Enabling development teams to move fast with PostgreSQL
About me

Valentine Gogichashvili
Head of Data Engineering @ZalandoTech
twitter: @valgog
google+: +valgog
e-mail: valentine.gogichashvili@zalando.de
One of Europe's largest online fashion retailers

- 15 countries
- 3 fulfillment centers
- 15+ million active customers
- 2.2 billion € revenue 2014
- 150,000+ products
- 8,000+ employees
Our tech-offices

More than 800 technologists

ambitious plans:
- +2000 developers by the end of 2016

Technology offices:
- Berlin
- Dortmund
- Dublin
- Helsinki
More numbers

Java, Scala, Clojure, Python, JS, Julia...

PostgreSQL:

- 120+ clusters (master + 2 or more slaves)
- 100+ independent databases
- 10 dedicated database engineers (team ACID)
More numbers

- > 6.0 TB in PostgreSQL databases
- > 100000 TX/sec in average
- plus the biggest database
  - eventlogdb (8TB)
  - 25 GB business events in a week
Biggest challenges

- Constantly growing
- Fast development cycles
- No downtimes are tolerated
Biggest challenges

- Constantly growing
- Fast development cycles
- No downtimes are tolerated
- Autonomous teams
Agenda

How we

- access data
- change data models without downtimes
- monitor and run databases
Agenda

How we

- access data
- change data models without downtimes
- monitor and run databases
Accessing data

- customer
  - bank account
  - order -> bank account
    - order position
  - return order -> order
    - return position -> order position

- financial document
  - financial transaction -> order
Accessing data

ORM

+ is well known to developers
+ CRUD operations are easy
+ all business logic inside your application
+ developers are in their comfort zone

- error prone transaction management
- you have to reflect your tables in your code
- business and data logic inside your application
- schema changes are not easy (versioning)
Accessing data

NoSQL

+ map your object hierarchy to a document
+ (de-)serialization is easy
+ transactions are not needed

− implicit schemas are tricky
− No SQL
Accessing data

Are there alternatives to ORM?

Pure SQL queries

+ return/receive entity aggregates
+ clear transaction scope

Examples

- Clojure YeSQL
- Embedded SQL in Scala Slick
Accessing data

Are there alternatives to ORM?

Stored Procedures

+ return/receive entity aggregates
+ clear transaction scope
+ more data consistency checks
+ independent from underlying data schema
Java Application

JDBC

Database Tables
Java Sproc Wrapper

Java Application
Sproc Wrapper
JDBC

Stored Procedure API
Database Tables
CREATE FUNCTION register_customer(p_email text,
p_gender z_data.gender)
RETURNS int
AS $$
INSERT INTO z_data.customer (c_email, c_gender)
VALUES (p_email, p_gender)
RETURNING c_id
$$
LANGUAGE 'sql' SECURITY DEFINER;
@SProcService
public interface CustomerSProcService {
    @SProcCall
    int registerCustomer(@SProcParam String email,
                         @SProcParam Gender gender);
}

CREATE FUNCTION register_customer(p_email text,
                                    p_gender z_data.gender)
    RETURNS int
AS $$
    INSERT INTO z_data.customer (c_email, c_gender)
    VALUES (p_email, p_gender)
    RETURNING c_id
$$
LANGUAGE 'sql' SECURITY DEFINER;
Java Sproc Wrapper

```java
@SProcService
public interface CustomerSProcService {
    @SProcCall
    int registerCustomer(@SProcParam String email,
                         @SProcParam Gender gender);
}
```

```sql
CREATE FUNCTION register_customer(p_email text,
                                  p_gender z_data.gender)
    RETURNS int
AS $$
    INSERT INTO z_data.customer (c_email, c_gender)
    VALUES (p_email, p_gender)
    RETURNING c_id
$$
LANGUAGE 'sql' SECURITY DEFINER;
```
Java Sproc Wrapper

@SProcCall
List<Order> findOrders(@SProcParam String email);

CREATE FUNCTION find_orders(p_email text,
    OUT order_id int,
    OUT order_created timestamptz,
    OUT shipping_address order_address)
  RETURNS SETOF record
AS $$
    SELECT o_id, o_created,
           ROW(oa_street, oa_city, oa_country)::order_address
    FROM z_data."order"
    JOIN z_data.order_address ON oa_order_id = o_id
    JOIN z_data.customer ON c_id = o_customer_id
    WHERE c_email = p_email
$$
LANGUAGE 'sql' SECURITY DEFINER;
@SProcCall
List<Order> findOrders(@SProcParam String email);

CREATE FUNCTION find_orders(p_email text,
    OUT order_id int,
    OUT order_created timestamptz,
    OUT shipping_address order_address)
    RETURNS SETOF record
AS $$
    SELECT o_id, o_created,
           ROW(oa_street, oa_city, oa_country)::order_address
    FROM z_data."order"
    JOIN z_data.order_address ON oa_order_id = o_id
    JOIN z_data.customer ON c_id = o_customer_id
    WHERE c_email = p_email
$$
LANGUAGE 'sql' SECURITY DEFINER;
Java Sproc Wrapper

Java Application

Sproc Wrapper

One big database
Sharding with Java Sproc Wrapper
Sharding with Java Sproc Wrapper

```java
@SProcCall
int registerCustomer(@SProcParam @ShardKey CustomerNumber customerNumber,
                     @SProcParam String email,
                     @SProcParam Gender gender);
```

```java
@SProcCall
Article getArticle(@SProcParam @ShardKey Sku sku);
```

```java
@SProcCall(runOnAllShards = true, parallel = true)
List<Order> findOrders(@SProcParam String email);
```
Sharding with Java Sproc Wrapper

```java
@SProcCall
int registerCustomer(@SProcParam @ShardKey CustomerNumber customerNumber,
                     @SProcParam String email,
                     @SProcParam Gender gender);
```

```java
@SProcCall
Article getArticle(@SProcParam @ShardKey Sku sku);
```

```java
@SProcCall(runOnAllShards = true, parallel = true)
List<Order> findOrders(@SProcParam String email);
```
Stored Procedures for developers

- CRUD operations need too much code
- No popular tooling for debugging
- You have to learn SQL
- Developers can write bad SQL

+ Use-case driven
+ You have to learn SQL
+ Developers learn how to write good SQL
Stored Procedure

API versioning

search_path = api_v15_01, public;

search_path = api_v15_02, public;

api_v15_01

api_v15_02

Database Tables
Stored Procedure

API versioning

search_path = api_v15_01, public;

search_path = api_v15_02, public;

Database Tables
Stored Procedure

API versioning

+ Tests are done to the whole API version
+ No API migrations needed
+ Deployments are fully automated
Agenda

How we

● access data
● change data models without downtimes
● monitor and run databases
Easy schema changes

PostgreSQL

+ Schema changes with minimal locks with:
  ADD/RENAME/DROP COLUMN
  ADD/DROP DEFAULT VALUE

+ CREATE/DROP INDEX CONCURRENTLY

− Constraints are still difficult to ALTER
Easy schema changes

Stored Procedure API layer

+ Can fill missing data on the fly

+ Helps to change data structure without application noticing it
Easy schema changes

- Read and write to old structure
- Write to both structures, old and new. Try to read from new, fallback to old
- Migrate data
- Read from new, write to old and new
Easy schema changes

Schema changes using SQL script files

- SQL scripts written by developers (DBDIFFs)
- registering DBDIFFs with Versioning
- can be reviewed by DB team members (on request)
- teams are trained to roll DB changes on the live system (trainings, tooling and monitoring tools from DB team)
BEGIN;
    SELECT _v.register_patch('ZEOS-15430.order');

    CREATE TABLE z_data.order_address (
        oa_id int SERIAL,
        oa_country z_data.country,
        oa_city varchar(64),
        oa_street varchar(128), ...
    );

    ALTER TABLE z_data."order" ADD o_shipping_address_id int 
    REFERENCES z_data.order_address (oa_id);
COMMIT;
BEGIN;
    SELECT _v.register_patch('ZEOS-15430.order');

\i order/database/order/10_tables/10_order_address.sql

ALTER TABLE z_data."order" ADD o_shipping_address_id int
    REFERENCES z_data.order_address (oa_id);
COMMIT;
BEGIN;
    SELECT _v.register_patch('ZEOS-15430.order');

    \i order/database/order/10_tables/10_order_address.sql

    SET statement_timeout TO '3s';

    ALTER TABLE z_data."order" ADD o_shipping_address_id int REFERENCES z_data.order_address (oa_id);

    COMMIT;
Easy schema changes

**pg_view**  [https://github.com/zalando/pg_view](https://github.com/zalando/pg_view)
- helps to monitor locks and load in real-time
- used during all DB schema change rollouts

**nice_updater**  [https://github.com/zalando/acid-tools](https://github.com/zalando/acid-tools)
- runs big migrations controlling database/system load
- used during automatic data migrations
Easy schema changes

No downtime due to migrations or deployment since we started using PostgreSQL
Agenda

How we

● access data
● change data models without downtimes
● monitor and run databases
https://github.com/zalando/pg_view

```
postgres@z-integrationdb:~$  134x29
Z-integrationdb up 117 days, 10:00:47 32 cores Linux 3.2.0-49-generic load average 0.73 0.63 0.51  22:10:13
sys: utime 1.7 stime 0.3 idle 98.0 iowait 0.0  ctxt 2500  run 3     block 0
mem: total 251.9GB free 34.0GB buffers 1.6GB cached 196.4GB dirty 10.0MB limit 127.8GB as 33.6GB left 94.2GB
integration93 9.3 database_connections: 0 of 800 allocated, 2 active

<table>
<thead>
<tr>
<th>type</th>
<th>dev</th>
<th>fill</th>
<th>total</th>
<th>left</th>
<th>read</th>
<th>write</th>
<th>await</th>
<th>path</th>
<th>size</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>mapper/vg01-data1</td>
<td>0.0</td>
<td>2.2TB</td>
<td>1.3TB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xlog</td>
<td>sda9</td>
<td>0.0</td>
<td>119.9GB</td>
<td>88.5GB</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
<td>64.0MB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

pid  type  s   utime  stime  guest  read  write  age  db       user  query
4595 backend S  0.0  0.0   0.0   0.0  0.01:37 integr..org db vgogichashvili idle in transaction
5019 backend S  0.0  0.0   0.0   0.0  0.01:02 integr..org db vgogichashvili select 'Cool tool' from pg_sleep(19000);
```
# PGOObserver

### perftables/

**Possible Table access/growth issues report**

<table>
<thead>
<tr>
<th>Host</th>
<th>Schema</th>
<th>Table</th>
<th>Date</th>
<th>Scan change %</th>
<th>Scans1</th>
<th>Scans2</th>
<th>Size1</th>
<th>Size2</th>
</tr>
</thead>
<tbody>
<tr>
<td>bm.db.zalando</td>
<td>zbm_data</td>
<td>sales_rule_set</td>
<td>2013-10-28</td>
<td>84.21 (50)</td>
<td>38</td>
<td>70</td>
<td>2022 MB</td>
<td>2022 MB</td>
</tr>
</tbody>
</table>

### perfindexes/

**Possible Index issues report**

<table>
<thead>
<tr>
<th>Host</th>
<th>Table name</th>
<th>Index name</th>
<th>Index size</th>
<th>% of table’s indexes</th>
<th>Table size</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog1.db.zalando</td>
<td>zcat.data.article_config</td>
<td>zcat.data.article_config_c1_c2_null_null_null_idx</td>
<td>0 bytes</td>
<td>0.0% of 87 MB</td>
<td>42 MB</td>
</tr>
</tbody>
</table>

**Invalid indexes (in total size of 0 bytes)**

**Duplicate indexes**

<table>
<thead>
<tr>
<th>Host</th>
<th>Table name</th>
<th>Table size</th>
<th>Index definition</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog1.db.zalando</td>
<td>zcat.data.article_simple</td>
<td>171 MB</td>
<td>CREATE INDEX X ON zcat_data.article_simple USING btree (as_simple_sku_id)</td>
<td>2</td>
</tr>
<tr>
<td>catalog1.db.zalando</td>
<td>zcat.data.article_config</td>
<td>130 MB</td>
<td>CREATE INDEX X ON zcat_data.article_config USING btree (ac_config_sku_idt)</td>
<td>2</td>
</tr>
<tr>
<td>catalog1.db.zalando</td>
<td>zcat.data.article_model</td>
<td>47 MB</td>
<td>CREATE INDEX X ON zcat_data.article_model USING btree (am_model_sku_idt)</td>
<td>2</td>
</tr>
<tr>
<td>catalog1.db.zalando</td>
<td>zcat.commons.size</td>
<td>12 MB</td>
<td>CREATE INDEX X ON zcat.commons.size USING btree (s_size_chart_code, s_code)</td>
<td>2</td>
</tr>
<tr>
<td>catalog1.db.zalando</td>
<td>zcat.commons.price_level</td>
<td>96 kB</td>
<td>CREATE INDEX X ON zcat.commons.price_level USING btree (pl_level)</td>
<td>2</td>
</tr>
</tbody>
</table>

**Unused indexes**

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Table name</th>
<th>Index name</th>
<th>Scans</th>
<th>Index size</th>
<th>% of table’s indexes</th>
<th>Table size</th>
</tr>
</thead>
</table>
https://github.com/zalando/PGObserver
https://github.com/zalando/ZMON
Patroni & Spilo

Patroni - High Availability Runner
- etcd or ZooKeeper for master election

Spilo - PostgreSQL AWS appliance
- Zalando Patroni for High Availability
- Docker for packaging
- Zalando STUPS for audit compliance
Thank you!
Links

SProcWrapper – Java library for stored procedure access
github.com/zalando/java-sproc-wrapper

PGObserver – monitoring web tool for PostgreSQL
github.com/zalando/PGObserver

pg_view – top-like command line activity monitor
github.com/zalando/pg_view

Patroni & Spilo – High Availability Runner and AWS appliance for PostgreSQL

ZMON monitoring tool
https://github.com/zalando/zmon

STUPS.io Zalando PaaS platform for AWS
https://github.com/zalando-stups
Where to Find Us:

Tech Blog: tech.zalando.com
GitHub: github.com/zalando
Twitter: @ZalandoTech
Instagram: zalandotech
Jobs: http://tech.zalando.com/jobs