators with the information required to plan for, install, and use SMF (System Management Facilities).

SMF is an optional feature of the IBM System/360 Operating System that can be selected at system generation for the Multiprogramming with a Fixed number of Tasks (MFT) or Multiprogramming with a Variable number of Tasks (MVT) option of the operating system.

llects system, job-management, and data-management
written routines that can monitor the

About Database Performance



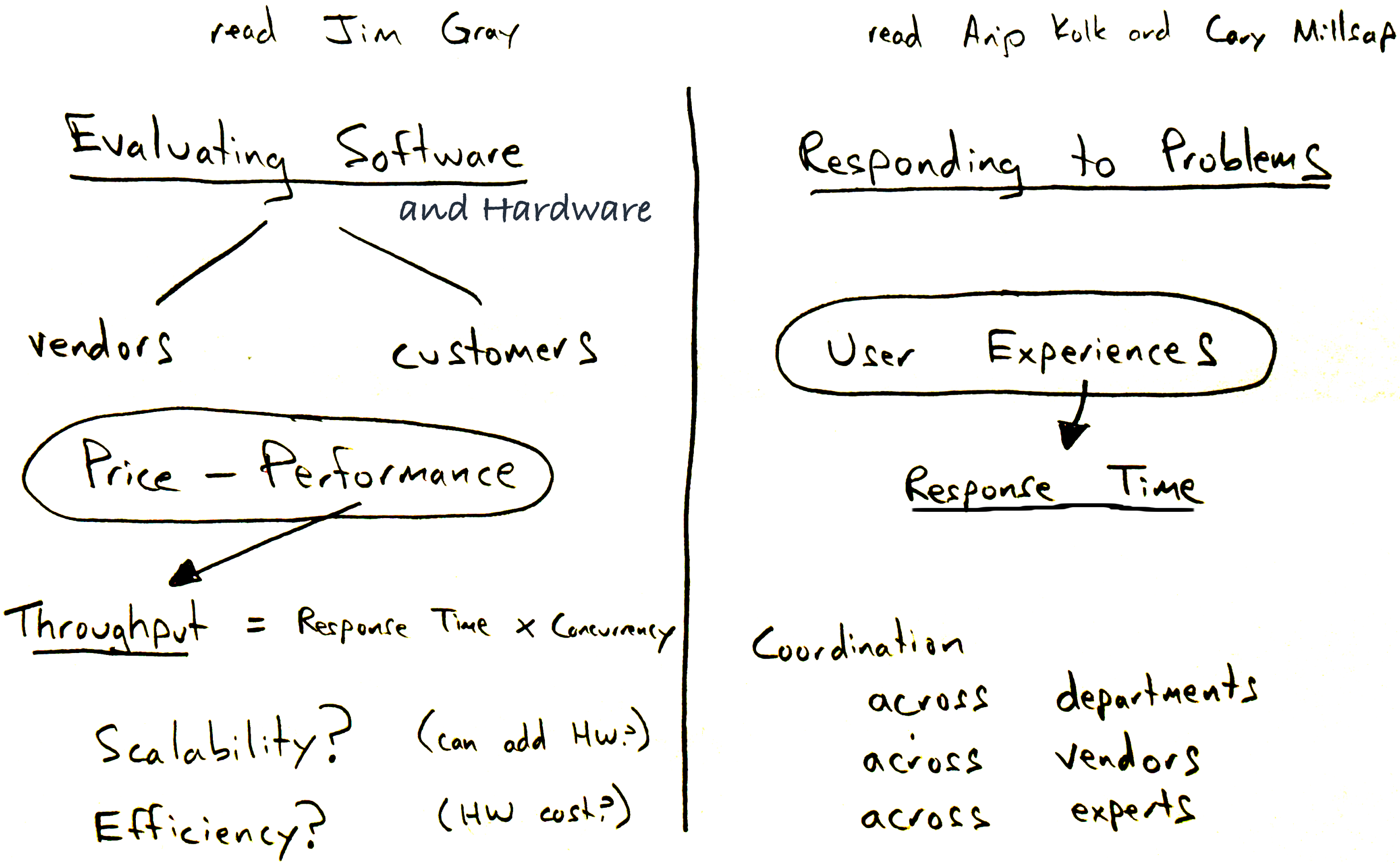
Documentation

- Linux Performance Analysis in 60,000 Milliseconds shows the first ten commands to use in an investigation (video, PDF). Written by myself and the performance engineering team at Netflix (2015).
- My post Performance Tuning Li
(2015).
- A post on Linux Load Averages:
the uninterruptible sleep state (201
- A gdb Debugging Full Example (
- Generating flame graphs on Linux
  - CPU Flame Graphs
  - Off-CPU Flame Graphs
  - Memory Flame Graphs
- Posts about eBPF, bcc, and bpftrac
  - Linux eBPF (2015)

Other Methodologies

While the USE Method may find 80% of server issues, latency-based methodologies (eg, Method R) can approach finding 100% of all issues. However, these can take much more time if you are unfamiliar with software internals. They may be more suited for database administrators or application developers, who already have this familiarity. The USE Method is more

About Database Performance



read Jim Gray

Evaluating Software

and Hardware

vendors

customers

Price – Performance

Throughput = Response Time × Concurrency

Scalability? (can add HW?)

Efficiency? (HW cost?)

read Anjo Kolk and Cary Millsap

Responding to Problems

User Experiences

Response Time

Coordination
across departments
across vendors
across experts

About Database Performance

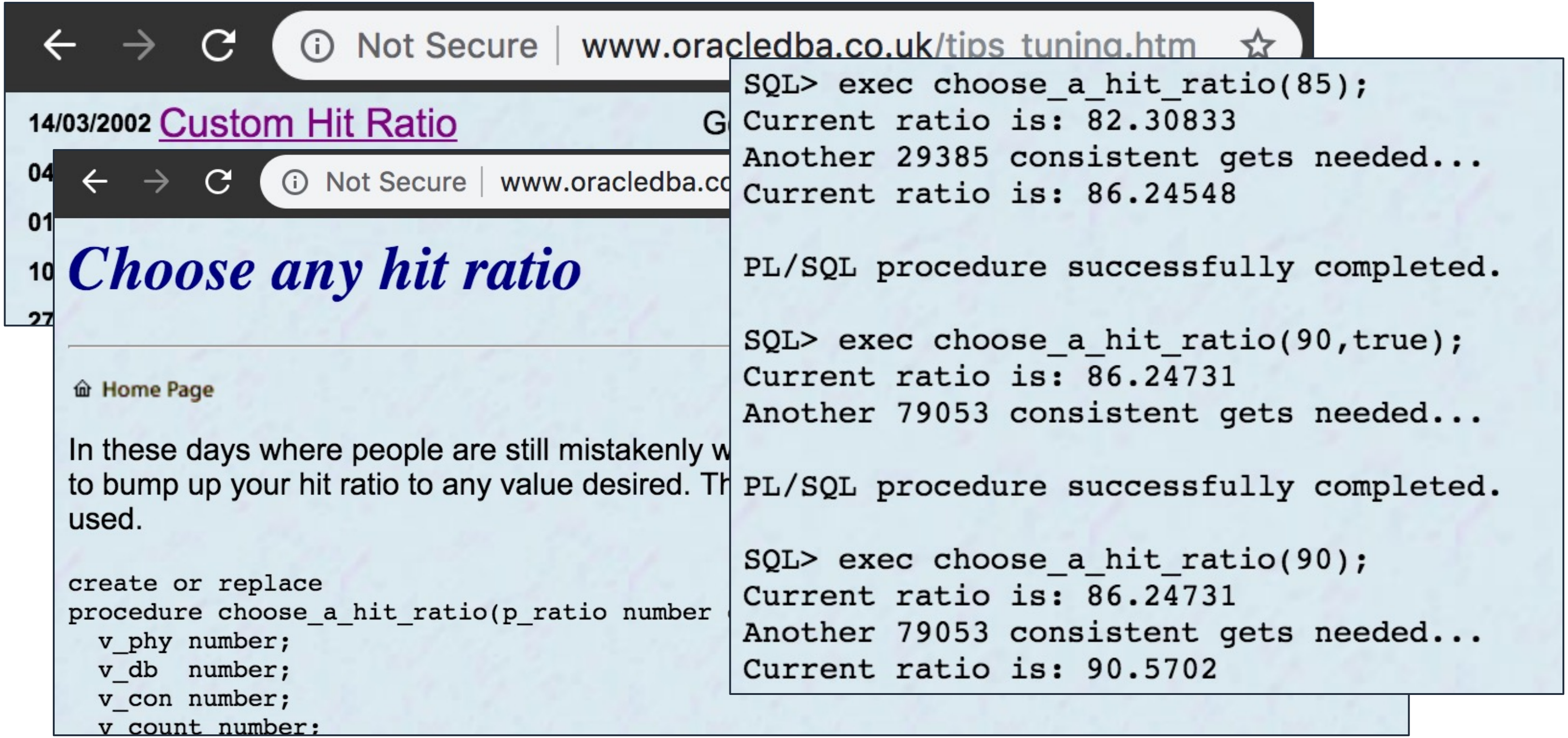
Delfador Chibi by Peileppe
CC0

1990’s Manager:

“Dear DBA: Expert consultants have taught us that if the Buffer Cache Hit Ratio (BCHR) is below 90% then the system immediately needs an expensive tuning engagement.

Please report any databases that have BCHR < 90%.”

About Database Performance



14/03/2002 Custom Hit Ratio G

04

01

10

27

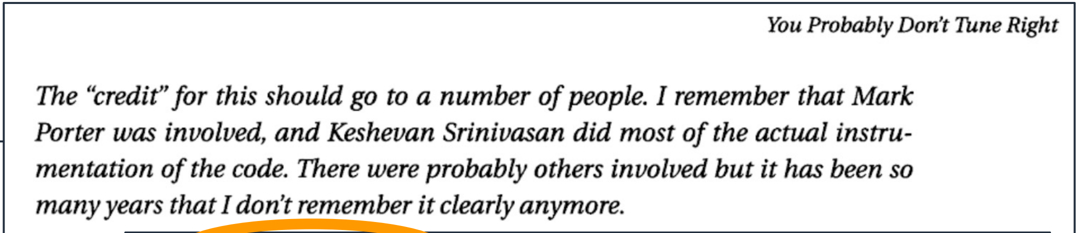
Choose any hit ratio

In these days where people are still mistakenly w
to bump up your hit ratio to any value desired. Th
used.

```
create or replace
procedure choose_a_hit_ratio(p_ratio number
  v_phy number;
  v_db  number;
  v_con number;
  v_count number;
```

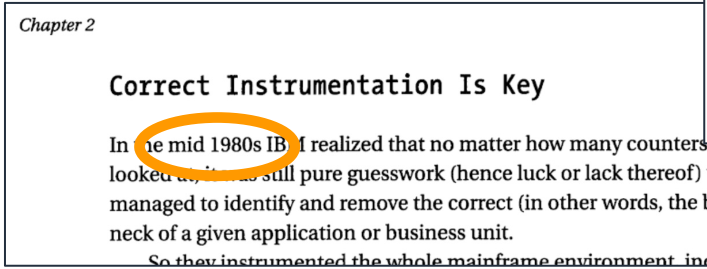
```
SQL> exec choose_a_hit_ratio(85);
Current ratio is: 82.30833
Another 29385 consistent gets needed...
Current ratio is: 86.24548

PL/SQL procedure successfully completed.

SQL> exec choose_a_hit_ratio(90,true);
Current ratio is: 86.24731
Another 79053 consistent gets needed...

PL/SQL procedure successfully completed.

SQL> exec choose_a_hit_ratio(90);
Current ratio is: 86.24731
Another 79053 consistent gets needed...
Current ratio is: 90.5702
```

About Database Performance



You Probably Don't Tune Right

The "credit" for this should go to a number of people. I remember that Mark Porter was involved, and Keshevan Srinivasan did most of the actual instrumentation of the code. There were probably others involved but it has been so many years that I don't remember it clearly anymore.

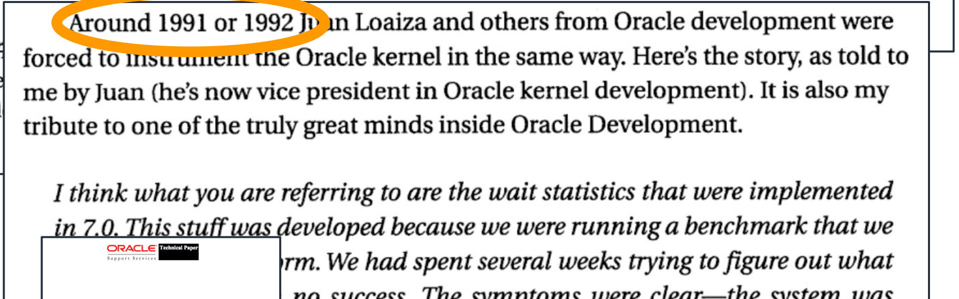


Chapter 2

Correct Instrumentation Is Key

In the mid 1980s IBM realized that no matter how many counters and ratios th looked at, it was still pure guesswork (hence luck or lack thereof) whether a pe managed to identify and remove the correct (in other words, the biggest) bottl neck of a given application or business unit.

So they instrumented the whole mainframe environment, including DB2



Around 1991 or 1992 Juan Loaiza and others from Oracle development were forced to instrument the Oracle kernel in the same way. Here's the story, as told to me by Juan (he's now vice president in Oracle kernel development). It is also my tribute to one of the truly great minds inside Oracle Development.

I think what you are referring to are the wait statistics that were implemented in 7.0. This stuff was developed because we were running a benchmark that we ...*rm. We had spent several weeks trying to figure out what* ... *no success. The symptoms were clear—the system was* ... *couldn't figure out why.*

...*tistics and ratios and kept coming up with theories, the* ...*e of them were right. So we wasted weeks tuning and fix-*

Nørgaard, Mogens et al. *Oracle Insights: Tales of the Oak Table*. Berkeley, CA: Apress/OakTable Press, 2004, p76-77.

Oracle Insights Tales of the
Troubleshooting Oracle Performance
Optimizing Oracle Performance
Oracle Database Release 2 Tuning Tips
Cost-Based Oracle Fundamentals
SQL Performance Explained
Forecasting Oracle Performance

ORACLE Support Services Technical Paper

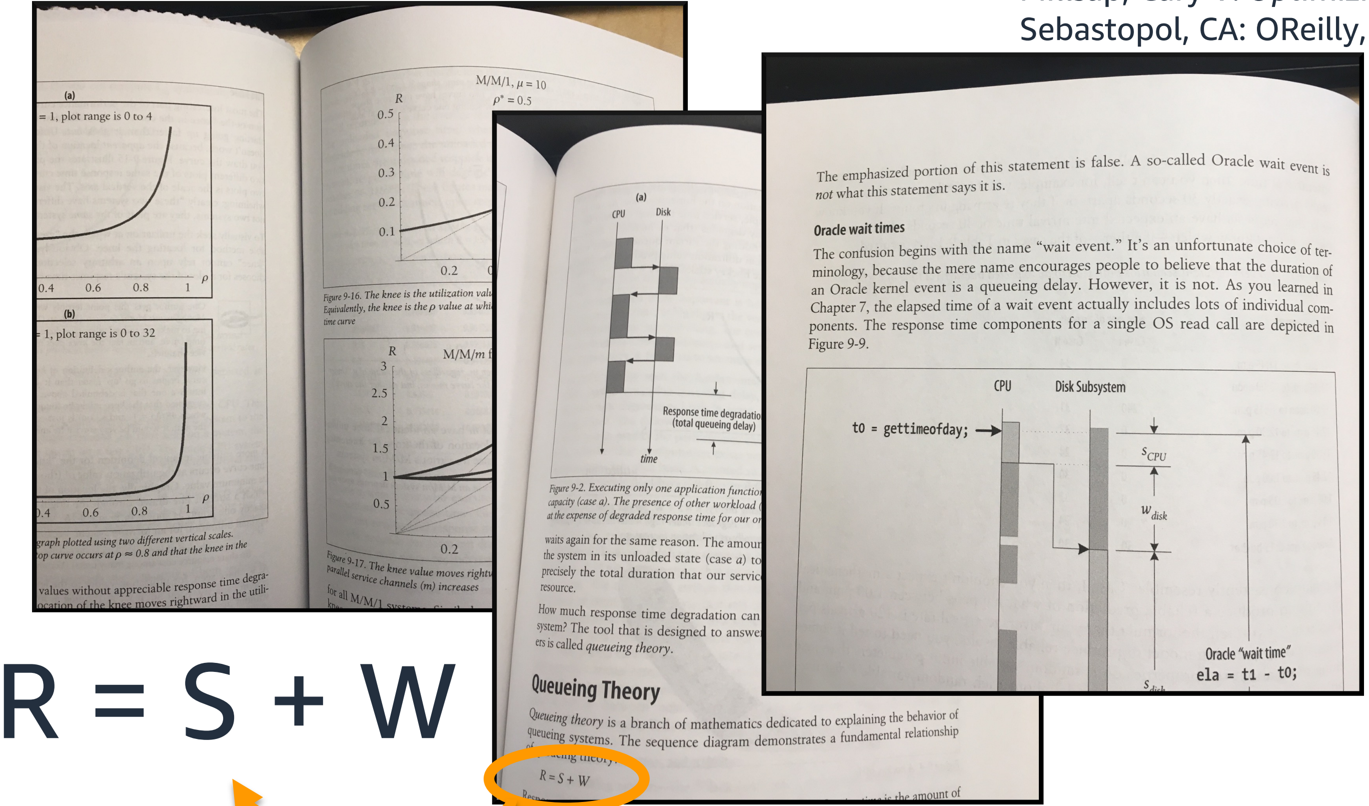
Yet Another Performance Profiling Method (Or YAPP-Method)

Anjo Kolk, Shari Yamaguchi – Data Server Applied Technologies
Jim Viscusi – Oracle Support Services Centers of Expertise

Oracle Corporation
June 1999

About Database Performance

Millsap, Cary V. *Optimizing Oracle Performance*. Sebastopol, CA: OReilly, 2003. p225, 240, 258-259



(a)
= 1, plot range is 0 to 4
0.4
0.6
0.8
1
ρ
(b)
= 1, plot range is 0 to 32
0.4
0.6
0.8
1
ρ
M/M/1, μ = 10
ρ* = 0.5
R
0.5
0.4
0.3
0.2
0.1
0.2
M/M/m f
3
2.5
2
1.5
1
0.5
0.2
(a)
CPU
Disk
Response time degradatio
(total queueing delay)
time
CPU
Disk Subsystem
s_{CPU}
w_{disk}
s_{disk}
Oracle "wait time"

graph plotted using two different vertical scales. op curve occurs at $\rho \approx 0.8$ and that the knee in the

values without appreciable response time degra- ocation of the knee moves rightward in the utili-

Figure 9-16. The knee is the utilization valu Equivalently, the knee is the ρ value at whi time curve

Figure 9-17. The knee value moves rightu parallel service channels (m) increases

for all M/M/1 system

Figure 9-2. Executing only one application functio capacity (case a). The presence of other workload (at the expense of degraded response time for our or

waits again for the same reason. The amour the system in its unloaded state (case *a*) to precisely the total duration that our servic resource.

How much response time degradation can system? The tool that is designed to answe ers is called *queueing theory*.

Queueing Theory

Queueing theory is a branch of mathematics dedicated to explaining the behavior of queueing systems. The sequence diagram demonstrates a fundamental relationship of queueing theory:

$$R = S + W$$

Respo ... time is the amount of

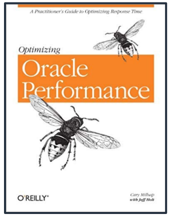
The emphasized portion of this statement is false. A so-called Oracle wait event is *not* what this statement says it is.

Oracle wait times

The confusion begins with the name "wait event." It's an unfortunate choice of terminology, because the mere name encourages people to believe that the duration of an Oracle kernel event is a queueing delay. However, it is not. As you learned in Chapter 7, the elapsed time of a wait event actually includes lots of individual components. The response time components for a single OS read call are depicted in Figure 9-9.

```
t0 = gettimeofday;
```

```
ela = t1 - t0;
```



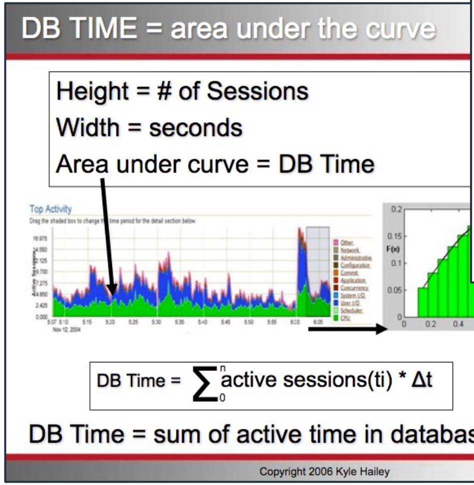
"How long the SQL takes to run"

See also:

- Shallahamer, Craig. *Forecasting Oracle Performance*. Berkeley, CA: Apress, 2007.

R = S + W

About Database Performance



DB TIME = area under the curve

Height = # of Sessions

Width = seconds

Area under curve = DB Time

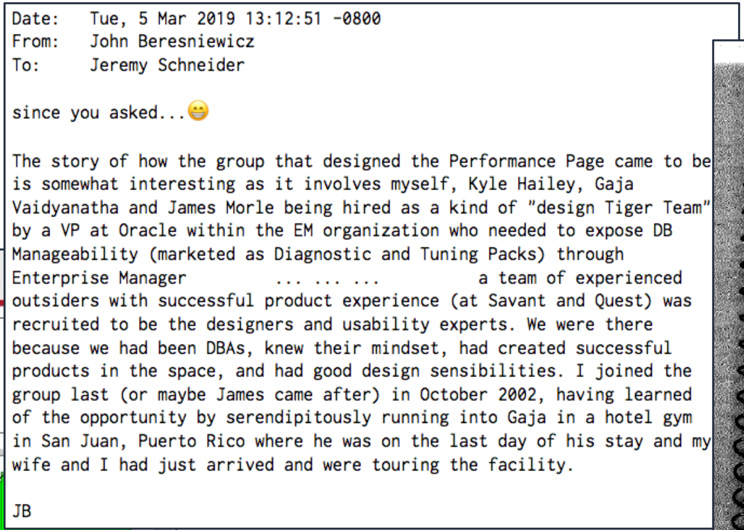
Top Activity
Drag the shaded box to change the time period for the detail section below
F(x)

DB Time = $\sum_{0}^{n}$ active sessions(ti) * Δt

DB Time = sum of active time in database

Copyright 2006 Kyle Hailey

#.20



```
Date:    Tue, 5 Mar 2019 13:12:51 -0800
From:    John Beresniewicz
To:      Jeremy Schneider
```

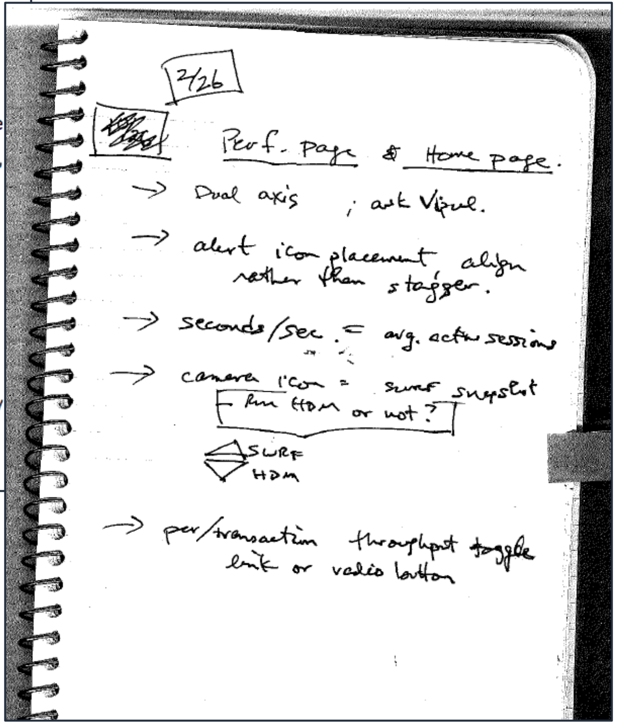
since you asked...😁

The story of how the group that designed the Performance Page came to be is somewhat interesting as it involves myself, Kyle Hailey, Gaja Vaidyanatha and James Morle being hired as a kind of "design Tiger Team" by a VP at Oracle within the EM organization who needed to expose DB Manageability (marketed as Diagnostic and Tuning Packs) through Enterprise Manager ... ... a team of experienced outsiders with successful product experience (at Savant and Quest) was recruited to be the designers and usability experts. We were there because we had been DBAs, knew their mindset, had created successful products in the space, and had good design sensibilities. I joined the group last (or maybe James came after) in October 2002, having learned of the opportunity by serendipitously running into Gaja in a hotel gym in San Juan, Puerto Rico where he was on the last day of his stay and my wife and I had just arrived and were touring the facility.

JB

Images & Quotes
Used With Permission

(JB's notebook, 2004)



2/26

Perf. Page & Home page.

→ Dual axis ; ask Vipul.

→ alert icon placement align rather than stagger.

→ seconds/sec = avg. active sessions

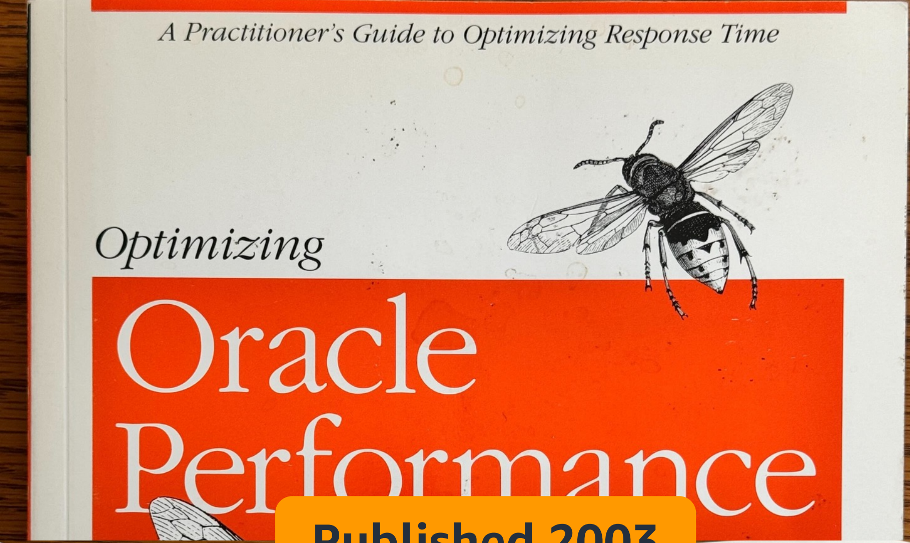
→ camera icon = same snapshot
[Run ADDM or not?]

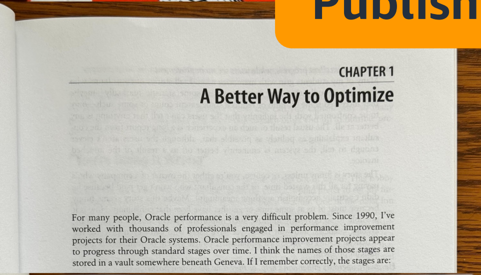
SWRF
ADDM

→ per/transaction throughput toggle link or radio button

Active Session Sampling

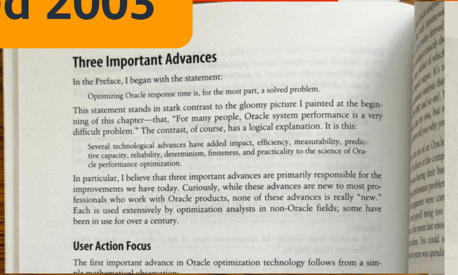
Published 2003




CHAPTER 1

A Better Way to Optimize

For many people, Oracle performance is a very difficult problem. Since 1990, I've worked with thousands of professionals engaged in performance improvement projects for their Oracle systems. Oracle performance improvement projects appear to progress through standard stages over time. I think the names of those stages are stored in a vault somewhere beneath Geneva. If I remember correctly, the stages are:


Three Important Advances

In the Preface, I began with the statement:

> Optimizing Oracle response time is, for the most part, a solved problem.

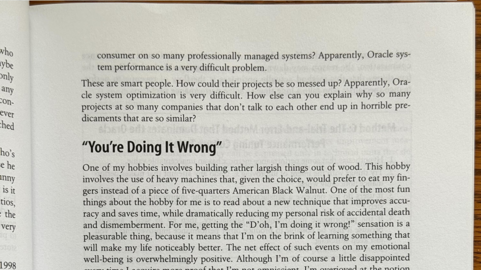
This statement stands in stark contrast to the gloomy picture I painted at the beginning of this chapter—that, "For many people, Oracle system performance is a very difficult problem." The contrast, of course, has a logical explanation. It is this:

> Several technological advances have added impact, efficiency, measurability, predictive capacity, reliability, determinism, finiteness, and practicality to the science of Oracle performance optimization.

In particular, I believe that three important advances are primarily responsible for the improvements we have today. Curiously, while these advances are new to most professionals who work with Oracle products, none of these advances is really "new." Each is used extensively by optimization analysts in non-Oracle fields; some have been in use for over a century.

User Action Focus

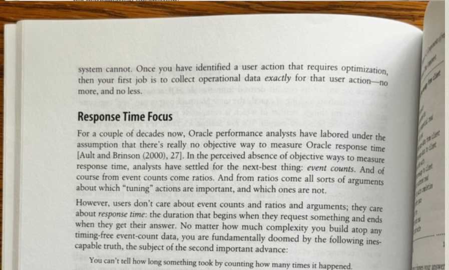
The first important advance in Oracle optimization technology follows from a sim-


consumer on so many professionally managed systems? Apparently, Oracle system performance is a very difficult problem.

These are smart people. How could their projects be so messed up? Apparently, Oracle system optimization is very difficult. How else can you explain why so many projects at so many companies that don't talk to each other end up in horrible predicaments that are so similar?

"You're Doing It Wrong"

One of my hobbies involves building rather largish things out of wood. This hobby involves the use of heavy machines that, given the choice, would prefer to eat my fingers instead of a piece of five-quarters American Black Walnut. One of the most fun things about the hobby for me is to read about a new technique that improves accuracy and saves time, while dramatically reducing my personal risk of accidental death and dismemberment. For me, getting the "D'oh, I'm doing it wrong!" sensation is a pleasurable thing, because it means that I'm on the brink of learning something that will make my life noticeably better. The net effect of such events on my emotional well-being is overwhelmingly positive. Although I'm of course a little disappointed


system cannot. Once you have identified a user action that requires optimization, then your first job is to collect operational data *exactly* for that user action—no more, and no less.

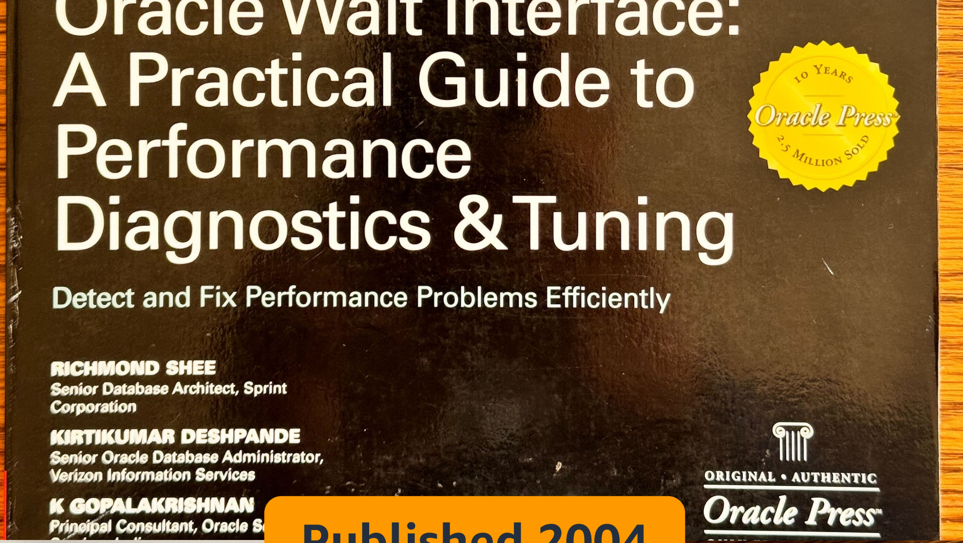
Response Time Focus

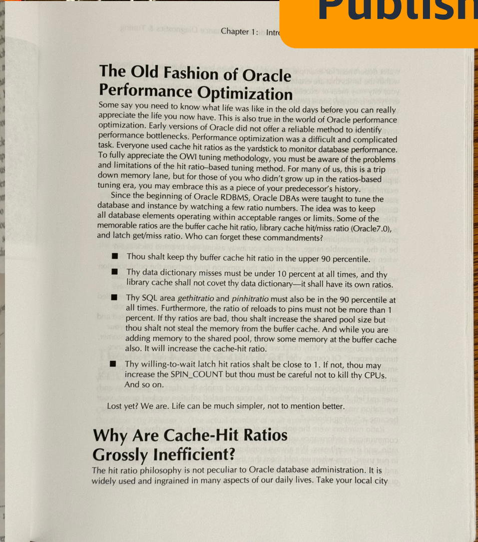
For a couple of decades now, Oracle performance analysts have labored under the assumption that there's really no objective way to measure Oracle response time [Ault and Brinson (2000), 27]. In the perceived absence of objective ways to measure response time, analysts have settled for the next-best thing: *event counts*. And of course from event counts come ratios. And from ratios come all sorts of arguments about which "tuning" actions are important, and which ones are not.

However, users don't care about event counts and ratios and arguments; they care about *response time*: the duration that begins when they request something and ends when they get their answer. No matter how much complexity you build atop any timing-free event-count data, you are fundamentally doomed by the following inescapable truth, the subject of the second important advance:

> You can't tell how long something took by counting how many times it happened.

Published 2004




Chapter 1: Intro

The Old Fashion of Oracle Performance Optimization

Some say you need to know what life was like in the old days before you can really appreciate the life you now have. This is also true in the world of Oracle performance optimization. Early versions of Oracle did not offer a reliable method to identify performance bottlenecks. Performance optimization was a difficult and complicated task. Everyone used cache hit ratios as the yardstick to monitor database performance. To fully appreciate the OWI tuning methodology, you must be aware of the problems and limitations of the hit ratio–based tuning method. For many of us, this is a trip down memory lane, but for those of you who didn't grow up in the ratios-based tuning era, you may embrace this as a piece of your predecessor's history.

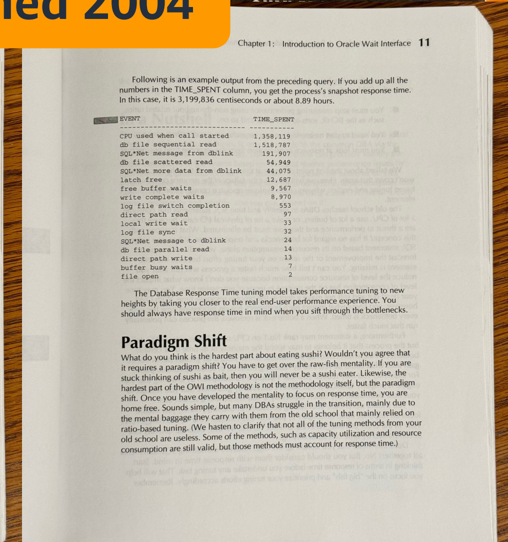
Since the beginning of Oracle RDBMS, Oracle DBAs were taught to tune the database and instance by watching a few ratio numbers. The idea was to keep all database elements operating within acceptable ranges or limits. Some of the memorable ratios are the buffer cache hit ratio, library cache hit/miss ratio (Oracle7.0), and latch get/miss ratio. Who can forget these commandments?

- Thou shalt keep thy buffer cache hit ratio in the upper 90 percentile.
- Thy data dictionary misses must be under 10 percent at all times, and thy library cache shall not covet thy data dictionary—it shall have its own ratios.
- Thy SQL area *gethitratio* and *pinhitratio* must also be in the 90 percentile at all times. Furthermore, the ratio of reloads to pins must not be more than 1 percent. If thy ratios are bad, thou shalt increase the shared pool size but thou shalt not steal the memory from the buffer cache. And while you are adding memory to the shared pool, throw some memory at the buffer cache also. It will increase the cache-hit ratio.
- Thy willing-to-wait latch hit ratios shalt be close to 1. If not, thou may increase the SPIN_COUNT but thou must be careful not to kill thy CPUs. And so on.

Lost yet? We are. Life can be much simpler, not to mention better.

Why Are Cache-Hit Ratios Grossly Inefficient?

The hit ratio philosophy is not peculiar to Oracle database administration. It is widely used and ingrained in many aspects of our daily lives. Take your local city


Chapter 1: Introduction to Oracle Wait Interface 11

Following is an example output from the preceding query. If you add up all the numbers in the TIME_SPENT column, you get the process's snapshot response time. In this case, it is 3,199,836 centiseconds or about 8.89 hours.

```
EVENT                          TIME_SPENT
------------------------------ -----------
CPU used when call started       1,358,119
db file sequential read          1,518,787
SQL*Net message from dblink        191,907
db file scattered read              54,949
SQL*Net more data from dblink       44,075
latch free                          12,687
free buffer waits                    9,567
write complete waits                 8,970
log file switch completion             553
direct path read                        97
local write wait                        33
log file sync                           32
SQL*Net message to dblink               24
db file parallel read                   14
direct path write                       13
buffer busy waits                        7
file open                                2
```

The Database Response Time tuning model takes performance tuning to new heights by taking you closer to the real end-user performance experience. You should always have response time in mind when you sift through the bottlenecks.

Paradigm Shift

What do you think is the hardest part about eating sushi? Wouldn't you agree that it requires a paradigm shift? You have to get over the raw-fish mentality. If you are stuck thinking of sushi as bait, then you will never be a sushi eater. Likewise, the hardest part of the OWI methodology is not the methodology itself, but the paradigm shift. Once you have developed the mentality to focus on response time, you are home free. Sounds simple, but many DBAs struggle in the transition, mainly due to the mental baggage they carry with them from the old school that mainly relied on ratio-based tuning. (We hasten to clarify that not all of the tuning methods from your old school are useless. Some of the methods, such as capacity utilization and resource consumption are still valid, but those methods must account for response time.)

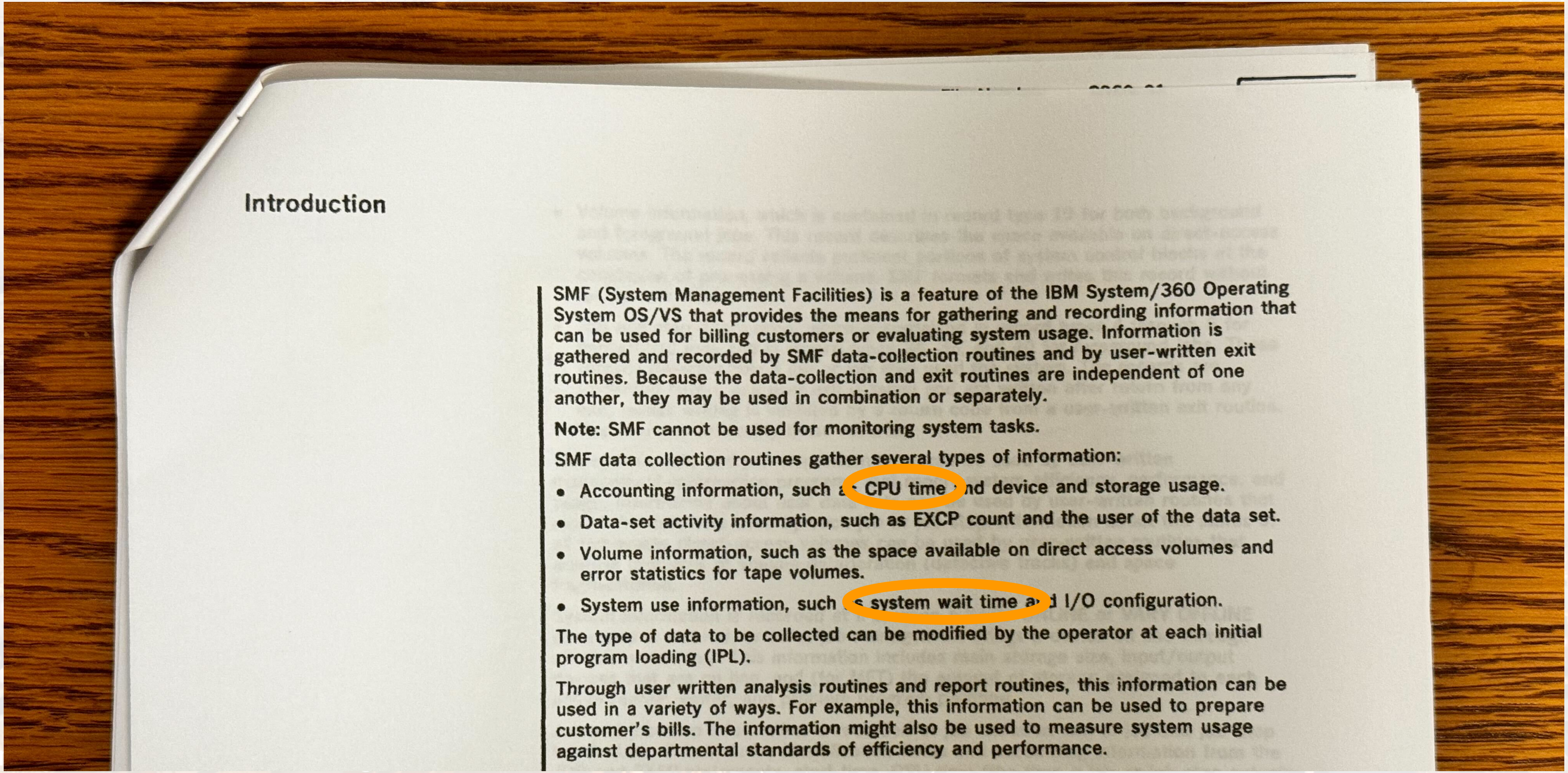
What about PostgreSQL?

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

Wait Events

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

Wait Events



Introduction

SMF (System Management Facilities) is a feature of the IBM System/360 Operating System OS/VS that provides the means for gathering and recording information that can be used for billing customers or evaluating system usage. Information is gathered and recorded by SMF data-collection routines and by user-written exit routines. Because the data-collection and exit routines are independent of one another, they may be used in combination or separately.

Note: SMF cannot be used for monitoring system tasks.

SMF data collection routines gather several types of information:

- Accounting information, such as CPU time and device and storage usage.
- Data-set activity information, such as EXCP count and the user of the data set.
- Volume information, such as the space available on direct access volumes and error statistics for tape volumes.
- System use information, such as system wait time and I/O configuration.

The type of data to be collected can be modified by the operator at each initial program loading (IPL).

Through user written analysis routines and report routines, this information can be used in a variety of ways. For example, this information can be used to prepare customer's bills. The information might also be used to measure system usage against departmental standards of efficiency and performance.

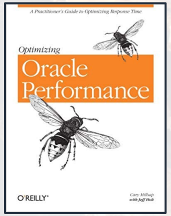
Marii

by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)

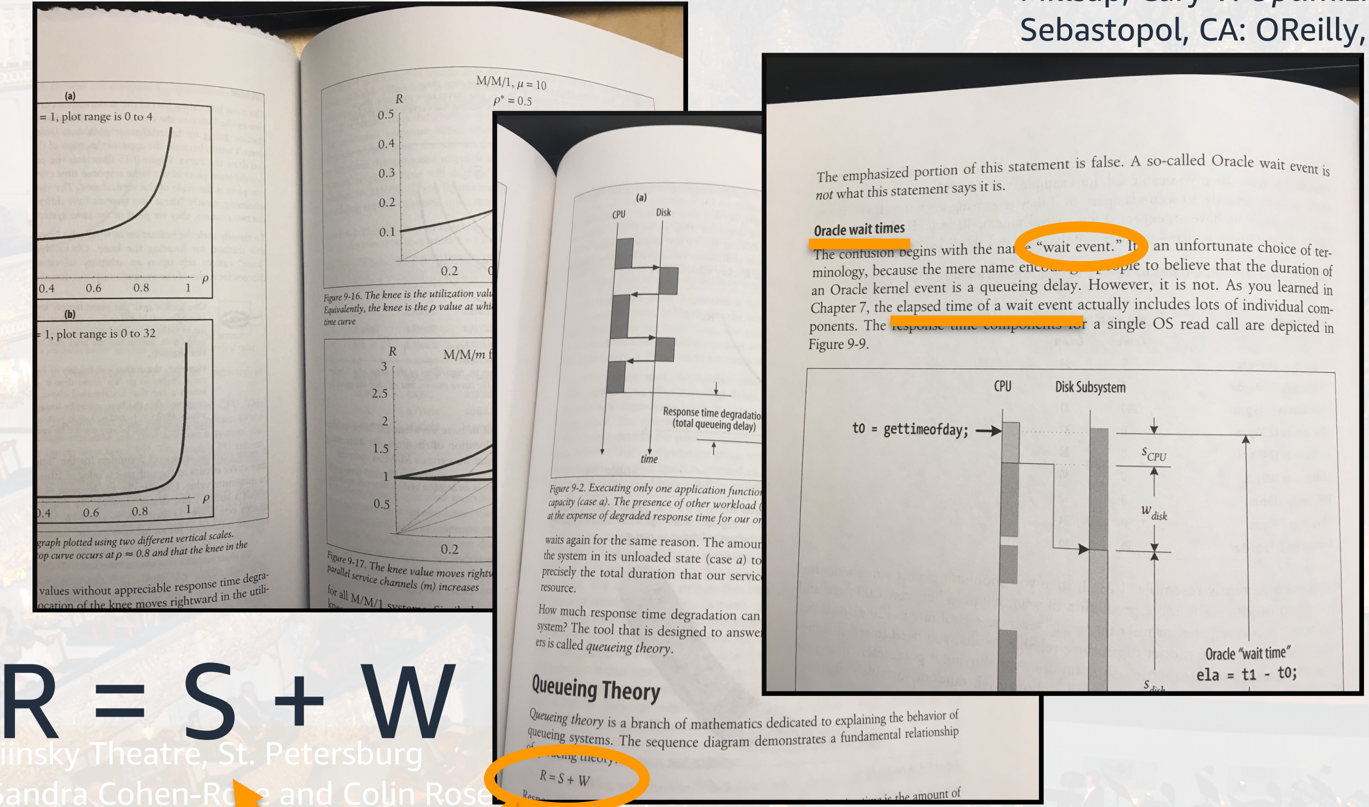
CC BY-SA

Wait Events

Millsap, Cary V. *Optimizing Oracle Performance*. Sebastopol, CA: OReilly, 2003. p225, 240, 258-259



A Practitioner's Guide to Optimizing Response Time
Optimizing
Oracle Performance
O'REILLY
Cary Millsap with Jeff Holt



(a)
= 1, plot range is 0 to 4
0.4
0.6
0.8
1
ρ
(b)
= 1, plot range is 0 to 32
M/M/1, μ = 10
ρ* = 0.5
R
0.5
0.4
0.3
0.2
0.1
0.2
M/M/m
3
2.5
2
1.5
1
0.5
CPU
Disk
Response time degradation (total queueing delay)
time
CPU
Disk Subsystem
t0 = gettimeofday;
s_{CPU}
w_{disk}
s_{disk}
Oracle "wait time"
ela = t1 - t0;

graph plotted using two different vertical scales. op curve occurs at ρ ≈ 0.8 and that the knee in the

values without appreciable response time degra- ocation of the knee moves rightward in the utili-

Figure 9-16. The knee is the utilization valu... Equivalently, the knee is the ρ value at whi... time curve

Figure 9-17. The knee value moves rightw... parallel service channels (m) increases

for all M/M/1 system...

Figure 9-2. Executing only one application function... capacity (case a). The presence of other workload... at the expense of degraded response time for our o...

waits again for the same reason. The amoun... the system in its unloaded state (case *a*) to... precisely the total duration that our servic... resource.

How much response time degradation can... system? The tool that is designed to answe... ers is called *queueing theory*.

Queueing Theory

Queueing theory is a branch of mathematics dedicated to explaining the behavior of queueing systems. The sequence diagram demonstrates a fundamental relationship of ...eing theory:

$R = S + W$

Resp... ...time is the amount of

The emphasized portion of this statement is false. A so-called Oracle wait event is *not* what this statement says it is.

Oracle wait times

The confusion begins with the name "wait event." It is an unfortunate choice of terminology, because the mere name encourages people to believe that the duration of an Oracle kernel event is a queueing delay. However, it is not. As you learned in Chapter 7, the elapsed time of a wait event actually includes lots of individual components. The response time components for a single OS read call are depicted in Figure 9-9.

“How long the SQL takes to run”

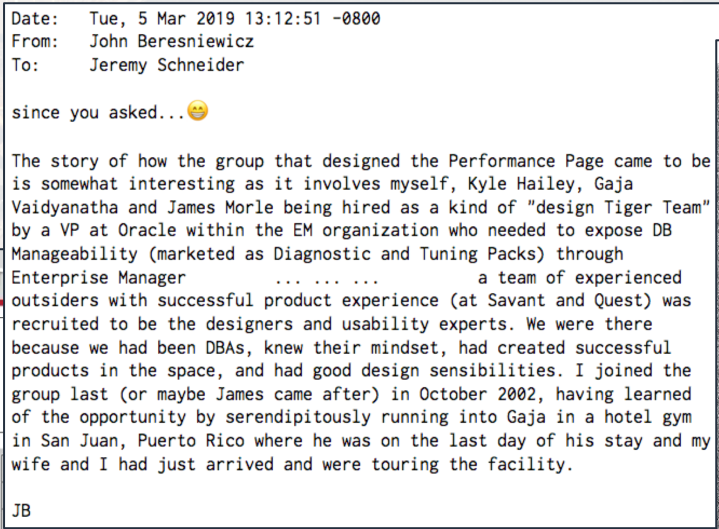
R = S + W

See also:

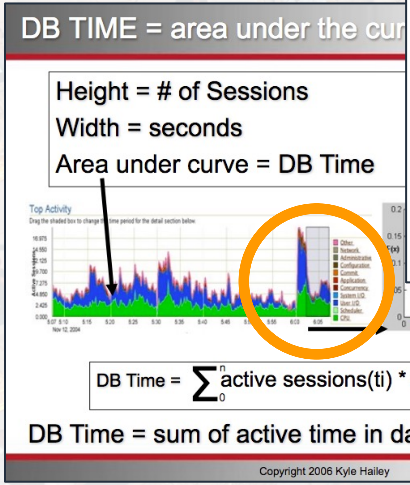
- Shallahamer, Craig. *Forecasting Oracle Performance*. Berkeley, CA: Apress, 2007.

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose
CC BY-SA

Wait Events



```
Date:   Tue, 5 Mar 2019 13:12:51 -0800
From:   John Beresniewicz
To:     Jeremy Schneider

since you asked...😁

The story of how the group that designed the Performance Page came to be
is somewhat interesting as it involves myself, Kyle Hailey, Gaja
Vaidyanatha and James Morle being hired as a kind of "design Tiger Team"
by a VP at Oracle within the EM organization who needed to expose DB
Manageability (marketed as Diagnostic and Tuning Packs) through
Enterprise Manager          ... ... ...          a team of experienced
outsiders with successful product experience (at Savant and Quest) was
recruited to be the designers and usability experts. We were there
because we had been DBAs, knew their mindset, had created successful
products in the space, and had good design sensibilities. I joined the
group last (or maybe James came after) in October 2002, having learned
of the opportunity by serendipitously running into Gaja in a hotel gym
in San Juan, Puerto Rico where he was on the last day of his stay and my
wife and I had just arrived and were touring the facility.

JB
```



DB TIME = area under the cur

Height = # of Sessions

Width = seconds

Area under curve = DB Time

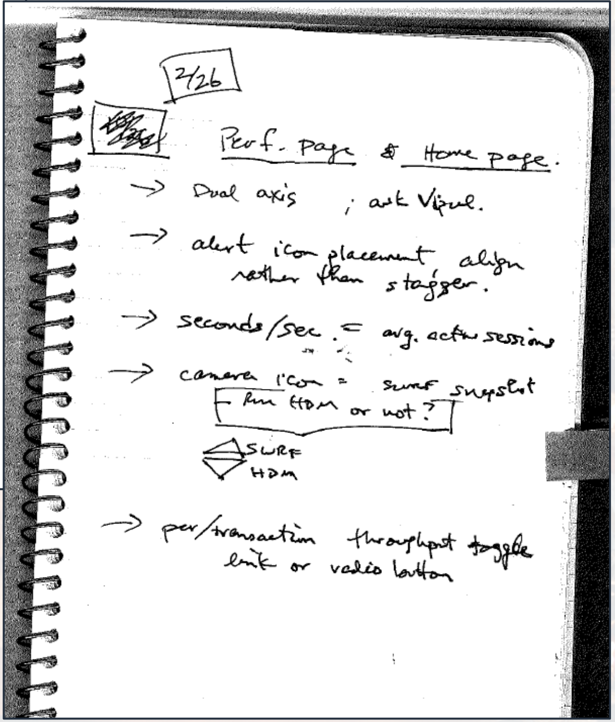
$$\text{DB Time} = \sum_{0}^{n} \text{active sessions}(ti) * \Delta t$$

DB Time = sum of active time in database

Copyright 2006 Kyle Hailey

#.20

(JB's notebook, 2004)



2/26

Perf. Page & Home page.

→ Dual axis ; ask Vipul.

→ alert icon placement align rather than stagger.

→ seconds/sec. = avg. active sessions

→ camera icon = SWRF snapshot

[Run ADDM or not?]

SWRF

ADDM

→ per/transaction throughput toggle link or radio button

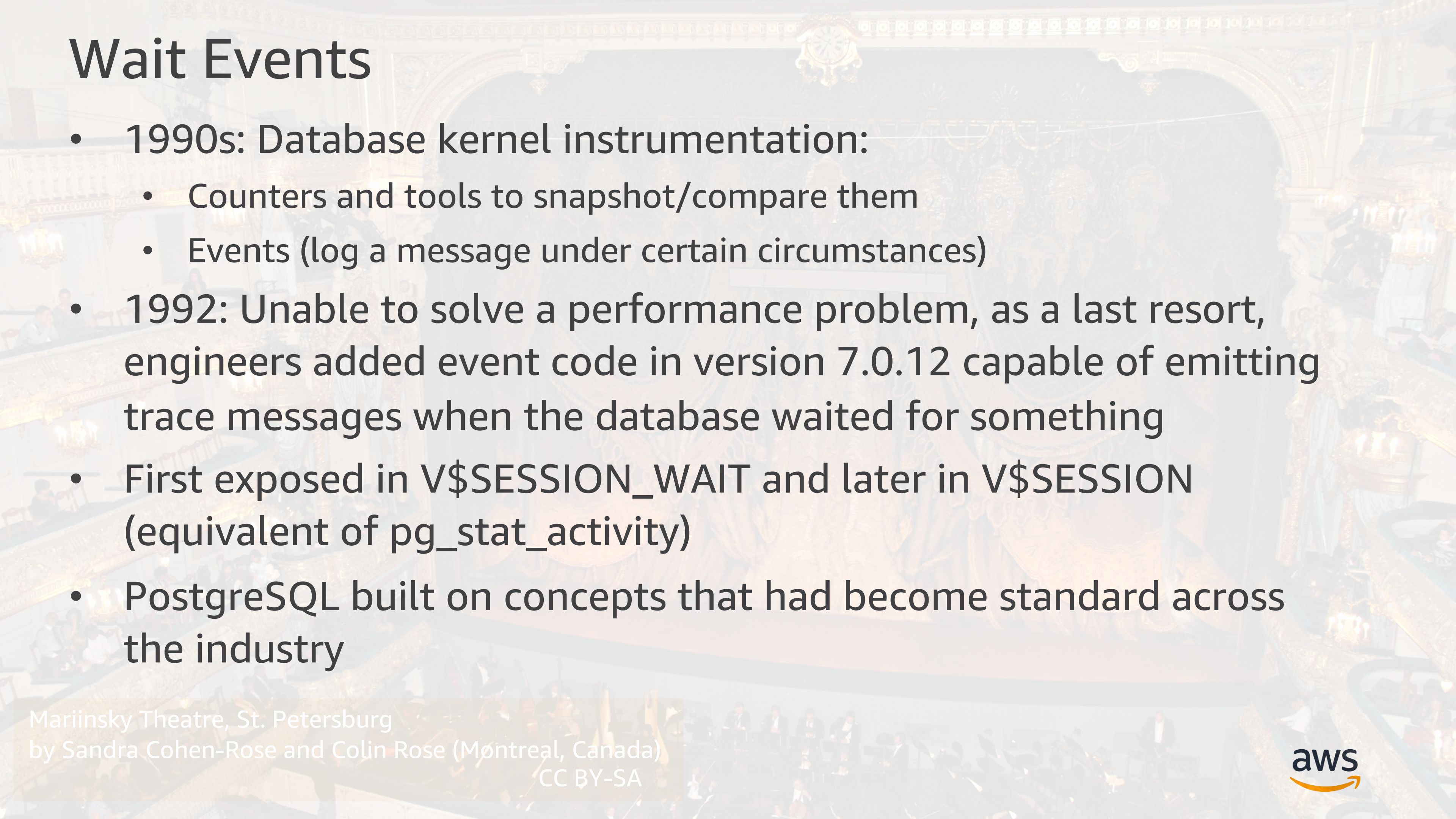
Images & Quotes
Used With Permission

Active Session Sampling

Mariinsky Theatre St Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

Wait Events

- 1990s: Database kernel instrumentation:
  - Counters and tools to snapshot/compare them
  - Events (log a message under certain circumstances)
- 1992: Unable to solve a performance problem, as a last resort, engineers added event code in version 7.0.12 capable of emitting trace messages when the database waited for something
- First exposed in V$SESSION_WAIT and later in V$SESSION (equivalent of pg_stat_activity)
- PostgreSQL built on concepts that had become standard across the industry



Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

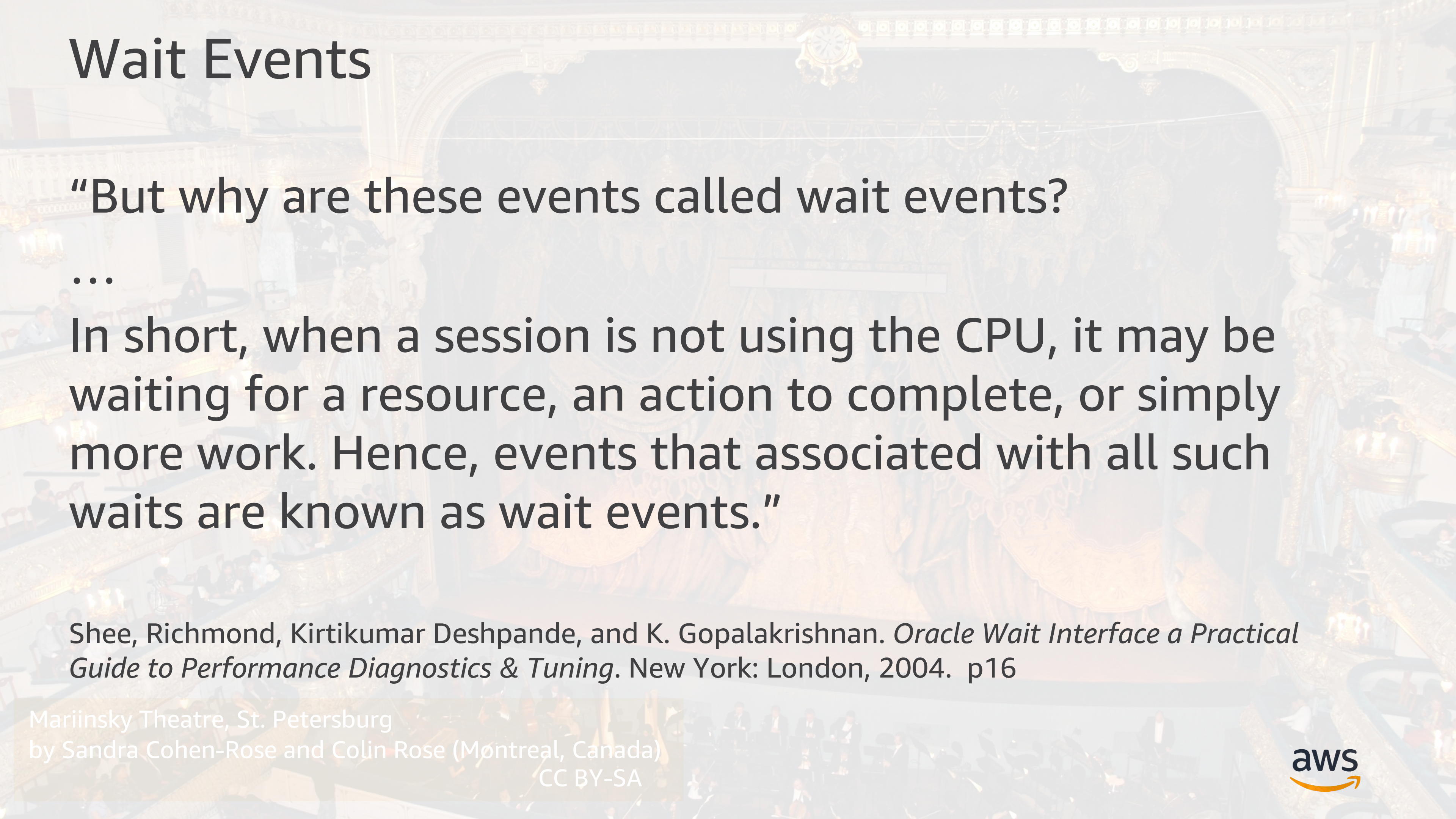
Wait Events

"But why are these events called wait events?

...

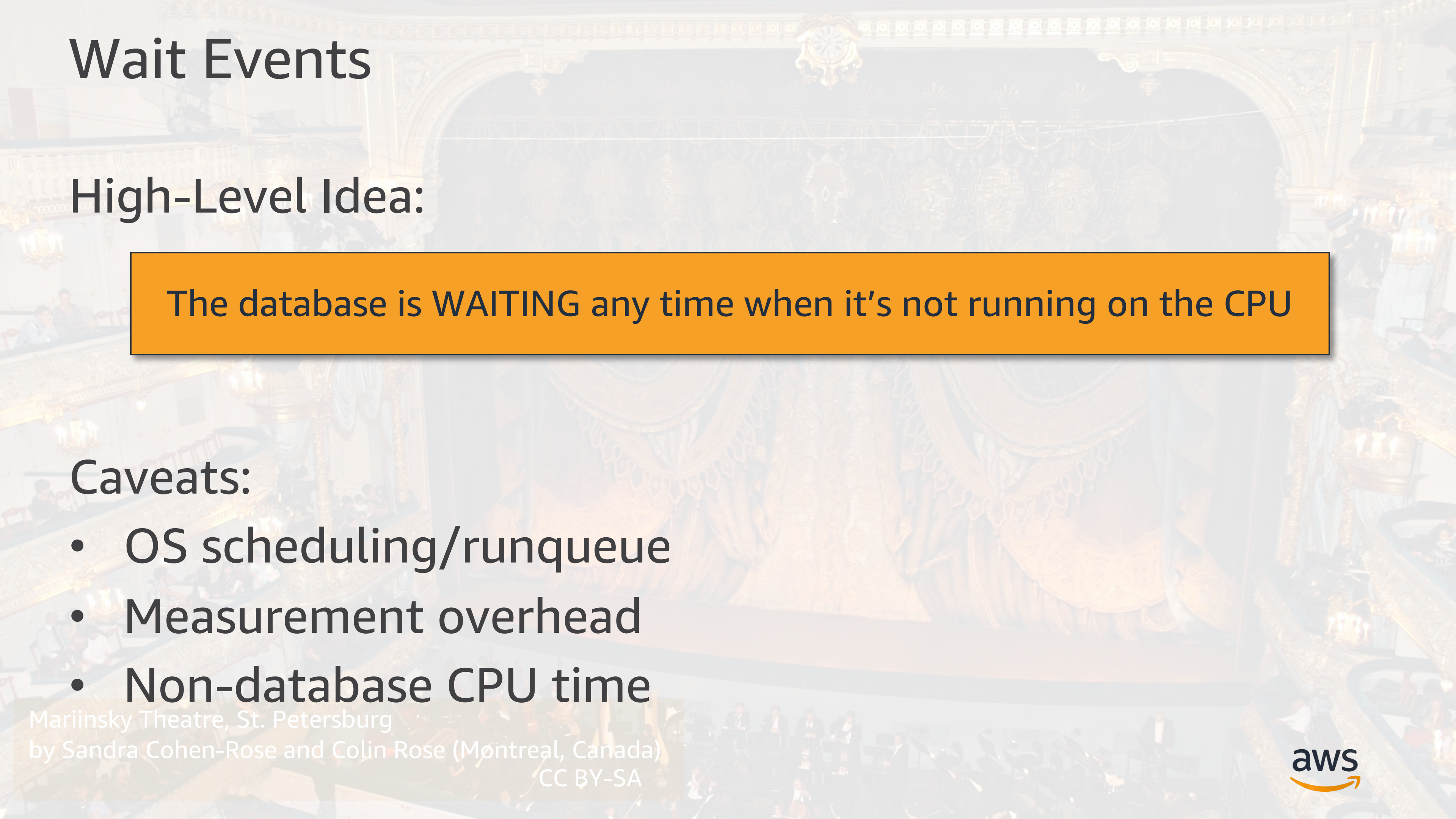
In short, when a session is not using the CPU, it may be waiting for a resource, an action to complete, or simply more work. Hence, events that associated with all such waits are known as wait events."

Shee, Richmond, Kirtikumar Deshpande, and K. Gopalakrishnan. *Oracle Wait Interface a Practical Guide to Performance Diagnostics & Tuning*. New York: London, 2004. p16



Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

Wait Events



Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

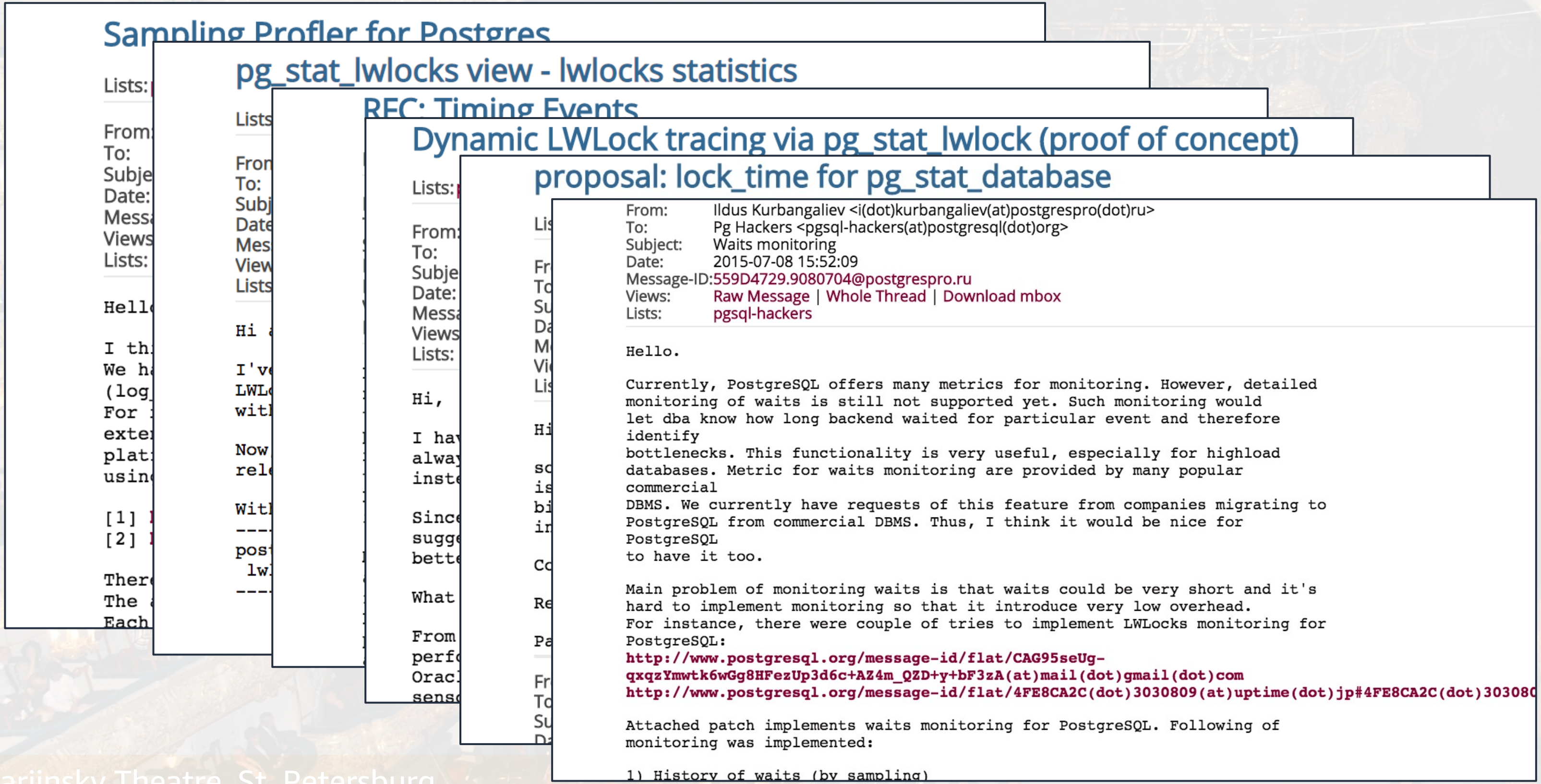
High-Level Idea:

The database is WAITING any time when it's not running on the CPU

Caveats:

- OS scheduling/runqueue
- Measurement overhead
- Non-database CPU time

Wait Events



Sampling Profler for Postgres

pg_stat_lwlocks view - lwlocks statistics

RFC: Timing Events

Dynamic LWLock tracing via pg_stat_lwlock (proof of concept)

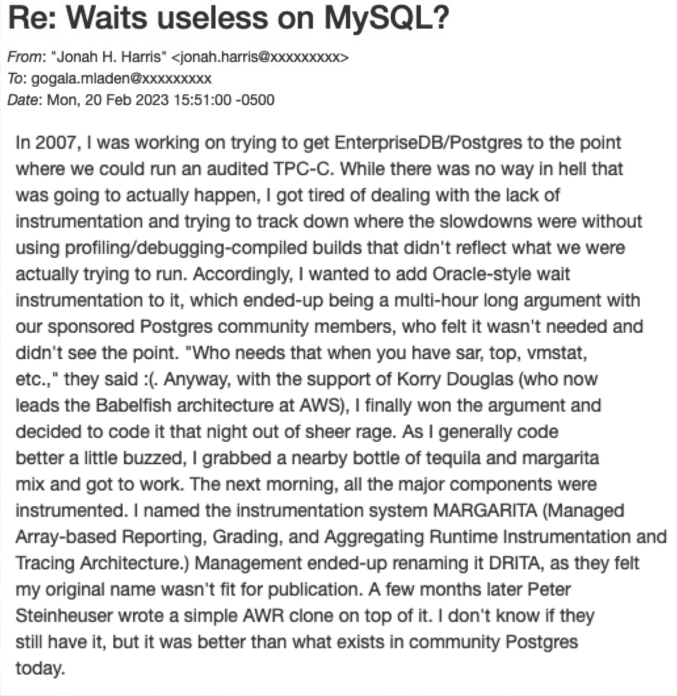
proposal: lock_time for pg_stat_database

From: Ildus Kurbangaliev <i(dot)kurbangaliev(at)postgrespro(dot)ru>
To: Pg Hackers <pgsql-hackers(at)postgresql(dot)org>
Subject: Waits monitoring
Date: 2015-07-08 15:52:09
Message-ID: 559D4729.9080704@postgrespro.ru
Views: Raw Message | Whole Thread | Download mbox
Lists: pgsql-hackers

```
Hello.

Currently, PostgreSQL offers many metrics for monitoring. However, detailed
monitoring of waits is still not supported yet. Such monitoring would
let dba know how long backend waited for particular event and therefore
identify
bottlenecks. This functionality is very useful, especially for highload
databases. Metric for waits monitoring are provided by many popular
commercial
DBMS. We currently have requests of this feature from companies migrating to
PostgreSQL from commercial DBMS. Thus, I think it would be nice for
PostgreSQL
to have it too.

Main problem of monitoring waits is that waits could be very short and it's
hard to implement monitoring so that it introduce very low overhead.
For instance, there were couple of tries to implement LWLocks monitoring for
PostgreSQL:
http://www.postgresql.org/message-id/flat/CAG95seUg-
qxqzYmwtk6wGg8HFezUp3d6c+AZ4m_QZD+y+bF3zA(at)mail(dot)gmail(dot)com
http://www.postgresql.org/message-id/flat/4FE8CA2C(dot)3030809(at)uptime(dot)jp#4FE8CA2C(dot)303080

Attached patch implements waits monitoring for PostgreSQL. Following of
monitoring was implemented:

1) History of waits (by sampling)
```

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

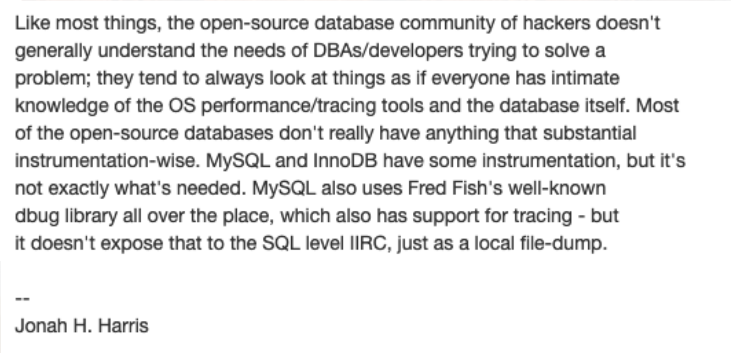
Wait Events



Re: Waits useless on MySQL?

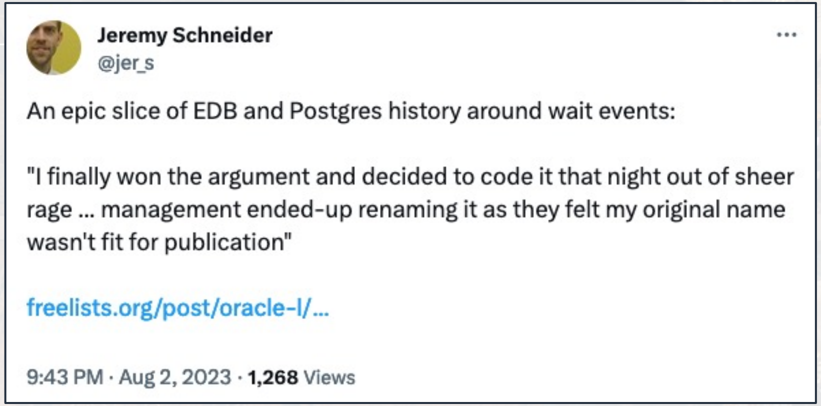
From: "Jonah H. Harris" <jonah.harris@xxxxxxxxx>
To: gogala.mladen@xxxxxxxxx
Date: Mon, 20 Feb 2023 15:51:00 -0500

In 2007, I was working on trying to get EnterpriseDB/Postgres to the point where we could run an audited TPC-C. While there was no way in hell that was going to actually happen, I got tired of dealing with the lack of instrumentation and trying to track down where the slowdowns were without using profiling/debugging-compiled builds that didn't reflect what we were actually trying to run. Accordingly, I wanted to add Oracle-style wait instrumentation to it, which ended-up being a multi-hour long argument with our sponsored Postgres community members, who felt it wasn't needed and didn't see the point. "Who needs that when you have sar, top, vmstat, etc.," they said :(. Anyway, with the support of Korry Douglas (who now leads the Babelfish architecture at AWS), I finally won the argument and decided to code it that night out of sheer rage. As I generally code better a little buzzed, I grabbed a nearby bottle of tequila and margarita mix and got to work. The next morning, all the major components were instrumented. I named the instrumentation system MARGARITA (Managed Array-based Reporting, Grading, and Aggregating Runtime Instrumentation and Tracing Architecture.) Management ended-up renaming it DRITA, as they felt my original name wasn't fit for publication. A few months later Peter Steinheuser wrote a simple AWR clone on top of it. I don't know if they still have it, but it was better than what exists in community Postgres today.



Like most things, the open-source database community of hackers doesn't generally understand the needs of DBAs/developers trying to solve a problem; they tend to always look at things as if everyone has intimate knowledge of the OS performance/tracing tools and the database itself. Most of the open-source databases don't really have anything that substantial instrumentation-wise. MySQL and InnoDB have some instrumentation, but it's not exactly what's needed. MySQL also uses Fred Fish's well-known dbug library all over the place, which also has support for tracing - but it doesn't expose that to the SQL level IIRC, just as a local file-dump.

--
Jonah H. Harris



Jeremy Schneider
@jer_s

An epic slice of EDB and Postgres history around wait events:

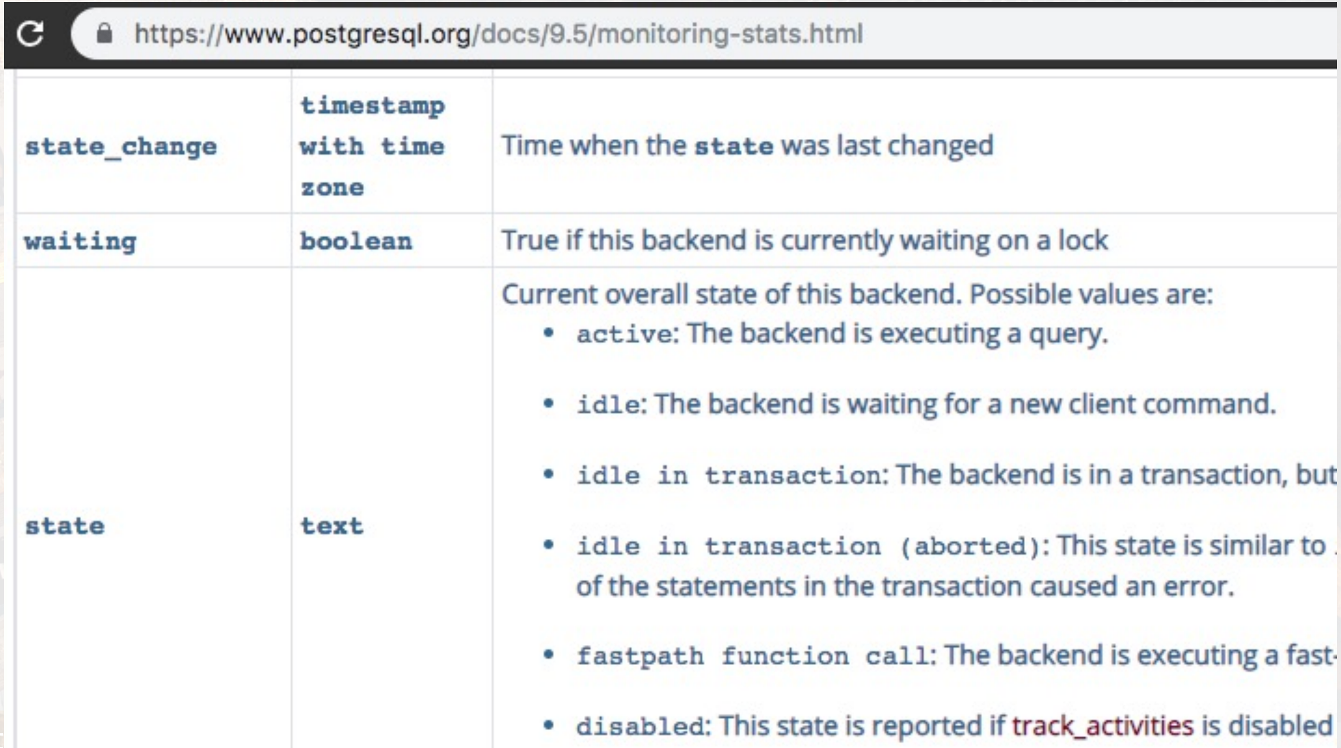
"I finally won the argument and decided to code it that night out of sheer rage ... management ended-up renaming it as they felt my original name wasn't fit for publication"

freelists.org/post/oracle-l/...

9:43 PM · Aug 2, 2023 · **1,268** Views

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

Wait Events



https://www.postgresql.org/docs/9.5/monitoring-stats.html

`state_change`	`timestamp with time zone`	Time when the `state` was last changed
`waiting`	`boolean`	True if this backend is currently waiting on a lock
`state`	`text`	Current overall state of this backend. Possible values are: - `active`: The backend is executing a query. - `idle`: The backend is waiting for a new client command. - `idle in transaction`: The backend is in a transaction, but - `idle in transaction (aborted)`: This state is similar to of the statements in the transaction caused an error. - `fastpath function call`: The backend is executing a fast- - `disabled`: This state is reported if track_activities is disabled

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

Wait Events

Significant Commits: Version 9.6

- Aa65de0 – 11 Sep 2015 – Autogenerate lwlocknames.[c|h]
- 53be0b1 – 10 Mar 2016 – Heavy/Lightweight Locks, Buffer Pins

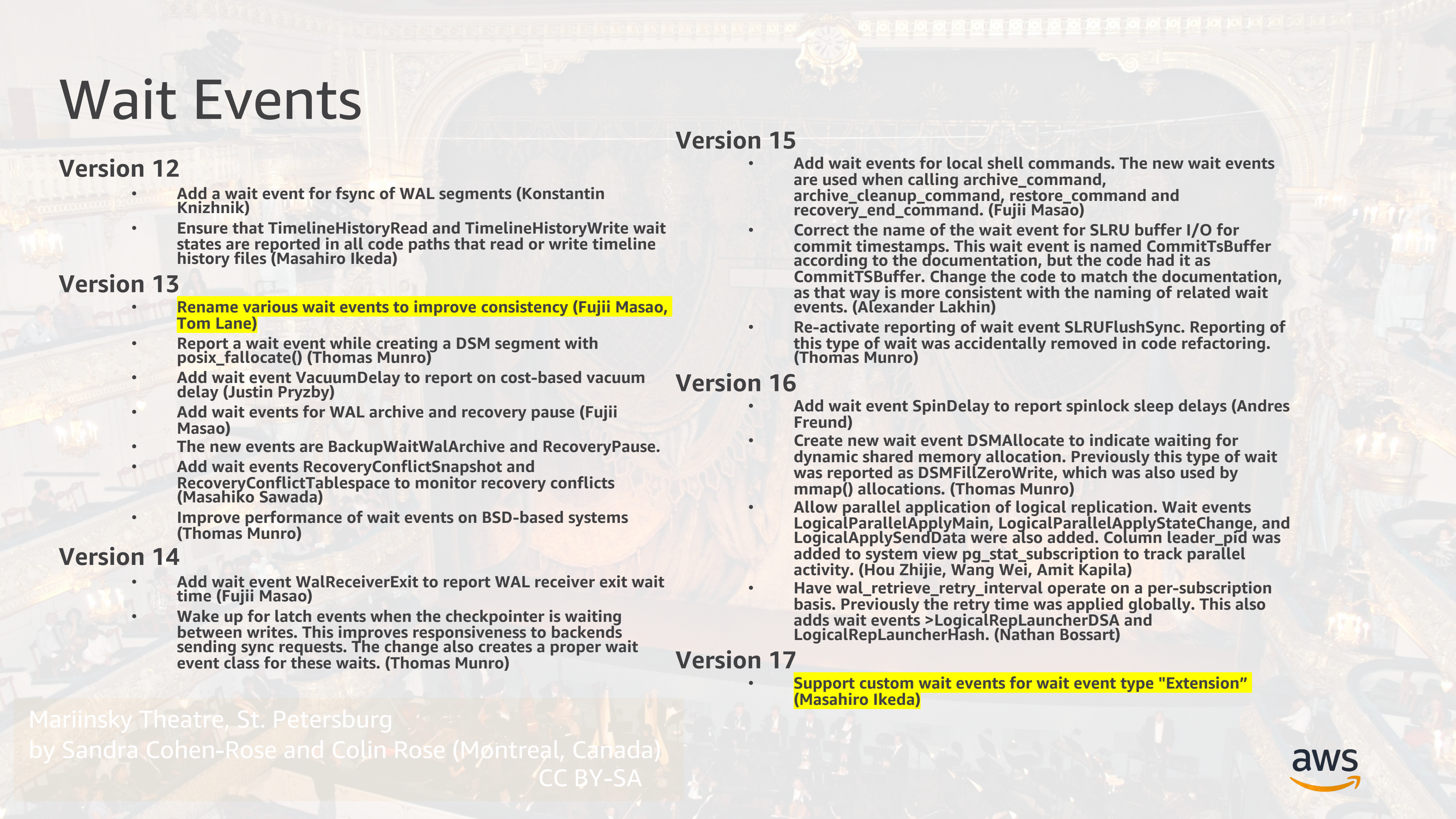
Version 10

- 6f3bd98 – 4 Oct 2016 – Latches & Sockets, Clients, Main Loops
- 249cf07 – 18 Mar 2017 – I/O
- Fc70a4b – 26 Mar 2017 – Background and Auxiliary Processes

Version 11

- 1804284 – 20 Dec 2017 – Parallel-Aware Hash Joins

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA



Wait Events

Version 12

- Add a wait event for fsync of WAL segments (Konstantin Knizhnik)
- Ensure that TimelineHistoryRead and TimelineHistoryWrite wait states are reported in all code paths that read or write timeline history files (Masahiro Ikeda)

Version 13

- Rename various wait events to improve consistency (Fujii Masao, Tom Lane)
- Report a wait event while creating a DSM segment with posix_fallocate() (Thomas Munro)
- Add wait event VacuumDelay to report on cost-based vacuum delay (Justin Pryzby)
- Add wait events for WAL archive and recovery pause (Fujii Masao)
- The new events are BackupWaitWalArchive and RecoveryPause.
- Add wait events RecoveryConflictSnapshot and RecoveryConflictTablespace to monitor recovery conflicts (Masahiko Sawada)
- Improve performance of wait events on BSD-based systems (Thomas Munro)

Version 14

- Add wait event WalReceiverExit to report WAL receiver exit wait time (Fujii Masao)
- Wake up for latch events when the checkpointer is waiting between writes. This improves responsiveness to backends sending sync requests. The change also creates a proper wait event class for these waits. (Thomas Munro)

Version 15

- Add wait events for local shell commands. The new wait events are used when calling archive_command, archive_cleanup_command, restore_command and recovery_end_command. (Fujii Masao)
- Correct the name of the wait event for SLRU buffer I/O for commit timestamps. This wait event is named CommitTsBuffer according to the documentation, but the code had it as CommitTSBuffer. Change the code to match the documentation, as that way is more consistent with the naming of related wait events. (Alexander Lakhin)
- Re-activate reporting of wait event SLRUFlushSync. Reporting of this type of wait was accidentally removed in code refactoring. (Thomas Munro)

Version 16

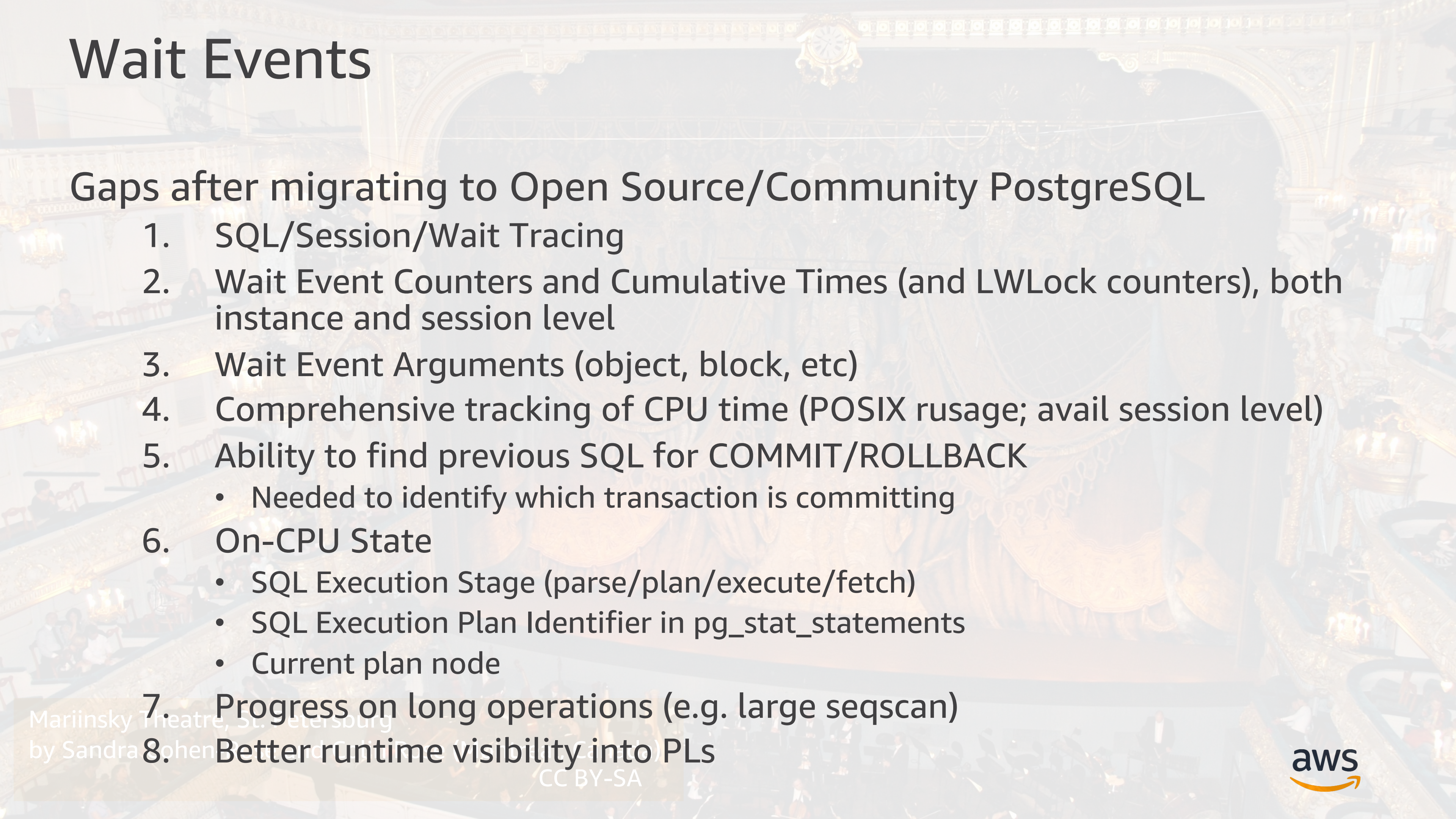
- Add wait event SpinDelay to report spinlock sleep delays (Andres Freund)
- Create new wait event DSMAllocate to indicate waiting for dynamic shared memory allocation. Previously this type of wait was reported as DSMFillZeroWrite, which was also used by mmap() allocations. (Thomas Munro)
- Allow parallel application of logical replication. Wait events LogicalParallelApplyMain, LogicalParallelApplyStateChange, and LogicalApplySendData were also added. Column leader_pid was added to system view pg_stat_subscription to track parallel activity. (Hou Zhijie, Wang Wei, Amit Kapila)
- Have wal_retrieve_retry_interval operate on a per-subscription basis. Previously the retry time was applied globally. This also adds wait events >LogicalRepLauncherDSA and LogicalRepLauncherHash. (Nathan Bossart)

Version 17

- Support custom wait events for wait event type "Extension" (Masahiro Ikeda)

Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

Wait Events



Gaps after migrating to Open Source/Community PostgreSQL

1. SQL/Session/Wait Tracing
2. Wait Event Counters and Cumulative Times (and LWLock counters), both instance and session level
3. Wait Event Arguments (object, block, etc)
4. Comprehensive tracking of CPU time (POSIX rusage; avail session level)
5. Ability to find previous SQL for COMMIT/ROLLBACK
    - Needed to identify which transaction is committing
6. On-CPU State
    - SQL Execution Stage (parse/plan/execute/fetch)
    - SQL Execution Plan Identifier in pg_stat_statements
    - Current plan node
7. Progress on long operations (e.g. large seqscan)
8. Better runtime visibility into PLs

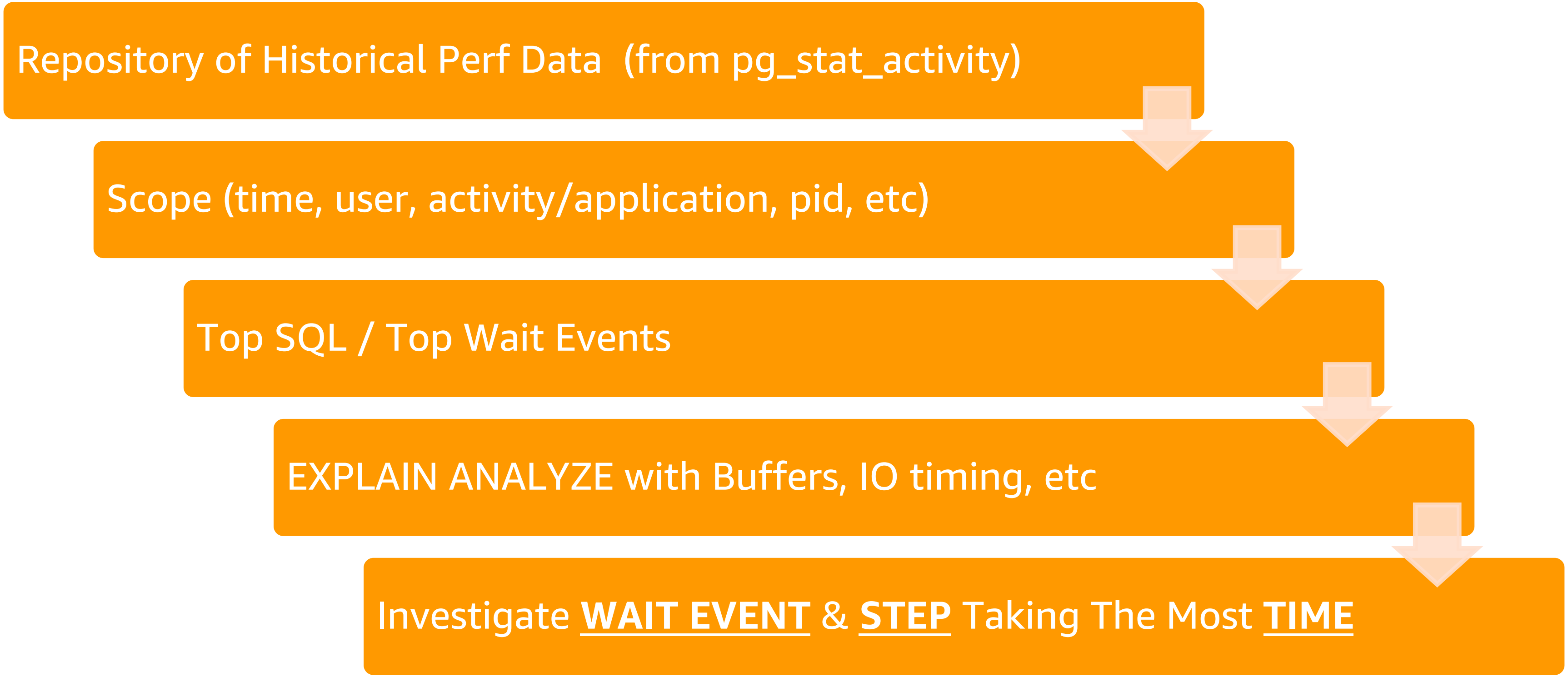
Mariinsky Theatre, St. Petersburg
by Sandra Cohen-Rose and Colin Rose (Montreal, Canada)
CC BY-SA

By Antony Griffiths (Flickr), CC BY

I can haz Wait Events?

Solving Problems with Wait Events in PostgreSQL

Solving Problems With Wait Events



Repository of Historical Perf Data (from pg_stat_activity)

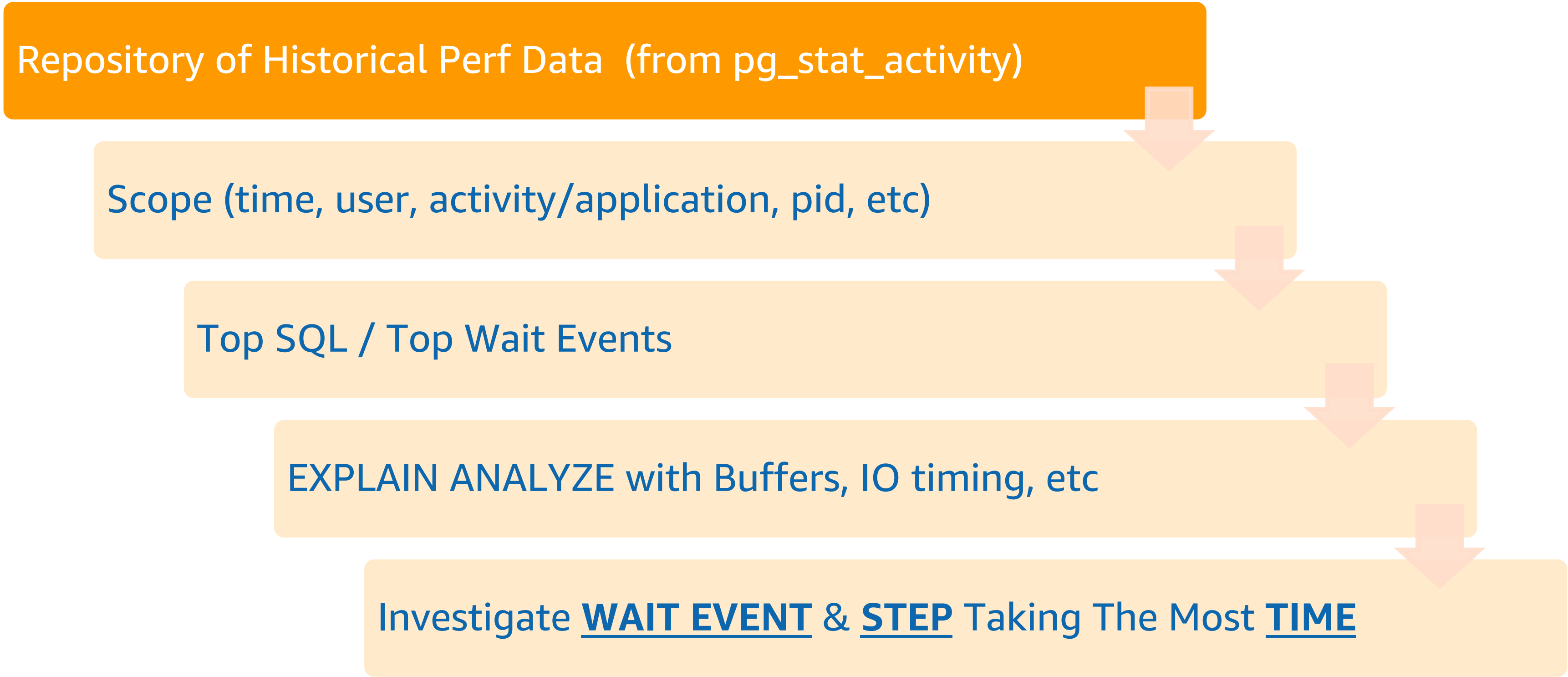
Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate **WAIT EVENT** & **STEP** Taking The Most **TIME**

Solving Problems With Wait Events



Repository of Historical Perf Data (from pg_stat_activity)

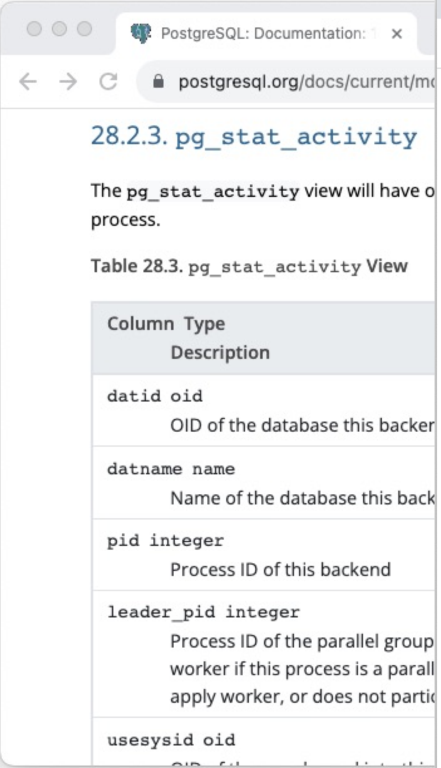
Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate **WAIT EVENT** & **STEP** Taking The Most **TIME**

Solving Problems With Wait Events



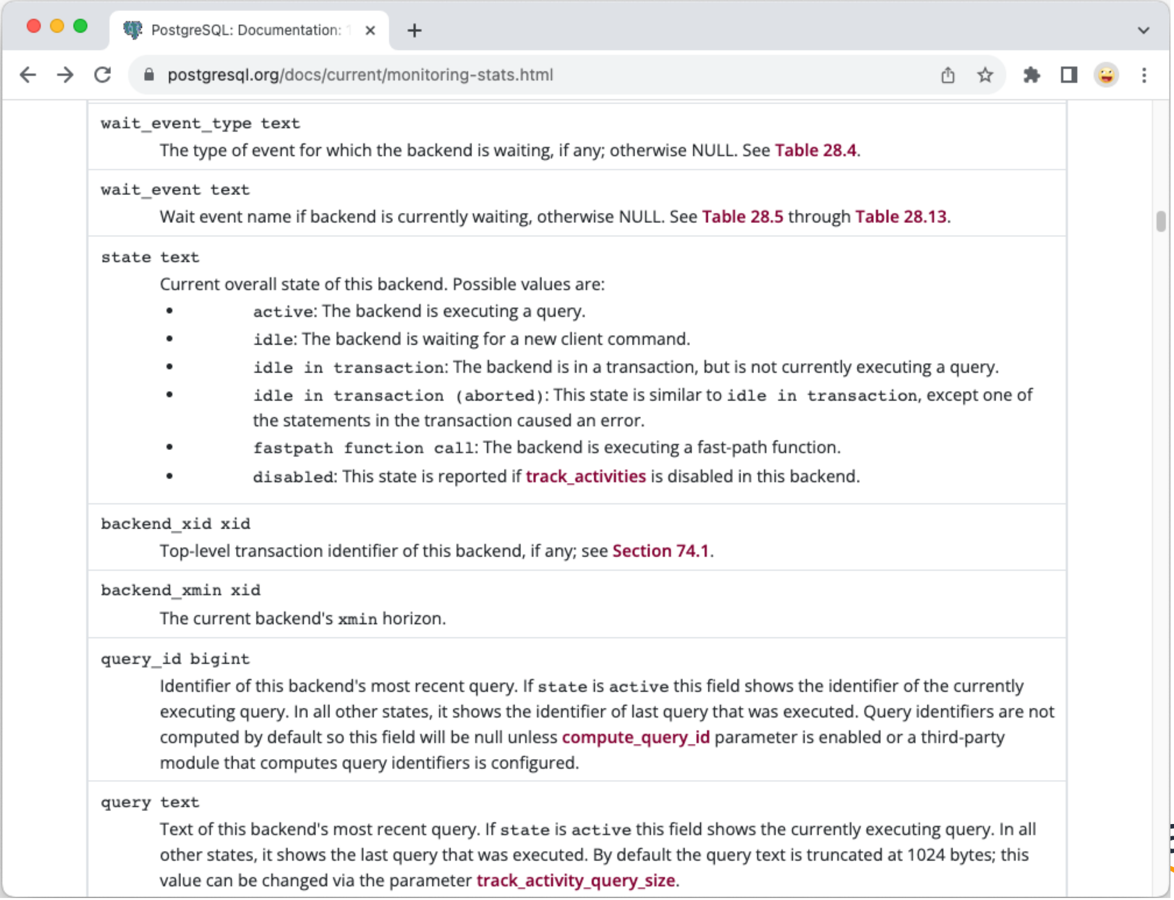
28.2.3. `pg_stat_activity`

The `pg_stat_activity` view will have o

process.

Table 28.3. `pg_stat_activity` View

Column Type Description
`datid oid` OID of the database this backer
`datname name` Name of the database this back
`pid integer` Process ID of this backend
`leader_pid integer` Process ID of the parallel group worker if this process is a parall apply worker, or does not partic
`usesysid oid`



PostgreSQL: Documentation: 1
postgresql.org/docs/current/monitoring-stats.html

`wait_event_type text`
The type of event for which the backend is waiting, if any; otherwise NULL. See Table 28.4.

`wait_event text`
Wait event name if backend is currently waiting, otherwise NULL. See Table 28.5 through Table 28.13.

`state text`
Current overall state of this backend. Possible values are:

- `active`: The backend is executing a query.
- `idle`: The backend is waiting for a new client command.
- `idle in transaction`: The backend is in a transaction, but is not currently executing a query.
- `idle in transaction (aborted)`: This state is similar to `idle in transaction`, except one of the statements in the transaction caused an error.
- `fastpath function call`: The backend is executing a fast-path function.
- `disabled`: This state is reported if track_activities is disabled in this backend.

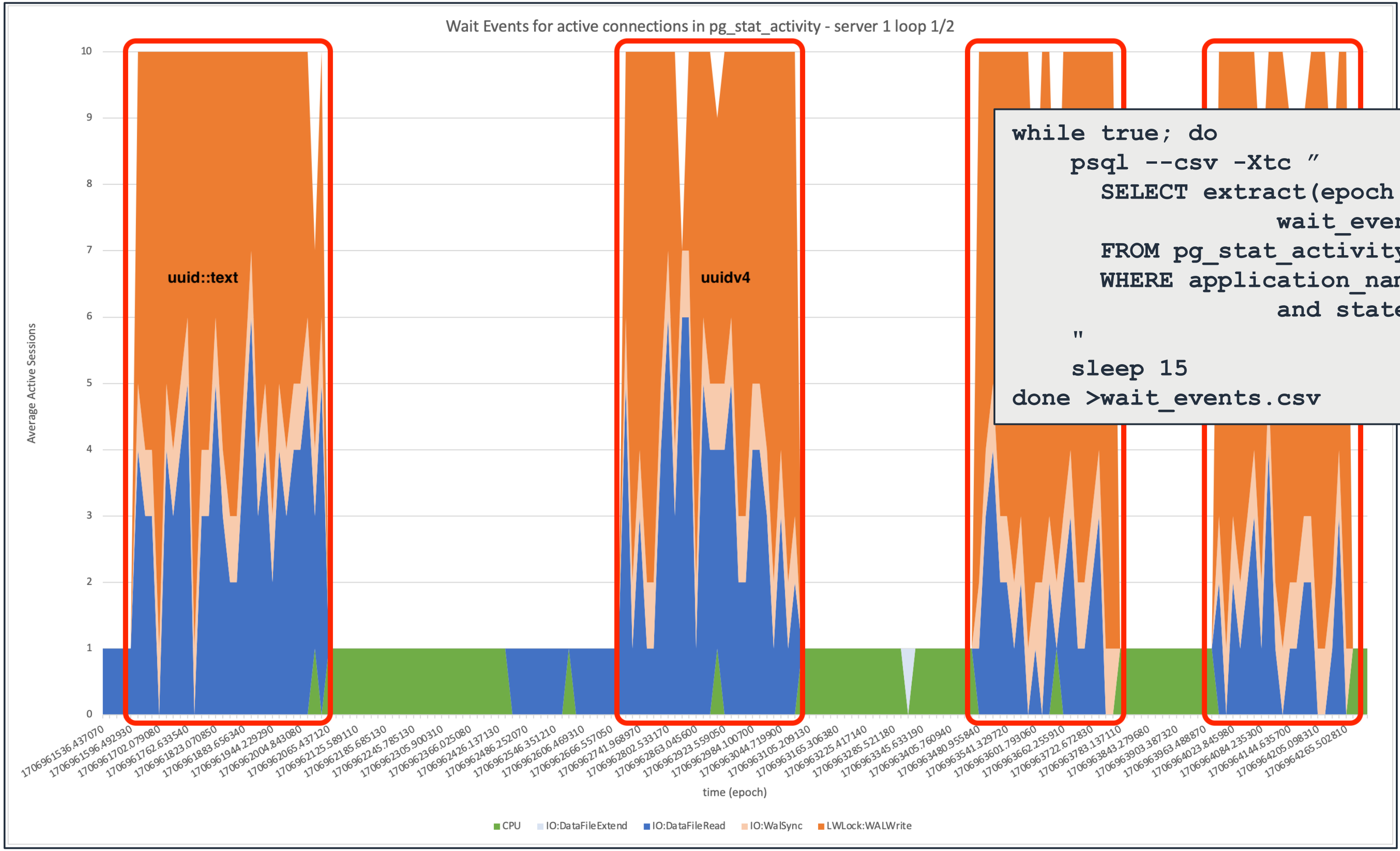
`backend_xid xid`
Top-level transaction identifier of this backend, if any; see Section 74.1.

`backend_xmin xid`
The current backend's `xmin` horizon.

`query_id bigint`
Identifier of this backend's most recent query. If `state` is `active` this field shows the identifier of the currently executing query. In all other states, it shows the identifier of last query that was executed. Query identifiers are not computed by default so this field will be null unless compute_query_id parameter is enabled or a third-party module that computes query identifiers is configured.

`query text`
Text of this backend's most recent query. If `state` is `active` this field shows the currently executing query. In all other states, it shows the last query that was executed. By default the query text is truncated at 1024 bytes; this value can be changed via the parameter track_activity_query_size.

Solving Problems With Wait Events



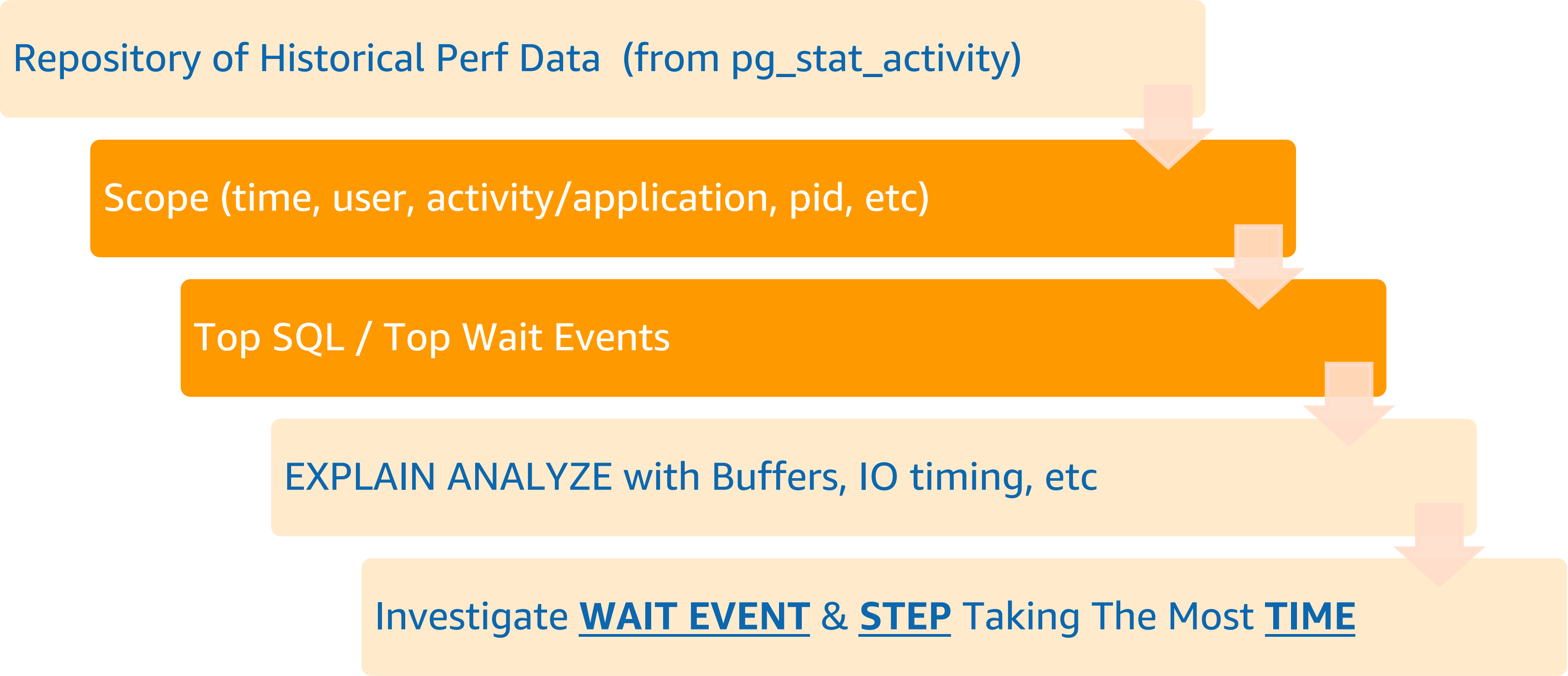
```
while true; do
    psql --csv -Xtc ”
      SELECT extract(epoch from now()), query,
                  wait_event_type, wait_event
      FROM pg_stat_activity
      WHERE application_name='pgbench'
                  and state='active’;
    "
    sleep 15
done >wait_events.csv
```

Solving Problems With Wait Events

Repositories of Historical Performance Data
(Active Session Sampling of Wait Events)

- https://wiki.postgresql.org/wiki/Monitoring
- Amazon RDS Performance Insights
  - RDS for PostgreSQL 10+
  - Aurora PostgreSQL-Compatible Edition 9.6+
    (v10 Wait Events were backported)
  - Rolling 7 days of history is free. Up to 2 years on paid tier.

Solving Problems With Wait Events



Repository of Historical Perf Data (from pg_stat_activity)

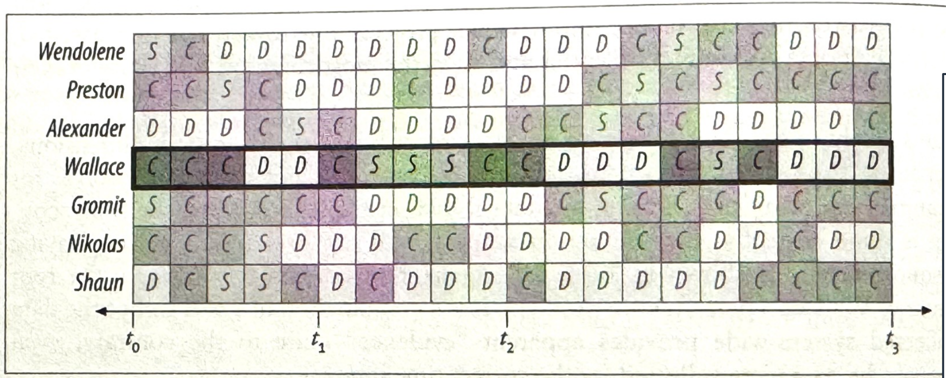
Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate **WAIT EVENT** & **STEP** Taking The Most **TIME**

Solving Problems With Wait Events

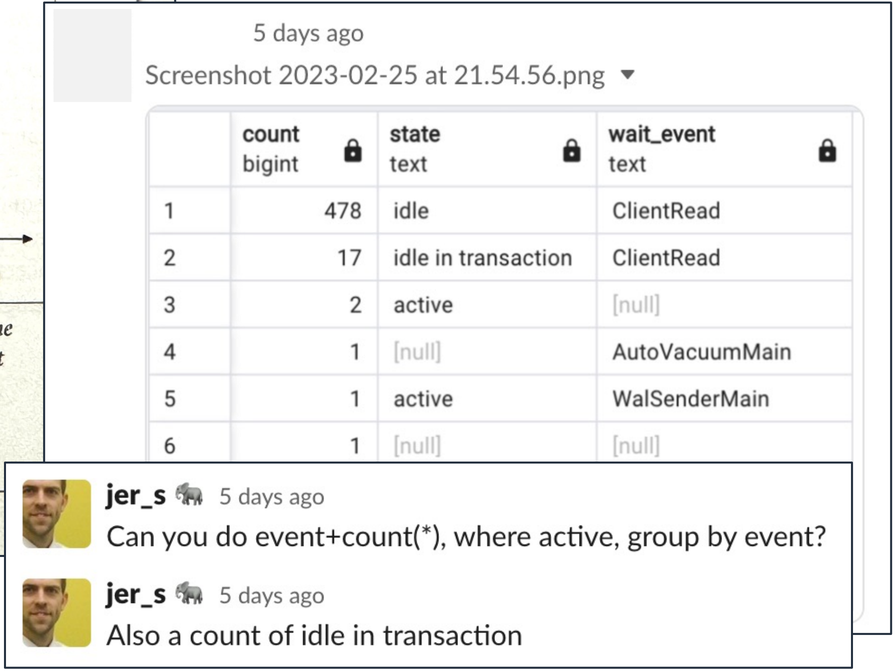


Wendolene	S	C	D	D	D	D	D	D	D	C	D	D	D	C	S	C	C	D	D	D
Preston	C	C	S	C	D	D	D	C	D	D	D	D	C	S	C	S	C	C	C	C
Alexander	D	D	D	C	S	C	D	D	D	D	C	C	S	C	C	D	D	D	D	C
Wallace	C	C	C	D	D	C	S	S	S	C	C	D	D	D	C	S	C	D	D	D
Gromit	S	C	C	C	C	C	D	D	D	D	D	C	S	C	C	C	D	C	C	C
Nikolas	C	C	C	S	D	D	D	C	C	D	D	D	D	C	C	D	D	C	D	D
Shaun	D	C	S	S	C	D	C	D	D	D	C	D	D	D	D	D	C	C	C	C

Figure 3-7. Collecting data that are scoped improperly on the time dimension also conceals the nature of Wallace's performance problem, even though the data were collected for the correct action scope

52 | **Chapter 3: Targeting the Right Diagnostic Data**

Millsap, Cary V. *Optimizing Oracle Performance.*
Sebastopol, CA: OReilly, 2003. p52



5 days ago

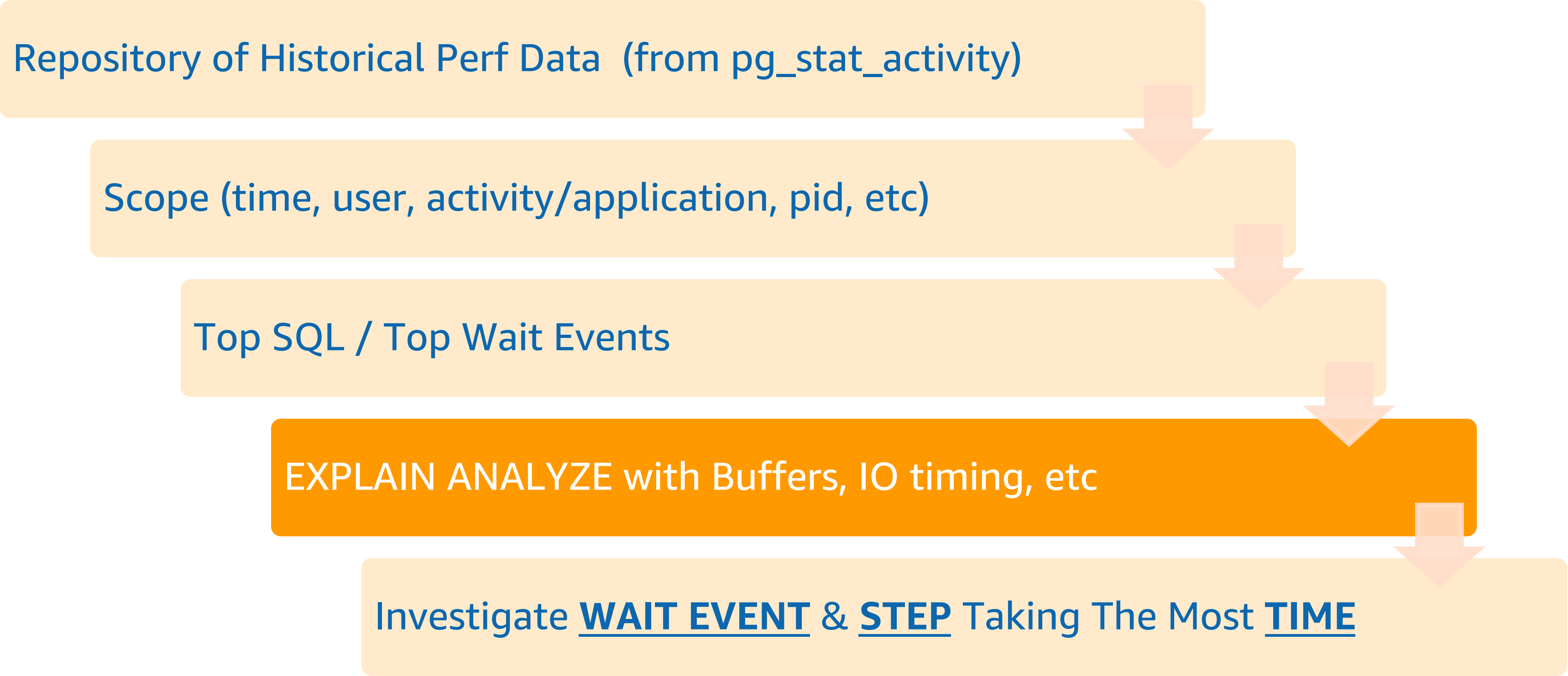
Screenshot 2023-02-25 at 21.54.56.png

	count bigint	state text	wait_event text
1	478	idle	ClientRead
2	17	idle in transaction	ClientRead
3	2	active	[null]
4	1	[null]	AutoVacuumMain
5	1	active	WalSenderMain
6	1	[null]	[null]

jer_s 🐘 5 days ago
Can you do event+count(*), where active, group by event?

jer_s 🐘 5 days ago
Also a count of idle in transaction

Solving Problems With Wait Events



Repository of Historical Perf Data (from pg_stat_activity)

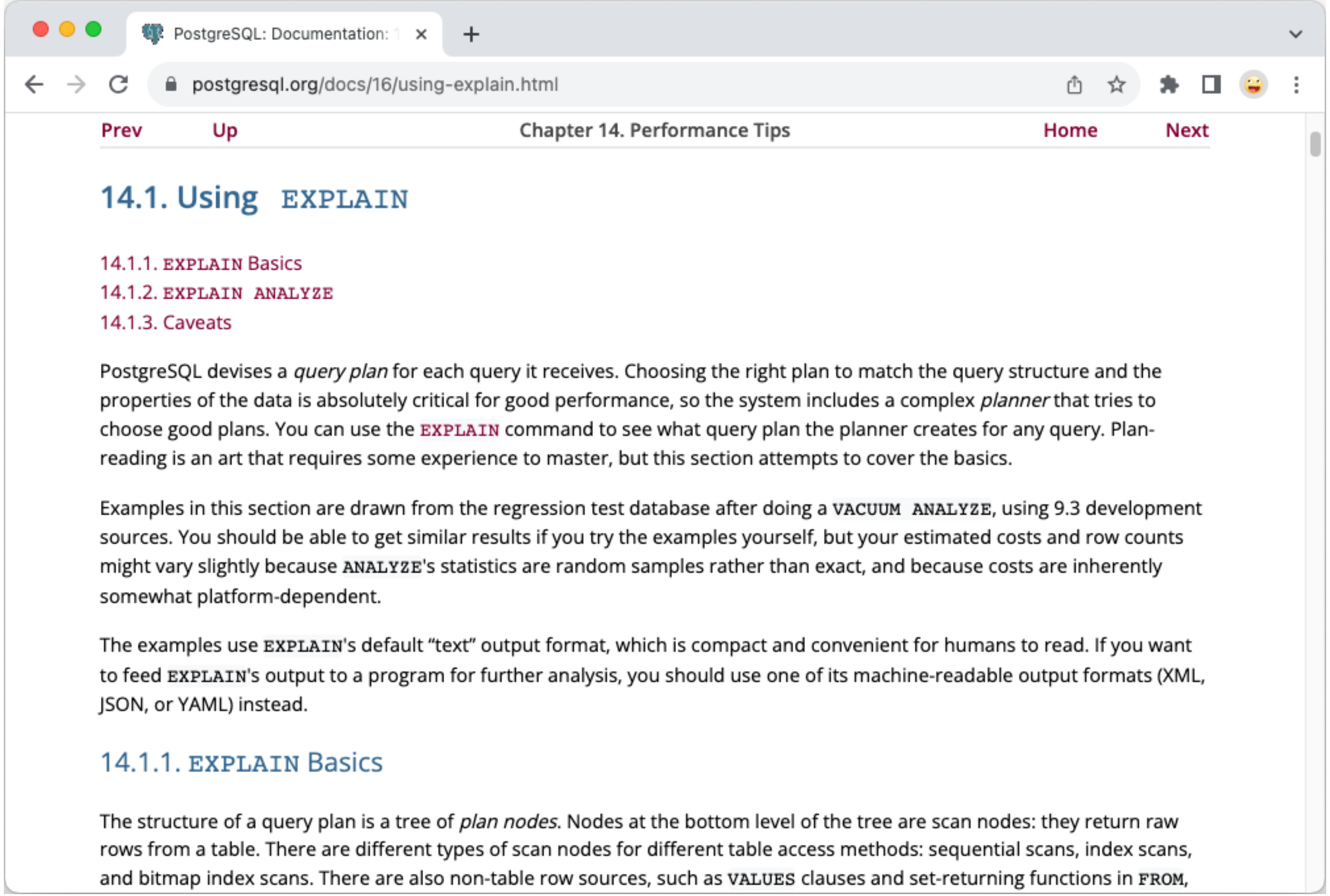
Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate **WAIT EVENT** & **STEP** Taking The Most **TIME**

Solving Problems With Wait Events



PostgreSQL: Documentation: 1
postgresql.org/docs/16/using-explain.html

14.1. Using EXPLAIN

PostgreSQL devises a *query plan* for each query it receives. Choosing the right plan to match the query structure and the properties of the data is absolutely critical for good performance, so the system includes a complex *planner* that tries to choose good plans. You can use the `EXPLAIN` command to see what query plan the planner creates for any query. Plan-reading is an art that requires some experience to master, but this section attempts to cover the basics.

Examples in this section are drawn from the regression test database after doing a `VACUUM ANALYZE`, using 9.3 development sources. You should be able to get similar results if you try the examples yourself, but your estimated costs and row counts might vary slightly because `ANALYZE`'s statistics are random samples rather than exact, and because costs are inherently somewhat platform-dependent.

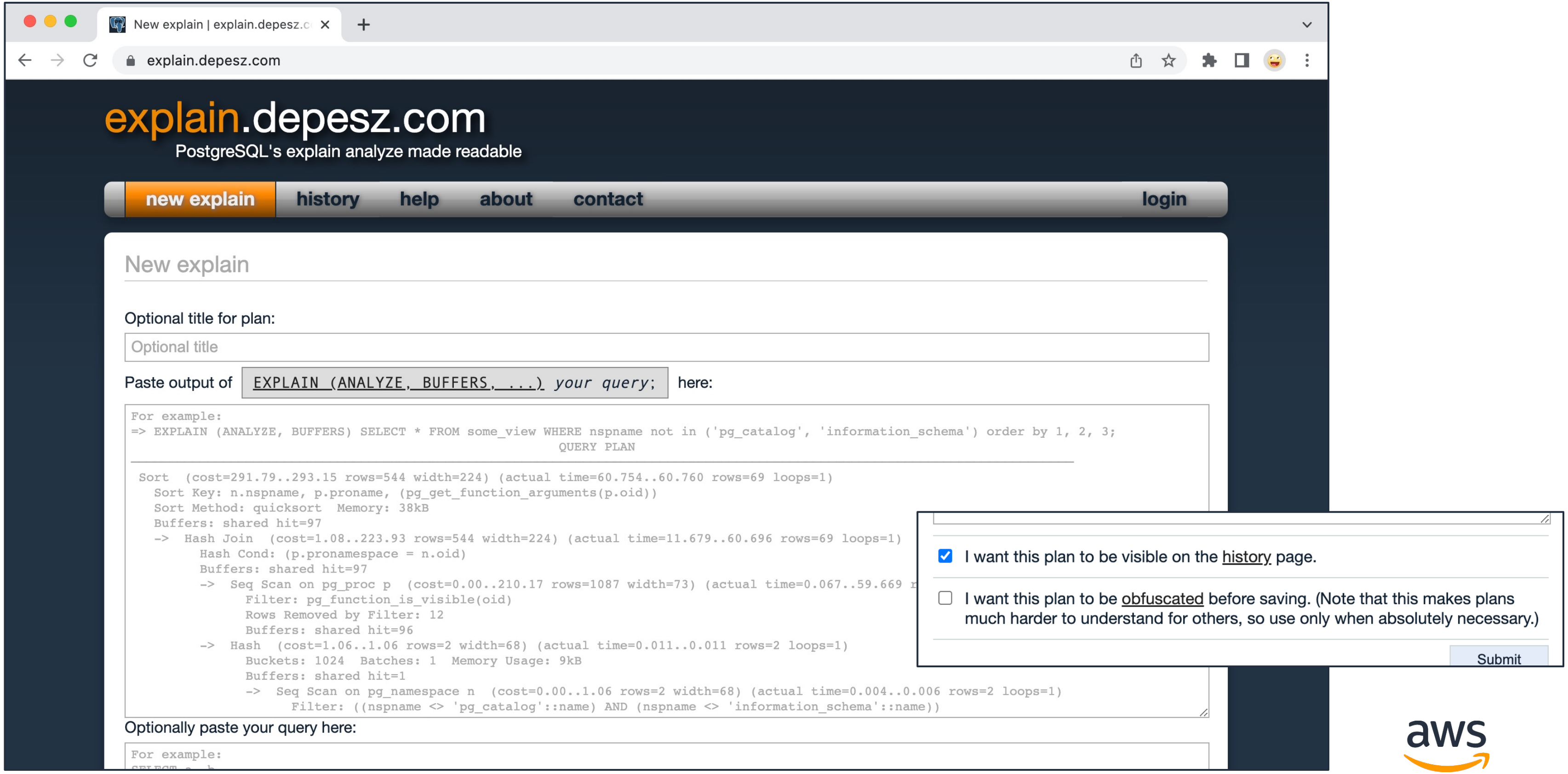
The examples use `EXPLAIN`'s default “text” output format, which is compact and convenient for humans to read. If you want to feed `EXPLAIN`'s output to a program for further analysis, you should use one of its machine-readable output formats (XML, JSON, or YAML) instead.

14.1.1. EXPLAIN Basics

The structure of a query plan is a tree of *plan nodes*. Nodes at the bottom level of the tree are scan nodes: they return raw rows from a table. There are different types of scan nodes for different table access methods: sequential scans, index scans, and bitmap index scans. There are also non-table row sources, such as `VALUES` clauses and set-returning functions in `FROM`,

Solving Problems With Wait Events

(one of many options)



explain.depesz.com

PostgreSQL's explain analyze made readable

New explain

Optional title for plan:

Optional title

Paste output of `EXPLAIN (ANALYZE, BUFFERS, ...) your query;` here:

```
For example:
=> EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM some_view WHERE nspname not in ('pg_catalog', 'information_schema') order by 1, 2, 3;
                                                             QUERY PLAN
─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
 Sort  (cost=291.79..293.15 rows=544 width=224) (actual time=60.754..60.760 rows=69 loops=1)
   Sort Key: n.nspname, p.proname, (pg_get_function_arguments(p.oid))
   Sort Method: quicksort  Memory: 38kB
   Buffers: shared hit=97
   ->  Hash Join  (cost=1.08..223.93 rows=544 width=224) (actual time=11.679..60.696 rows=69 loops=1)
         Hash Cond: (p.pronamespace = n.oid)
         Buffers: shared hit=97
         ->  Seq Scan on pg_proc p  (cost=0.00..210.17 rows=1087 width=73) (actual time=0.067..59.669 r
               Filter: pg_function_is_visible(oid)
               Rows Removed by Filter: 12
               Buffers: shared hit=96
         ->  Hash  (cost=1.06..1.06 rows=2 width=68) (actual time=0.011..0.011 rows=2 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               Buffers: shared hit=1
               ->  Seq Scan on pg_namespace n  (cost=0.00..1.06 rows=2 width=68) (actual time=0.004..0.006 rows=2 loops=1)
                     Filter: ((nspname <> 'pg_catalog'::name) AND (nspname <> 'information_schema'::name))
```

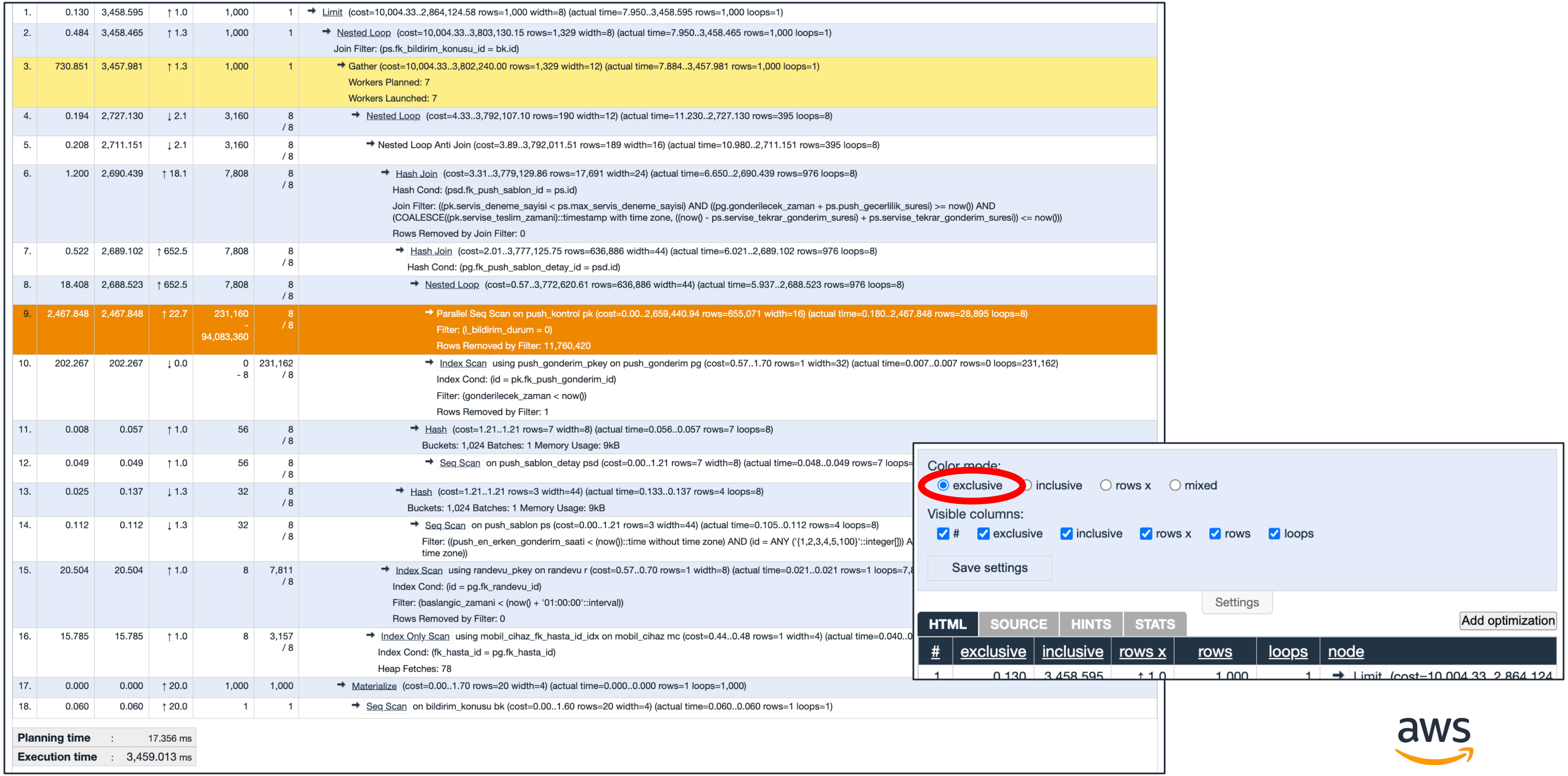
Optionally paste your query here:

```
For example:
```

☑ I want this plan to be visible on the history page.

☐ I want this plan to be obfuscated before saving. (Note that this makes plans much harder to understand for others, so use only when absolutely necessary.)

Solving Problems With Wait Events



1.	0.130	3,458.595	↑ 1.0	1,000	1	→ Limit (cost=10,004.33..2,864,124.58 rows=1,000 width=8) (actual time=7.950..3,458.595 rows=1,000 loops=1)
2.	0.484	3,458.465	↑ 1.3	1,000	1	→ Nested Loop (cost=10,004.33..3,803,130.15 rows=1,329 width=8) (actual time=7.950..3,458.465 rows=1,000 loops=1) Join Filter: (ps.fk_bildirim_konusu_id = bk.id)
3.	730.851	3,457.981	↑ 1.3	1,000	1	→ Gather (cost=10,004.33..3,802,240.00 rows=1,329 width=12) (actual time=7.884..3,457.981 rows=1,000 loops=1) Workers Planned: 7 Workers Launched: 7
4.	0.194	2,727.130	↓ 2.1	3,160	8 / 8	→ Nested Loop (cost=4.33..3,792,107.10 rows=190 width=12) (actual time=11.230..2,727.130 rows=395 loops=8)
5.	0.208	2,711.151	↓ 2.1	3,160	8 / 8	→ Nested Loop Anti Join (cost=3.89..3,792,011.51 rows=189 width=16) (actual time=10.980..2,711.151 rows=395 loops=8)
6.	1.200	2,690.439	↑ 18.1	7,808	8 / 8	→ Hash Join (cost=3.31..3,779,129.86 rows=17,691 width=24) (actual time=6.650..2,690.439 rows=976 loops=8) Hash Cond: (psd.fk_push_sablon_id = ps.id) Join Filter: ((pk.servis_deneme_sayisi < ps.max_servis_deneme_sayisi) AND ((pg.gonderilecek_zaman + ps.push_gecerlilik_suresi) >= now()) AND (COALESCE((pk.servise_teslim_zamani)::timestamp with time zone, ((now() - ps.servise_tekrar_gonderim_suresi) + ps.servise_tekrar_gonderim_suresi)) <= now())) Rows Removed by Join Filter: 0
7.	0.522	2,689.102	↑ 652.5	7,808	8 / 8	→ Hash Join (cost=2.01..3,777,125.75 rows=636,886 width=44) (actual time=6.021..2,689.102 rows=976 loops=8) Hash Cond: (pg.fk_push_sablon_detay_id = psd.id)
8.	18.408	2,688.523	↑ 652.5	7,808	8 / 8	→ Nested Loop (cost=0.57..3,772,620.61 rows=636,886 width=44) (actual time=5.937..2,688.523 rows=976 loops=8)
9.	2,467.848	2,467.848	↑ 22.7	231,160 - 94,083,360	8 / 8	→ Parallel Seq Scan on push_kontrol pk (cost=0.00..2,659,440.94 rows=655,071 width=16) (actual time=0.180..2,467.848 rows=28,895 loops=8) Filter: (l_bildirim_durum = 0) Rows Removed by Filter: 11,760,420
10.	202.267	202.267	↓ 0.0	0 - 8	231,162 / 8	→ Index Scan using push_gonderim_pkey on push_gonderim pg (cost=0.57..1.70 rows=1 width=32) (actual time=0.007..0.007 rows=0 loops=231,162) Index Cond: (id = pk.fk_push_gonderim_id) Filter: (gonderilecek_zaman < now()) Rows Removed by Filter: 1
11.	0.008	0.057	↑ 1.0	56	8 / 8	→ Hash (cost=1.21..1.21 rows=7 width=8) (actual time=0.056..0.057 rows=7 loops=8) Buckets: 1,024 Batches: 1 Memory Usage: 9kB
12.	0.049	0.049	↑ 1.0	56	8 / 8	→ Seq Scan on push_sablon_detay psd (cost=0.00..1.21 rows=7 width=8) (actual time=0.048..0.049 rows=7 loops=
13.	0.025	0.137	↓ 1.3	32	8 / 8	→ Hash (cost=1.21..1.21 rows=3 width=44) (actual time=0.133..0.137 rows=4 loops=8) Buckets: 1,024 Batches: 1 Memory Usage: 9kB
14.	0.112	0.112	↓ 1.3	32	8 / 8	→ Seq Scan on push_sablon ps (cost=0.00..1.21 rows=3 width=44) (actual time=0.105..0.112 rows=4 loops=8) Filter: ((push_en_erken_gonderim_saati < (now())::time without time zone) AND (id = ANY ('{1,2,3,4,5,100}'::integer[])) A time zone))
15.	20.504	20.504	↑ 1.0	8	7,811 / 8	→ Index Scan using randevu_pkey on randevu r (cost=0.57..0.70 rows=1 width=8) (actual time=0.021..0.021 rows=1 loops=7,8 Index Cond: (id = pg.fk_randevu_id) Filter: (baslangic_zamani < (now() + '01:00:00'::interval)) Rows Removed by Filter: 0
16.	15.785	15.785	↑ 1.0	8	3,157 / 8	→ Index Only Scan using mobil_cihaz_fk_hasta_id_idx on mobil_cihaz mc (cost=0.44..0.48 rows=1 width=4) (actual time=0.040..0 Index Cond: (fk_hasta_id = pg.fk_hasta_id) Heap Fetches: 78
17.	0.000	0.000	↑ 20.0	1,000	1,000	→ Materialize (cost=0.00..1.70 rows=20 width=4) (actual time=0.000..0.000 rows=1 loops=1,000)
18.	0.060	0.060	↑ 20.0	1	1	→ Seq Scan on bildirim_konusu bk (cost=0.00..1.60 rows=20 width=4) (actual time=0.060..0.060 rows=1 loops=1)

Planning time	:	17.356 ms
Execution time	:	3,459.013 ms

Color mode:

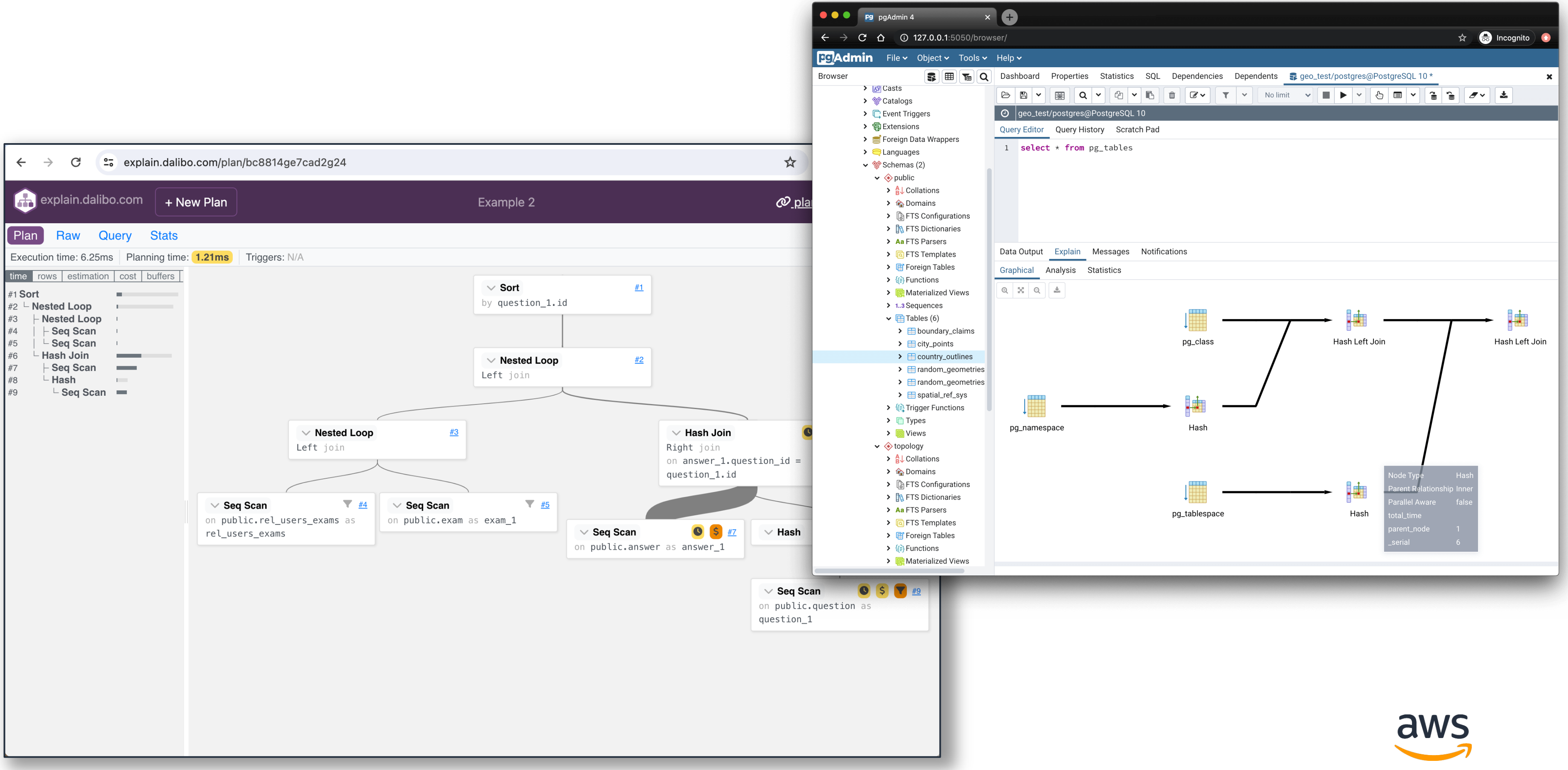
● exclusive ○ inclusive ○ rows x ○ mixed

Visible columns:

☑ # ☑ exclusive ☑ inclusive ☑ rows x ☑ rows ☑ loops

#	exclusive	inclusive	rows x	rows	loops	node
1	0.130	3,458.595	↑ 1.0	1,000	1	→ Limit (cost=10,004.33..2,864,124

Solving Problems With Wait Events



explain.dalibo.com/plan/bc8814ge7cad2g24

Example 2

Execution time: 6.25ms Planning time: 1.21ms Triggers: N/A

```
#1 Sort
#2 └ Nested Loop
#3   ├ Nested Loop
#4   │ ├ Seq Scan
#5   │ └ Seq Scan
#6   └ Hash Join
#7     ├ Seq Scan
#8     └ Hash
#9       └ Seq Scan
```

```
Sort #1
by question_1.id

Nested Loop #2
Left join

Nested Loop #3
Left join

Hash Join
Right join
on answer_1.question_id =
question_1.id

Seq Scan #4
on public.rel_users_exams as
rel_users_exams

Seq Scan #5
on public.exam as exam_1

Seq Scan #7
on public.answer as answer_1

Hash

Seq Scan #9
on public.question as
question_1
```

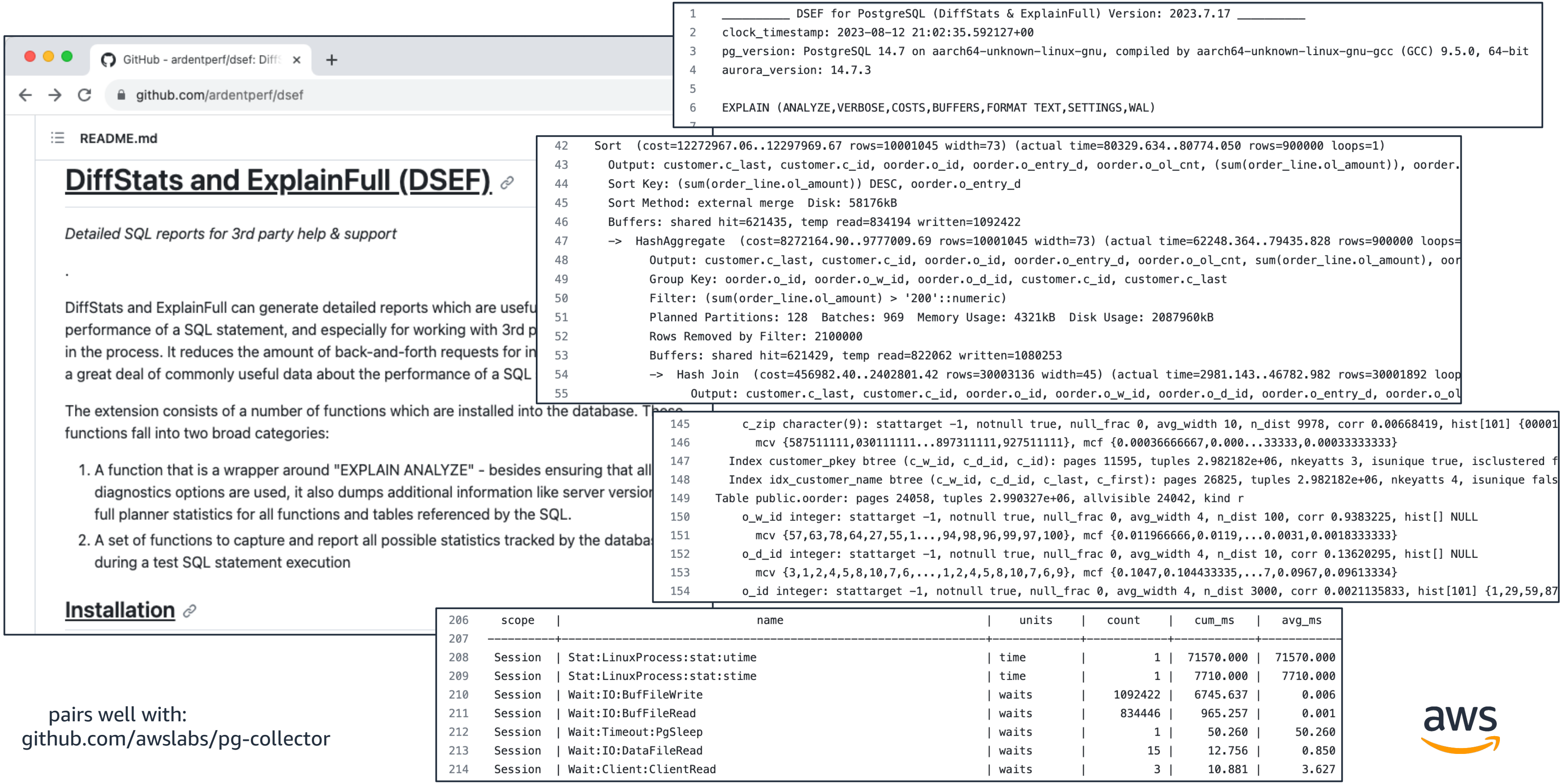
127.0.0.1:5050/browser/

geo_test/postgres@PostgreSQL 10

```
1  select * from pg_tables
```

Node Type	Hash
Parent Relationship	Inner
Parallel Aware	false
total_time	
parent_node	1
_serial	6

Solving Problems With Wait Events



DiffStats and ExplainFull (DSEF)

Detailed SQL reports for 3rd party help & support

.

DiffStats and ExplainFull can generate detailed reports which are usefu
performance of a SQL statement, and especially for working with 3rd p
in the process. It reduces the amount of back-and-forth requests for in
a great deal of commonly useful data about the performance of a SQL

The extension consists of a number of functions which are installed into the database. These functions fall into two broad categories:

1. A function that is a wrapper around "EXPLAIN ANALYZE" - besides ensuring that all diagnostics options are used, it also dumps additional information like server versior full planner statistics for all functions and tables referenced by the SQL.
2. A set of functions to capture and report all possible statistics tracked by the databa during a test SQL statement execution

Installation

```
__________ DSEF for PostgreSQL (DiffStats & ExplainFull) Version: 2023.7.17 __________
clock_timestamp: 2023-08-12 21:02:35.592127+00
pg_version: PostgreSQL 14.7 on aarch64-unknown-linux-gnu, compiled by aarch64-unknown-linux-gnu-gcc (GCC) 9.5.0, 64-bit
aurora_version: 14.7.3

EXPLAIN (ANALYZE,VERBOSE,COSTS,BUFFERS,FORMAT TEXT,SETTINGS,WAL)
```

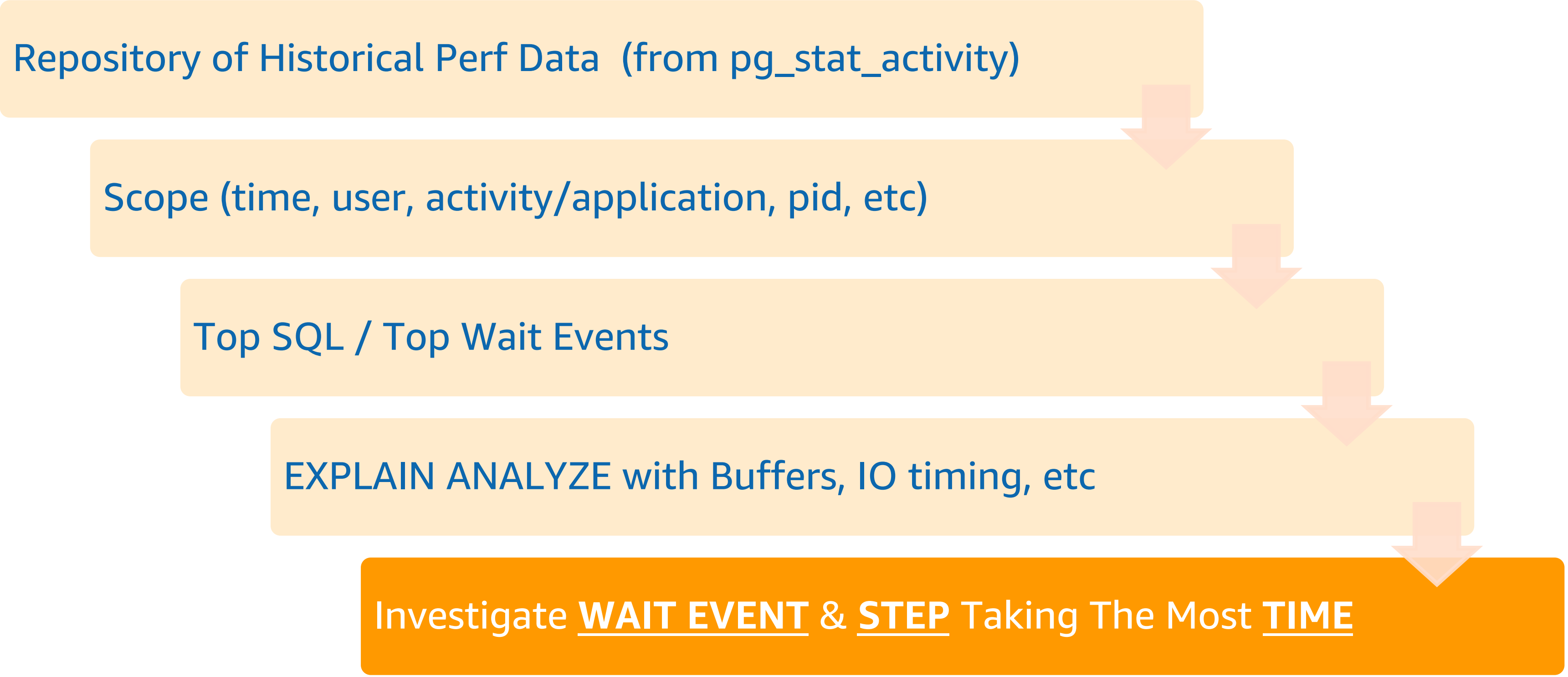
```
Sort  (cost=12272967.06..12297969.67 rows=10001045 width=73) (actual time=80329.634..80774.050 rows=900000 loops=1)
  Output: customer.c_last, customer.c_id, oorder.o_id, oorder.o_entry_d, oorder.o_ol_cnt, (sum(order_line.ol_amount)), oorder.
  Sort Key: (sum(order_line.ol_amount)) DESC, oorder.o_entry_d
  Sort Method: external merge  Disk: 58176kB
  Buffers: shared hit=621435, temp read=834194 written=1092422
  ->  HashAggregate  (cost=8272164.90..9777009.69 rows=10001045 width=73) (actual time=62248.364..79435.828 rows=900000 loops=
        Output: customer.c_last, customer.c_id, oorder.o_id, oorder.o_entry_d, oorder.o_ol_cnt, sum(order_line.ol_amount), oor
        Group Key: oorder.o_id, oorder.o_w_id, oorder.o_d_id, customer.c_id, customer.c_last
        Filter: (sum(order_line.ol_amount) > '200'::numeric)
        Planned Partitions: 128  Batches: 969  Memory Usage: 4321kB  Disk Usage: 2087960kB
        Rows Removed by Filter: 2100000
        Buffers: shared hit=621429, temp read=822062 written=1080253
        ->  Hash Join  (cost=456982.40..2402801.42 rows=30003136 width=45) (actual time=2981.143..46782.982 rows=30001892 loop
              Output: customer.c_last, customer.c_id, oorder.o_id, oorder.o_w_id, oorder.o_d_id, oorder.o_entry_d, oorder.o_ol
```

```
      c_zip character(9): stattarget -1, notnull true, null_frac 0, avg_width 10, n_dist 9978, corr 0.00668419, hist[101] {00001
        mcv {587511111,030111111...897311111,927511111}, mcf {0.00036666667,0.000...33333,0.00033333333}
    Index customer_pkey btree (c_w_id, c_d_id, c_id): pages 11595, tuples 2.982182e+06, nkeyatts 3, isunique true, isclustered f
    Index idx_customer_name btree (c_w_id, c_d_id, c_last, c_first): pages 26825, tuples 2.982182e+06, nkeyatts 4, isunique fals
  Table public.oorder: pages 24058, tuples 2.990327e+06, allvisible 24042, kind r
      o_w_id integer: stattarget -1, notnull true, null_frac 0, avg_width 4, n_dist 100, corr 0.9383225, hist[] NULL
        mcv {57,63,78,64,27,55,1...,94,98,96,99,97,100}, mcf {0.011966666,0.0119,...0.0031,0.0018333333}
      o_d_id integer: stattarget -1, notnull true, null_frac 0, avg_width 4, n_dist 10, corr 0.13620295, hist[] NULL
        mcv {3,1,2,4,5,8,10,7,6,...,1,2,4,5,8,10,7,6,9}, mcf {0.1047,0.104433335,...7,0.0967,0.09613334}
      o_id integer: stattarget -1, notnull true, null_frac 0, avg_width 4, n_dist 3000, corr 0.0021135833, hist[101] {1,29,59,87
```

```
  scope   |                           name                            |   units    |   count    |   cum_ms   |   avg_ms
----------+-----------------------------------------------------------+------------+------------+------------+-----------
 Session  | Stat:LinuxProcess:stat:utime                              | time       |          1 |  71570.000 |  71570.000
 Session  | Stat:LinuxProcess:stat:stime                              | time       |          1 |   7710.000 |   7710.000
 Session  | Wait:IO:BufFileWrite                                      | waits      |    1092422 |   6745.637 |      0.006
 Session  | Wait:IO:BufFileRead                                       | waits      |     834446 |    965.257 |      0.001
 Session  | Wait:Timeout:PgSleep                                      | waits      |          1 |     50.260 |     50.260
 Session  | Wait:IO:DataFileRead                                      | waits      |         15 |     12.756 |      0.850
 Session  | Wait:Client:ClientRead                                    | waits      |          3 |     10.881 |      3.627
```

pairs well with:
github.com/awslabs/pg-collector

Solving Problems With Wait Events



Repository of Historical Perf Data (from pg_stat_activity)

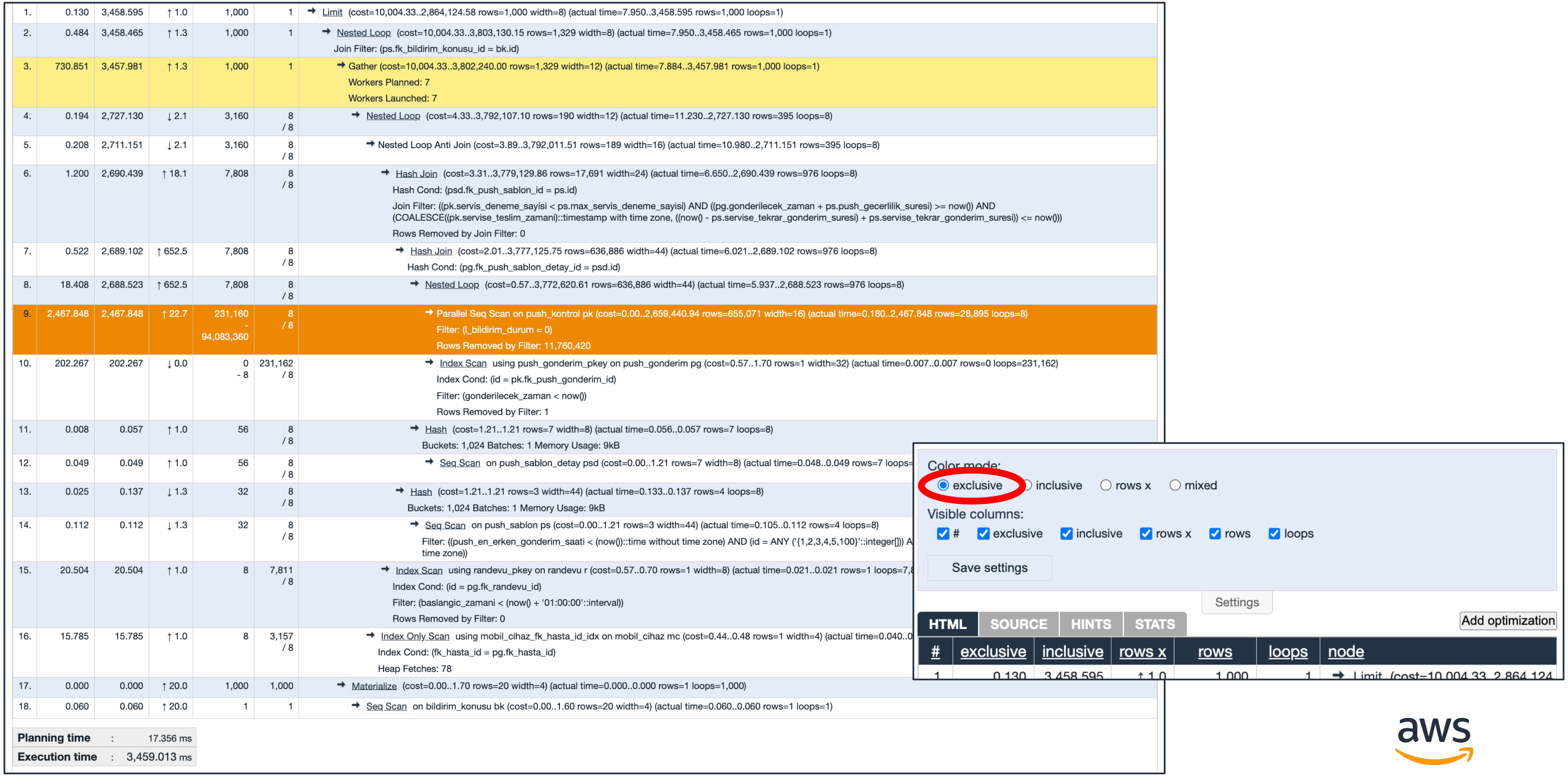
Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate **WAIT EVENT** & **STEP** Taking The Most **TIME**

Solving Problems With Wait Events



1.	0.130	3,458.595	↑ 1.0	1,000	1	→ Limit (cost=10,004.33..2,864,124.58 rows=1,000 width=8) (actual time=7.950..3,458.595 rows=1,000 loops=1)
2.	0.484	3,458.465	↑ 1.3	1,000	1	→ Nested Loop (cost=10,004.33..3,803,130.15 rows=1,329 width=8) (actual time=7.950..3,458.465 rows=1,000 loops=1) Join Filter: (ps.fk_bildirim_konusu_id = bk.id)
3.	730.851	3,457.981	↑ 1.3	1,000	1	→ Gather (cost=10,004.33..3,802,240.00 rows=1,329 width=12) (actual time=7.884..3,457.981 rows=1,000 loops=1) Workers Planned: 7 Workers Launched: 7
4.	0.194	2,727.130	↓ 2.1	3,160	8 / 8	→ Nested Loop (cost=4.33..3,792,107.10 rows=190 width=12) (actual time=11.230..2,727.130 rows=395 loops=8)
5.	0.208	2,711.151	↓ 2.1	3,160	8 / 8	→ Nested Loop Anti Join (cost=3.89..3,792,011.51 rows=189 width=16) (actual time=10.980..2,711.151 rows=395 loops=8)
6.	1.200	2,690.439	↑ 18.1	7,808	8 / 8	→ Hash Join (cost=3.31..3,779,129.86 rows=17,691 width=24) (actual time=6.650..2,690.439 rows=976 loops=8) Hash Cond: (psd.fk_push_sablon_id = ps.id) Join Filter: ((pk.servis_deneme_sayisi < ps.max_servis_deneme_sayisi) AND ((pg.gonderilecek_zaman + ps.push_gecerlilik_suresi) >= now()) AND (COALESCE((pk.servise_teslim_zamani)::timestamp with time zone, ((now() - ps.servise_tekrar_gonderim_suresi) + ps.servise_tekrar_gonderim_suresi)) <= now())) Rows Removed by Join Filter: 0
7.	0.522	2,689.102	↑ 652.5	7,808	8 / 8	→ Hash Join (cost=2.01..3,777,125.75 rows=636,886 width=44) (actual time=6.021..2,689.102 rows=976 loops=8) Hash Cond: (pg.fk_push_sablon_detay_id = psd.id)
8.	18.408	2,688.523	↑ 652.5	7,808	8 / 8	→ Nested Loop (cost=0.57..3,772,620.61 rows=636,886 width=44) (actual time=5.937..2,688.523 rows=976 loops=8)
9.	2,467.848	2,467.848	↑ 22.7	231,160 - 94,083,360	8 / 8	→ Parallel Seq Scan on push_kontrol pk (cost=0.00..2,659,440.94 rows=655,071 width=16) (actual time=0.180..2,467.848 rows=28,895 loops=8) Filter: (l_bildirim_durum = 0) Rows Removed by Filter: 11,760,420
10.	202.267	202.267	↓ 0.0	0 - 8	231,162 / 8	→ Index Scan using push_gonderim_pkey on push_gonderim pg (cost=0.57..1.70 rows=1 width=32) (actual time=0.007..0.007 rows=0 loops=231,162) Index Cond: (id = pk.fk_push_gonderim_id) Filter: (gonderilecek_zaman < now()) Rows Removed by Filter: 1
11.	0.008	0.057	↑ 1.0	56	8 / 8	→ Hash (cost=1.21..1.21 rows=7 width=8) (actual time=0.056..0.057 rows=7 loops=8) Buckets: 1,024 Batches: 1 Memory Usage: 9kB
12.	0.049	0.049	↑ 1.0	56	8 / 8	→ Seq Scan on push_sablon_detay psd (cost=0.00..1.21 rows=7 width=8) (actual time=0.048..0.049 rows=7 loops=
13.	0.025	0.137	↓ 1.3	32	8 / 8	→ Hash (cost=1.21..1.21 rows=3 width=44) (actual time=0.133..0.137 rows=4 loops=8) Buckets: 1,024 Batches: 1 Memory Usage: 9kB
14.	0.112	0.112	↓ 1.3	32	8 / 8	→ Seq Scan on push_sablon ps (cost=0.00..1.21 rows=3 width=44) (actual time=0.105..0.112 rows=4 loops=8) Filter: ((push_en_erken_gonderim_saati < (now())::time without time zone) AND (id = ANY ('{1,2,3,4,5,100}'::integer[])) A time zone))
15.	20.504	20.504	↑ 1.0	8	7,811 / 8	→ Index Scan using randevu_pkey on randevu r (cost=0.57..0.70 rows=1 width=8) (actual time=0.021..0.021 rows=1 loops=7, Index Cond: (id = pg.fk_randevu_id) Filter: (baslangic_zamani < (now() + '01:00:00'::interval)) Rows Removed by Filter: 0
16.	15.785	15.785	↑ 1.0	8	3,157 / 8	→ Index Only Scan using mobil_cihaz_fk_hasta_id_idx on mobil_cihaz mc (cost=0.44..0.48 rows=1 width=4) (actual time=0.040..0 Index Cond: (fk_hasta_id = pg.fk_hasta_id) Heap Fetches: 78
17.	0.000	0.000	↑ 20.0	1,000	1,000	→ Materialize (cost=0.00..1.70 rows=20 width=4) (actual time=0.000..0.000 rows=1 loops=1,000)
18.	0.060	0.060	↑ 20.0	1	1	→ Seq Scan on bildirim_konusu bk (cost=0.00..1.60 rows=20 width=4) (actual time=0.060..0.060 rows=1 loops=1)

Planning time	:	17.356 ms
Execution time	:	3,459.013 ms

Color mode:

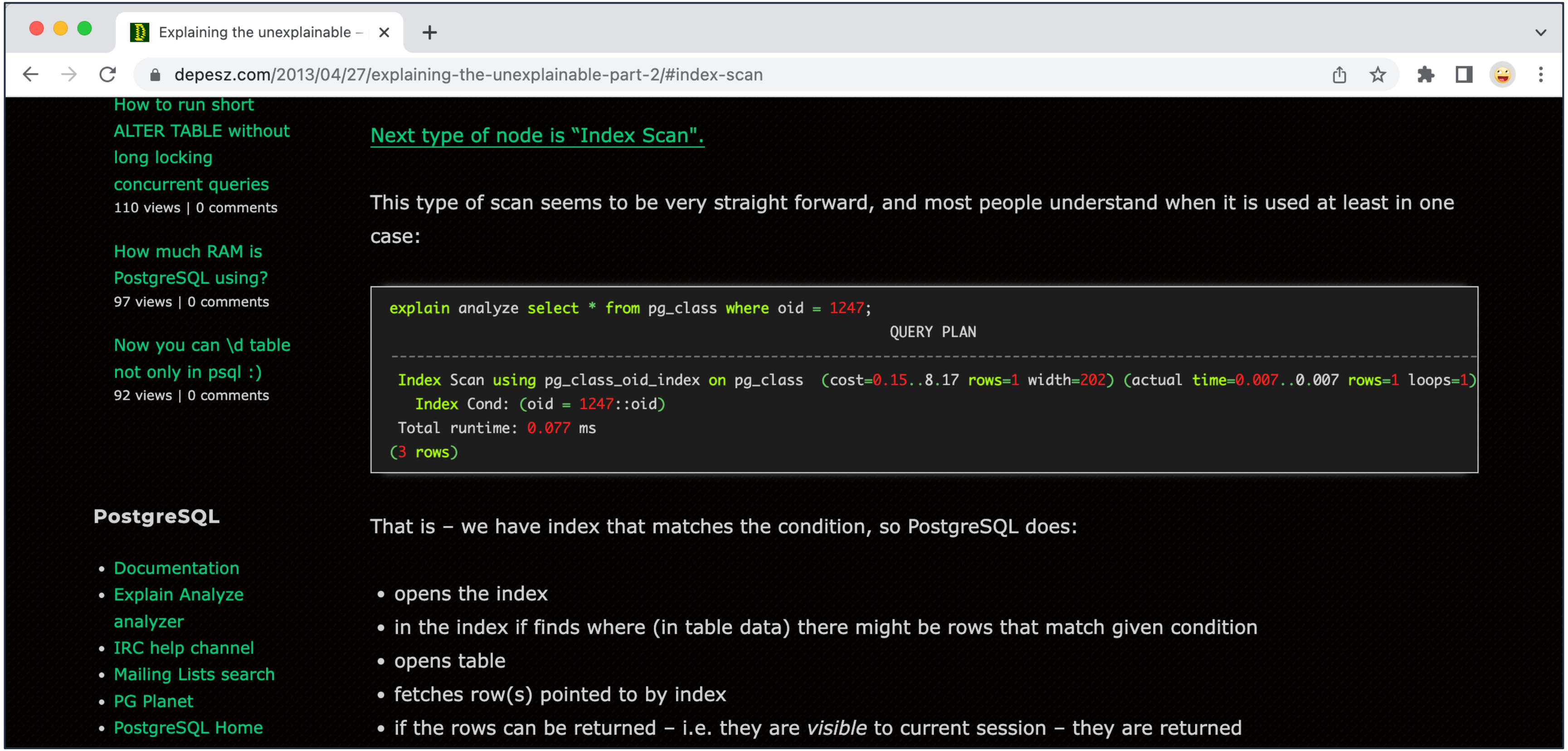
● exclusive ○ inclusive ○ rows x ○ mixed

Visible columns:

☑ # ☑ exclusive ☑ inclusive ☑ rows x ☑ rows ☑ loops

#	exclusive	inclusive	rows x	rows	loops	node
1	0.130	3,458.595	↑ 1.0	1,000	1	→ Limit (cost=10,004.33..2,864,124

Solving Problems With Wait Events


Explaining the unexplainable –
depesz.com/2013/04/27/explaining-the-unexplainable-part-2/#index-scan

How to run short ALTER TABLE without long locking concurrent queries
110 views | 0 comments

How much RAM is PostgreSQL using?
97 views | 0 comments

Now you can \d table not only in psql :)
92 views | 0 comments

PostgreSQL

- Documentation
- Explain Analyze analyzer
- IRC help channel
- Mailing Lists search
- PG Planet
- PostgreSQL Home

Next type of node is "Index Scan".

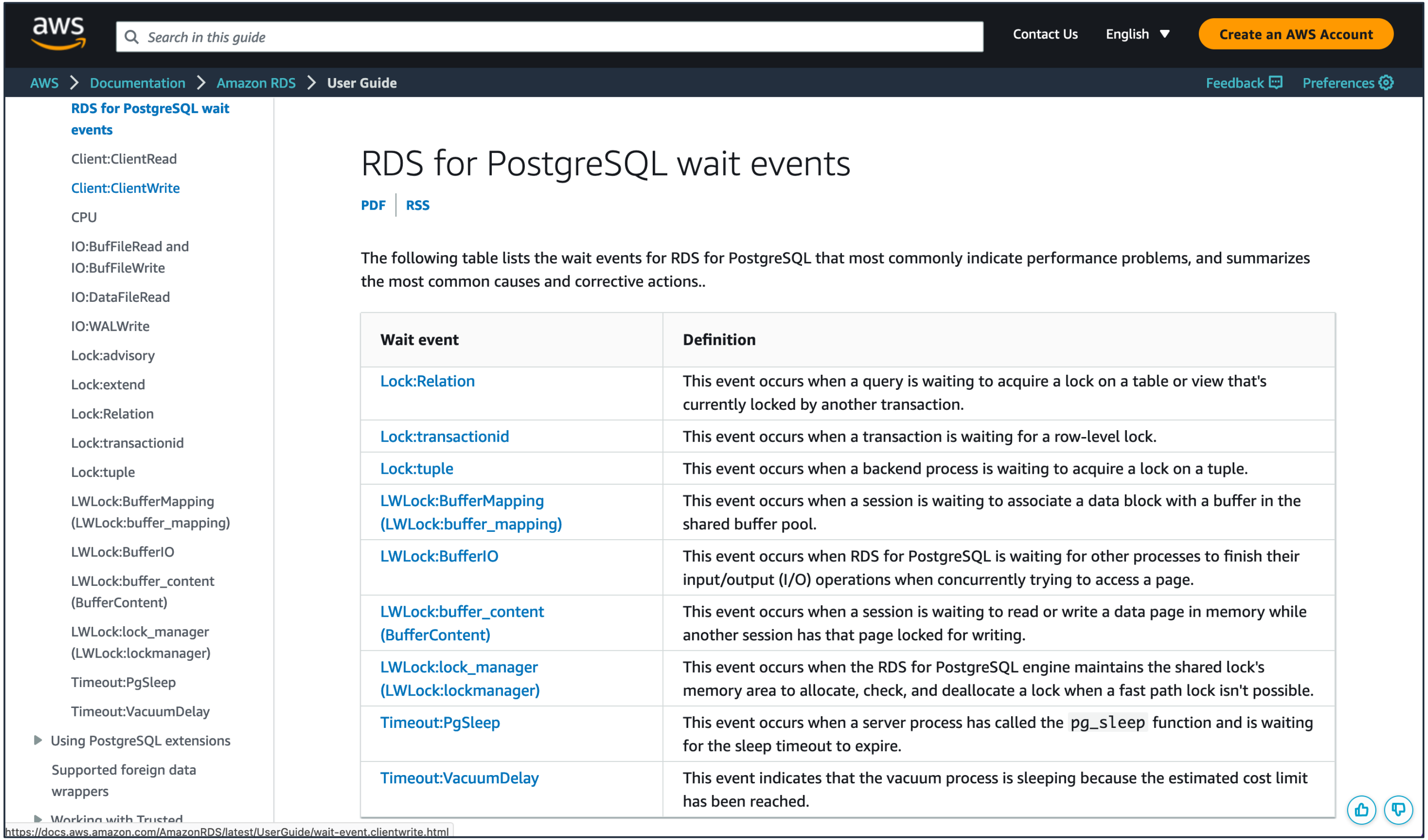
This type of scan seems to be very straight forward, and most people understand when it is used at least in one case:

```
explain analyze select * from pg_class where oid = 1247;
                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Index Scan using pg_class_oid_index on pg_class  (cost=0.15..8.17 rows=1 width=202) (actual time=0.007..0.007 rows=1 loops=1)
   Index Cond: (oid = 1247::oid)
 Total runtime: 0.077 ms
(3 rows)
```

That is – we have index that matches the condition, so PostgreSQL does:

- opens the index
- in the index if finds where (in table data) there might be rows that match given condition
- opens table
- fetches row(s) pointed to by index
- if the rows can be returned – i.e. they are *visible* to current session – they are returned

Solving Problems With Wait Events



RDS for PostgreSQL wait events

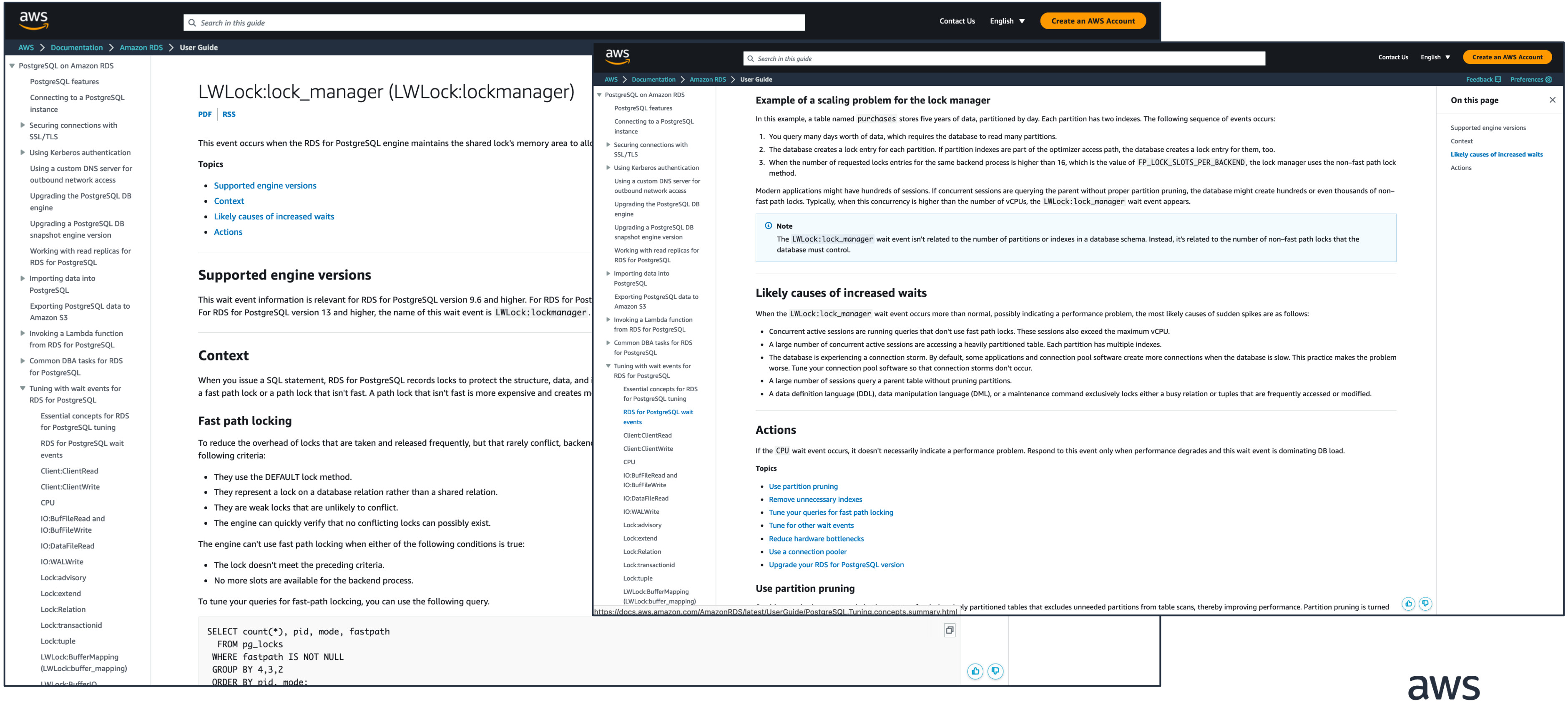
PDF | RSS

The following table lists the wait events for RDS for PostgreSQL that most commonly indicate performance problems, and summarizes the most common causes and corrective actions..

Wait event	Definition
Lock:Relation	This event occurs when a query is waiting to acquire a lock on a table or view that's currently locked by another transaction.
Lock:transactionid	This event occurs when a transaction is waiting for a row-level lock.
Lock:tuple	This event occurs when a backend process is waiting to acquire a lock on a tuple.
LWLock:BufferMapping (LWLock:buffer_mapping)	This event occurs when a session is waiting to associate a data block with a buffer in the shared buffer pool.
LWLock:BufferIO	This event occurs when RDS for PostgreSQL is waiting for other processes to finish their input/output (I/O) operations when concurrently trying to access a page.
LWLock:buffer_content (BufferContent)	This event occurs when a session is waiting to read or write a data page in memory while another session has that page locked for writing.
LWLock:lock_manager (LWLock:lockmanager)	This event occurs when the RDS for PostgreSQL engine maintains the shared lock's memory area to allocate, check, and deallocate a lock when a fast path lock isn't possible.
Timeout:PgSleep	This event occurs when a server process has called the `pg_sleep` function and is waiting for the sleep timeout to expire.
Timeout:VacuumDelay	This event indicates that the vacuum process is sleeping because the estimated cost limit has been reached.

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/wait-event.clientwrite.html

Solving Problems With Wait Events


aws
Search in this guide
Contact Us
English
Create an AWS Account
AWS > Documentation > Amazon RDS > User Guide
PostgreSQL on Amazon RDS
PostgreSQL features
Connecting to a PostgreSQL instance
Securing connections with SSL/TLS
Using Kerberos authentication
Using a custom DNS server for outbound network access
Upgrading the PostgreSQL DB engine
Upgrading a PostgreSQL DB snapshot engine version
Working with read replicas for RDS for PostgreSQL
Importing data into PostgreSQL
Exporting PostgreSQL data to Amazon S3
Invoking a Lambda function from RDS for PostgreSQL
Common DBA tasks for RDS for PostgreSQL
Tuning with wait events for RDS for PostgreSQL
Essential concepts for RDS for PostgreSQL tuning
RDS for PostgreSQL wait events
Client:ClientRead
Client:ClientWrite
CPU
IO:BufFileRead and IO:BufFileWrite
IO:DataFileRead
IO:WALWrite
Lock:advisory
Lock:extend
Lock:Relation
Lock:transactionid
Lock:tuple
LWLock:BufferMapping (LWLock:buffer_mapping)
LWLock:BufferIO
PDF
RSS

LWLock:lock_manager (LWLock:lockmanager)

This event occurs when the RDS for PostgreSQL engine maintains the shared lock's memory area to allo

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is relevant for RDS for PostgreSQL version 9.6 and higher. For RDS for Post
For RDS for PostgreSQL version 13 and higher, the name of this wait event is `LWLock:lockmanager`.

Context

When you issue a SQL statement, RDS for PostgreSQL records locks to protect the structure, data, and
a fast path lock or a path lock that isn't fast. A path lock that isn't fast is more expensive and creates m

Fast path locking

To reduce the overhead of locks that are taken and released frequently, but that rarely conflict, backend
following criteria:

- They use the DEFAULT lock method.
- They represent a lock on a database relation rather than a shared relation.
- They are weak locks that are unlikely to conflict.
- The engine can quickly verify that no conflicting locks can possibly exist.

The engine can't use fast path locking when either of the following conditions is true:

- The lock doesn't meet the preceding criteria.
- No more slots are available for the backend process.

To tune your queries for fast-path lockcing, you can use the following query.

```
SELECT count(*), pid, mode, fastpath
  FROM pg_locks
 WHERE fastpath IS NOT NULL
 GROUP BY 4,3,2
 ORDER BY pid, mode;
```

aws
Search in this guide
Contact Us
English
Create an AWS Account
AWS > Documentation > Amazon RDS > User Guide
Feedback
Preferences
PostgreSQL on Amazon RDS
PostgreSQL features
Connecting to a PostgreSQL instance
Securing connections with SSL/TLS
Using Kerberos authentication
Using a custom DNS server for outbound network access
Upgrading the PostgreSQL DB engine
Upgrading a PostgreSQL DB snapshot engine version
Working with read replicas for RDS for PostgreSQL
Importing data into PostgreSQL
Exporting PostgreSQL data to Amazon S3
Invoking a Lambda function from RDS for PostgreSQL
Common DBA tasks for RDS for PostgreSQL
Tuning with wait events for RDS for PostgreSQL
Essential concepts for RDS for PostgreSQL tuning
RDS for PostgreSQL wait events
Client:ClientRead
Client:ClientWrite
CPU
IO:BufFileRead and IO:BufFileWrite
IO:DataFileRead
IO:WALWrite
Lock:advisory
Lock:extend
Lock:Relation
Lock:transactionid
Lock:tuple
LWLock:BufferMapping (LWLock:buffer_mapping)
On this page
Supported engine versions
Context
Likely causes of increased waits
Actions

Example of a scaling problem for the lock manager

In this example, a table named `purchases` stores five years of data, partitioned by day. Each partition has two indexes. The following sequence of events occurs:

1. You query many days worth of data, which requires the database to read many partitions.
2. The database creates a lock entry for each partition. If partition indexes are part of the optimizer access path, the database creates a lock entry for them, too.
3. When the number of requested locks entries for the same backend process is higher than 16, which is the value of `FP_LOCK_SLOTS_PER_BACKEND`, the lock manager uses the non–fast path lock method.

Modern applications might have hundreds of sessions. If concurrent sessions are querying the parent without proper partition pruning, the database might create hundreds or even thousands of non–fast path locks. Typically, when this concurrency is higher than the number of vCPUs, the `LWLock:lock_manager` wait event appears.

> **Note**
> The `LWLock:lock_manager` wait event isn't related to the number of partitions or indexes in a database schema. Instead, it's related to the number of non–fast path locks that the database must control.

Likely causes of increased waits

When the `LWLock:lock_manager` wait event occurs more than normal, possibly indicating a performance problem, the most likely causes of sudden spikes are as follows:

- Concurrent active sessions are running queries that don't use fast path locks. These sessions also exceed the maximum vCPU.
- A large number of concurrent active sessions are accessing a heavily partitioned table. Each partition has multiple indexes.
- The database is experiencing a connection storm. By default, some applications and connection pool software create more connections when the database is slow. This practice makes the problem worse. Tune your connection pool software so that connection storms don't occur.
- A large number of sessions query a parent table without pruning partitions.
- A data definition language (DDL), data manipulation language (DML), or a maintenance command exclusively locks either a busy relation or tuples that are frequently accessed or modified.

Actions

If the `CPU` wait event occurs, it doesn't necessarily indicate a performance problem. Respond to this event only when performance degrades and this wait event is dominating DB load.

Topics

- Use partition pruning
- Remove unnecessary indexes
- Tune your queries for fast path locking
- Tune for other wait events
- Reduce hardware bottlenecks
- Use a connection pooler
- Upgrade your RDS for PostgreSQL version

Use partition pruning

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/PostgreSQL.Tuning.concepts.summary.html ly partitioned tables that excludes unneeded partitions from table scans, thereby improving performance. Partition pruning is turned

Scenario:

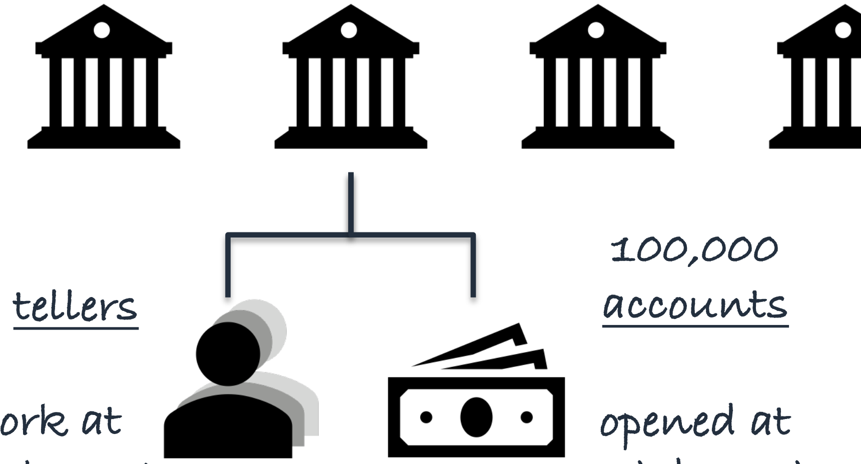
Small Bank, Lots of Business

Small Bank, Lots of Business

Our bank is small because we only have 10 branches

```
1. BEGIN;
2. UPDATE accounts
        SET abalance = abalance + :delta
        WHERE account = :account;
3. SELECT balance FROM accounts
        WHERE account = :account;
4. UPDATE tellers
        SET balance = balance + :delta
        WHERE teller = :teller;
5. UPDATE branches
        SET balance = balance + :delta
        WHERE branch = :branch;
6. INSERT INTO history
        VALUES (:teller, :branch, :account,
                        :delta, CURRENT_TIMESTAMP);
7. END; (COMMIT TRANSACTION)
```

Bank branches (can scale this)



10 tellers

work at each branch

100,000 accounts

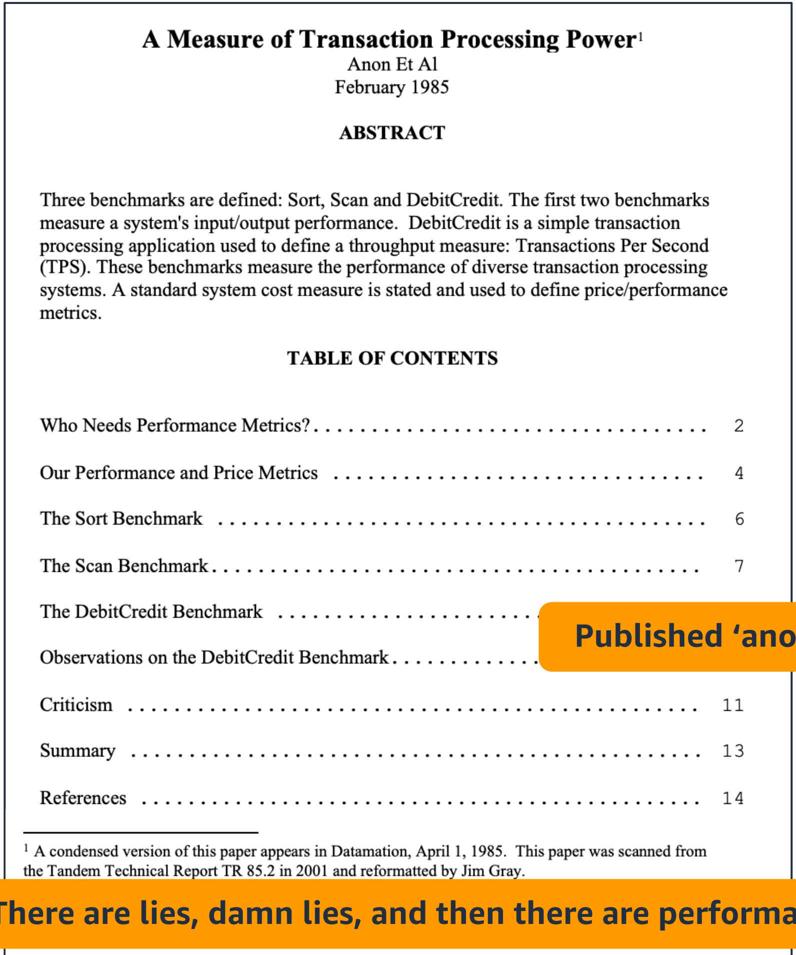
opened at each branch

Very important for regulators!

audit history

Small Bank, Lots of Business

Jim Gray
April fools day 1985
Changed DB field forever



A Measure of Transaction Processing Power[1]

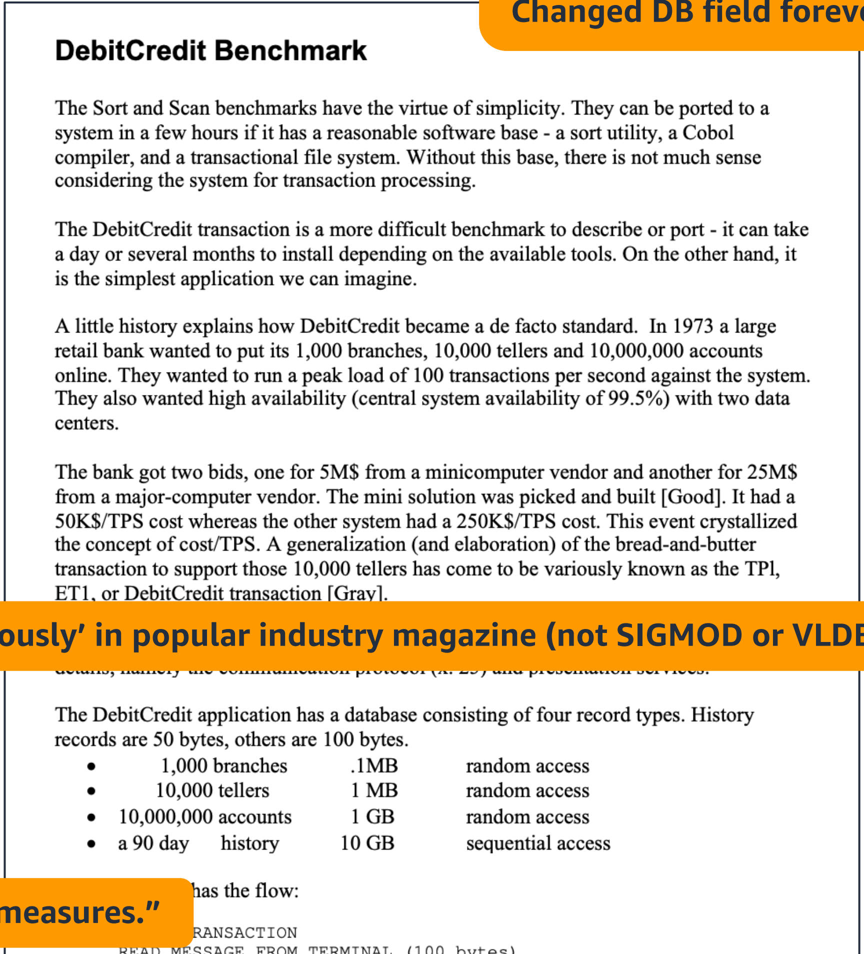
Anon Et Al
February 1985

ABSTRACT

Three benchmarks are defined: Sort, Scan and DebitCredit. The first two benchmarks measure a system's input/output performance. DebitCredit is a simple transaction processing application used to define a throughput measure: Transactions Per Second (TPS). These benchmarks measure the performance of diverse transaction processing systems. A standard system cost measure is stated and used to define price/performance metrics.

TABLE OF CONTENTS

[1] A condensed version of this paper appears in Datamation, April 1, 1985. This paper was scanned from the Tandem Technical Report TR 85.2 in 2001 and reformatted by Jim Gray.



DebitCredit Benchmark

The Sort and Scan benchmarks have the virtue of simplicity. They can be ported to a system in a few hours if it has a reasonable software base - a sort utility, a Cobol compiler, and a transactional file system. Without this base, there is not much sense considering the system for transaction processing.

The DebitCredit transaction is a more difficult benchmark to describe or port - it can take a day or several months to install depending on the available tools. On the other hand, it is the simplest application we can imagine.

A little history explains how DebitCredit became a de facto standard. In 1973 a large retail bank wanted to put its 1,000 branches, 10,000 tellers and 10,000,000 accounts online. They wanted to run a peak load of 100 transactions per second against the system. They also wanted high availability (central system availability of 99.5%) with two data centers.

The bank got two bids, one for 5M$ from a minicomputer vendor and another for 25M$ from a major-computer vendor. The mini solution was picked and built [Good]. It had a 50K$/TPS cost whereas the other system had a 250K$/TPS cost. This event crystallized the concept of cost/TPS. A generalization (and elaboration) of the bread-and-butter transaction to support those 10,000 tellers has come to be variously known as the TPl, ET1, or DebitCredit transaction [Gray].

details, namely the communication protocol (x. 25) and presentation services.

The DebitCredit application has a database consisting of four record types. History records are 50 bytes, others are 100 bytes.

•	1,000 branches	.1MB	random access
•	10,000 tellers	1 MB	random access
•	10,000,000 accounts	1 GB	random access
•	a 90 day history	10 GB	sequential access

has the flow:

```
RANSACTION
    READ MESSAGE FROM TERMINAL (100 bytes)
```

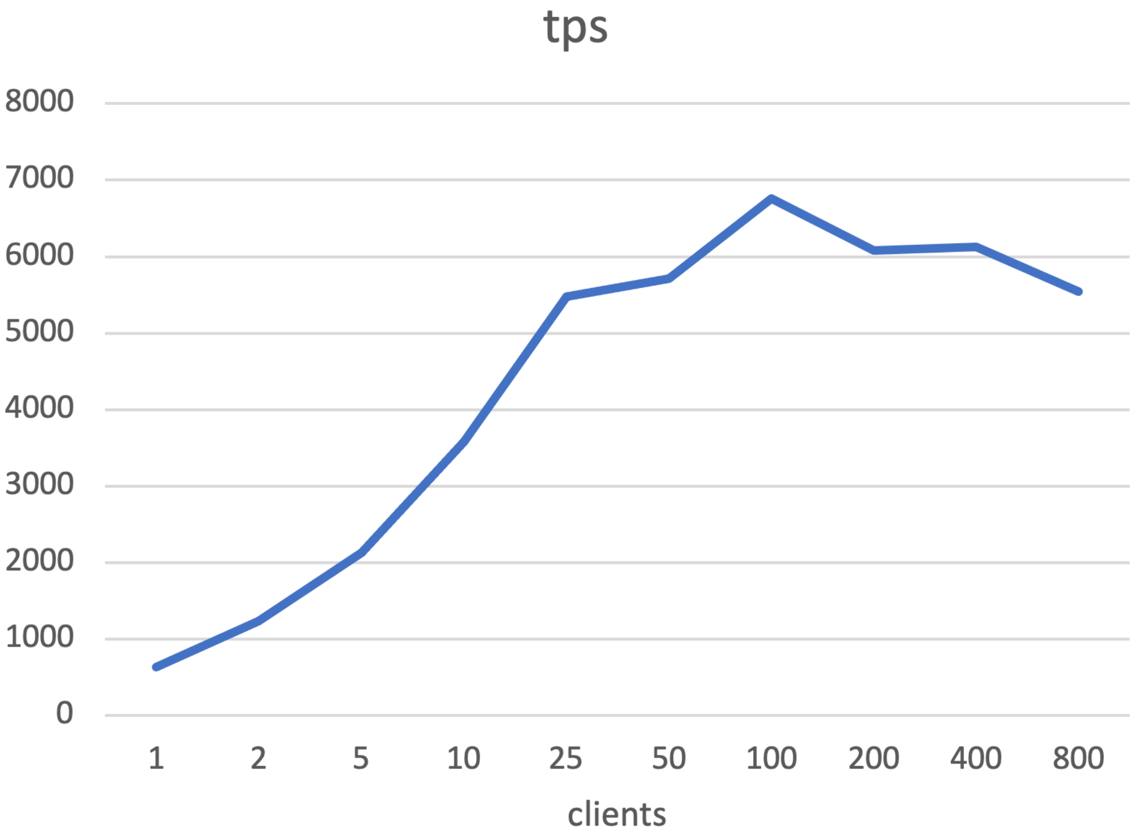
Published 'anonymously' in popular industry magazine (not SIGMOD or VLDB)

"There are lies, damn lies, and then there are performance measures."

Small Bank, Lots of Business

```
pgbench --initialize --scale=10


for CLIENT in 1 2 5 10 25 50
              100 200 400 800; do

  pgbench --client=$CLIENT
          --progress=2 --time=30

done 2>&1 | tee benchmark.log


egrep '(clients:|progress: 30)'
      benchmark.log | paste - -
```

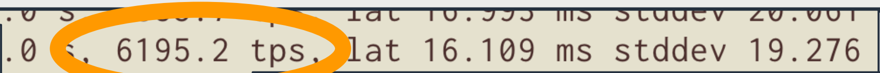
```
starting
progress:
progress:
progress:
progress:
progress:
progress:
progress:
progress: 16.0 s, 629.5 tps, lat 1.589 ms stddev 0.070
progress: 18.0 s, 637.5 tps, lat 1.569 ms stddev 0.087
progress: 20.0 s, 634.5 tps, lat 1.576 ms stddev 0.075
progress: 22.0 s, 630.0 tps, lat 1.587 ms stddev 0.342
progress: 24.0 s, 633.5 tps, lat 1.578 ms stddev 0.076
progress: 26.0 s, 625.5 tps, lat 1.599 ms stddev 0.138
progress: 28.0 s, 614.5 tps, lat 1.627 ms stddev 0.559
progress: 30.0 s, 638.0 tps, lat 1.567 ms stddev 0.204
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 10
query mode: simple
number of clients: 1
number of threads: 1
duration: 30 s
number of transactions actually processed: 18947
latency average = 1.583 ms
latency stddev = 0.245 ms
tps = 631.552734 (including connections establishing)
tps = 631.696406 (excluding connections establishing)
```



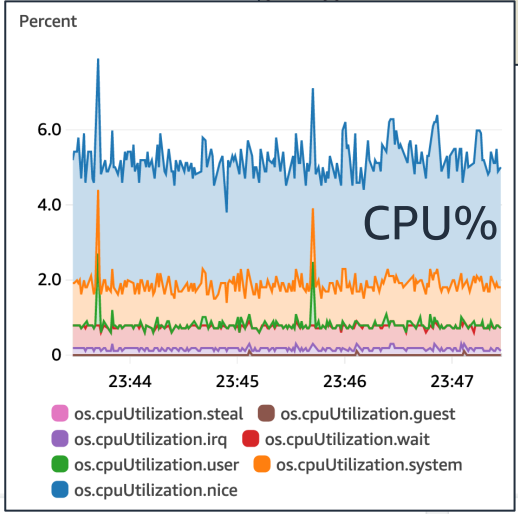
Xeon Ice Lake, 1 socket (processor), 32 cores, 64 threads
512 GiB memory

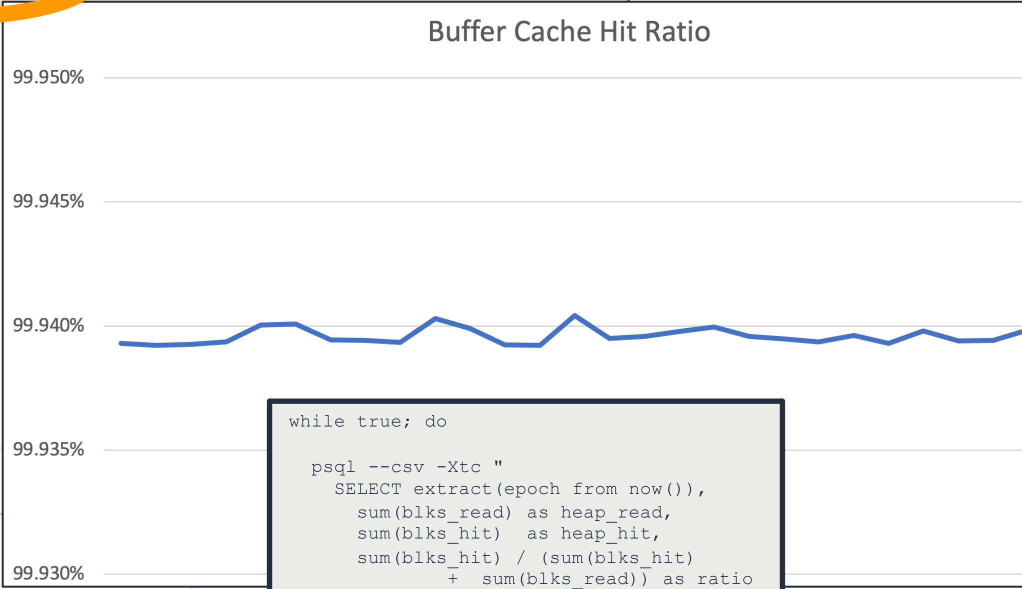
The Old Method: Counter and Ratio Metrics

```
pgbench --client=100 --progress=5 --time=9999
```

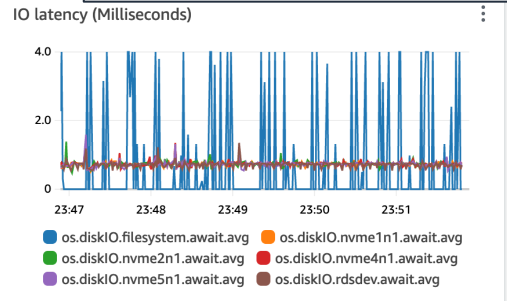


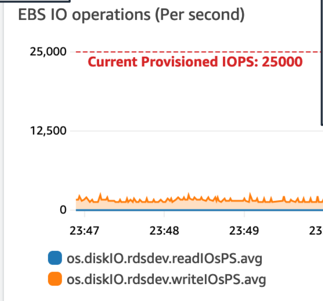
```
progress: 605.0 s        lat 16.995 ms stddev 20.061
        .0 s, 6195.2 tps, lat 16.109 ms stddev 19.276
```



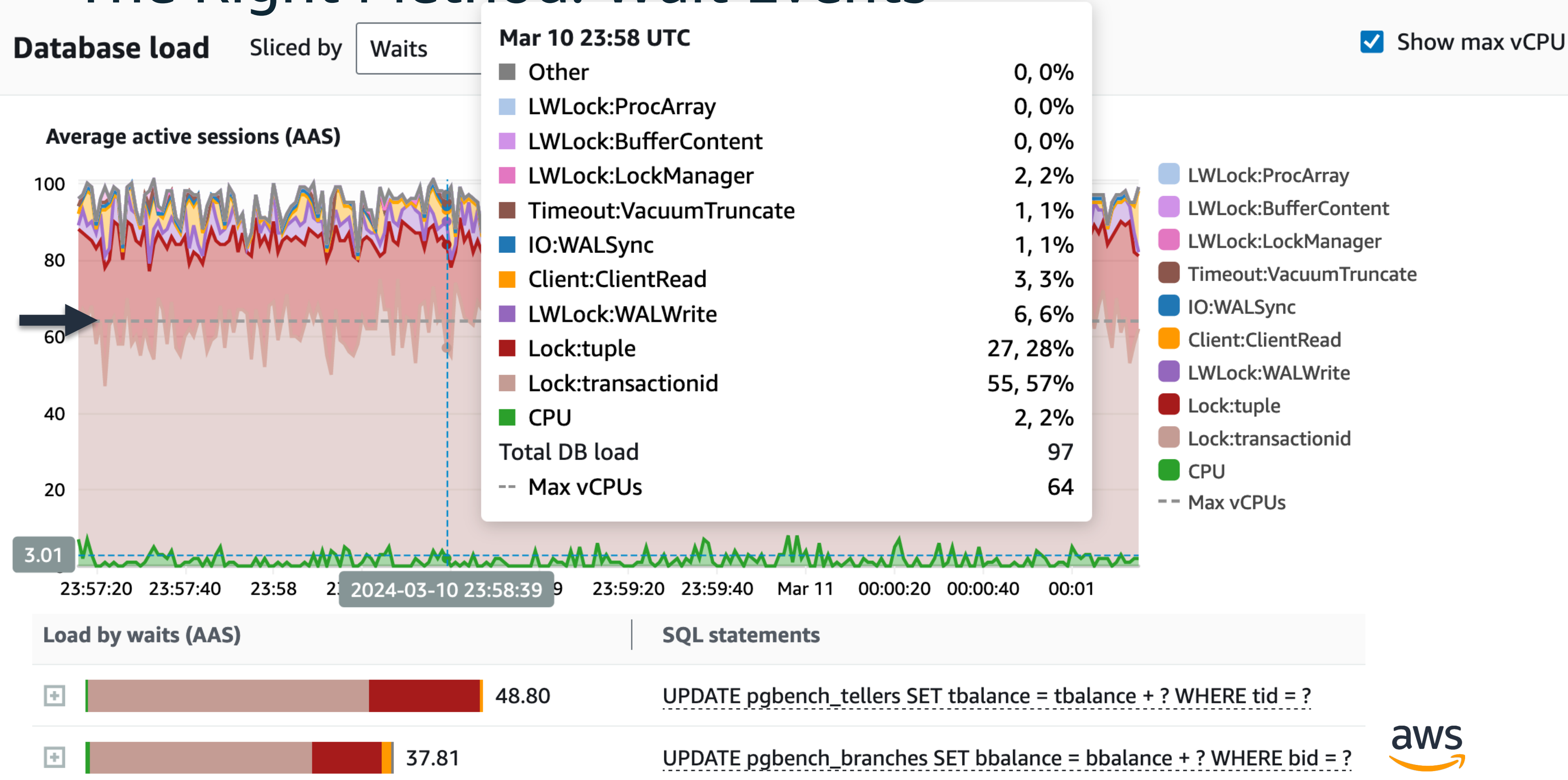


```
while true; do

  psql --csv -Xtc "
    SELECT extract(epoch from now()),
      sum(blks_read) as heap_read,
      sum(blks_hit)  as heap_hit,
      sum(blks_hit) / (sum(blks_hit)
            + sum(blks_read)) as ratio
    FROM pg_stat_database;
  " -c "
    SELECT pg_stat_reset();
  "

  sleep 5;
done;
```





The Right Method: Wait Events



Database load Sliced by Waits

☑ Show max vCPU

Average active sessions (AAS)

Mar 10 23:58 UTC

Other	0, 0%
LWLock:ProcArray	0, 0%
LWLock:BufferContent	0, 0%
LWLock:LockManager	2, 2%
Timeout:VacuumTruncate	1, 1%
IO:WALSync	1, 1%
Client:ClientRead	3, 3%
LWLock:WALWrite	6, 6%
Lock:tuple	27, 28%
Lock:transactionid	55, 57%
CPU	2, 2%
Total DB load	97
-- Max vCPUs	64

- LWLock:ProcArray
- LWLock:BufferContent
- LWLock:LockManager
- Timeout:VacuumTruncate
- IO:WALSync
- Client:ClientRead
- LWLock:WALWrite
- Lock:tuple
- Lock:transactionid
- CPU
- -- Max vCPUs

Load by waits (AAS)	**SQL statements**
⊞ 48.80	UPDATE pgbench_tellers SET tbalance = tbalance + ? WHERE tid = ?
⊞ 37.81	UPDATE pgbench_branches SET bbalance = bbalance + ? WHERE bid = ?

The Right Method: Wait Events

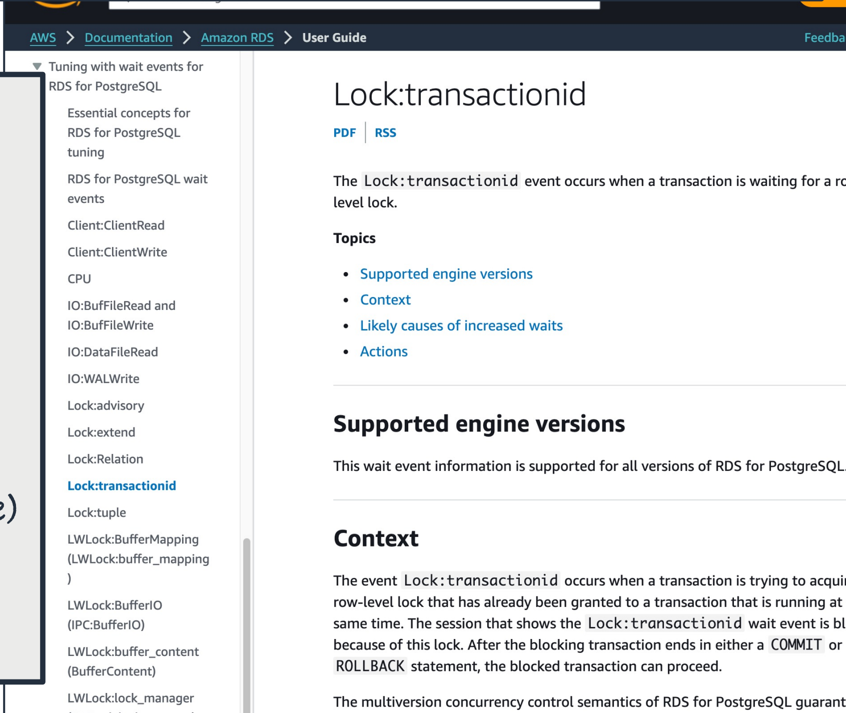
```
1. BEGIN;
2. UPDATE accounts
        SET abalance = abalance + :delta
        WHERE account = :account;
3. SELECT balance FROM accounts
        WHERE account = :account;
~~4. UPDATE tellers~~
~~        SET balance = b~~
~~        WHERE teller =~~
~~5. UPDATE branches~~
~~        SET balance = b~~
~~        WHERE branch~~
6. INSERT INTO histo
        VALUES (:teller,
                  :del
7. END; (COMMIT TRA
```

```
-- Scheduled job, run every few seconds

UPDATE tellers
  SET balance = (SELECT SUM(delta)
           FROM history
           WHERE h.teller=t.teller
           GROUP BY teller)

UPDATE branches
  SET balance = (SELECT SUM(balance)
           FROM accounts
           WHERE a.branch=b.branch
           GROUP BY branch)
```

Top SQL: UPDATE **tellers** & UPDATE **branches**

Top Waits Events: **Lock:transactionid** & **Lock:tuple**

= you have hot records in your table



AWS > Documentation > Amazon RDS > User Guide

Feedba

- Tuning with wait events for RDS for PostgreSQL
  - Essential concepts for RDS for PostgreSQL tuning
  - RDS for PostgreSQL wait events
  - Client:ClientRead
  - Client:ClientWrite
  - CPU
  - IO:BufFileRead and IO:BufFileWrite
  - IO:DataFileRead
  - IO:WALWrite
  - Lock:advisory
  - Lock:extend
  - Lock:Relation
  - **Lock:transactionid**
  - Lock:tuple
  - LWLock:BufferMapping (LWLock:buffer_mapping)
  - LWLock:BufferIO (IPC:BufferIO)
  - LWLock:buffer_content (BufferContent)
  - LWLock:lock_manager

Lock:transactionid

PDF | RSS

The `Lock:transactionid` event occurs when a transaction is waiting for a ro level lock.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

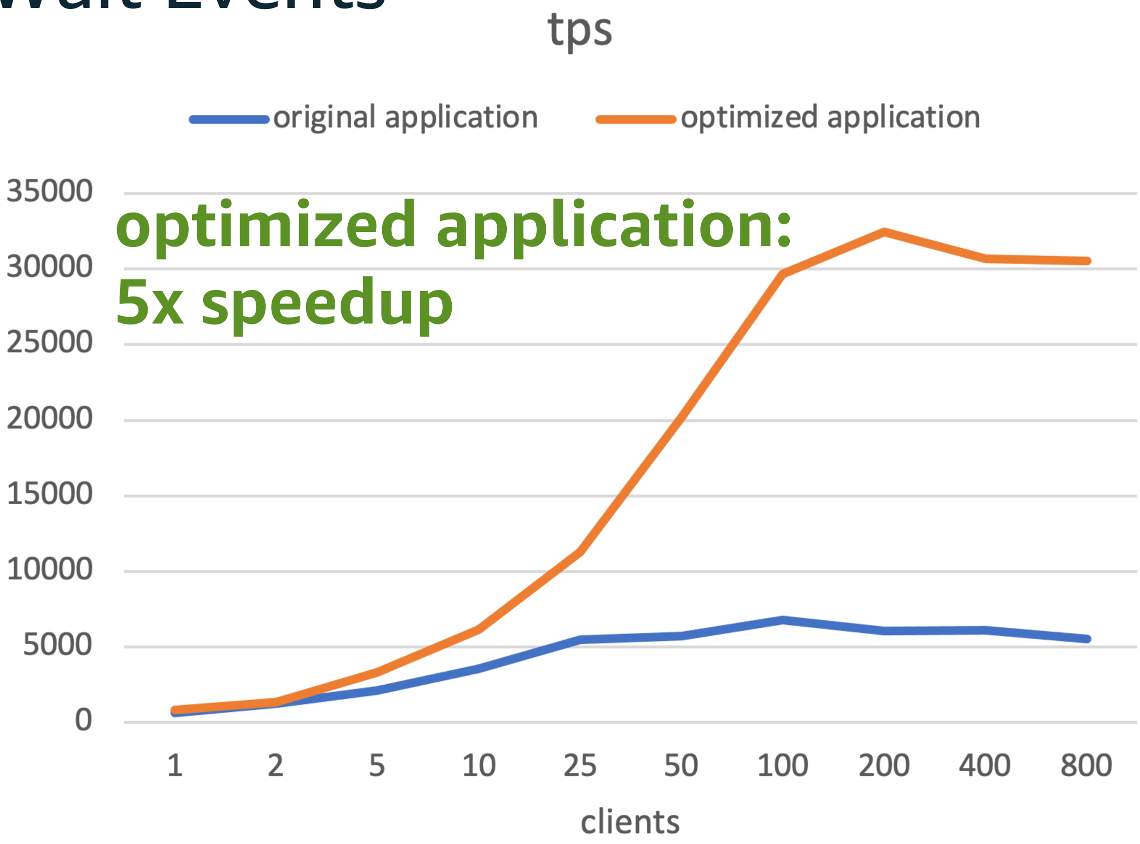
The event `Lock:transactionid` occurs when a transaction is trying to acqui row-level lock that has already been granted to a transaction that is running at same time. The session that shows the `Lock:transactionid` wait event is bl because of this lock. After the blocking transaction ends in either a `COMMIT` or `ROLLBACK` statement, the blocked transaction can proceed.

The multiversion concurrency control semantics of RDS for PostgreSQL guarant

The Right Method: Wait Events

```
pgbench-step4-step5-job.sql:

UPDATE pgbench_tellers t
  SET tbalance = (SELECT sum(h.delta)
                  FROM pgbench_history h
                  WHERE h.tid=t.tid
                  GROUP BY h.tid)
;
UPDATE pgbench_branches b
  SET bbalance = (SELECT sum(a.abalance)
                  FROM pgbench_accounts a
                  WHERE a.bid=b.bid
                  GROUP BY a.bid)
;

pgbench --initialize --scale=10

pgbench --no-vacuum --client=1 --rate=1
        --progress=5 --time=9999
        --file=pgbench-step4-step5-job.sql
```

```
for CLIENT in 1 2 5 10 25 50
              100 200 400 800; do

  pgbench --client=$CLIENT
          --progress=2 --time=30
          --builtin=simple-update

done 2>&1 | tee benchmark-2.log


egrep '(clients:|progress: 30)'
      benchmark-2.log | paste - -
```



optimized application:
5x speedup

The Right Method: Wait Events

Midjourney: May 2023 Incident

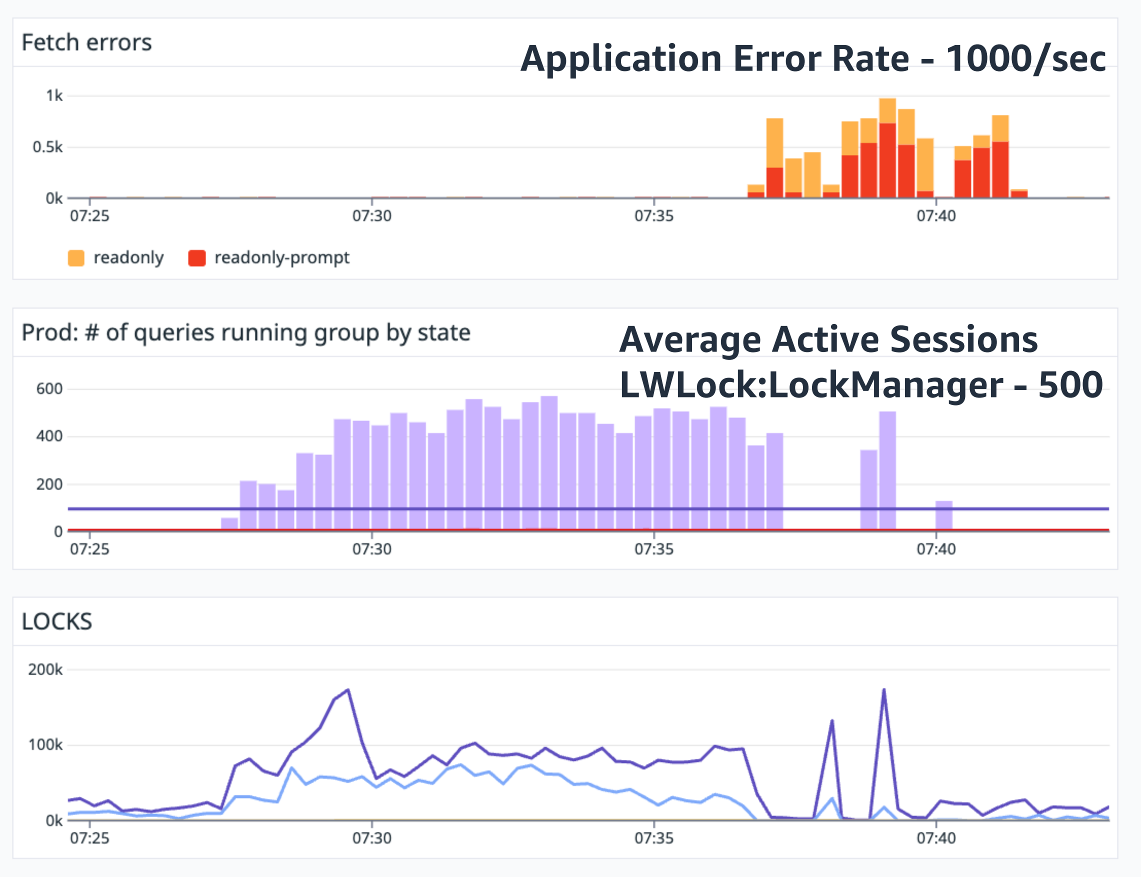
8 TB non-partitioned table
8,000-10,000 QPS
Outbound Logical Replication

Four weeks after minimal downtime migration to a partitioned table

Two incidents (4 days apart) of critically elevated application error rates, caused by severe & sudden database performance degradation

AAS & Wait Events History used to quickly identify contention point, greatly accelerating mitigation.

source: www.kylehailey.com



Fetch errors

Application Error Rate - 1000/sec

Prod: # of queries running group by state

Average Active Sessions
LWLock:LockManager - 500

LOCKS

More Scenarios

CPU

- Overall rate of work: Review SQL execution plans, check for plan flips and optimize total blocks accessed.

DataFileRead, BufferIO

- I/O Read Path: Review SQL execution plans, check for plan flips and optimize total blocks accessed.

WALWrite

- I/O Write Path: Check commit rate, volume of change.

For more info, RDS docs on wait events

transactionid, relation, etc.

- Hot records: check top SQL or pg_locks during contention, review application flow of updating records.

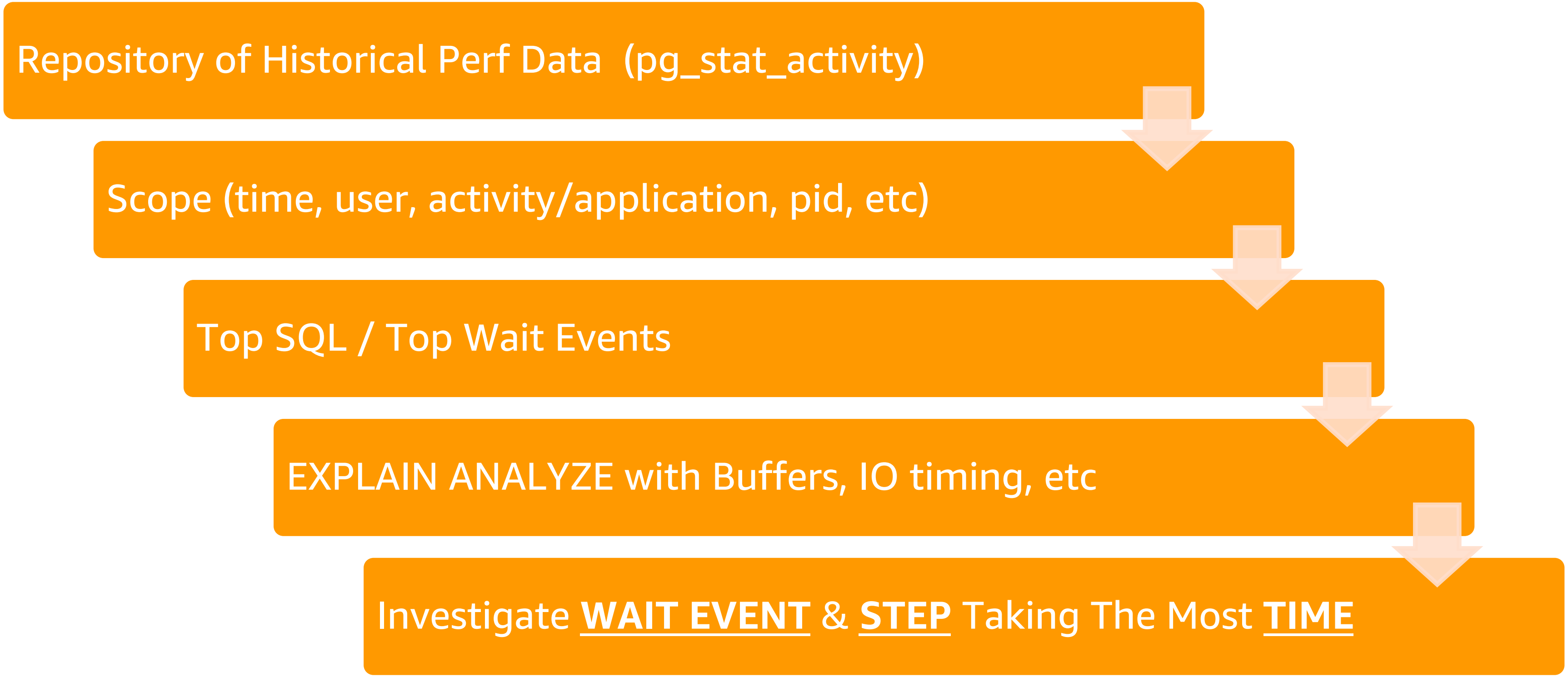
BufferContent

- Hot Block in Memory: check foreign keys, optimize contention (e.g. schema redesign, fillfactor, etc).

LockManager

- Lock System High Pressure: Check total number of indexes and partitions involved in tables used by the query, reduce query execution rate, use replicas

Solving Problems With Wait Events



Repository of Historical Perf Data (pg_stat_activity)

Scope (time, user, activity/application, pid, etc)

Top SQL / Top Wait Events

EXPLAIN ANALYZE with Buffers, IO timing, etc

Investigate **WAIT EVENT** & **STEP** Taking The Most **TIME**

PostgreSQL Happiness Hints

version:
jer_s/2022-04-26

Checksums and Huge Pages Enabled

Connection Pooling

- Centralized (e.g. pgbouncer) and decentralized (e.g. JDBC) architectures
- Recycle server connections (e.g. server_lifetime)
- Limit or avoid dynamic growth when practical – queue at a tier above the DB

Default Limits: Temp Usage, Statement & Idle Transaction Timeout

- Timeouts 5-15 minutes or lower, increase at session level if needed

Scaling

- Measure conn count in hundreds (not thousands), table count in thousands (not hundreds of thousands), relation size in GB (not TB), indexes per table in single digits (not double digits)
- Higher ranges work, but often require budget for experienced & expensive PostgreSQL staff
- To scale workloads, shard across instances or carefully partition tables

Updates and Upgrades

- PostgreSQL quarterly stable “minors” = security and critical fixes only
  - On Aurora: minors can have new development work
- Before major version upgrade, compare plans and latencies of top SQL on upgraded test copy
- Remember to upgrade extensions; it’s not automatic
- Stats/analyze after major version upgrade

Logging

- Minimum 1 month retention (on AWS: use max retention and publish to Cloudwatch)
- Log autovacuum minimum duration = 10 seconds or lower
- Log lock waits
- Log temp usage when close to the default limit
- On AWS: autovacuum force logging level = WARNING

Multiple Physical Data Centers (= Multi-AZ on AWS)

Physical Backups

- Minimum 1 month retention
- Regular restore testing

Logical Backups (at least one)

- Scheduled exports/dumps and redrive/replay
- Logical replication

Active Session Monitoring (= Performance Insights on AWS)

- Save snapshots of pg_stat_activity making sure to include wait events
- Keep historical data, minimum 1 month retention (hopefully much more)

SQL and Catalog and Other Database Statistics Monitoring

- Preload pg_stat_statements
- Save snapshots of pg_stat_statements and key statistics
  - Exec plans (eg. auto_explain or others), relation sizes (bytes & rows incl catalogs), unused indexes
  - Rates: tuple fetch & return, WAL record & fpi & byte, DDL, XID, subtransaction, multixact, conn
- Keep historical data, minimum 1 month retention (hopefully much more)

OS Monitoring (= Enhanced Monitoring on AWS)

- Granularity of 10 seconds or lower (1 second if possible)
- Keep historical data, minimum 1 month retention (hopefully much more)

Alarms

- **Average active sessions** (= dbload cloudwatch metric on AWS)
- Memory / swap
- Disk space: %space and %inodes (and free local storage on Aurora)
- Hot standby & logical replication lag / WAL size (disk space) on primary
- Unexpected errors in the logs, both database and application tier
- Maximum used transaction IDs (aka time to wraparound)
- Checkpoint: time since latest & warnings in log (doesn’t apply to Aurora)

Thank you!

aws.amazon.com/rds/postgresql