Moderne Datenbankentwicklung

Tools & Konzepte

Thomas Koch

Senior Database Specialist

Deutsche Bahn Connect GmbH

Wer bin ich ?

Thomas Koch

- Dipl.-Inf. (FH) & M.Sc.
- Lehrauftrag für Datenbanken & Java an der Beuth Hochschule für Technik Berlin
- Sprecher auf verschiedenen Konferenzen

Arbeitsstellen

- Software Developer
- Database Specialist
- Database & Software Architect

DB, thinkproject!, FONDSNET, SIEMENS, TECHNISCHE FACHHOCHSCHULE BERLIN University of Applied Sciences, GROUPON, searchmetrics Your Web Intelligence, idealo.de, zanox.

Daten und Fakten - wir stellen uns vor!

Mitarbeiter: rd. 520

Mobilitätsdienstleister: rd. 26.800 Fahrzeuge inkl. Partnern und 13.000 Räder

Kunden: deutschlandweit über 1,2 Mio. Endkunden

Hast du es schon live ausgeführt?
Wenn nicht, STOP ...

Kein Problem –
Ich habe ja mein psql.

Schau mal bitte irgendetwas
füllt die Festplatte ...

Ich brauch mal schnell einen Index,
… ist gerade langsam.

Warum ist die Tabelle …
nicht auf meiner Testdatenbank?

Seit wann heißt die
Spalte … anders?

Probleme

- Manuell → ist fehleranfällig
- Nicht nachverfolgbar
- Nicht testbar (automatisiert)
- Kein Monitoring
- Fördert Vergesslichkeit

Kannst du Kollegen …
bitte mal alle Rechte geben.
Er muss dringend … ausführen.

Link: https://grepora.com/2015/06/05/whats-is-the-main-characteristicsskills-of-a-dba-part-2/

Moderne Software Entwicklung

Implementierung

- Coding Style
- Patterns
- Versionierung

Test

- Test – Framework
- Automatisierung

Monitoring

- Icinga / Nagios
- Grafana

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
- Grafana

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)

Security – SET ROLE

```
CREATE ROLE reader WITH NOLOGIN NOSUPERUSER INHERIT;

CREATE ROLE app WITH NOLOGIN NOSUPERUSER INHERIT;
CREATE ROLE app_noinherit WITH NOLOGIN NOSUPERUSER NOINHERIT IN ROLE app;
```

```
CREATE USER web
  WITH LOGIN INHERIT
  ENCRYPTED PASSWORD '...'
  IN ROLE app;

CREATE USER "thomas.koch"
  WITH LOGIN INHERIT
  ENCRYPTED PASSWORD '...'
  IN ROLE reader, app_noinherit;
```

```
-- hat reader Privilegien
SELECT * FROM table;

-- mit app Privilegien
SET ROLE app;
INSERT INTO …

RESET ROLE;

-- wieder "nur" reader Privilegien
```

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 – Audit-Spalten

- Jede Tabelle hat die Spalten created_at, changed_at, changed_info (als JSON)
- Trigger sorgt für Befüllung

```
CREATE FUNCTION tr_set_audit_values() RETURNS trigger
    LANGUAGE plpgsql  AS $$
    DECLARE info TEXT DEFAULT '';
    BEGIN
        IF (TG_OP = 'INSERT') THEN
            NEW.created_at := now();
            IF NEW.changed_info IS NOT NULL AND NEW.changed_info != '{}' THEN
                info := concat('"info":', NEW.changed_info, ',');
            END IF;
        END IF;
        IF (TG_OP = 'UPDATE') THEN
            IF NEW.changed_info != OLD.changed_info AND NEW.changed_info != '{}' THEN
                info := concat('"info":', NEW.changed_info, ',');
            END IF;
        END IF;
        NEW.changed_at := now();
        NEW.changed_info := concat('{', info, '"user":"', SESSION_USER, '",
          "pid":', pg_backend_pid(), '}');
        RETURN NEW;
    END;
    $$;

CREATE TRIGGER triu_tableX_set_audit_values
  BEFORE INSERT OR UPDATE ON tableX FOR EACH ROW EXECUTE PROCEDURE tr_set_audit_values();
```

Implementierung – Historisierung

- Datensatz als JSON
- Erstellung durch eine Funktion
- Historisierung durch Trigger

```
CREATE FUNCTION tr_log_table() RETURNS trigger
   ...
   _sql := format(
     'INSERT INTO log.%s_%s (operation, data) VALUES (%L, %L);',
        TG_TABLE_SCHEMA, TG_RELNAME, TG_OP::log.operation, row_to_json(OLD)
   );
   ...

CREATE FUNCTION create_log(in_schema name, in_table name, VARIADIC in_ignore_columns name[] DEFAULT '{}'::name[])
  RETURNS void
  ...
  BEGIN
  ...
  _sql := format('
     CREATE TABLE log.%I (
       operation log.operation NOT NULL,
       data JSONB NOT NULL
  )
  ', _table);
  EXECUTE _sql;
  ...
  -- Index auf PK der Ursprungstabelle (json-feld) -
  ...
  -- Trigger auf Ursprungstabelle setzen
  _sql := format('
    CREATE TRIGGER trU_%s AFTER UPDATE %s ON %I.%I FOR EACH ROW
      WHEN (OLD.* IS DISTINCT FROM NEW.*) EXECUTE PROCEDURE log.tr_log_table()
    ',
  ...
```

Implementierung – Konstanten

Magic Numbers / Strings gilt als schlechter Programmierstil

→ Konstanten verwenden

```
SELECT name FROM city WHERE country_id = 12;

SELECT name FROM city WHERE country_id IN (
  SELECT country_id FROM country WHERE name LIKE 'Deutschland');

SELECT name FROM city WHERE country_id = 'Deutschland'::country::INT
```

```
CREATE TYPE country AS ENUM (
  'Deutschland', …
);
CREATE FUNCTION cast_constant(in_constant county) RETURNS integer
  LANGUAGE sql IMMUTABLE STRICT
  AS $$
    SELECT
      CASE in_constant
        WHEN 'Deutschland' THEN 12
        ...
      END;
$$;
CREATE CAST (country AS integer) WITH FUNCTION cast_constant(country);
```

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
- Großes Thema für ein anderen Vortrag

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

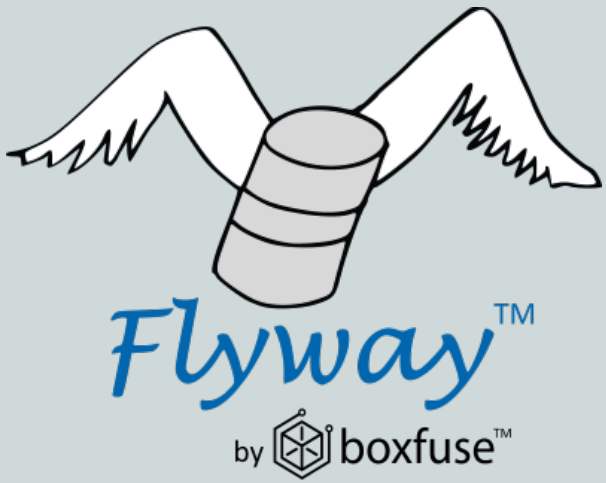
Versionierung von sämtlichen Datenbank-Code

- Änderungsskripte während der Implementierung
  - Nutzen eines Deployment Tools
  - Review durch DBA / Kollegen
- Regelmäßiger Export der aktuellen Datenbank-Struktur
  - Historie über die Änderung einzelner Datenbank-Objekte

```
python3 pg_extractor.py \
        --host=$host --port=$port --username=$user --dbname=$db \
        --schemadir --dbnamedir="./$directory/$db" --schema_exclude="extern" \
        --orreplace --delete --jobs=20 \
        --getall
```

Database Deployment Tools

- Liquibase
- Flyway
- Dbdeploy
- MigrateDB



Liquibase

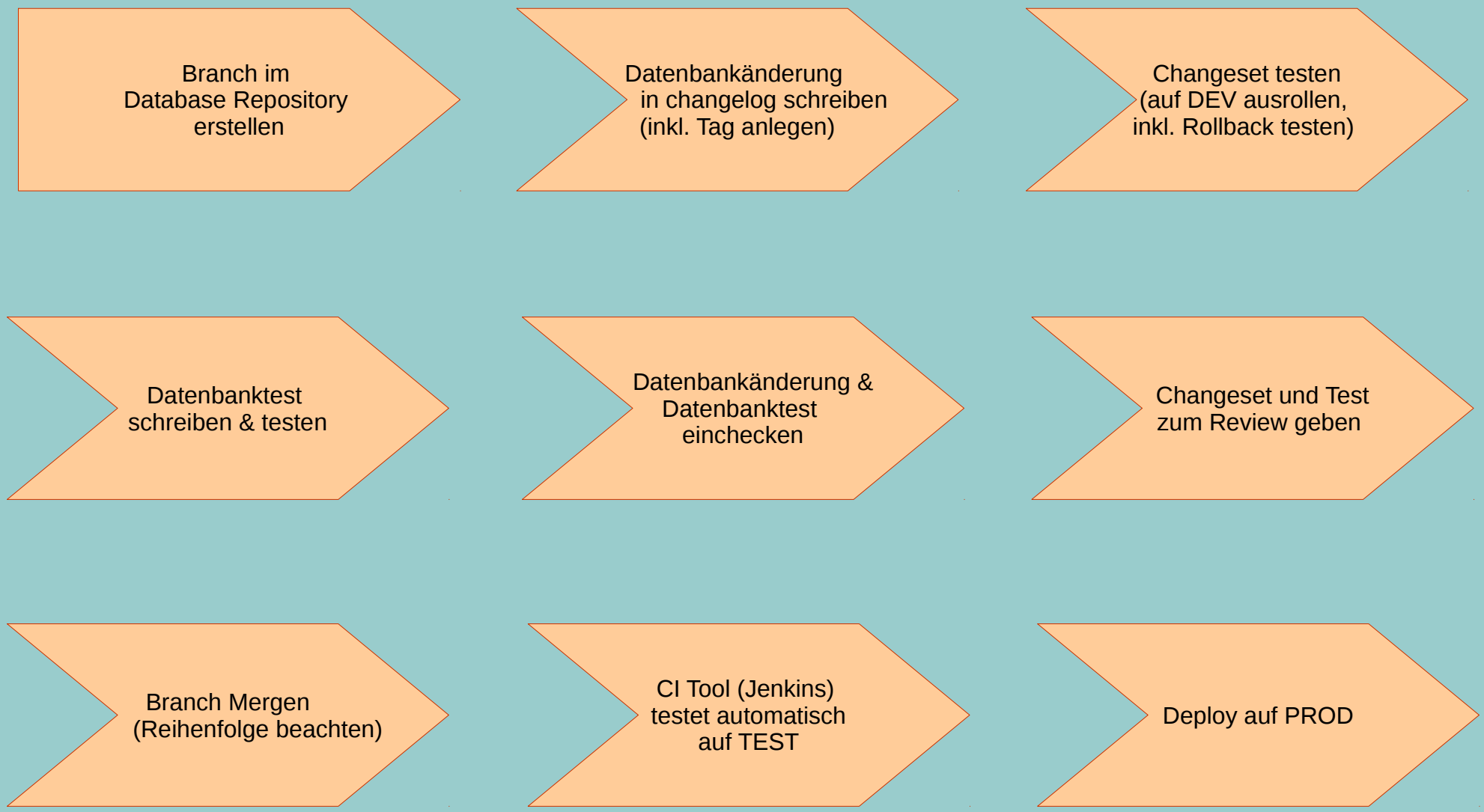
- Supports **code branching and merging**
- Supports **multiple developers**
- Supports multiple database types
- Supports XML, YAML, JSON and SQL formats
- Supports **context-dependent logic**
- Generate Database change **documentation**
- Generate Database "diffs"
- **Run through your build process**, 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 …
  - Datenbank-Code auschecken
  - Neue Änderungsskripte ausführen (Update)
  - Automatisierte Tests ausführen
  - Rollback & Update
- Separat für jede Umgebung (Test & Staging) automatisiert nach jedem Checkin
- Für PROD wahrscheinlich eher auf Knopfdruck bzw. einzelne Schritte manuell

Möglicher Datenbank Workflow



Database Setup Concept

Randbedingung

- Jeder Microservice hat seine eigene Datenbank
  → viele kleine Datenbanken
- Gilt nicht nur für PostgreSQL → für andere RDBMS adaptierbar (z.B. MariaDB)
- Berücksichtigt Anforderungen von DBA und Entwicklerteam

Anforderung DBA

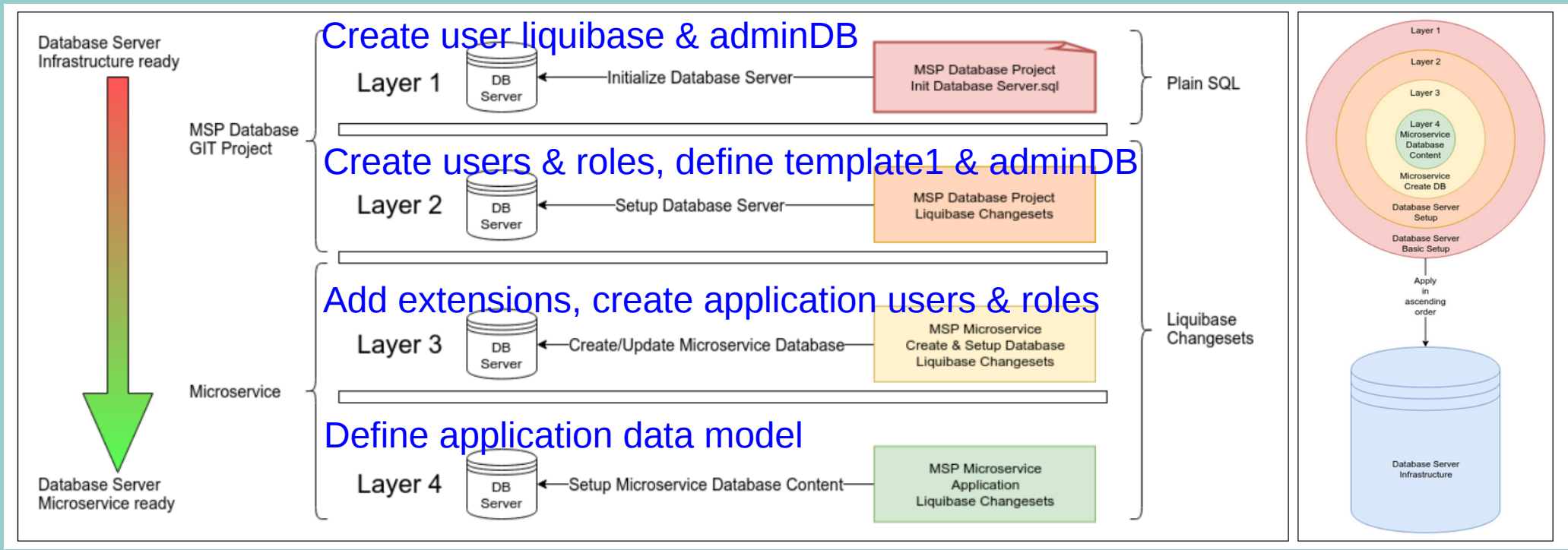
- Überall gleicher Deploymentprozess
- Alle Änderungen sind in Git versioniert
- Alle Skripte an einer definierten Stelle

Anforderung Entwicklerteam

- Microservice definiert Datenbankstruktur (OR-Mapper)
- Docker und Non-Docker fähig
- Unterstützt unabhängige Wegwerf-Test-Datenbanken

Database Setup Concept

- Layer 1 & 2 → Verantwortung DBA
- Layer 3 & 4 → Verantwortung Team & Unterstützung durch DBA
- Docker (für lokale Entwicklung & Test) benötigt nicht Layer 2
- Admin-Datenbank und ein Admin-Schema je Microservice-Datenbank



Monitoring - pgbadger

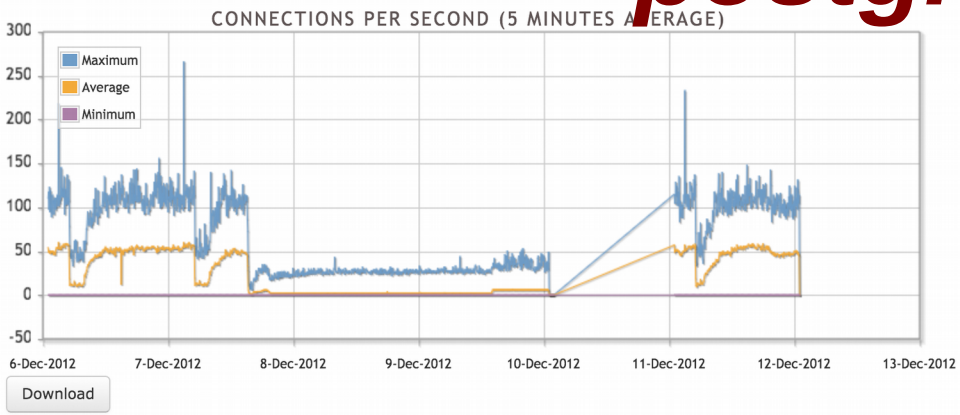
Auswertung postgres.log

pgBadger | Overview | Connections | Sessions | Checkpoints | Temp Files | Vacuums | Locks | Queries | Top | Events | PgBouncer

Most Frequent Errors/Events

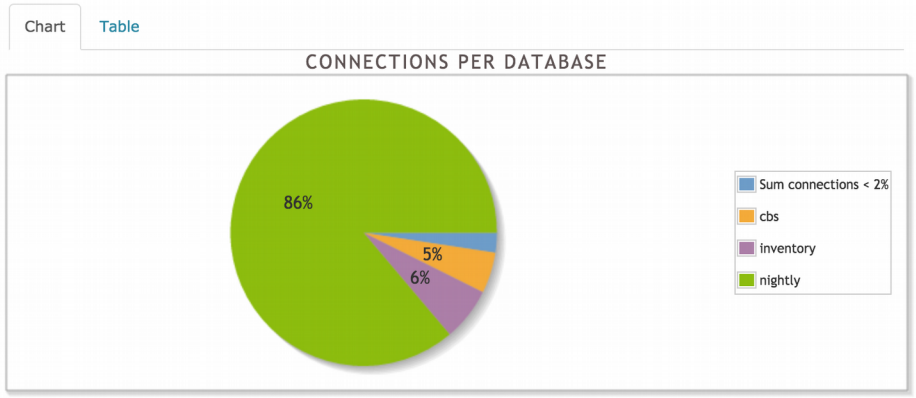
Established Connections

Rank	KEY VALUES
1	**266 connections** Connection Peak **2012-12-07 02:02:02** Date



Connections per database

Rank	KEY VALUES
1	**nightly** Main Database **10,828,340 connections** Total
2	




```
, suiteid,
test, osversion )
```

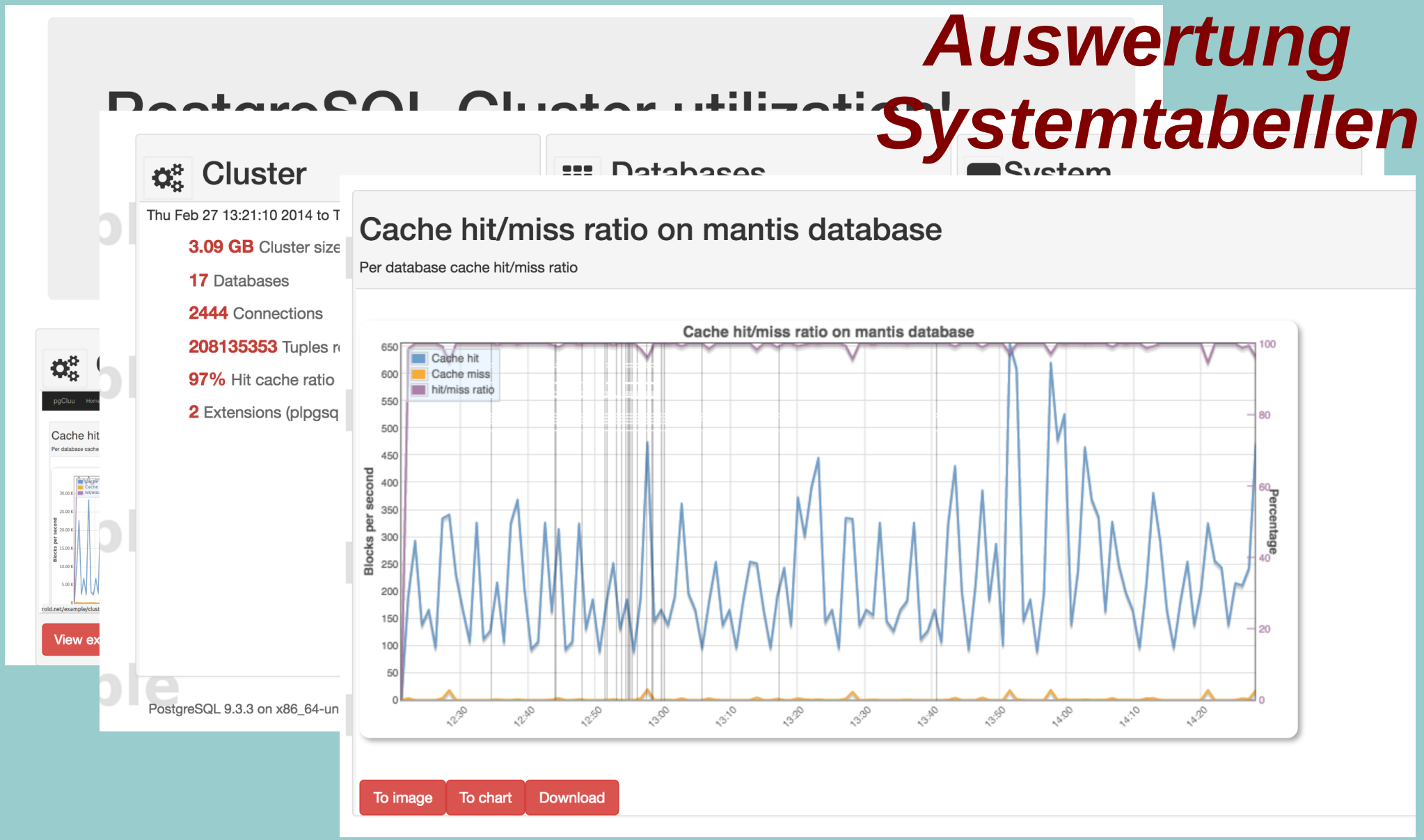
```
uild_type, branch,
```

```
uild_type, branch,
```

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

Monitoring - pgcluu

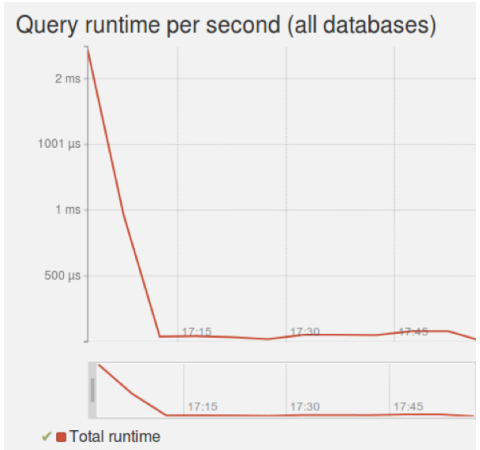
Auswertung Systemtabellen



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

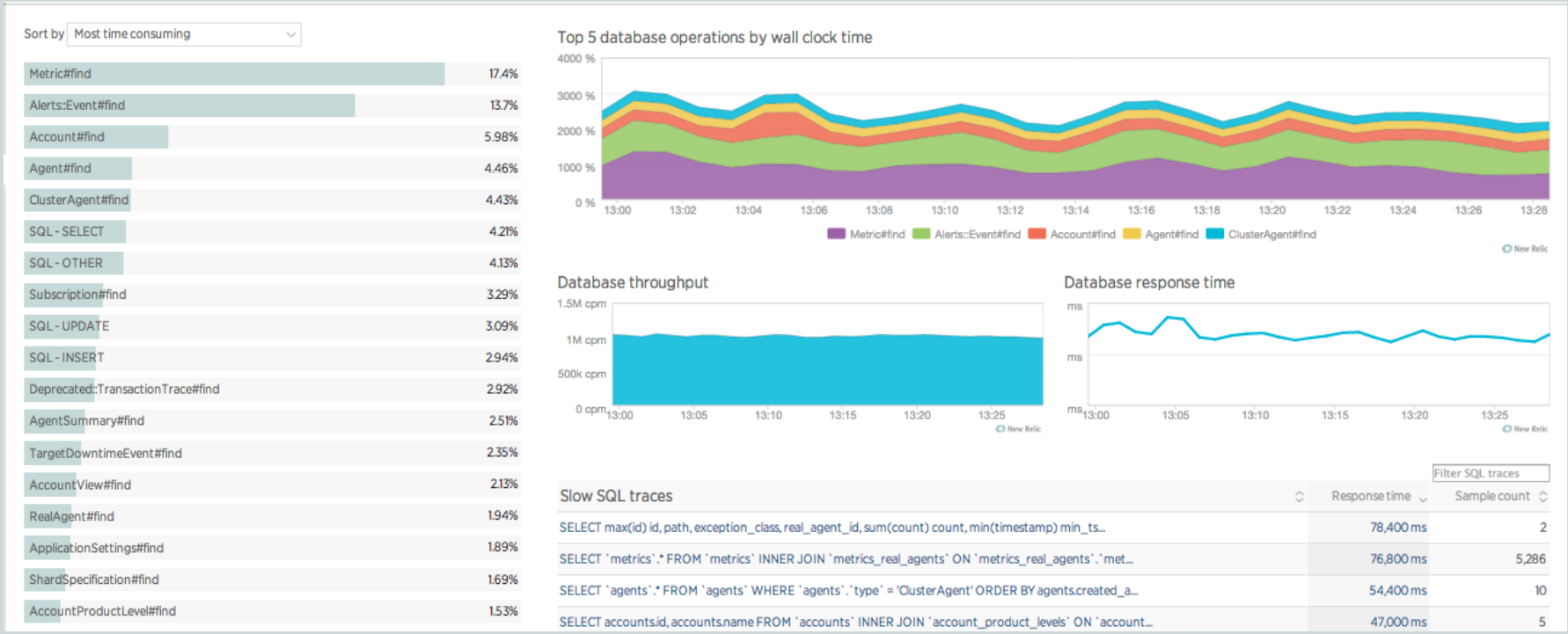
```
HashAggregate  (cost=20159.84..20159.97 rows=10 width=13)
  Group Key: com.id
  ->  Nested Loop  (cost=31.32..20159.34 rows=100 width=13)
        ->  Index Only Scan using client_pkey on client cli
(cost=0.29..6.30 rows=1 width=8)
              Index Cond: (id = 7026)
        ->  Hash Join  (cost=31.04..20152.04 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: https://powa.readthedocs.io/en/latest/

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

Zusammenfassung

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

- Betrieb
  - Infrastruktur
  - Backup / Recovery
  - Replikation
  - pgbouncer
- Monitoring
  - Log → pgbadger
  - Systemtabellen → pgcluu
  - real time → PoWa
- Dokumentation
  - Datenbankmodell
  - Konzepte
  - Gründe & Entscheidungen

Moderne Datenbankentwicklung