Yes! The size matters!

PostgreSQL implementation on a large operational database,. . .

Thomas BOUSSEKEY (@ThomasBoussekey)

DECATHLON, retail IT

2017-10-27

posdata

A better way to share sales data.

Agenda

- Introduction: The context
- Building Version 1
- Version1 GO PROD
- Emergency DDL update
- Next steps

Introduction: The context

My background

16+ years as DBA on different engines

- MSSQL (2001 - 2016)
  - Starting on distributed environments all over EUROPE
  - Then stepping into onPremise datacenter for CRM database
- ORACLE on a BI datamart (2008 - 2014)
  - 1 Billion mails sent to customers
  - One of the largest database of the company
    - Thanks the data management into the editor's app!
- 2012: Discovering POSTGRESQL
  - Building the new CRM database on `PostgreSQL9.2`

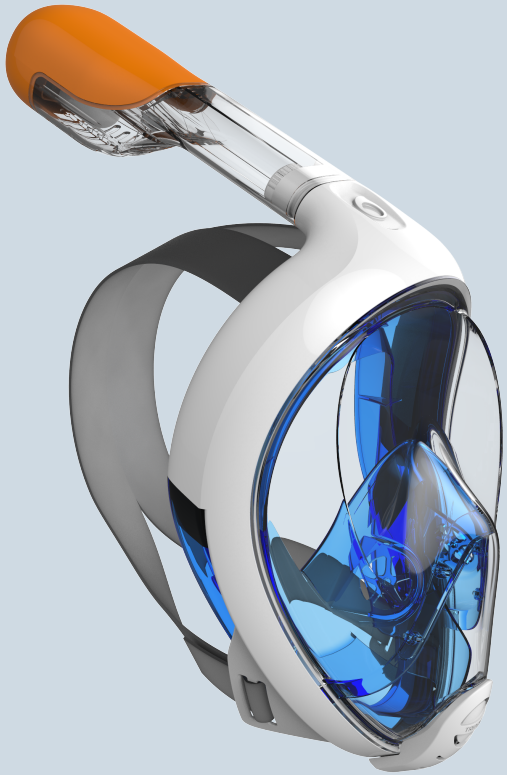
DECATHLON

DECATHLON in figures

- **102.032** teammates (2017-07)
  - 910 IT teammates, AND 500 external resources
- 2018: 130 IT people to recruit!
- Worlwide company: stores in **34** countries

Activities

Sport product design



Retail

- Click
  - 23 websites
  - **5** distinct CMS / e-commerce solutions used
- Mortar
  - 1403 stores
  - **3** distinct cashing softwares

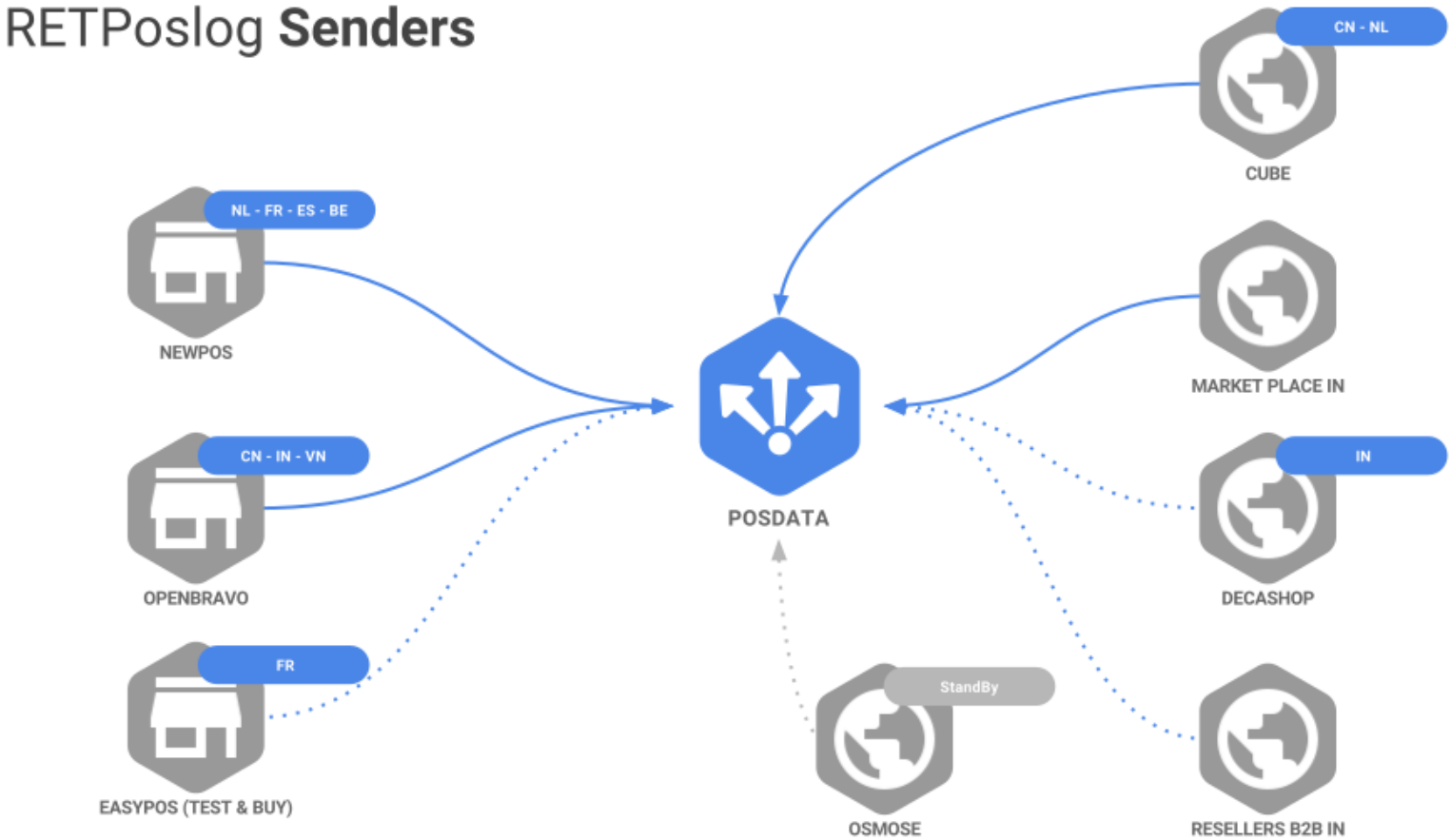
The Posdata project

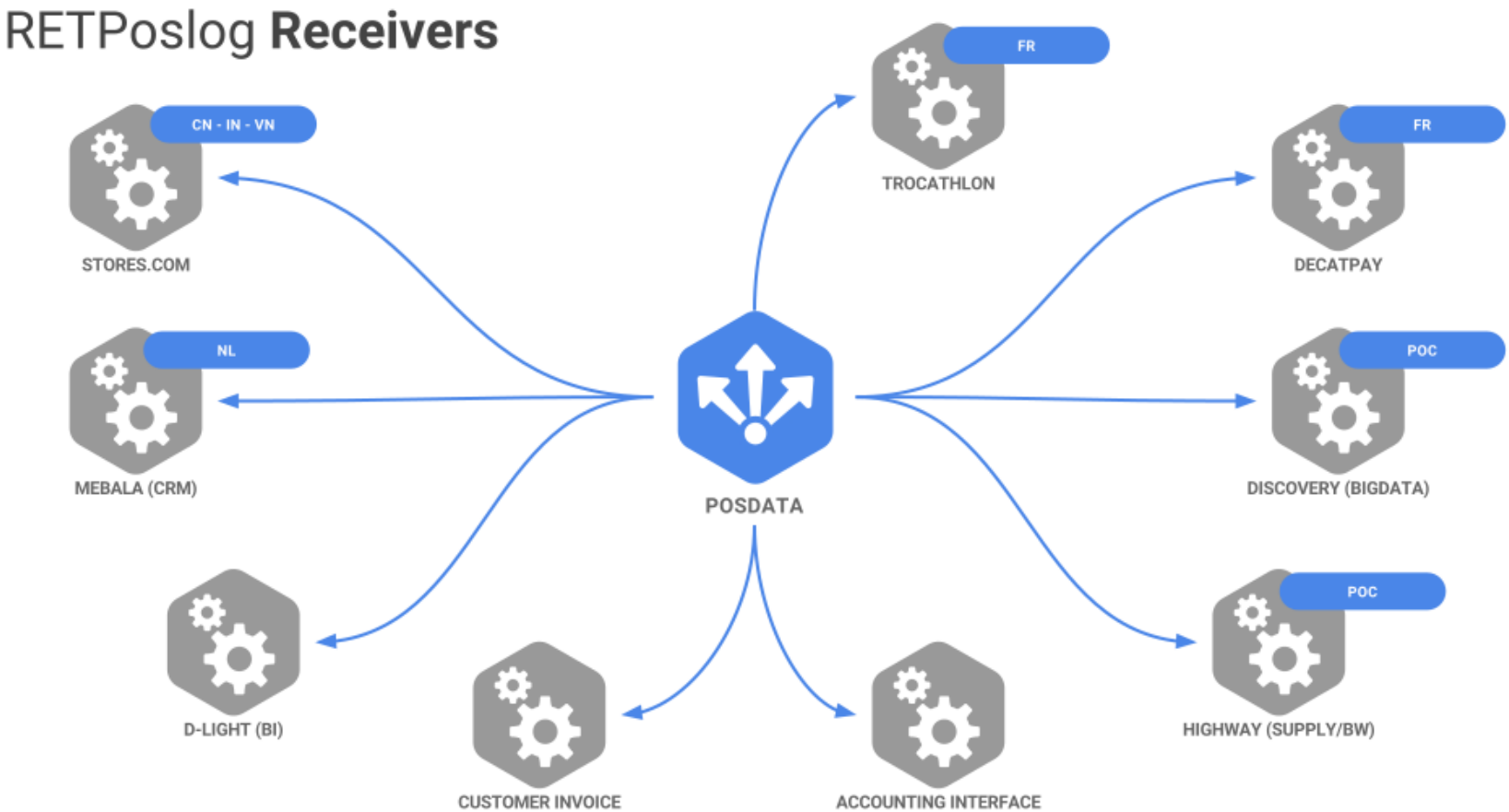
AIM

- Collect
- Store
- Distribute

. . . all sales transactions into:

- on ONE single application
- installed on 3 different datacenters.


RETPoslog Senders
CN - NL
CUBE
NL - FR - ES - BE
NEWPOS
MARKET PLACE IN
CN - IN - VN
OPENBRAVO
POSDATA
IN
DECASHOP
FR
EASYPOS (TEST & BUY)
StandBy
OSMOSE
RESELLERS B2B IN


RETPoslog Receivers
CN - IN - VN
STORES.COM
NL
MEBALA (CRM)
D-LIGHT (BI)
FR
TROCATHLON
POSDATA
CUSTOMER INVOICE
ACCOUNTING INTERFACE
FR
DECATPAY
POC
DISCOVERY (BIGDATA)
POC
HIGHWAY (SUPPLY/BW)

Alert

We need a common pivot format to collect these data.

Alert

We need a common pivot format to collect these data.

Solution

Use of POSLOG format

POSLOG Format

Definition

- Standard released by the Association for Retail Technology Standards [a]
- Highly compatible

POSLOG Format

Definition

- Standard released by the Association for Retail Technology Standards [a]
- Highly compatible

- Up to 7 versions for e-commerce command
- Updated when customer returns products from initial sale.

a. https://en.wikipedia.org/wiki/Association_for_Retail_Technology_Standards

The cashtill ticket

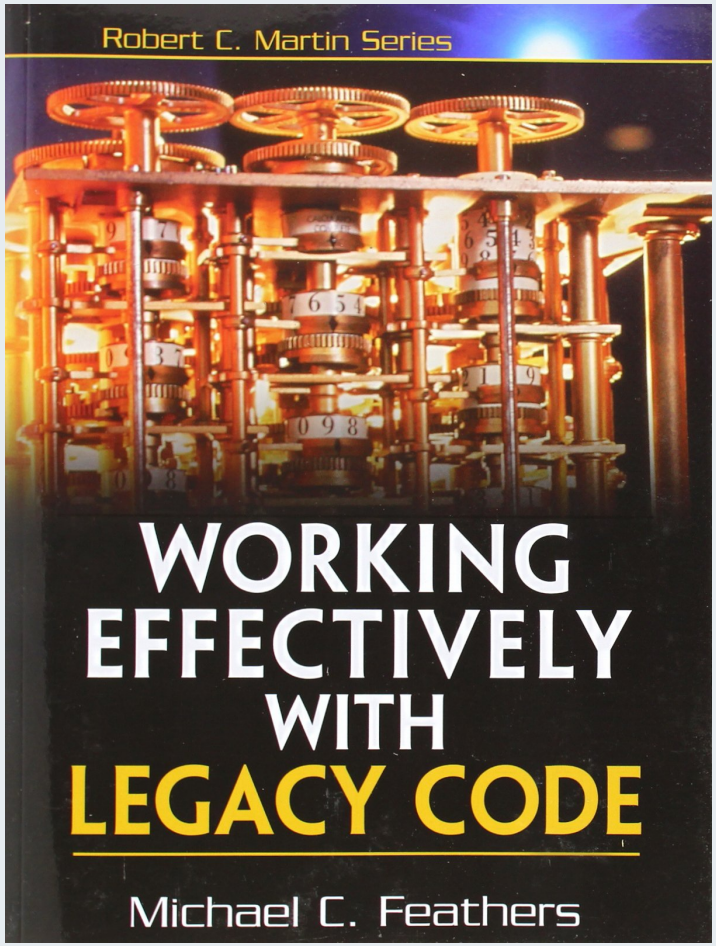
```
Magasin BTWIN village [.../...]

 TROTTINETTE B1 COQUE
 747411                                2,99 €
 RFID : 0001270829
 TROTTINETTE B1 STRUC
 746943                               20,00 €
 RFID : 0002815436
 SCOOTER RACK
 503679                                4,99 €
 RFID : 0000065726
 SCOOTER RACK
 503679                                4,99 €
 RFID : 0000065725
                                    ==========

 TOTAL      32,97 €
```

```
 Decat Pay                              32,97 €

 TAUX                MTT TVA             MTT HT
 20%                 5,50 €              27,48 €

Numéro de carte:  2098000079995

Vos informations carte Décathlon :
CARTE DECATHLONIEN
Merci M. BOUSSEKEY [.../...]

----------------------------------------
 Le  06/10/17 13:56 Caisse  1 Tran 9664
 Hôte(sse) : MCZ   Magasin :  700648

 HORAIRES D OUVERTURE  [.../...]
```

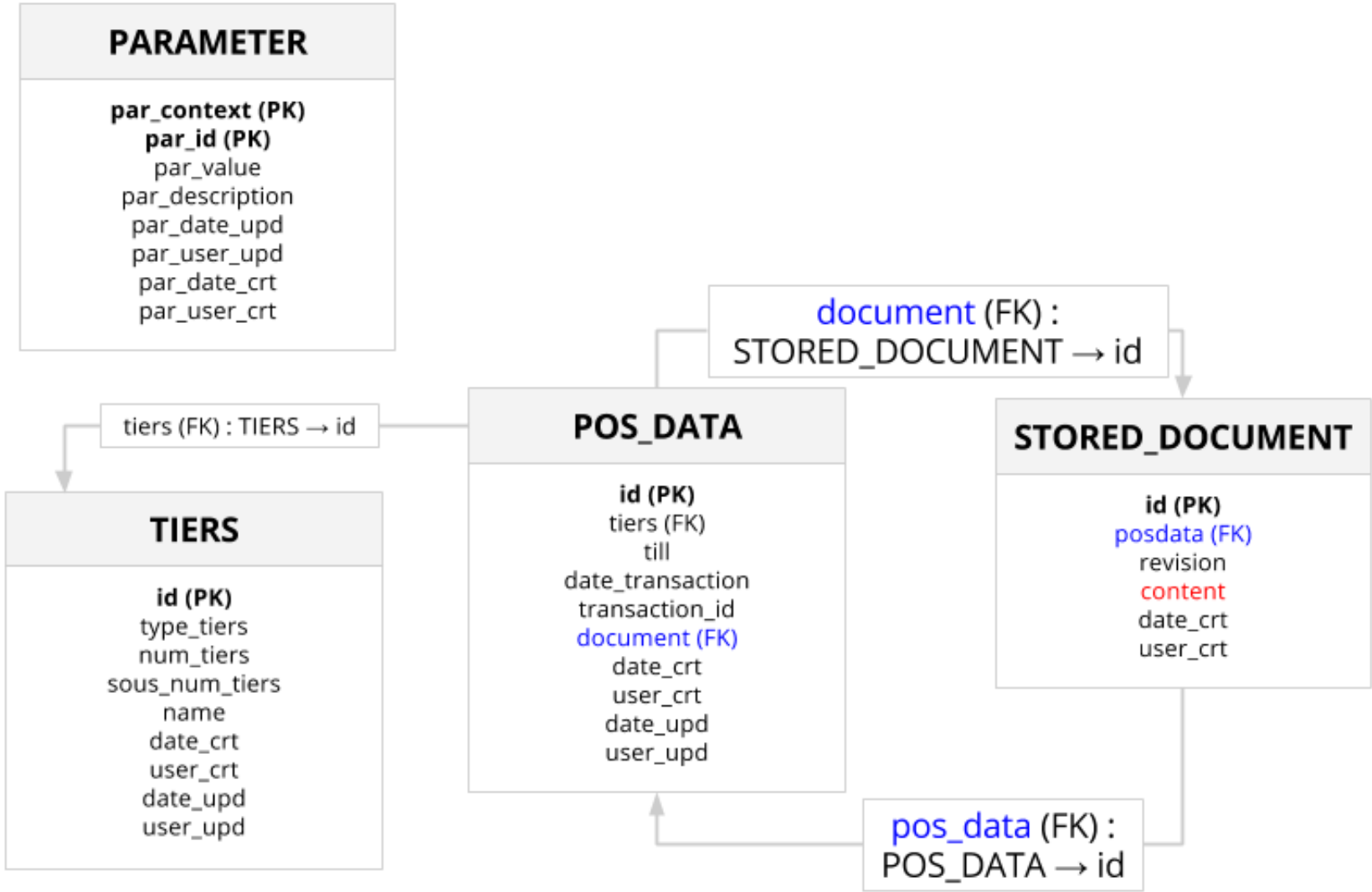
POSLOG structure

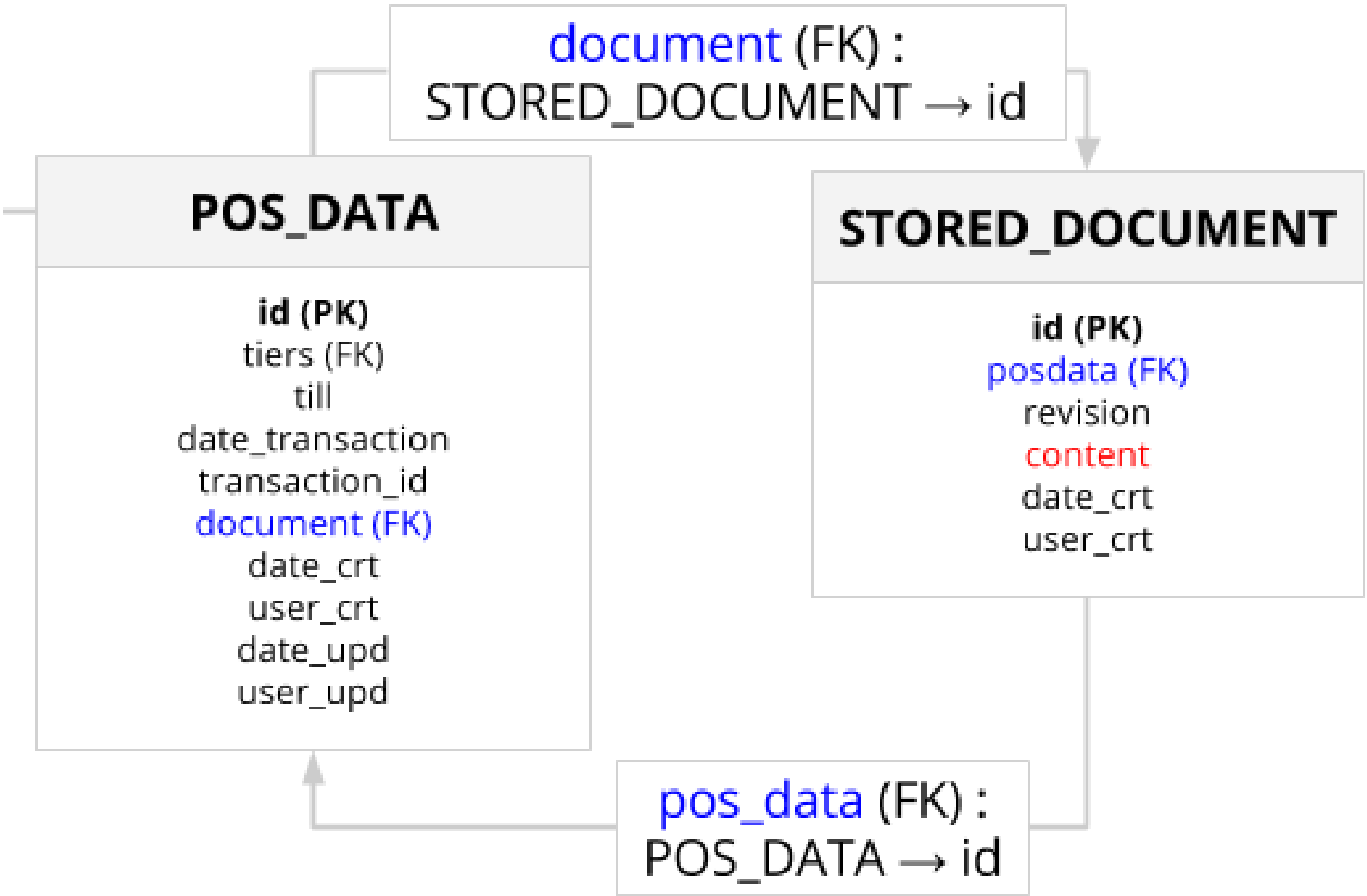
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<POSLog ><!--XML schema information-->
<Transaction >
  <!-- [Store, Channel, Date information]-->
  <RetailTransaction>
    <!-- Content of the transaction: items-->
  </RetailTransaction>
  <TrailerText>
    <Text> <!-- BASE64 till ticket encoding--> </Text>
  </TrailerText>
  <ReceiptNumber> <!--Receipt ID--> </ReceiptNumber>
  <RetailTransaction TransactionStatus="Finished">
</Transaction >
```

My arrival on the project



- After 12 years on CRM...
- Entering into the cashing IT departement
- Project was in PROTOTYPE status, close to PRODUCTION-READY


PARAMETER
par_context (PK)
par_id (PK)
par_value
par_description
par_date_upd
par_user_upd
par_date_crt
par_user_crt
document (FK) : STORED_DOCUMENT → id
tiers (FK) : TIERS → id
POS_DATA
id (PK)
tiers (FK)
till
date_transaction
transaction_id
document (FK)
date_crt
user_crt
date_upd
user_upd
STORED_DOCUMENT
id (PK)
posdata (FK)
revision
content
date_crt
user_crt
TIERS
id (PK)
type_tiers
num_tiers
sous_num_tiers
name
date_crt
user_crt
date_upd
user_upd
pos_data (FK) : POS_DATA → id


document (FK) :
STORED_DOCUMENT → id
POS_DATA
id (PK)
tiers (FK)
till
date_transaction
transaction_id
document (FK)
date_crt
user_crt
date_upd
user_upd
STORED_DOCUMENT
id (PK)
posdata (FK)
revision
content
date_crt
user_crt
pos_data (FK) :
POS_DATA → id

Feedback from the design side.

- 1 table STORED_DOCUMENT represents 95% of the database size.
  - Storing the XML POSLOG as **text**
- Double foreign key between the 2 main tables
  - Generates multiple UPDATES
  - ==> **DEAD TUPLES** on POS_DATA table
- Rename fields for a better understanding
  - Skip **"id"** field name
- Managing TIER in a different way
  - Human readable

key drivers:: Data model

- Strategy
  - Become PRODUCTION-READY
  - Industrialize the application
  - Ready to be deployed on the Cloud
    - India
    - China
- Data model
  - Spread data between many tables

key drivers:: ACID compliant

- Managing the XML POSLOG field
  - Using JSONB datatype
    - decrease size by **7%**
  - Spread XML nodes in separate tables.

key drivers:: ACID compliant

- Managing the `XML POSLOG` field
  - Using `JSONB` datatype
    - decrease size by **7%**
  - Spread XML nodes in separate tables.

NoGO on Poslog field modification

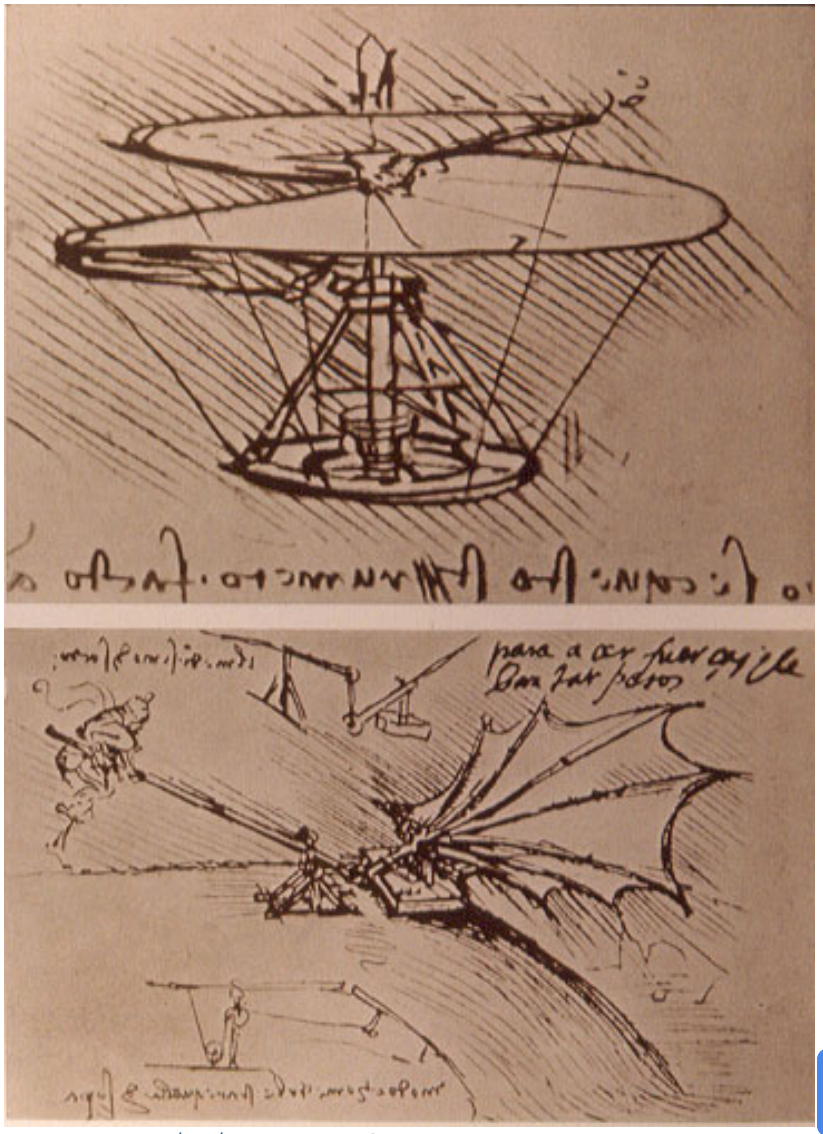
To prevent POSLOG alterations, We need to keep it the EXACT format, it was received by the application.

- Technology side

Domain	Existing platform	2016Q2
OS	RHEL 6	RHEL 7
Pg version	9.2	9.5
Config mgmt	Puppet V2	Puppet V4 + Foreman
DDL updates	Manual	Flyway [1]
Deployment	multiple RUNDECK	JENKINS

1. https://flywaydb.org/

Building Version 1



public domain, http://www.lascaux.fr, https://commons.wikimedia.org/w/index.php?curid=674315

Size considerations

Volume of Records

	Physical	E-commerce
POSLOG Length	12 kB	8 kB
POSLOG per Tx	1	7
Days opened per year	325	365
Tx per year (M)	683.96	6.71
Poslog per year (M)	683.96	47.01

Tx = Transaction

Database size estimation for EUROPE only

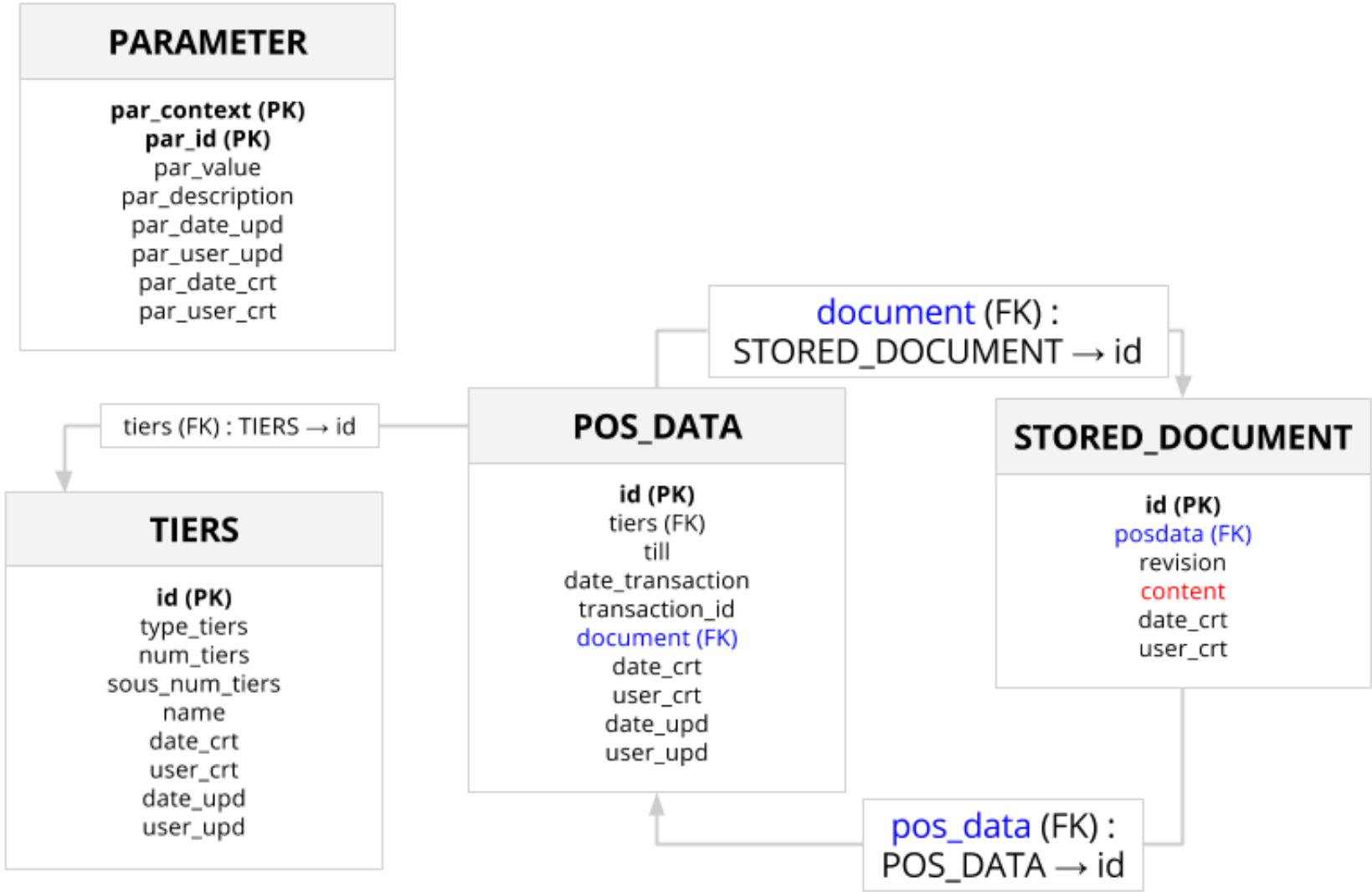
	Transactions (M)	DB Size (GB)	Yearly growth (%)
End 2017	80	350	2300
End 2018	480	2.000	570
End 2019	1.000	5.000	250
End 2020	1.600	8.000	160

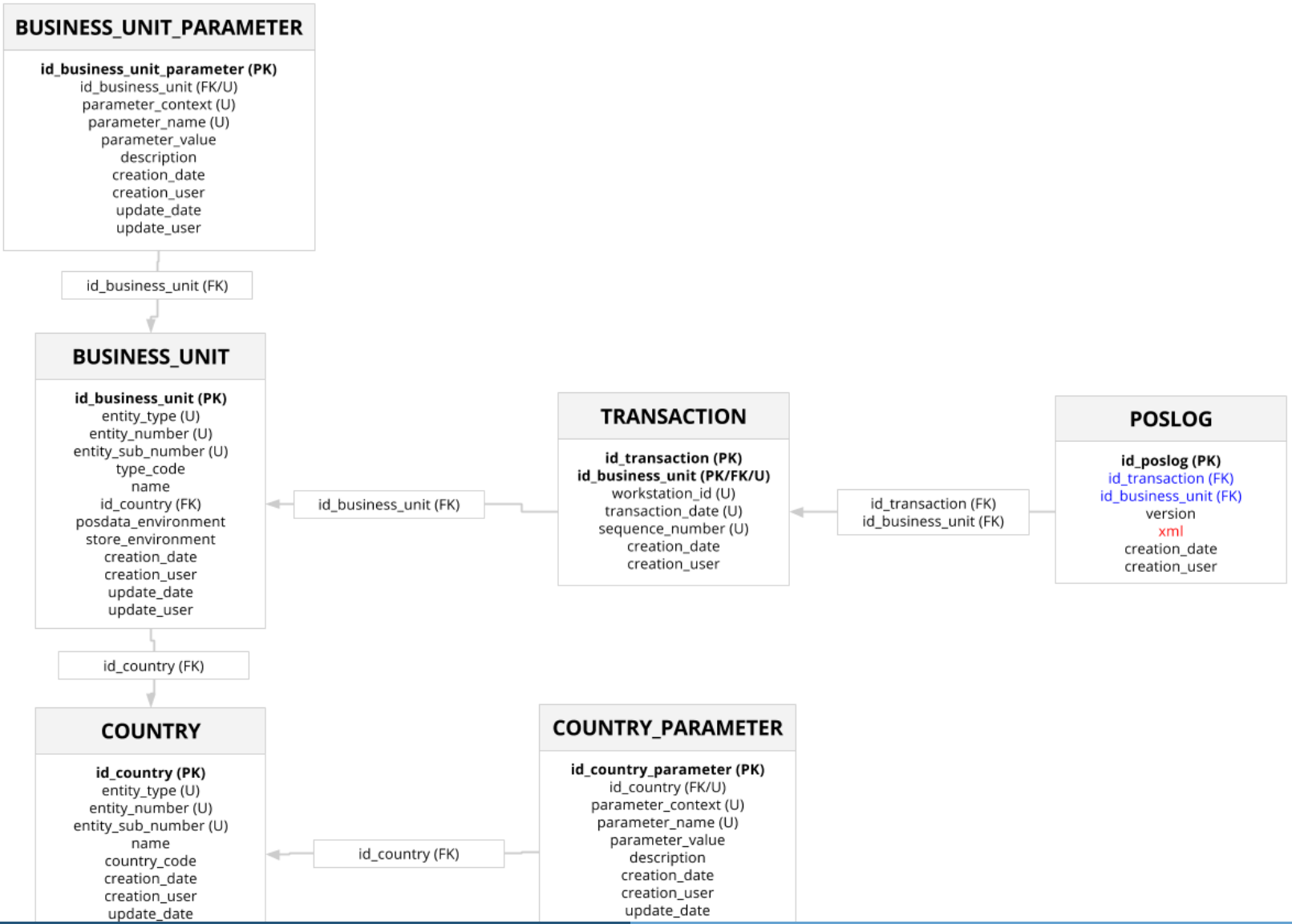
Industrialisation

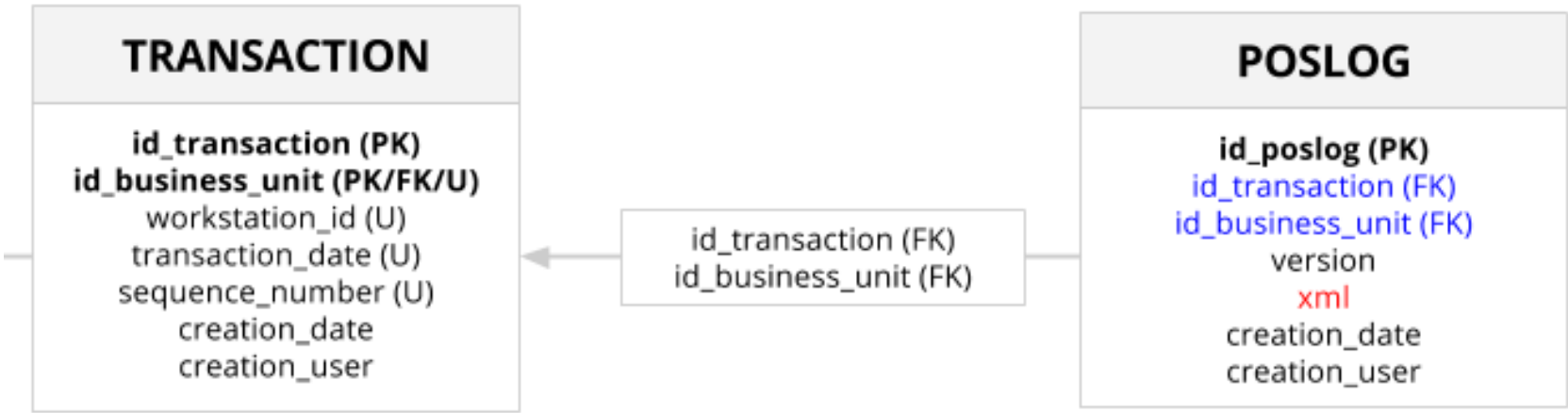
- Implementing PostgreSQL puppet Brick
  - Derivated from puppet forge
- Modified for multiple instance purposes.

PUPPET PGSQL Brick structure

```
classes:
  dkt_postgresql:
    instances:
      cash:
        instance_port: 5432
        pg_conf:
          [config_options @ postgresql.conf]
        pg_hba:
          [hba_entries @ pg_hba.conf]
        databases:
          [database_config]
```


PARAMETER
par_context (PK)
par_id (PK)
par_value
par_description
par_date_upd
par_user_upd
par_date_crt
par_user_crt
document (FK) : STORED_DOCUMENT → id
tiers (FK) : TIERS → id
POS_DATA
id (PK)
tiers (FK)
till
date_transaction
transaction_id
document (FK)
date_crt
user_crt
date_upd
user_upd
STORED_DOCUMENT
id (PK)
posdata (FK)
revision
content
date_crt
user_crt
TIERS
id (PK)
type_tiers
num_tiers
sous_num_tiers
name
date_crt
user_crt
date_upd
user_upd
pos_data (FK) : POS_DATA → id


BUSINESS_UNIT_PARAMETER
id_business_unit_parameter (PK)
id_business_unit (FK/U)
parameter_context (U)
parameter_name (U)
parameter_value
description
creation_date
creation_user
update_date
update_user
id_business_unit (FK)
BUSINESS_UNIT
id_business_unit (PK)
entity_type (U)
entity_number (U)
entity_sub_number (U)
type_code
name
id_country (FK)
posdata_environment
store_environment
creation_date
creation_user
update_date
update_user
id_business_unit (FK)
TRANSACTION
id_transaction (PK)
id_business_unit (PK/FK/U)
workstation_id (U)
transaction_date (U)
sequence_number (U)
creation_date
creation_user
id_transaction (FK)
id_business_unit (FK)
POSLOG
id_poslog (PK)
id_transaction (FK)
id_business_unit (FK)
version
xml
creation_date
creation_user
id_country (FK)
COUNTRY
id_country (PK)
entity_type (U)
entity_number (U)
entity_sub_number (U)
name
country_code
creation_date
creation_user
update_date
id_country (FK)
COUNTRY_PARAMETER
id_country_parameter (PK)
id_country (FK/U)
parameter_context (U)
parameter_name (U)
parameter_value
description
creation_date
creation_user
update_date


TRANSACTION
id_transaction (PK)
id_business_unit (PK/FK/U)
workstation_id (U)
transaction_date (U)
sequence_number (U)
creation_date
creation_user
id_transaction (FK)
id_business_unit (FK)
POSLOG
id_poslog (PK)
id_transaction (FK)
id_business_unit (FK)
version
xml
creation_date
creation_user

Risk management

Spread data in multiple tables to limit risks, locking ...

Possible solutions:

- Design the application to store data for each pos in a different table.
- Partitionning
  - manually
  - Using extensions
    - pg_partman
    - pg_pathman

Both extensions were easy to install and configure

Decision

When we had to take a decision, pg_partman was chosen:

- Extension maturity (IN 2016 Q2):
  - pg_partman was already "mature"
  - pg_pathman was in **version 0.4**
- had many stars on **GITHUB**
- **"almost"** compliant with our cloud DB managed service
- Project was up-to-date (regarding Pg releases)

Breaking news

PostgreSQL10

Partitioning is now fully integrated into PostgreSQL 10!
Native partitioning feature

Link to PostgreSQL WIKI

Partitioning

Build a partition key for point of sales

3 columns to define a **Point of sale**

- Type of TIER
- TIER number
- TIER sub-number

Merge these 3 columns into a single column

Partitioning

Build a partition key for point of sales

3 columns to define a **Point of sale**

- Type of TIER
- TIER number
- TIER sub-number

Merge these 3 columns into a single column

Huge interest for database statistics

Breaking news

PostgreSQL 10

Cross-column Statistics feature is now AVAILABLE

Link to PostgreSQL WIKI

Type of value for the partitioning key

- **INTEGER** format preserved, for:
  - search efficiency
  - ID partitioning
- VARCHAR might be:
  - more difficult to handle
  - less performant for searches and joining

Range (Exact-ID) partitionning

Need to size partitioning range to 10 (minimum set by `pg_partman` extension)

Range (Exact-ID) partitionning

Need to size partitioning range to 10 (minimum set by `pg_partman` extension)

NEED

Get a ‘human readable identifier‘
Create a ‘bigint‘, grouping the 3 fields and a final "0"

Range (Exact-ID) partitionning

Need to size partitioning range to 10 (minimum set by `pg_partman` extension)

NEED

Get a ‘human readable identifier‘
Create a ‘bigint‘, grouping the 3 fields and a final "0"

Example: **7** **00118** **00118** **0** for:

- Type of tier: **7**, 2 digits (7, 48, 50)
- Tier number: **00118**, 5 possible digits
- Sub-tier number: **00118**, 5 possible digits
- **Plus**, the trailing **0**, for range partitioning

Build a partition key for daily retention

Create 2 partitioned tables named **transaction_live** and **poslog_live** to store records and ease time-window searches.
These partitions have a 30 day retention time.

All partitions

Table Name	partitioning	Data purge	specifity
transaction	**BU**	**NEVER**	none
transaction_live	**day**	> 2 weeks	none
poslog	**BU**	**NEVER**	**XML poslog**
poslog_live	**day**	> 1 week	none

BU = Business Unit

SQL commands

Integer range partitioning

```
SELECT partman.create_parent(
  p_parent_table := 'posdata.transaction',
  p_control := 'id_business_unit',
  p_type := 'id',
  p_interval := '10',
  p_premake := 1);
```

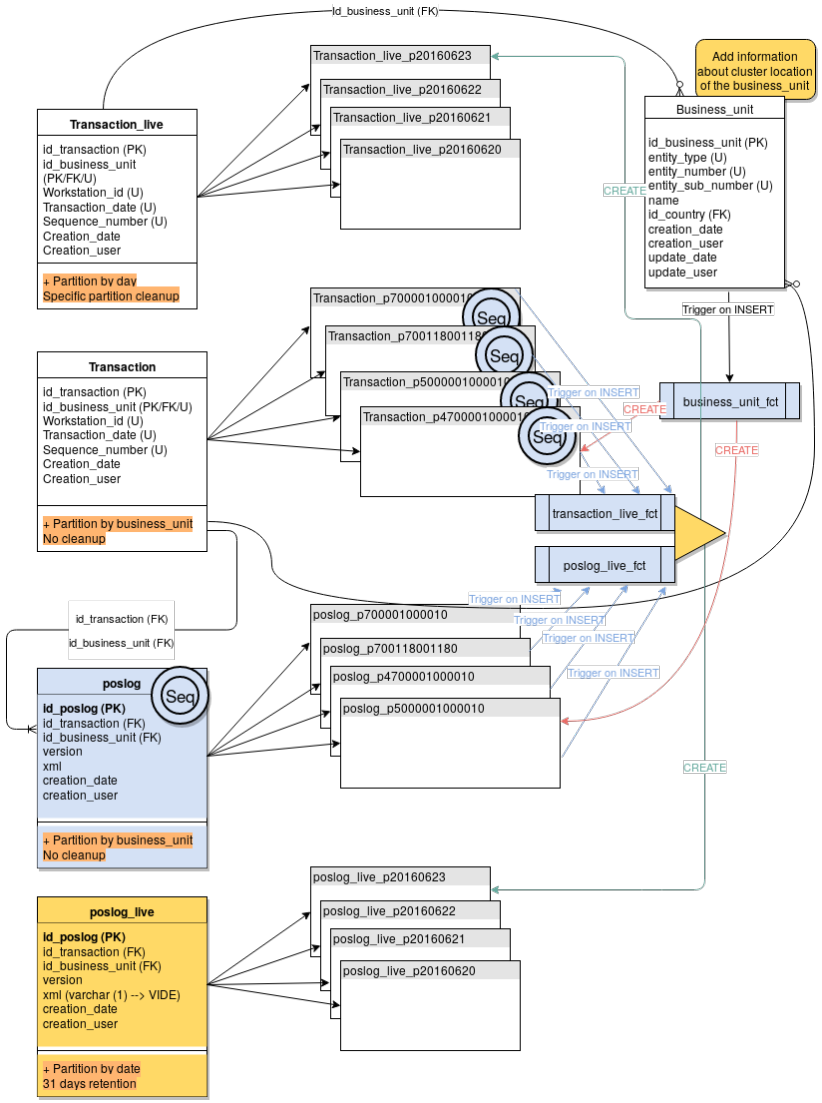
Timestamp daily partitioning

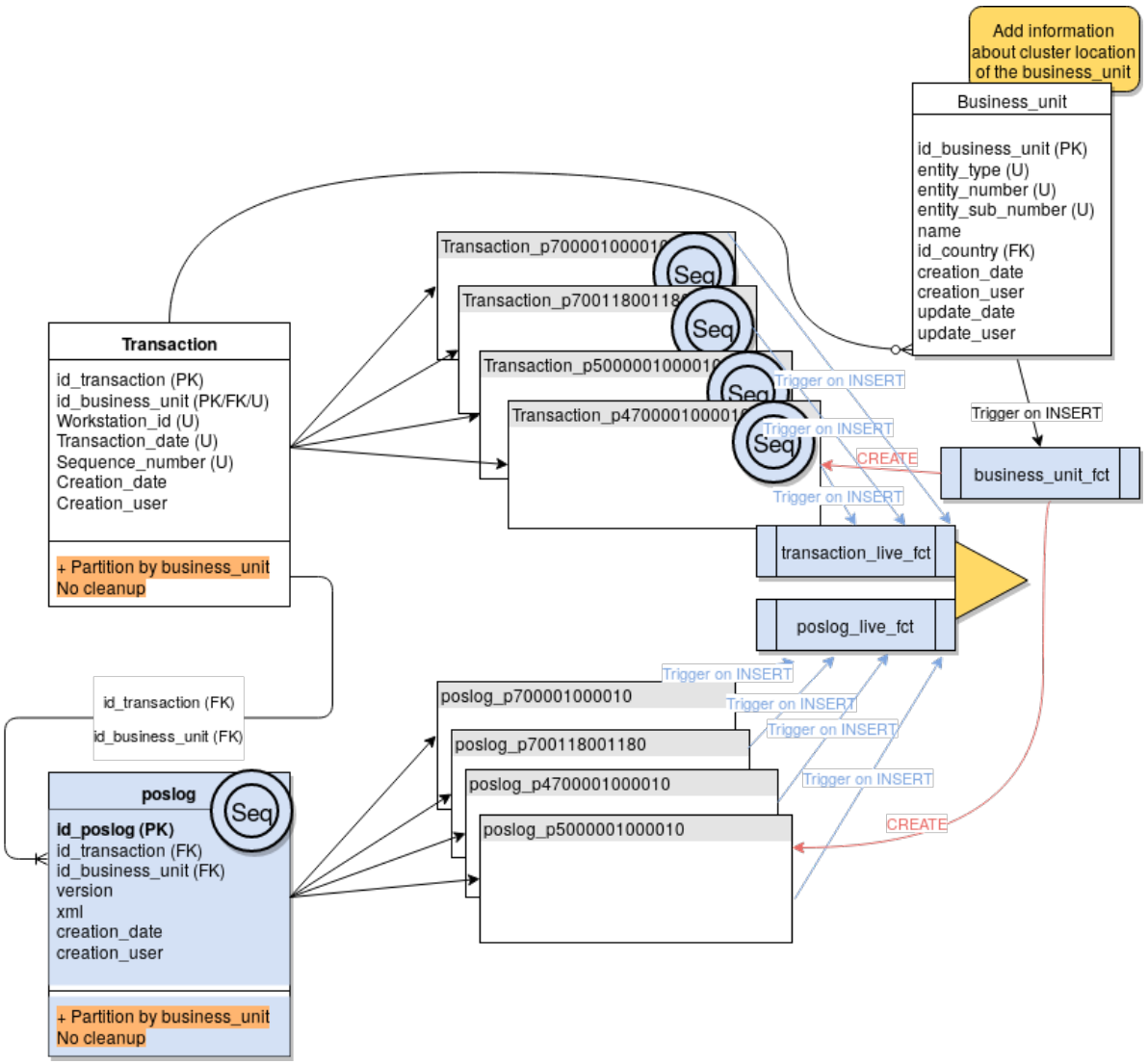
```
SELECT partman.create_parent(
  p_parent_table := 'posdata.transaction_live',
  p_control := 'creation_date',
  p_type := 'time',
  p_interval := 'daily');
```

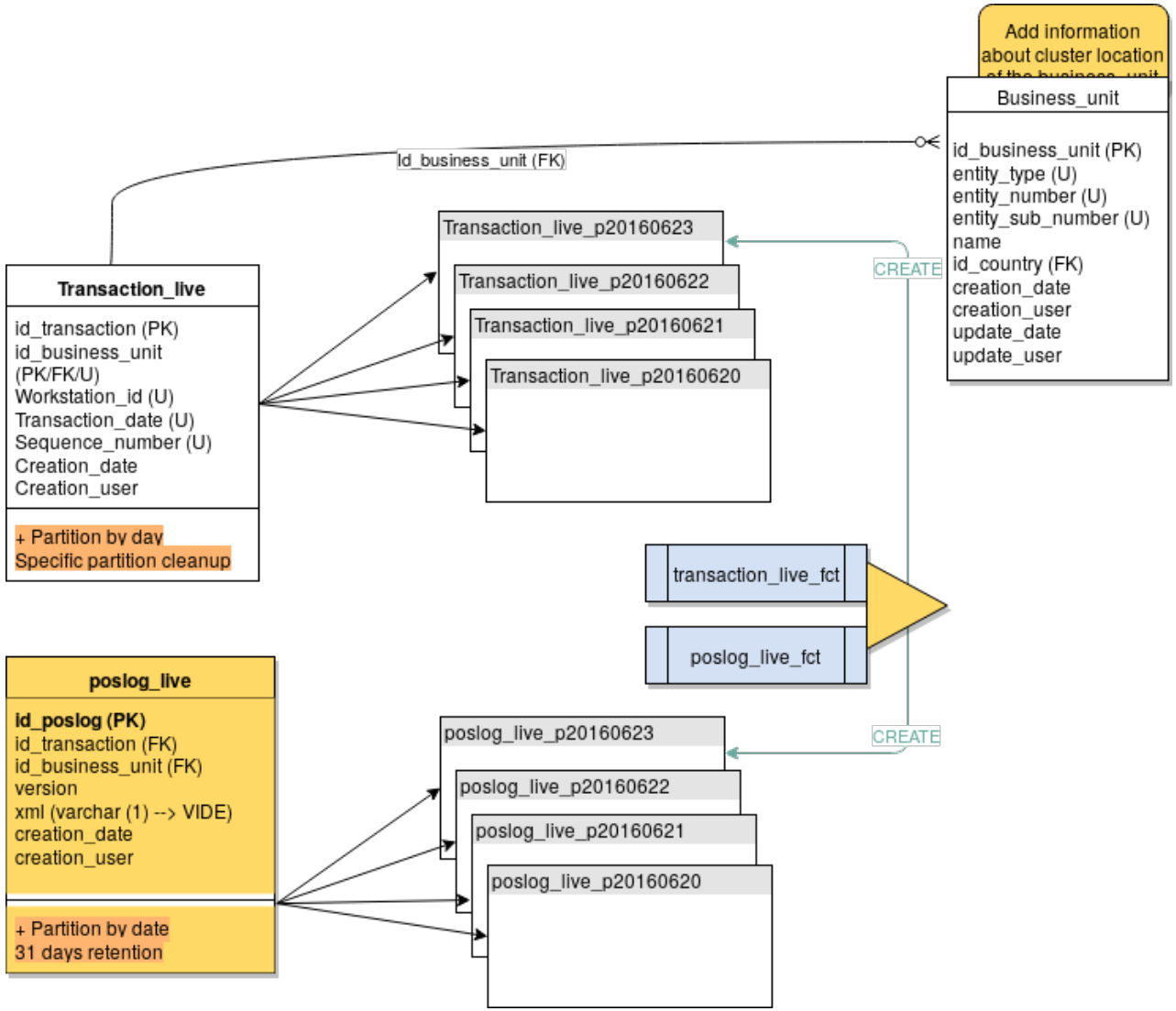
Additionnal tooling

For adding:

- point of sales
- new partitions for new days


id_business_unit (FK)
Add information about cluster location of the business_unit
Transaction_live
id_transaction (PK)
id_business_unit (PK/FK/U)
Workstation_id (U)
Transaction_date (U)
Sequence_number (U)
Creation_date
Creation_user
+ Partition by day
Specific partition cleanup
Transaction_live_p20160623
Transaction_live_p20160622
Transaction_live_p20160621
Transaction_live_p20160620
Business_unit
id_business_unit (PK)
entity_type (U)
entity_number (U)
entity_sub_number (U)
name
id_country (FK)
creation_date
creation_user
update_date
update_user
CREATE
Trigger on INSERT
Transaction
id_transaction (PK)
id_business_unit (PK/FK/U)
Workstation_id (U)
Transaction_date (U)
Sequence_number (U)
Creation_date
Creation_user
+ Partition by business_unit
No cleanup
Seq
business_unit_fct
transaction_live_fct
poslog_live_fct
id_transaction (FK)
id_business_unit (FK)
poslog
id_poslog (PK)
id_transaction (FK)
id_business_unit (FK)
version
xml
creation_date
creation_user
+ Partition by business_unit
No cleanup
poslog_p700001000010
poslog_p700118001180
poslog_p4700001000010
poslog_p5000001000010
poslog_live
id_poslog (PK)
id_transaction (FK)
id_business_unit (FK)
version
xml (varchar (1) --> VIDE)
creation_date
creation_user
+ Partition by date
31 days retention
poslog_live_p20160623
poslog_live_p20160622
poslog_live_p20160621
poslog_live_p20160620


Add information about cluster location of the business_unit
Business_unit
id_business_unit (PK)
entity_type (U)
entity_number (U)
entity_sub_number (U)
name
id_country (FK)
creation_date
creation_user
update_date
update_user
Transaction
id_transaction (PK)
id_business_unit (PK/FK/U)
Workstation_id (U)
Transaction_date (U)
Sequence_number (U)
Creation_date
Creation_user
+ Partition by business_unit
No cleanup
Seq
Trigger on INSERT
CREATE
business_unit_fct
transaction_live_fct
poslog_live_fct
id_transaction (FK)
id_business_unit (FK)
poslog
id_poslog (PK)
id_transaction (FK)
id_business_unit (FK)
version
xml
creation_date
creation_user
+ Partition by business_unit
No cleanup


Add information about cluster location
Business_unit
id_business_unit (PK)
entity_type (U)
entity_number (U)
entity_sub_number (U)
name
id_country (FK)
creation_date
creation_user
update_date
update_user
id_business_unit (FK)
CREATE
Transaction_live
id_transaction (PK)
id_business_unit (PK/FK/U)
Workstation_id (U)
Transaction_date (U)
Sequence_number (U)
Creation_date
Creation_user
+ Partition by day
Specific partition cleanup
Transaction_live_p20160623
Transaction_live_p20160622
Transaction_live_p20160621
Transaction_live_p20160620
transaction_live_fct
poslog_live_fct
poslog_live
id_poslog (PK)
id_transaction (FK)
id_business_unit (FK)
version
xml (varchar (1) --> VIDE)
creation_date
creation_user
+ Partition by date
31 days retention
CREATE
poslog_live_p20160623
poslog_live_p20160622
poslog_live_p20160621
poslog_live_p20160620

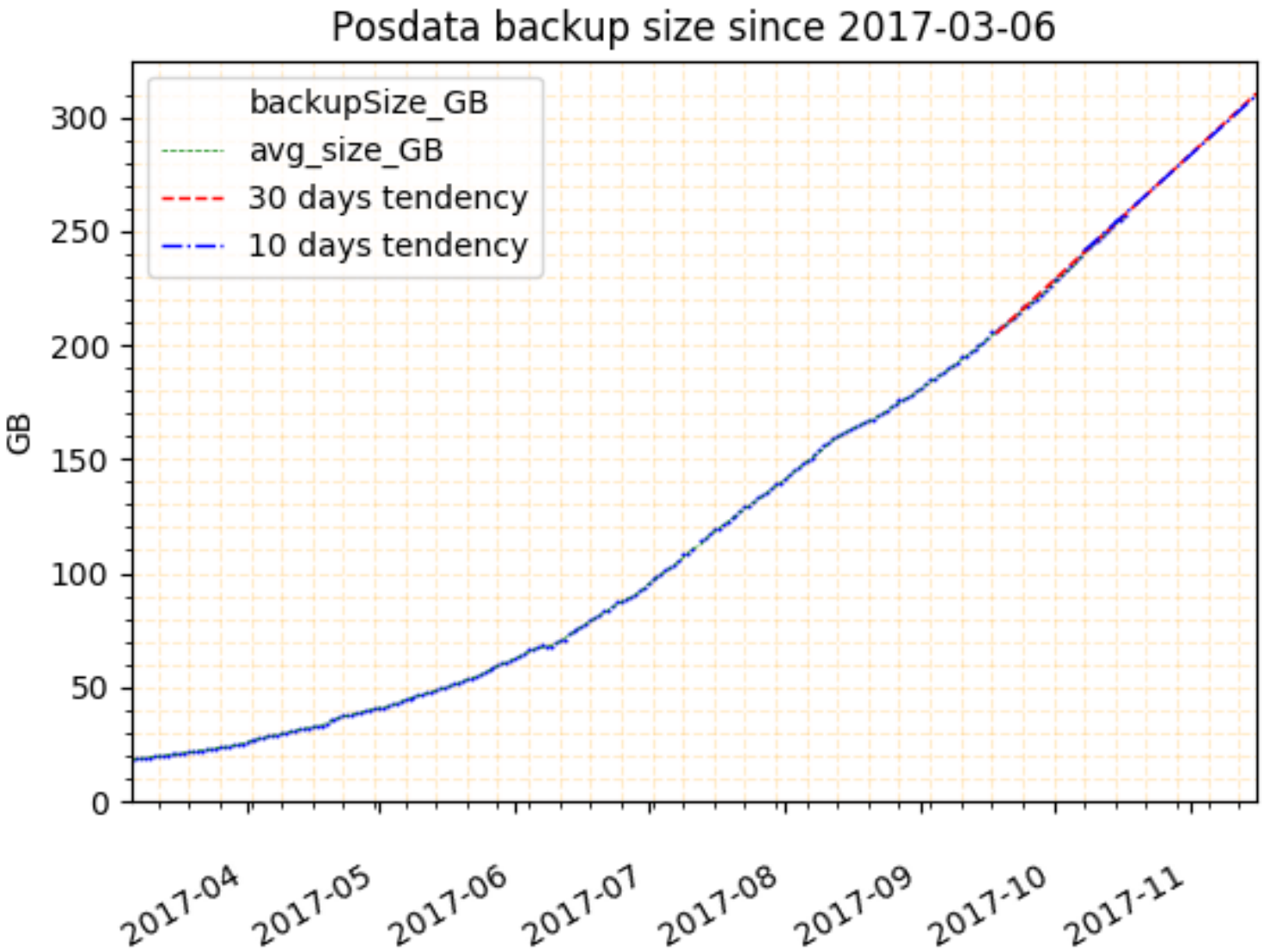
Updating configuration for pg_partman

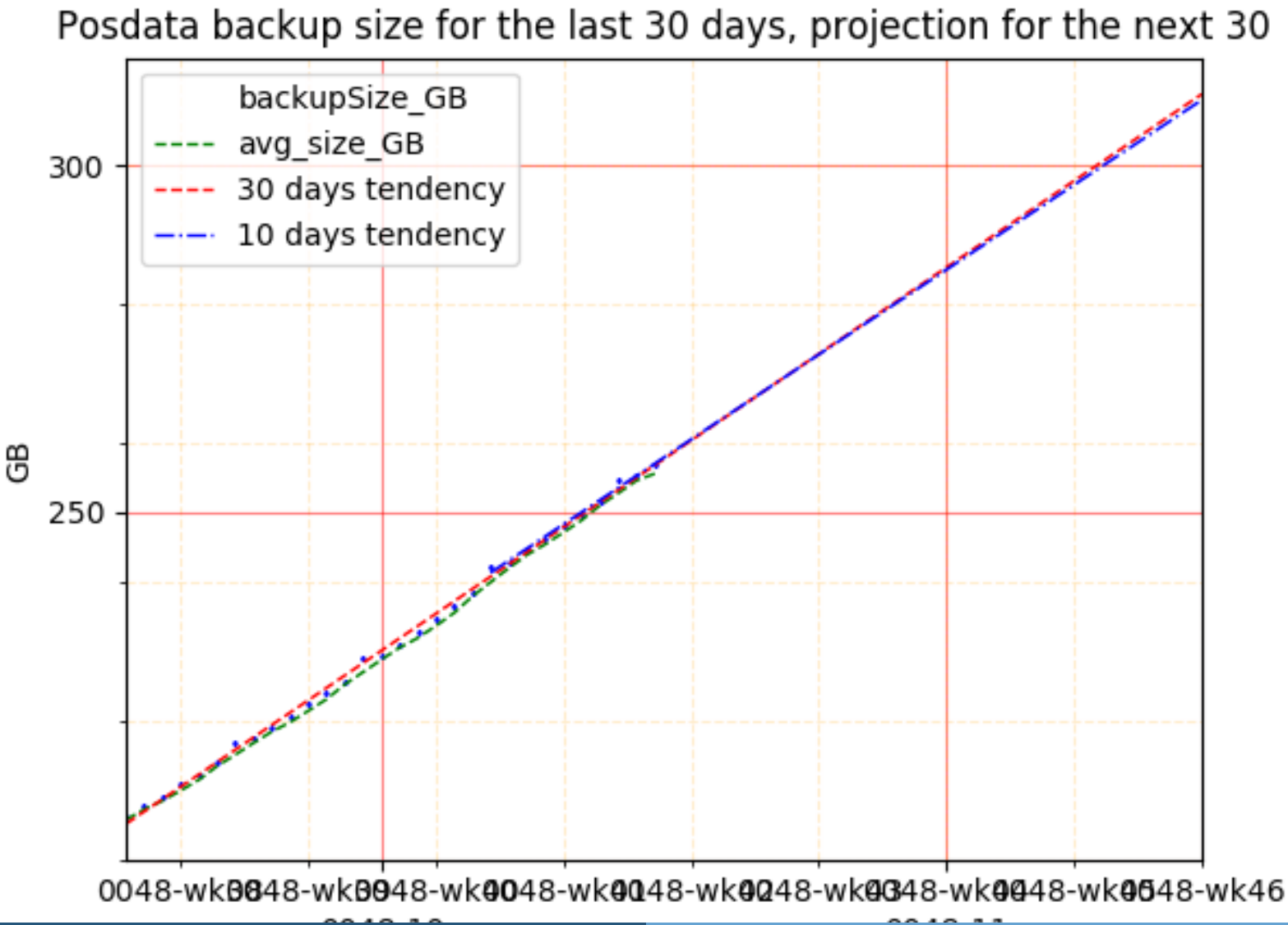
```
instances:
  cash:
    pg_conf:
      [.../...]
      max_locks_per_transaction: 512
      max_pred_locks_per_transaction: 256
      shared_preload_libraries: [...],pg_partman_bgw
    databases:
      posdata:
        database_name: posdata
          extensions:
            pg_partman:
              ensure: present
              schema: partman
            [.../...]
```

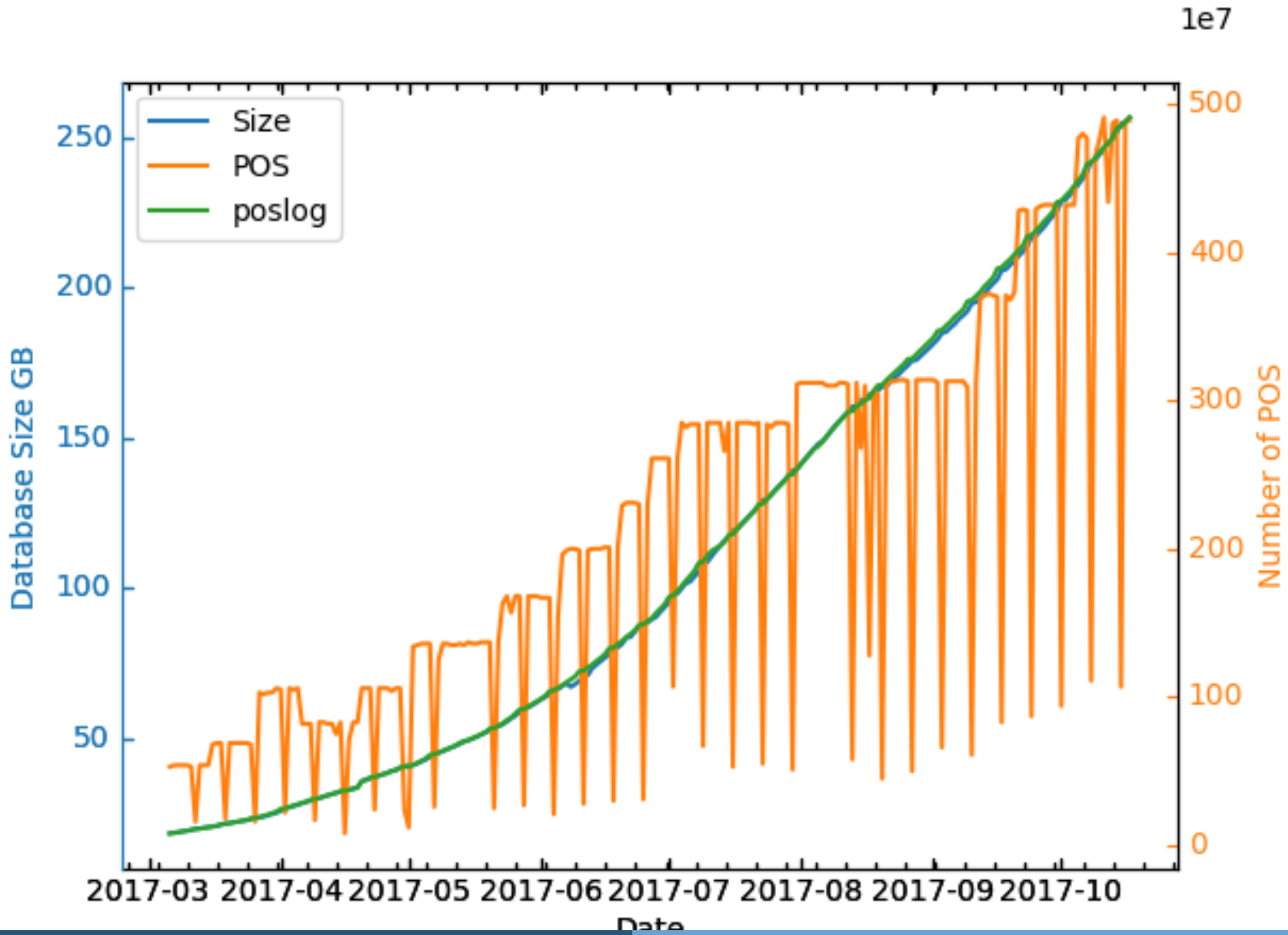
Create partitions for new business units:

- On modification on the parameter table `business_unit`
  - Fire the trigger `business_unit_fct`
    - If the partitions doesn't exist, they are created

Version1 GO PROD

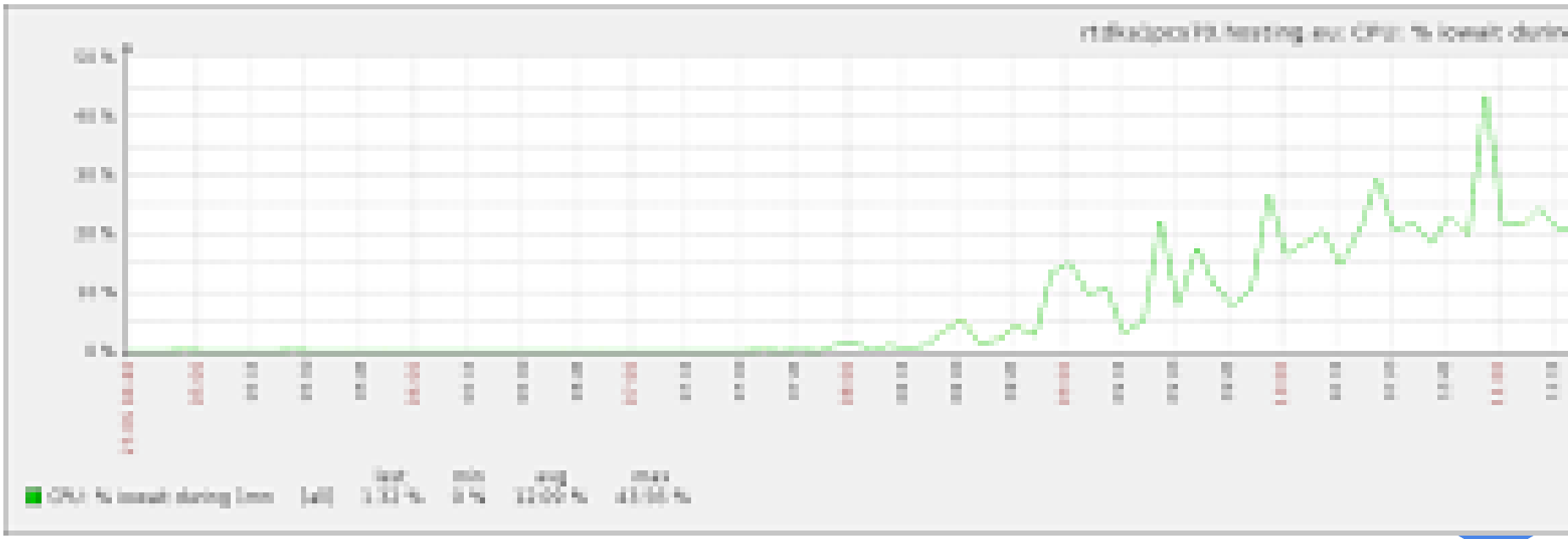

Posdata backup size since 2017-03-06
backupSize_GB
avg_size_GB
30 days tendency
10 days tendency
GB
0
50
100
150
200
250
300
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11


Posdata backup size for the last 30 days, projection for the next 30
backupSize_GB
avg_size_GB
30 days tendency
10 days tendency
300
250
GB
0048-wk46


1e7
Size
POS
poslog
Database Size GB
Number of POS
250
200
150
100
50
500
400
300
200
100
0
2017-03
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
Date

Why do we encounter `IOWait`, 6 months after live on production ?

- IOWAIT strong increase discoverd on Friday JUNE 23rd morning on Production



Searching a ROOT-cause!

- Here are your 3 new friends:
  - `pg_buffercache`
  - `pg_stat_statements`
  - `EXPLAIN`

Working with PG_BUFFERCACHE

```
SELECT c.relname as "Name", count(*) as "buffers",
  round(count(*)*8.0/1024,2) AS Buffer_MB,
  round(count(*)*100.0/(SELECT count(*)
      FROM pg_buffercache),2) as "%"
FROM pg_buffercache b
  INNER JOIN pg_class c
    ON b.relfilenode = pg_relation_filenode(c.oid)
WHERE b.reldatabase IN (0, (SELECT oid FROM pg_database
                WHERE datname = (current_database())))
  GROUP BY c.relname
  ORDER BY 2 DESC
    LIMIT 50;
```

Clue #1

- Small amount of **heap** partitioned POSLOG were found into the BUFFERS
  - Can be improved by **index creation**
  - Which query needs to be indexed?

Working with PG_STAT_STATEMENTS

- Having a good sample
  - Do not investigate on `N-1` version queries
  - Avoid DBA & Dev queries
- A small tip
  - On Friday afternoon, Run a `SELECT pg_stat_statements_reset();`
  - On Monday morning, **"Snapshot"** pg_stat_statements table

Query used to focus on IOWait

```
SELECT left(pss.query, 120), pss.calls, pss.total_time,
  pss.mean_time, pss.stddev_time, pss.shared_blks_hit,
  pss.shared_blks_dirtied
from pg_stat_statements pss
where pss.dbid =
  (SELECT oid
    from pg_database
    where datname = current_database())
order by pss.total_time desc
limit 80;
```

CULPRIT identified:

```
SELECT COALESCE(max(p.version), 0)
  FROM posdata.poslog_p700346003460 p
  WHERE p.id_business_unit= 700346003460
    AND p.id_transaction= 1234;
```

Execution plan

```
EXPLAIN (ANALYZE, BUFFERS) SELECT COALESCE(max(p.version),
  WHERE p.id_business_unit= 700346003460 AND p.id_transacti
```

Execution plan

```
EXPLAIN (ANALYZE, BUFFERS) SELECT COALESCE(max(p.version),
  WHERE p.id_business_unit= 700346003460 AND p.id_transacti

Aggregate  (cost=8797.93..8797.94 rows=1 width=4) (actual t
Buffers: shared hit=**5504**
->  **Seq Scan** on poslog_p700346003460 p  (cost=0.00..879
Filter: ((id_business_unit = '700346003460'::bigint) AND (i
Rows Removed by Filter: 219595
Buffers: shared hit=5504
Planning time: 0.321 ms
Execution time: **67.162 ms**
```

How to improve?

- How to index it efficiently?
  - B-Tree index
  - BRIN index
    - Introduced in PostgreSQL 9.5

BRIN index

Definition

- BRIN stands for Block Range Index, default range is 1MB.
- BRIN is designed for handling very large tables in which certain columns have some natural correlation with their physical location within the table.
- Store `MIN` and `MAX` values for the related block.

Test it before

execution plan, once the BRIN index created:

```
Aggregate  (cost=16.03..16.04 rows=1 width=4) (actual time=
Buffers: shared hit=**20**
->  **Bitmap Heap Scan** on poslog_p700346003460 p  (cost=1
Recheck Cond: ((id_business_unit = '700346003460'::bigint)
Buffers: shared hit=20
->  **Bitmap Index Scan** on poslog_p700346003460_brin_idx(
Index Cond: ((id_business_unit = '700346003460'::bigint) AN
Buffers: shared hit=20
Planning time: 0.406 ms
Execution time: **0.198 ms**
```

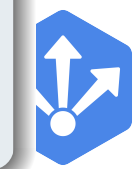
Compare the 2 kinds of index

	Size (kB)	Blocks read
Heap	736.000	5.504
B-Tree INDEX	8.738	6
BRIN INDEX	48	20

BRIN index needs less IO on INSERT.

Industrialize index creation

```
CREATE OR REPLACE FUNCTION add_brin_index_on_poslog_tables(
DECLARE
    live_part_table RECORD;
BEGIN
    RAISE NOTICE 'Creating BRIN indexes on poslog partition
    FOR live_part_table IN SELECT tab.relname as "tab_relna
        RAISE NOTICE 'Updating constraint for table % ...',
        EXECUTE 'CREATE INDEX IF NOT EXISTS  ' || quote_ide
        --- CREATE INDEX CONCURRENTLY IF NOT EXISTS poslog_
    END LOOP;
    RAISE NOTICE 'Done creating BRIN indexes on poslog part
    RETURN 0;
END;
$$ LANGUAGE plpgsql;
```



Other thing, I've learned at this point

- PL/pgsql LANGUAGE doesn't allow:
  - `VACUUM` inside a transaction block.
  - `CREATE INDEX CONCURRENTLY` inside a transaction block.
  - `COMMIT / ROLLBACK` inside a transaction block.

Oooopps!!! new IOWait peak several weeks after !!!

RTFM!

Oooopps!!! new IOWait peak several weeks after !!!

RTFM!

BRIN index maintenance:

Tuples on a new heap page are not summarized into the BRIN index automatically

Oooopps!!! new IOWait peak several weeks after !!!

RTFM!

BRIN index maintenance:

Tuples on a new heap page are not summarized into the BRIN index automatically

- 2 possible ways to update the BRIN index:
  - Using procedure: `brin_summarize_new_values(regclass)`
  - Perform a VACUUM on the table. [a]

a. https://www.postgresql.org/docs/9.5/static/brin-intro.html

Solution

```
CREATE OR REPLACE FUNCTION refresh_brin_index_on_poslog_tab
DECLARE
  live_part_table RECORD;
BEGIN
  FOR live_part_table IN SELECT tab.relname as "tab_relname
    RAISE NOTICE 'Refreshing BRIN index for table % ...', c
    EXECUTE 'SELECT brin_summarize_new_values(''' || quote_
  END LOOP;
  RAISE NOTICE 'Done refreshing BRIN indexes on poslog part
  RETURN 0;
END;
$$ LANGUAGE plpgsql;
```

Emergency DDL update

Possible solutions

1. Create a new table for ReceiptNumber
2. Add a new column for ReceiptNumber into Poslog table

Impact analysis

Tested on integration, on ~300k rows

Size impact (MB)	+ Column	+ Table	Delta (%)
Poslog HEAP	13.75	0	
Poslog INDEX	0	0	
Receipt HEAP	0	31.79	
Receipt INDEX	0	7.69	
Total size	13.75	39.48	**+287**
Insert 75k rows (IO)	60,526	62,124	**+3**

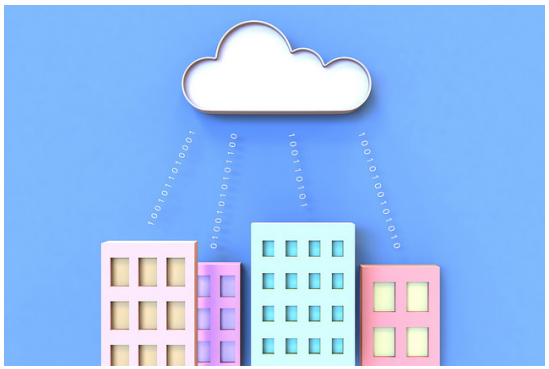
Time impact (ms)	+ Column	+ Table	Delta (%)
Creation time	40	114162	
Time for insertion	99236	0	
Vacuum time	30981	0	
Total	130257	114162	**+14/-14**

Improve XML storage

- Try on JSONB, 7% space reduced
  - Offers full text search improvement
- Quick look over `XMLTABLE`[2] feature on **PostgreSQL 10**

2. https://blog.2ndquadrant.com/xmltable-intro/

Next steps



Cloud Image from [Chris Potter](www.seywut.com/Chris).

- Technology

State of the art	2016Q1	2017Q3
Location	onPremise	Cloud
OS	RHEL 7	CentOS 7
Pg version	9.5	10 OR Postgres_XL
Config mgmt	Puppet V4 Foreman	Terraform + foreman
SQL scripts	Flyway	Flyway
Deployment	JENKINS	JENKINS

New hosting solution

- Leave onPremise datacenters to go to the cloud.
- Improve industrialization
- Start contenerization

Terraform

- Infrastructure as code
- Ability to perform operations on multiple hosting solutions
  - cloud computing
  - onPremise datacenters
- Interact with FOREMAN & PUPPET

Postgres_XL

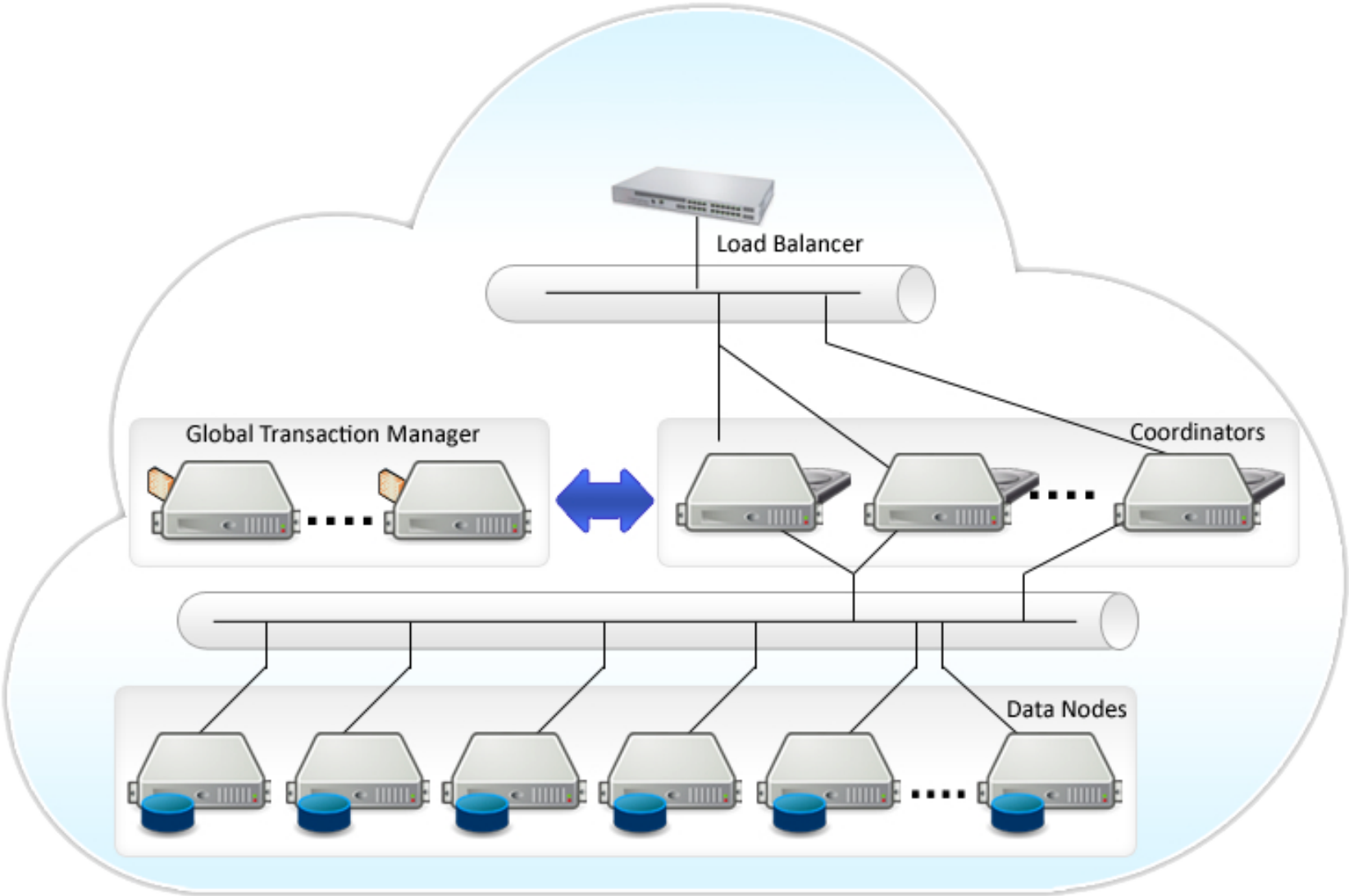
Key drivers

- Critical size estimated to 1 database for 200 databases.
  - Needing 7 different databases for production (regarding DB size)
  - Adding complex platform management to spread and search data accross the platforms
- Scalable

```
ALTER NODE data_5 WITH (TYPE = 'datanode');
```

- We can define the sharding structure for **EACH** table

```
ALTER TABLE disttab ADD NODE (data_5);
```


Load Balancer
Global Transaction Manager
Coordinators
Data Nodes

Features

- Fully ACID
- Open Source
- Cluster-wide Consistency
- Multi-tenant Security
- PostgreSQL-based

Pros

- No new engine
  - Only a few system tables & columns difference
- Table location strategy
  - **DISTRIBUTED:** for large table, align distribution startegy for joins
  - **REPLICATED:** for parameter tables

```
SELECT xc_node_id, count(*)
FROM transaction
GROUP BY xc_node_id;
```

Pros

- No new engine
  - Only a few system tables & columns difference
- Table location strategy
  - **DISTRIBUTED:** for large table, align distribution startegy for joins
  - **REPLICATED:** for parameter tables

```
SELECT xc_node_id, count(*)
FROM transaction
GROUP BY xc_node_id;
```

Cons

- Prevent cross datanodes queries

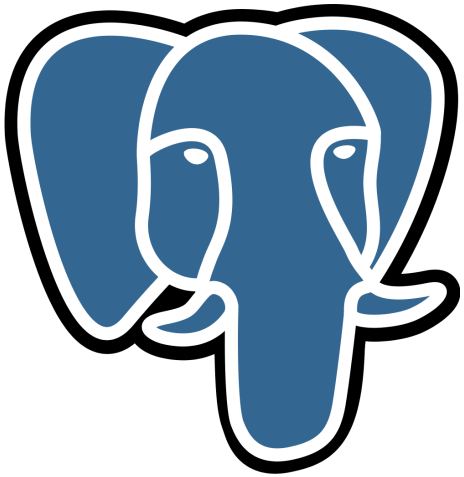
Poslog storage

- Stored in raw mode
  - No specific indexing on its content
  - Only field to contain personal information
- Can be stored into a cloud file storage solution
  - with encryption, to comply with **GDPR** regulation

Other new functionnalities / tools

- Improved monitoring
  - TICK
  - ELK
- Use containers for JAVA application
- New flow solution

Thanks for your attention!



Any question?

Created with opensource tools:

- markdown
- pandoc
- laTEX
- beamer
- and PostgreSQL

Appendix

The 7 e-commerce statuses

- **Totaled:** Order created without payment
- **Authorized:** Order validated, payment ok or authorized
- **Canceled:** Order fully cancelled
- **PartialShip:** Only a part of Shipping group is in status shipped or more.
- **Shipped:** All shipping group are shipped
- **Delivered:** All shipping group are in delivered
- **Finished:** Order is completed

Partition management scripts

```
DECLARE
    v_sqlcmd  text;
BEGIN
    -- 1 Create partition tables
    PERFORM partman.create_partition_id('posdata.transactio
    PERFORM partman.create_partition_id('posdata.poslog'::t
    -- 2 Drop existing FK
    v_sqlcmd := format('ALTER TABLE posdata.poslog_p%s DROP
    EXECUTE v_sqlcmd;
    RAISE NOTICE 'Script generated: %', v_sqlcmd;
    -- 3 Create FK
    v_sqlcmd := format('ALTER TABLE posdata.poslog_p%s ADD
    EXECUTE v_sqlcmd;
    RAISE NOTICE 'Script generated: %', v_sqlcmd;
```

```
DECLARE
    v_bigint_BusinessID2 BIGINT;
    v_str_InstanceName VARCHAR;
BEGIN
    --# Check if business_unit exists
    SELECT id_business_unit INTO v_bigint_BusinessID2  FRON
    IF NOT FOUND THEN raise exception 'NO_DATA_FOUND'; END
    --# Check if InstanceName exists
    SELECT di_instance INTO STRICT v_str_InstanceName FROM
    IF NOT FOUND THEN raise exception 'NO_DATA_FOUND'; END
    UPDATE posdata.business_unit
      SET posdata_environment = v_str_InstanceName
      WHERE id_business_unit = v_bigint_BusinessID;
EXCEPTION
WHEN no_data_found THEN
    RAISE 'InstanceName does not exist';
```

```
DECLARE
    v_table_count int;
    v_sqlcmd  text;
    v_instanceName text;
BEGIN
  -- IDENTIFY the current instance
  SELECT di_instance INTO v_instanceName FROM posdata.tech_
  /* Check if the partition is on this instance */
  IF NEW.posdata_environment = v_instanceName THEN
    v_sqlcmd := format('SELECT count(*) AS n FROM ONLY pg_c
    EXECUTE v_sqlcmd INTO v_table_count;
    CASE v_table_count
      WHEN 0 THEN
                -- no table --> OK
                RAISE NOTICE 'No table exist for business_uni
                PERFORM posdata.create_business_unit_tables(N
```