Stored procedure wrapper for Java & PGObservable
Outline

• Introduction
• Stored procedure wrapper
  – Problems before the wrapper
  – How it works
  – How to use it
  – More features including sharding
• PGOObserver
Zalando

- 14 countries
- 471 Mio € revenue
- 1st half of 2012
- 3 warehouses
- Europe's largest online fashion retailer

tech.zalando.org
Zalando platform

- Modern open source software stack
- Mostly Java
- PostgreSQL database backend
- > 150 developers

[Link to Tech.Zalando.org](http://tech.zalando.org)
PostgreSQL setup

- ~ 20+ Servers PostgreSQL master servers
- ~ 4,000 GB of data
- Started with PostgreSQL 9.0 rc1
- Now running version 9.0 to 9.2
  - cascading replication very welcome
  - maintenance improvements great (drop concurrently)
  - Index only scan, pg_stat_statement improvements

Machine setup

- 8- to 48- cores, 16GB to 128GB
- SAN, no SAN with (2x2x RAID 1, 4x RAID 10) preferred
PostgreSQL availability

- BoneCP as Java connection pool
- All databases use streaming replication
  - Service IP for switching
- Failover is manual task
  - Monitored by Java app, Web frontend
- Significant replication delays sometimes
  - Fullpage writes, Nested Transactions, Slave load
Stored procedure experience

- Performance benefits
- Easy to change live behavior
- Makes moving to new software version easy
- Validation close to data
- Run a very simplistic transaction scope
- Cross language API layer
- More than 1000 stored procedures
  - More plpgsql than SQL than plpython
Outline

• Introduction
• Stored procedure wrapper
  – Problems before the wrapper
  – How it works
  – How to use it
  – More features including sharding
• PGOObserver
Execution of stored procedures

- Using spring's BaseStoredProcedure
  - Initially a lot of work per stored procedure
  - One class per stored procedure
  - Write row mappers for domain object mapping

- Missing type mapper on Java side
  - Spring type mapper insufficient
    - Enums, array of types, nesting, and hstore missing

- JdbcTemplate or alternatives lack ease of use
Goals of our wrapper

● Write as little code as possible on Java side

● One location for procedures of same topic

● One call path to any stored procedure

● “Natural” feeling for using stored procedures
  - Procedure call should look like Java method call
  - RPC like
Brief example

```sql
CREATE OR REPLACE FUNCTION create_customer ( p_customer t_customer )
    RETURNS SETOF t_customer
AS
$$
   -- Procedure definition
$$
LANGUAGE 'plpgsql' SECURITY DEFINER;
```
Brief example

CREATE OR REPLACE FUNCTION create_customer ( p_customer t_customer )
  RETURNS SETOF t_customer
AS
$$
  -- Procedure definition
$$
  LANGUAGE 'plpgsql' SECURITY DEFINER;

@SProcService
public interface CustomerExampleSProcService {

  @SProcCall
  Customer createCustomer(@SProcParam Customer customer);
}
Brief example

```java
@SProcService
public interface CustomerExampleSProcService {
    @SProcCall
    Customer createCustomer(@SProcParam Customer customer);
}
```

```java
Customer c = new Customer();
c.setFirstName("Jan");
c.setName("Name");
Customer result = service.createCustomer(c);
```
Outline

• Introduction

• Stored procedure wrapper
  – Problems before the wrapper
  – **How it works**
  – How to use it
  – More features including sharding

• PGObservable
Under the hood

Service Object

Invoke method()

Proxy Object

.StoredProcedure lookup

.StoredProcedure Object

Datasource Provider

Typemapper

Datasource

JDBC Connection
Features

• New spring compatible type mapper
  – From simple types to nested domain objects
  – Supports PG enum to Java enum

• Accessing sharded data supported
  – Result “aggregation” across shards
  – Parallel query issuing

• Advisory locking via annotation

• Set custom timeout per stored procedure
Type mapper

- Annotations for class and member variables
  - `@DatabaseType` and `@DatabaseField`
- CamelCase to camel_case conversion
- JPA 2.0 `@Column` annotation supported
- Addition type conversions include:
  - Nested PostgreSQL types to Java objects
  - hstore to Map<String, String>
  - PostgreSQL enum to Java enum (by name)
  - PostgreSQL array[] to List<?>()
Outline

- Introduction
- Stored procedure wrapper
  - Problems before the wrapper
  - How it works
  - How to use it
  - More features including sharding
- PGObservable
Using the wrapper

• Considere Java to PostgreSQL plpgsql
• First define the Java interface

```java
@SProcService
public interface CustomerExampleSProcService {

    @SProcCall
    public Customer loadCustomer(@SProcParam int id);

    @SProcCall
    public Customer createCustomer(@SProcParam Customer customer);

    @SProcCall
    public int addNewAddress(@SProcParam int customerId,
                              @SProcParam Address address);
}
```
Using the wrapper

- Create class implementing previous interface

```java
@Repository
public class CustomerExampleSProcServiceImpl
    extends AbstractSProcService<CustomerExampleSProcService,
    SingleDataSourceProvider>
    implements CustomerExampleSProcService {

    @Autowired
    public CustomerExampleSProcServiceImpl(
        @Qualifier("testDataCustomerExampleProvider")
        final SingleDataSourceProvider p) {

        super(p, CustomerExampleSProcService.class);
    }

    @Override
    public Customer loadCustomer(int id) {
        return sproc.loadCustomer(id);
    }
```
Using the wrapper

- Define DTO classes if necessary
  - Input parameters
  - ResultSet mapping

```java
@DatabaseType(name="t_customer")
public class Customer {

@DatabaseField
protected Integer id;

@DatabaseField
protected String name;
@DatabaseField
protected String firstName;
@DatabaseField
protected List<Address> addresses;

@DatabaseField
protected Address defaultAddress;
```
Using the wrapper

• Next create analogous PostgreSQL types

    CREATE TYPE t_customer AS ( id int,
                               name text,
                               address t_address[] );

• Or use “OUT” columns

    CREATE FUNCTION load_customer( INOUT id int,
                                    OUT name text,
                                    OUT address t_address[] )
    RETURNS SETOF record AS

• Implement stored procedures
Putting it together

• Integration test

```java
public class CustomerExampleIT {

    @Autowired
    private CustomerExampleSProcService service;

    @Test
    public void testCreate() {
        Customer c = new Customer();
        c.setFirstName("Jan");
        c.setName("Name");

        Customer result = service.createCustomer(c);

        assertEquals( true, (int)result.getId() > 0 );
        assertEquals( null, result.getDefaultAddress() );
    }
}
```
Outline

• Introduction
• Stored procedure wrapper
  – Problems before the wrapper
  – How it works
  – How to use it
  – More features including sharding
• PGObserver
Running SQL queries

- `@SProcCall(sql="[…]")` may run any query
  - Benefit from type mapper
  - Relatively easy to use
  - Although mixing SQL into Java source

```java
@SProcCall(sql="UPDATE t SET name = ?"
  + " WHERE id = ? "
  + " RETURNING id")
int updateName(@SProcParam String newName,
               @SProcParam int userId);

// allows you then to do:
int r = service.updateName('Jan',1001);
```
Sharding support

- Parameter annotation `@ShardKey`
- `@ShardKey` and `@SProcParam` may overlap

```java
@SProcCall
Customer getCustomer(@ShardKey int shardId,
                     @SProcParam String cnumber)
```

```java
@SProcCall
Article getArticle(@ShardKey @SProcParam ean)
```

- `ShardedObject` interface for custom classes
- Added datasource providers for translation
Different datasource providers

DataSource Provider

Java code and annotations

Spring context config
Search and “Merge” result set

- Use `searchShards` where you do not know the shard
  - will run on all shards return on first find
- Use `runOnAllShards` execute on all shards
  - Search `name like 'Na%'` and return one collection
Auto partitioning

- Java method called with one large collection
  - Wrapper will split collection according to key
  - Execute SQL for split collection on each shard
- Default behavior if @ShardKey is a collection
Java bean validation

- Annotation based validation (JSR 303)
  ```java
  @DatabaseField
  @NotNull
  public String a;
  
  @DatabaseField
  @Min(4)
  @Max(6)
  @NotNull
  public Integer b;
  ```

- Relying on hibernate validator
- Automatically checked inside wrapper
  - Less boiler plate code
  - `@SProcService(validate = true)`
Value transformers

• Global registry for type conversions
  - e.g. for use with JodaTime class
  - Enables transparent handling of legacy types

• Usefull for ::text to Java class conversion
  - Type safe domain classes
  - ::text => class EAN
Per stored procedure timeout

• Trouble with global statement timeout
  – Long running queries and supposedly fast ones

• Added `@SProcCall(timeout=x)`
  – X is timeout in ms
  – Allows overwrite for long running jobs
  – Ensures limited run time for “fast” functions
    • Search functions with too few constraints
Concurrency with advisory locks

• Single database serves many Java instances
  – Synchronization may be required

• Wrapper features one enum for different locks
  – `@SProcCall(advisoryLockType=LOCK1)`
  – Easy locking
  – One enum warns developers of existing locks
Transaction support

- Spring's `@.Transactional` should work
  - More or less datasource dependent
  - Sharded environment more complicated

- For multi shard operations wrapper provides
  - Context is one procedure equals one transaction
  - Immediate commit on each shard
  - Commit only if all executions were successful
  - Use two phase commit

- Enabled on SProcService or SProcCall level
Outline

• Introduction

• Stored procedure wrapper
  – Problems before the wrapper
  – How it works
  – How to use it
  – More features including sharding

• PGObservable
PGObserver

• Build to monitor PostgreSQL performance
  – Stored procedures as execution unit
  – Track table statistics to assist identifying causes

• Infrastructure
  – One Java data gatherer
  – Web frontend in using Python
  – Metric data is stored in PostgreSQL
  – Per service configuration of all gather intervals
PGObserver database view

CPU vs Sproc Load

IO related stats

Top 10 by runtime

Top 10 by calls

<table>
<thead>
<tr>
<th>Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Avg. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>311</td>
<td>11m</td>
<td>58.561s</td>
<td>2.310s</td>
</tr>
<tr>
<td>9141</td>
<td>9m</td>
<td>36.958s</td>
<td>0.063s</td>
</tr>
<tr>
<td>3552</td>
<td>7m</td>
<td>8.279s</td>
<td>0.121s</td>
</tr>
<tr>
<td>3552</td>
<td>7m</td>
<td>7.633s</td>
<td>0.120s</td>
</tr>
<tr>
<td>1258</td>
<td>4m</td>
<td>3.139s</td>
<td>0.008s</td>
</tr>
<tr>
<td>285</td>
<td>4m</td>
<td>27.231s</td>
<td>0.738s</td>
</tr>
<tr>
<td>7104</td>
<td>4m</td>
<td>21.282s</td>
<td>0.307s</td>
</tr>
<tr>
<td>20677</td>
<td>4m</td>
<td>14.168s</td>
<td>0.012s</td>
</tr>
<tr>
<td>18940</td>
<td>3m</td>
<td>9.183s</td>
<td>0.013s</td>
</tr>
<tr>
<td>18261</td>
<td>2m</td>
<td>44.869s</td>
<td>0.009s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Avg. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>246101</td>
<td>1m</td>
<td>19.093s</td>
<td>0.000s</td>
</tr>
<tr>
<td>76563</td>
<td>3.604s</td>
<td>0.000s</td>
<td></td>
</tr>
<tr>
<td>50645</td>
<td>13.003s</td>
<td>0.000s</td>
<td></td>
</tr>
<tr>
<td>40588</td>
<td>41.182s</td>
<td>0.001s</td>
<td></td>
</tr>
<tr>
<td>20519</td>
<td>31.521s</td>
<td>0.001s</td>
<td></td>
</tr>
<tr>
<td>20677</td>
<td>4m</td>
<td>14.168s</td>
<td>0.012s</td>
</tr>
<tr>
<td>18940</td>
<td>3m</td>
<td>9.183s</td>
<td>0.013s</td>
</tr>
<tr>
<td>1261</td>
<td>2m</td>
<td>44.869s</td>
<td>0.009s</td>
</tr>
<tr>
<td>10882</td>
<td>28.045s</td>
<td>0.003s</td>
<td></td>
</tr>
<tr>
<td>10558</td>
<td>1m</td>
<td>15.699s</td>
<td>0.007s</td>
</tr>
</tbody>
</table>
Sequential scan in live env.

Total runtime per monitored 15min

Avg. run time per call

Avg. self time per call
Table I/O data excerpt

Table size

Index size

Sequential scans
Summary

- Stored procedures can improve performance
- Type mapper great library to reduce map code
- Wrapper makes procedure usage a lot easier
- Stored procedure and general PostgreSQL performance monitoring is very important
- Wrapper and PGOObserver available soon!

Visit us on:
- http://www.github.com/zalando
- http://tech.zalando.org
Thank you for listening