Moderne Datenbankentwicklung

Tools & Konzepte

Thomas Koch

Moderne Software Entwicklung

Implementierung

- Coding Style
- Patterns
- Versionierung

Test

- Test – Framework
- Automatisierung

Monitoring

- Icinga / Nagios
- Logs

Deployment

- Continuous Integration
- Continuous Delivery

Betrieb

- Infrastruktur
- Konfiguration
- Wartung

Dokumentation

- Code – Doku
- Wiki – Doku
- Automatisierung

Security

- Benutzer
- Sicherheitslücken

Moderne Datenbank Entwicklung

Implementierung

- Coding Style
- Patterns
- Versionierung

Test

- Test – Framework
- Automatisierung

Monitoring

- Icinga / Nagios
- Logs

Deployment

- Continuous Integration
- Continuous Delivery

Betrieb

- Infrastruktur
- Konfiguration
- Wartung

Dokumentation

- Code – Doku
- Wiki – Doku
- Automatisierung

Security

- Benutzer
- Sicherheitslücken

Bewertungskatalog

Kategorie	Bewertung
Implementation	3
Test	3
Security	4
Deployment	1
Betrieb	3
Monitoring	1
Dokumentation	2
Gesamt	**2**

- Überblick über Zustand der Datenbank
- Ableiten von weiteren Maßnahmen
- Nachverfolgung & Bericht
- Unterteilung nach Kategorien
- Bewertung
  - 0 bis 5 Punkte je Stichpunkt einer Kategorie
  - Durchschnitt je Kategorie

Security

- Schema public und Rolle public
- Rollen
  - Unterscheidung Entwickler, Admins und Software
  - Gut sind Rollen für Reader, Editor, Admins und Superadmins
- Benutzung von (NO)INHERIT
- Owner von Tabellen (und anderen Objekten)
- pg_hba.conf – vermeiden von
  - trust
  - md5 (und Passwort im Code & Versionskontrolle)
  - All (databases & IP address)

Implementierung

- Coding Convention
  - Einrückungen
  - Groß/Kleinschreibung
  - Namen
  - Template für Funktionen
  - Verwendung von NULL
  - Datum statt BOOLEAN (manchmal)
- DB-Schema
  - Umgang mit Schema public
  - Verteilung DB-Objekte je Schema
  - Trennung nach Komponenten
    (Export, Kunden- und Produktverwaltung usw.)

    ODER / UND

    Art der Daten (Stammdaten, OLTP, OLAP, Archivdaten usw.)

Implementierung

- Pattern (Konzepte)
  - Audit-Spalten → create_dt, change_dt, change_info
  - Historisierung (Protokollierung)
  - Konstanten (Magic Numbers/Strings)

    ```
    country_id = 12 → 'Deutschland'::Country::INT
    ```
  - Views & Funktionen als Schnittstelle
  - Prozesstabellen

Test

- Auswahl eines passenden Test-Frameworks → pgTAP
- Test von
  - Struktur → Existenz von Tabellen, Funktionen, Constraints …
  - Logik → Arbeitsweise von Funktionen, Trigger, Views ...
  - Performance
- Ausführung auf verschiedenen Umgebungen
  - Dev (meist local) → alle Tests
  - Test → alle Tests
  - Staging → Struktur und Performance Tests
  - Live → Struktur Test
- Siehe Vortrag „Teste die Datenbank!“ (letztes Jahr)

3 Rules for Database Work

1. Verwende niemals eine gemeinsame Datenbank für die Entwicklung im Team.
2. Habe immer eine einzig maßgebliche Quelle für das Schema.
3. Versioniere immer die Datenbank.

Quelle:
http://odetocode.com/blogs/scott/archive/2008/01/30/three-rules-for-database-work.aspx
http://www.heise.de/developer/artikel/Kontinuierliche-Datenbankmigration-mit-Liquibase-und-Flyway-1857773.html

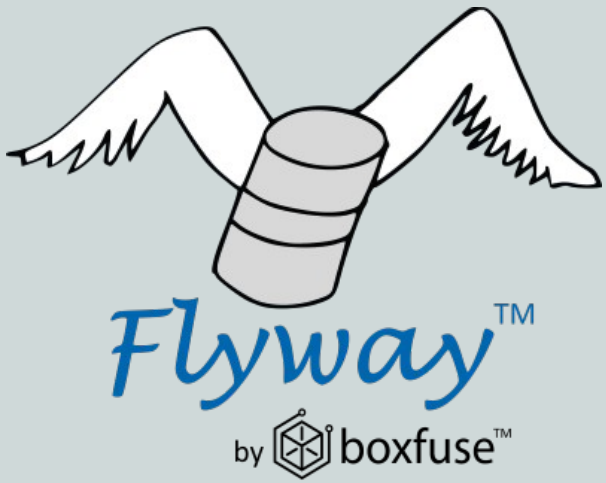
Datenbankversionierung

Versionierung von sämtlichen DB-Code

- Änderungsskripte während der Implementierung
  - Nutzen eines Deployment Tools
  - Review durch DBA / Kollegen
- Regelmäßiger Export der aktuellen DB-Struktur
  - Historie über die Änderung einzelner DB-Objekte
  - Skript / Befehl für den Export in kleine Dateien ?

DB Deployment Tools

- Liquibase
- Flyway
- Dbdeploy
- MigrateDB



Liquibase

- Supports **code branching and merging**
- Supports **multiple developers**
- Supports multiple database types
- Supports XML, YAML, JSON and SQL formats
- Supports **context-dependent logic**
- Generate Database change **documentation**
- Generate Database "diffs"
- Run through your build process, embedded in your application or on demand
- Automatically generate SQL scripts for DBA code review

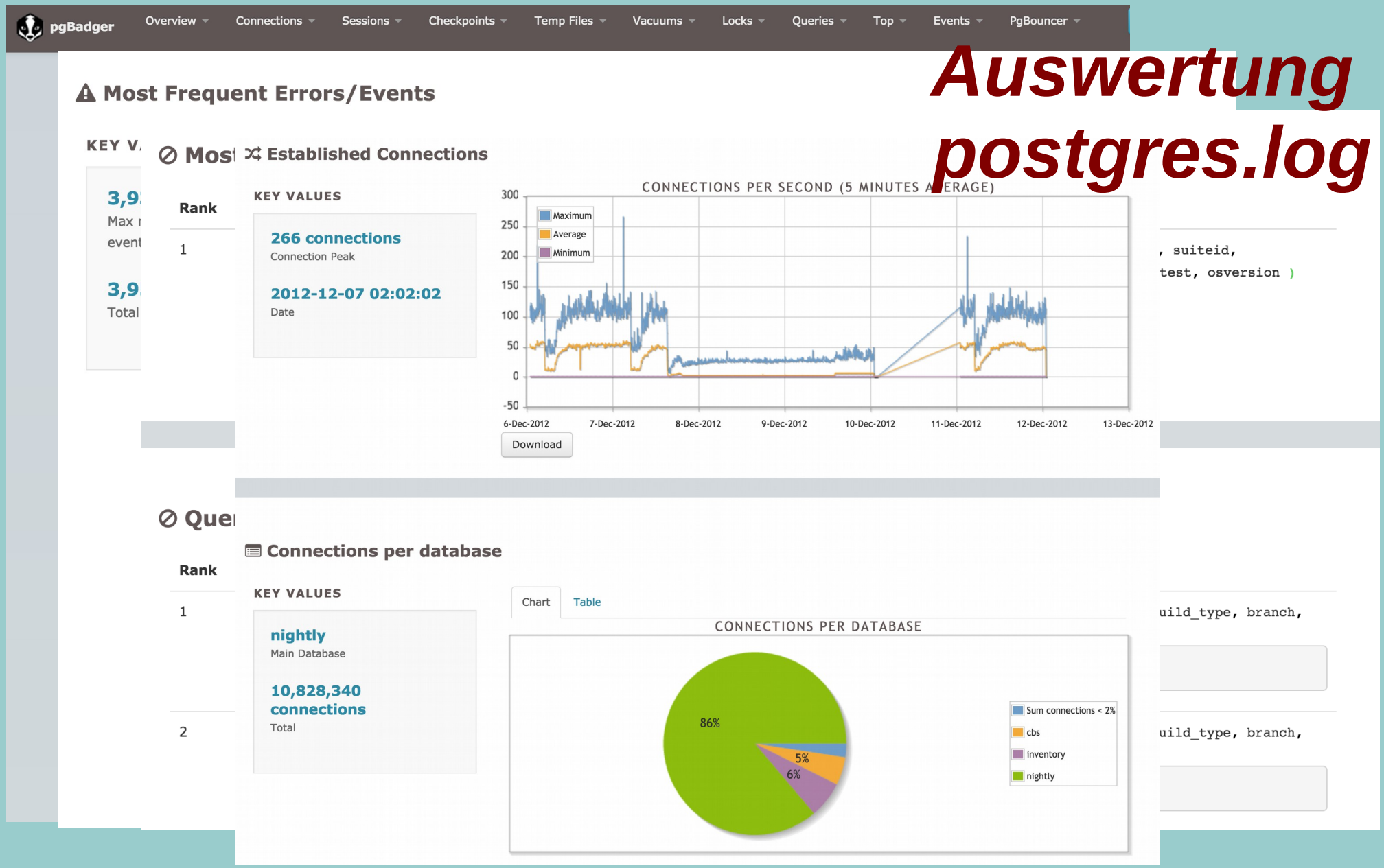
Quelle: http://www.liquibase.org

Deployment mit CI-Tool

- Entwickeln einer Deployment-Pipeline in Jenkins, Bamboo …
  - DB-Code auschecken
  - Neue Änderungsskripe ausführen (Update)
  - Automatisierte Tests ausführen
  - Rollback & Update
- Separat für jede Umgebung (Test & Staging) automatisiert nach jedem Checkin
- Für LIVE wahrscheinlich eher auf Knopfdruck bzw. einzelne Schritte manuell

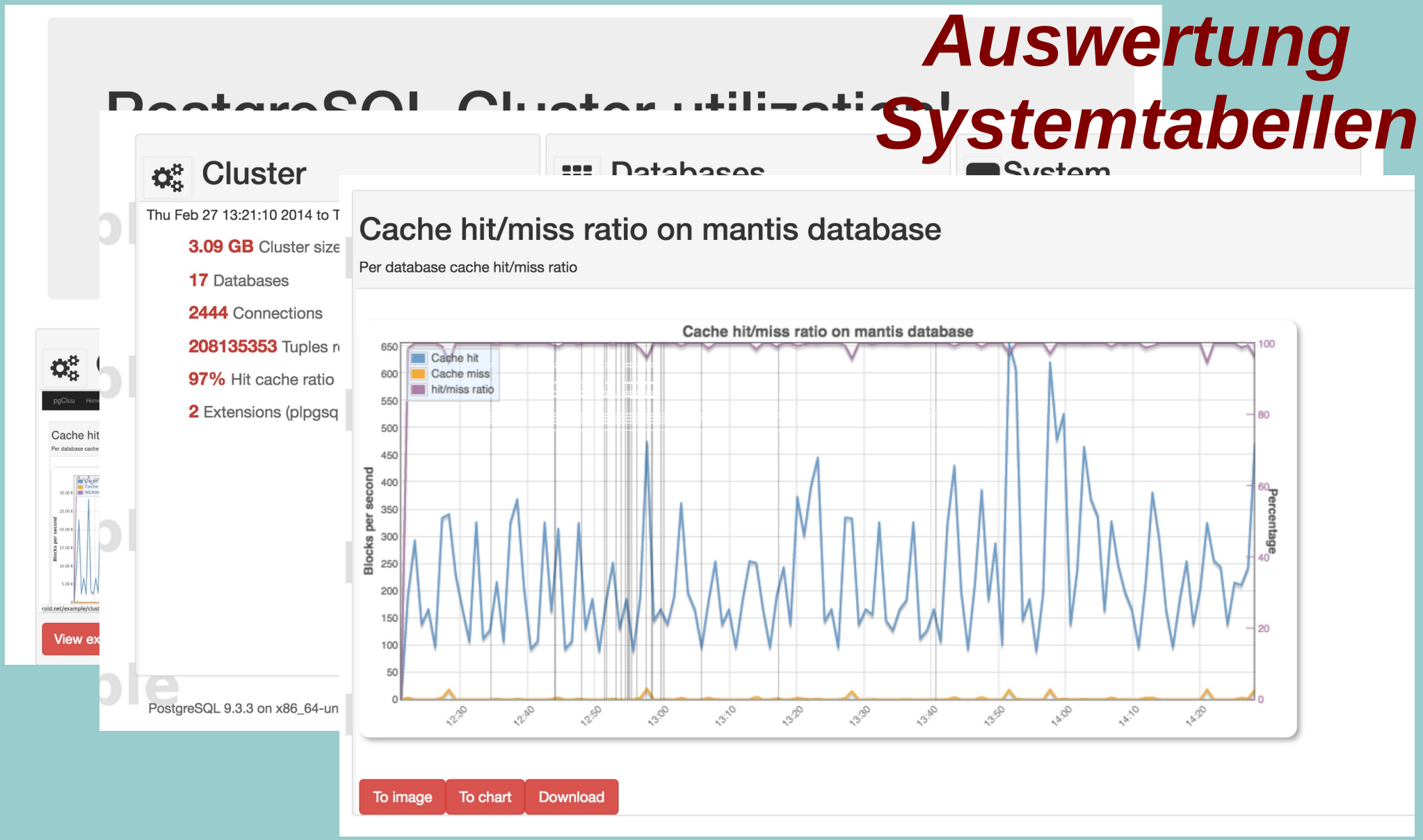
Monitoring - pgbadger

Auswertung postgres.log



Monitoring - pgcluu

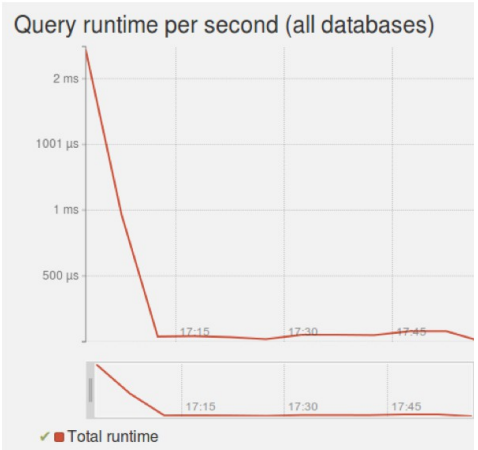
Auswertung Systemtabellen



Monitoring - PoWa

PoWA is PostgreSQL Workload Analyzer that gathers performance stats and provides real-time charts and graphs to help monitor and tune your PostgreSQL servers.

GLOBAL OVERVIEW



Details for all databases

Database	#Calls	Runtime	Avg runtime	Block
pgbench	4415778	1 min 45 s 782 ms	0 ms	323.
powa	8278	2 s 410 ms	0 ms	4.5
obvious	1084	1 min 46 s 94 ms	97 ms	0
postgres	24	12 ms	0 ms	48.

VALIDATE SUGGESTED INDEX

The following indexes would be used:

```
CREATE INDEX ON "public"."command"(id_client)
```

EXPLAIN plan **without** suggested indexes:

```
HashAggregate  (cost=22114.03..22114.15 rows=10 width=13)
  Group Key: com.id
  ->  Hash Join  (cost=1992.53..22113.53 rows=100 width=13)
        Hash Cond: (c_l.id_command = com.id)
        ->  Seq Scan on command_line c_l  (cost=0.00..16370.00
rows=1000000 width=13)
        ->  Hash  (cost=1992.40..1992.40 rows=10 width=8)
              ->  Nested Loop  (cost=0.29..1992.40 rows=10
width=8)
                    ->  Index Only Scan using client_pkey on
client cli  (cost=0.29..6.30 rows=1 width=8)
                          Index Cond: (id = 7026)
                    ->  Seq Scan on command com
(cost=0.00..1986.00 rows=10 width=16)
                          Filter: (id_client = 7026)
```

EXPLAIN plan **with** suggested index

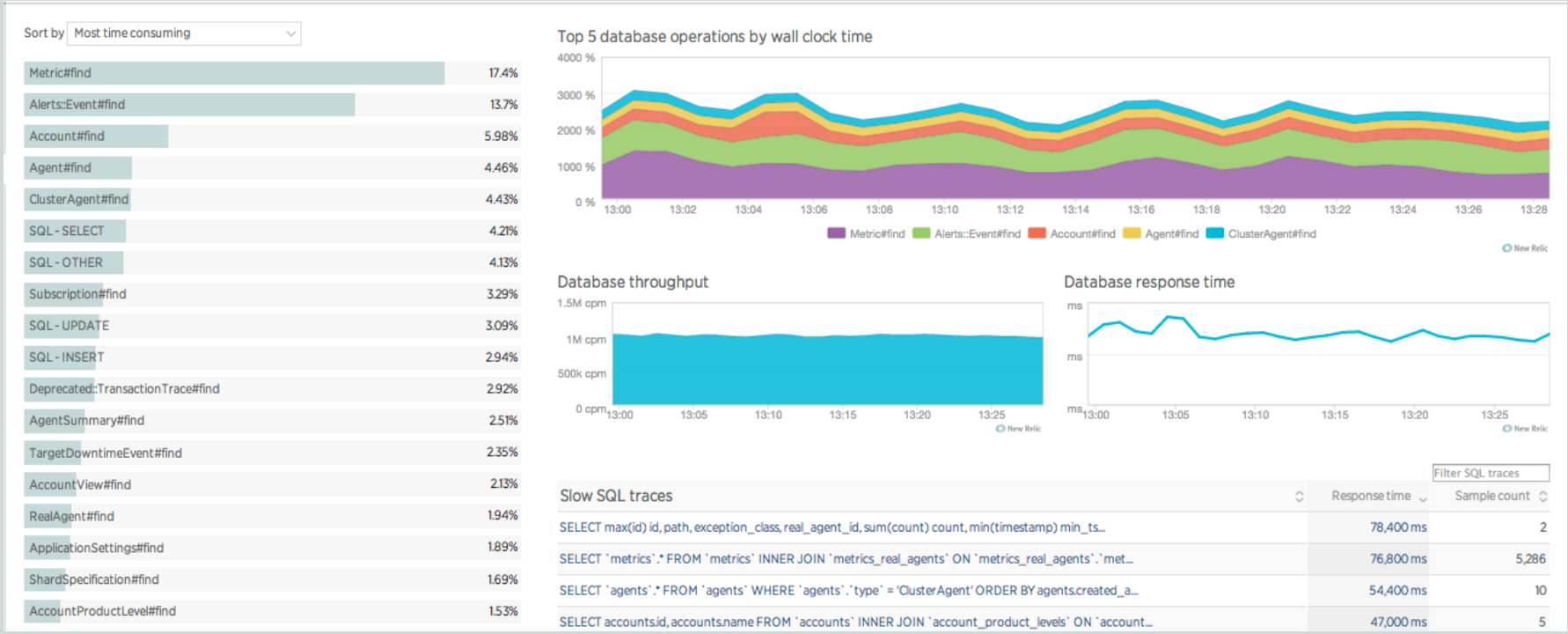
```
HashAggregate  (cost=20159.84..20159.97 rows=10 width=13)
  Group Key: com.id
  ->  Nested Loop  (cost=31.32..20159.34 rows=100 width=13)
        ->  Index Only Scan using client_pkey on client cli
(cost=0.29..6.30 rows=1 width=8)
              Index Cond: (id = 7026)
        ->  Hash Join  (cost=31.04..20152.04 rows=100 width=21)
              Hash Cond: (c_l.id_command = com.id)
              ->  Seq Scan on command_line c_l
(cost=0.00..16370.00 rows=1000000 width=13)
              ->  Hash  (cost=30.91..30.91 rows=10 width=16)
                    ->  Bitmap Heap Scan on command com
(cost=3.12..30.91 rows=10 width=16)
                          Recheck Cond: (id_client = 7026)
                          ->  Bitmap Index Scan on
<259966>btree_command_id_client  (cost=0.00..3.12 rows=10
width=0)
                                Index Cond: (id_client = 7026)
```

Query cost gain factor with hypothetical index: **8.84 %**

Quelle: http://http://dalibo.github.io/powa/

Monitoring

- Plugins für Nagios, Icinga, Zabbix und …
- check_postgres
- New Relic – Application Monitoring (java, .net, php u.a.)



- Weitere unter https://wiki.postgresql.org/wiki/Monitoring

Quelle: https://newrelic.com

Betrieb

- Infrastruktur
  - Virtualisierung
  - CPU
  - Platten
  - RAM
  - Netzwerk
- Backup / Recovery
  - Prozess
  - Und wie oft getestet?
- Replikation
- Hot Standby

Betrieb

- DB-Version / Upgrade-Strategie
  - Major-Versionen
  - Minor-Versionen
- Konfiguration
  - Manuell
  - Über Puppet, Chef oder …
  - Versionierung
- Cronjobs (Wartung)
  - Konfiguration
  - Versionierung
  - Monitoring

Dokumentation

- Mit welchen Tool?
- Zustand & Qualität (Wie aktuell?)
- Inhalt
  - Struktur der Dokumentation (Chaos)
  - DB-Modell
  - Konzepte & Ideen
  - Gründe & Entscheidungen

Zusammenfassung

- Implementierung
  - Schema
  - Coding Convention
  - Konzepte (audit, Konstanten etc.)
- Test
  - Automatisiert mit pgTAP
- Deployment
  - Deployment-Tool
  - CI-Server
- Security
  - User & Rollen

- Betrieb
  - Infrastruktur
  - Backup / Recovery
  - Replikation
  - Hot Standby
- Monitoring
  - Log → pgbadger
  - Systemtabellen → pgcluu
  - real time → PoWa
- Dokumentation
  - DB-Modell
  - Konzepte
  - Gründe & Entscheidungen

Moderne Datenbankentwicklung