Infrastructure at your Service.

PostgreSQL as a central reporting hub

About me

Daniel Westermann

Senior Consultant

Open Infrastructure Technology Leader

+41 79 927 24 46

daniel.westermann@dbi-services.com

Who we are
dbi services

Experts At Your Service

> Over 45 specialists in IT infrastructure

> Certified, experienced, passionate

Based In Switzerland

> 100% self-financed Swiss company

> Over CHF6 mio. turnover

Leading In Infrastructure Services

> More than 120 customers in CH, D, & F

> Over 40 SLAs dbi FlexService contracted

Agenda

Agenda

1. Some wordings to begin with
2. A (hypothetical) use case
3. What PostgreSQL can offer
4. Sample setups
5. Demo

Terminology

Some wordings to begin with

When people talk about reporting databases they usually use one or a combination of these terms

- Analytic database
- OLAP
- BI
- DSS
- DM
- ETL

What do they mean and how are they different from each other?

Some wordings to begin with

OLAP

- Online Analytical Processing
- Business reporting
  - Sales
  - Marketing
  - Management
  - Forecasting
  - Financials
  - ...
- Aggregation of data
- Roll up / Drill down
- Multidimensional cubes

Some wordings to begin with

Analytic database

- A (usually) read only database
- Stores historical data
- Supports Business Intelligence
- Can be part of a data warehouse
- The data is updated on a regular basis
- Big data

"We don't let people access the data warehouse - that would slow it down too much"

Some wordings to begin with

Business Intelligence

> Transform raw data in meaningful information for business decisions
> Works on large amounts of unstructured data
> Easy interpretation of data
> De-Normalization
> Relies on meta data

"Finance here - we're not sure about this Hadoop thing... Could you just dump it all into Excel for us?"
TimoElliott.com

Some wordings to begin with

Decision Support (Systems)

- Serves the management, operations and planning levels of an organization
- Usually for mid and higher management
- "interactive computer-based systems which help decision-makers utilize data bases and models to solve ill-structured problems"

HI
TWITTER WAS DOWN THIS MORNING. COULD YOU JUST TELL ME, WHAT YOU HAD FOR BREAKFAST? THANKS!
geek and poke
THE REAL FOLLOWERS

Some wordings to begin with

Data Mining

- Discover data in large data sets
- Extract data and transform it into understandable structures
- Knowledge discovery in databases
- Anomaly detection

"When you two have finished arguing your opinions, I actually have data!"

Some wordings to begin with

ETL

- Extract, Transform, Load
  - Extract data from homogeneous or heterogeneous data sources
  - Transform the data to a format you want work with
  - Load the data into the final database
- Heavy batch loads involved

© 2001 Randy Glasbergen www.glasbergen.com
$80,000 + 0 = $800,000
"That's right, I've decided to give myself zero pay raise this year."

Some wordings to begin with
What are we talking about now?

So what are we talking about? Analytic database, OLAP, BI, DSS, DM, ETL?

They all mean more or less the same

The important message is

> You need to be able to extract the data

> You need to be able to load and maybe transform the data

> You need to be able to do reporting on top of the data

A (hypothetical) use case

A (hypothetical) use case
How it begins ...

Some time in the future Bill, Larry and Maria decide that time has come to merge their companies

A new company called "the database" is born

Everybody is enthusiastic and thousands of people are happy to work for the new company

Surprisingly nobody has to leave the new company and everybody is involved in building the open future

A (hypothetical) use case
How it evolves ...

... but then, when the first fiscal year is over and the "the database" has to report its figures to the shareholders ...

... there is a huge issue:

Neither Larry nor Bill nor Maria thought about data integration

Immediately a task force is created to align and agree on the next steps

A (hypothetical) use case
The reality (1) ...

Larry says: All our sales figures are stored in the Oracle cloud in an Oracle database

A (hypothetical) use case
The reality (2) ...

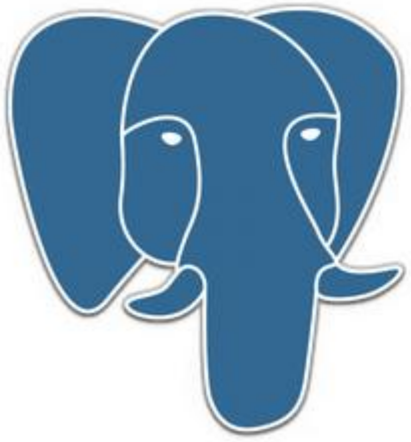
Bill says: All our sales figures are stored in Azure in a MS SQL Server instance

A (hypothetical) use case
The reality (3) ...

Maria says: All our sales figures are stored in Amazon AWS in a central MariaDB instance

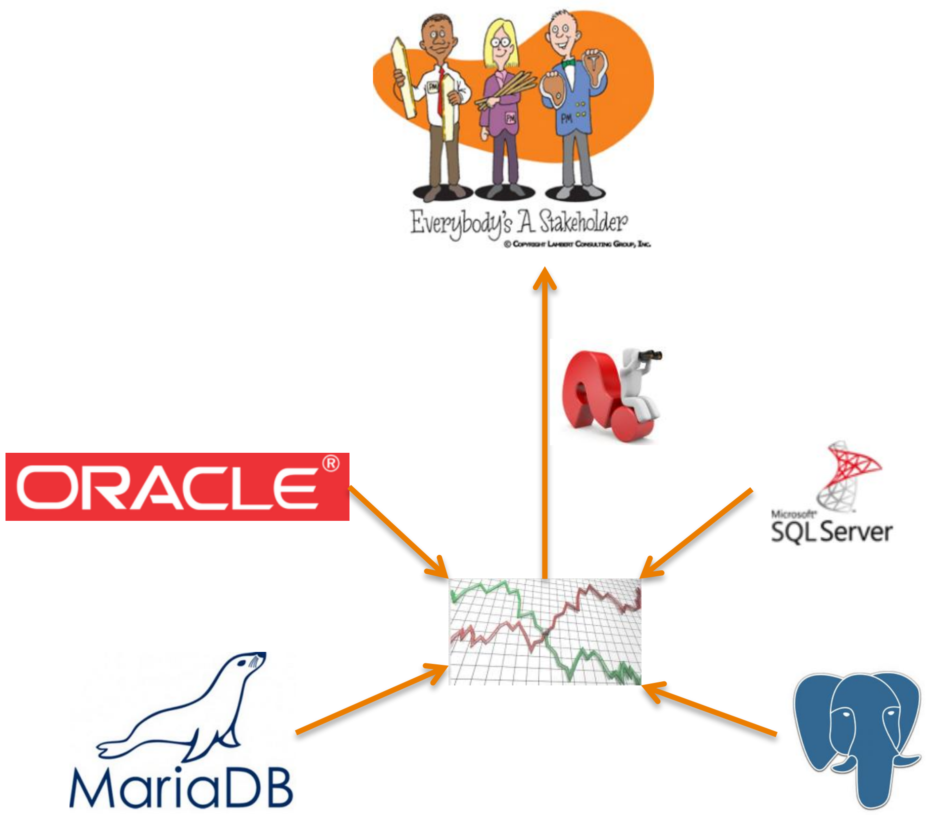
A (hypothetical) use case
The reality (4) ...

And suddenly someone raises his fingers and says: Well, there is a PostgreSQL instance which holds the sales figures for Skype



A (hypothetical) use case
The issue ...

So "the database" is faced with this:



A (hypothetical) use case
Thoughts?

What to do?

Implement a logical replication solution?

Is there any capable of replicating between all vendors?

How much time is required for implementation?

A (hypothetical) use case
Solution?

What PostgreSQL can offer

What PostgreSQL can offer
Back in 2011

When PostgreSQL 9.1 was released in 2001 support for SQL/MED was added to PostgreSQL core

MED = Modification of External Data

Foreign data wrappers (fdw) where born

FDWs allow to access data that is outside of PostgreSQL using standard SQL queries

This data is referred to as foreign data

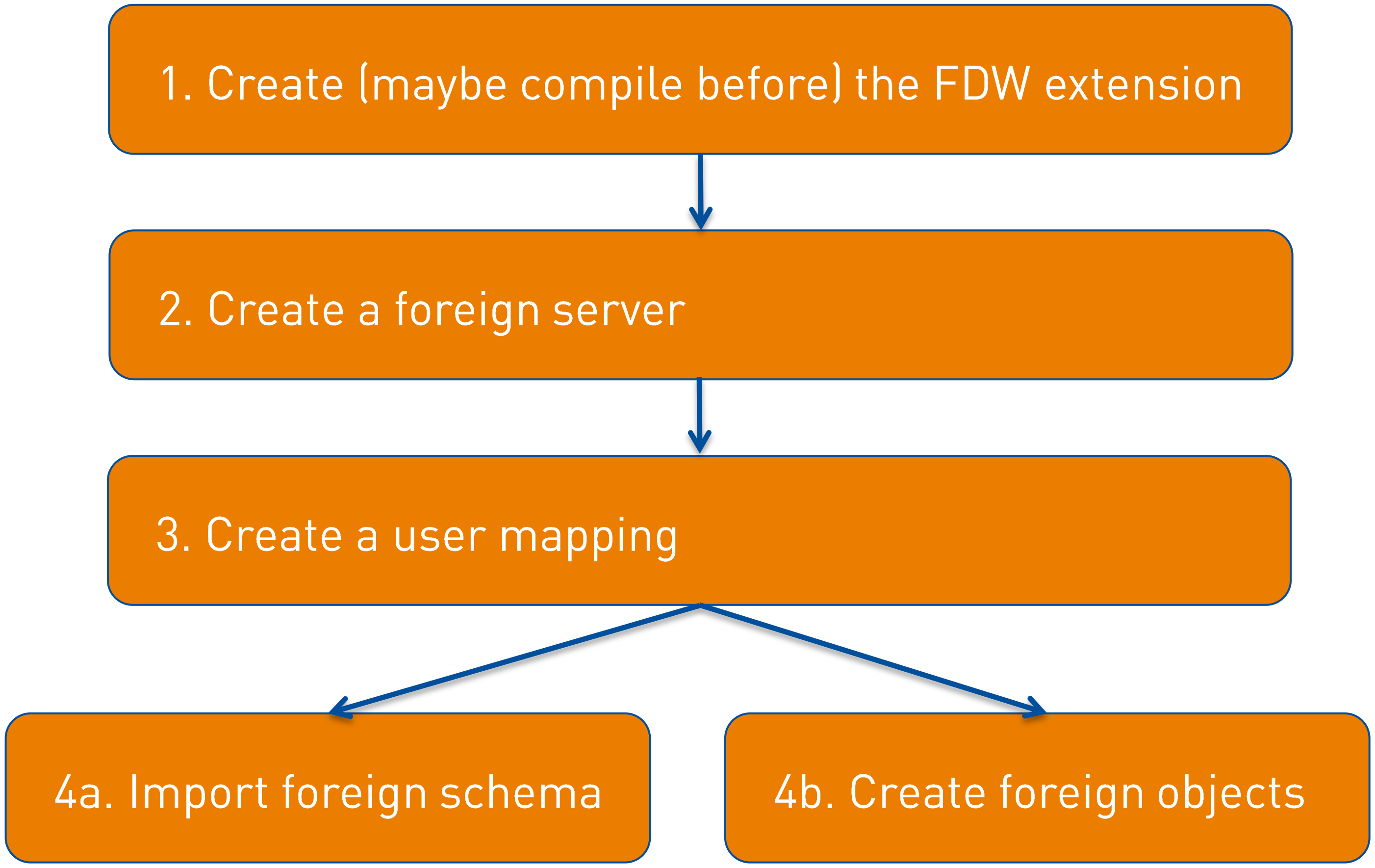
What PostgreSQL can offer
Now in 2016

There are more than 70 Foreign Data Wrappers, e.g. for accessing data in

- Oracle, MS SQL Server/Sybase, MySQL/MariaDB, Informix, Firebird
- Files on disk (csv, xml, json, tar, zip, ...)
- MongoDB, Cassandra, CouchDB, Redis
- Twitter
- LDAP
- GIT, RSS, IMAP, www
- Hadoop, Hive, HDFS
- Docker

https://wiki.postgresql.org/wiki/FDW?nocache=1

What PostgreSQL can offer
FDW setup


1. Create (maybe compile before) the FDW extension
2. Create a foreign server
3. Create a user mapping
4a. Import foreign schema
4b. Create foreign objects

What PostgreSQL can offer
Languages

There are many languages you can use to work on your data

- PL/Java
- PL/Perl
- PL/Python
- PL/R
- PL/pgSQL
- PL/Ruby
- PL/Scheme
- PL/sh
- PL/Tcl
- PL/v8

Choose the one that fits best for you, you are free in your choice

What PostgreSQL can offer
NoSQL – the best of both worlds

You can have NoSQL in PostgreSQL

- Schema less data types
  - json
  - jsonb
  - hstore

Index support on schema less data types

No drawbacks

- Full transactional schema changes
- Durability by default
- Point In Time Recovery

What PostgreSQL can offer
Analytic features

PostgreSQL provides many analytical features by default

- Aggregates
- Window functions
- Bitmap heap scans
- Tablespaces
- Partitioning
- Materialized Views
- Common table expressions
- BRIN Indexes
- Grouping sets, Rollup, Cube

What PostgreSQL can offer
Streaming replication

Hot standby databases can be used for read scaling

There can be multiple of these standby databases

You can use pgpool-II to load balance queries between the master and the slaves

- Writes go to the master
- Reads are balanced between the nodes

What PostgreSQL can offer
Configuration best practices

Use the latest version of PostgreSQL

- There are many improvements with each major release
- There is (usually) one major release every year
- 9.2, 9.3, 9.4, 9.5 are all major releases

Use a system with as many disks as you can

- Use SSDs whenever possible

Put your WAL (write ahead log) on separate disks

What PostgreSQL can offer
Configuration best practices

Usually there are not many connections to reporting databases

- max_connections

Pump your memory limits

- shared_buffers: 25% of the memory of the server
- work_mem: depends on the queries, 1GB or more
  - This is per operation, be careful
- maintenance_work_mem: at least 512M, 1GB, 2GB?
- temp_buffers: avoid sorts on disks, 1GB?
- effective_cache_size: 75% of the memory of the server

What PostgreSQL can offer
Configuration best practices

Partition your tables

- PostgreSQL supports table partitioning through table inheritance for a long time
- Partition according to your queries
  - By week?
  - By month?
  - By year?

Make use of tablespaces

- You can spread your data over various disks / mountpoints
- What are your biggest tables?
- Use temp_tablespaces for temporary tables
- Take care with backups when using tablespaces

What PostgreSQL can offer
Configuration best practices

If you can always rebuild your data turn of the following during bulk loads

> fsync

> synchronous_commit

> full_page_writes

Adjust checkpointing behavior during data loads

> checkpoint_segments: use a high setting during data load, 64 or more (<= PG 9.4)

> max_wal_size: use a high setting during data load (>= PG 9.5)

> checkpoint_timeout: use a high setting during data load

> autovacuum: turn it off and do manual vacuum after or during the data load

What PostgreSQL can offer
Configuration best practices

For data loading

- Use as many parallel processes as possible
  - Not more than the number of CPUs
- Add indexes and constraints after the load
- Do not commit too often, bulk operations are faster
- Use unlogged tables where appropriate
- Whenever possible truncate and insert instead of updating

Constraints, think about

- Do you really need foreign keys in the reporting database?
- Do you really need check constraints in the reporting database?
- Is the data that is coming in already consistent?

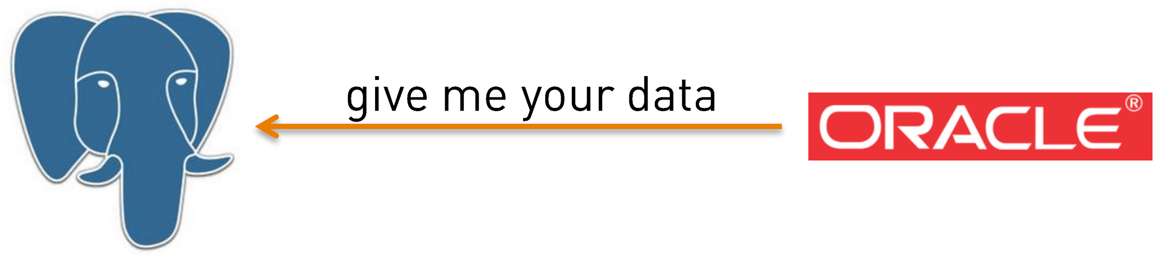
What PostgreSQL can offer
Now in 9.6 beta

From the release notes

- Parallel sequential scans
- Parallel joins
- Parallel aggregates
- Substantial performance improvements, especially in the area of improving scalability on many-CPU servers
- Elimination of repetitive scanning of old data by autovacuum
- If a GROUP BY clause includes all columns of a non-deferred primary key, as well as other columns of the same relation, those other columns are redundant and can be dropped from the grouping. This saves computation in many common cases.
- Improve aggregate-function performance by sharing calculations across multiple aggregates if they have the same arguments and transition functions
- Any many more: http://www.postgresql.org/docs/9.6/static/release-9-6.html

Sample setups

Sample setups
oracle_fdw



Sample setups
oracle_fdw

The oracle_fdw can be obtained in two ways

- From GitHub
  - https://github.com/laurenz/oracle_fdw
- From the PostgreSQL Extension Network
  - http://pgxn.org/dist/oracle_fdw

oracle_fdw requires the Oracle client libraries for being able to connect to an Oracle instance

- Oracle Instant Client is free and can be downloaded from
  - http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html
  - Use the latest version whenever possible

Sample setups
oracle_fdw

A sample setup using the Oracle Instant Client rpms

```
postgres@pg$ which pg_config
/u01/app/postgres/product/95/db_2/bin/pg_config
postgres@pg$ unzip oracle_fdw-1.4.0.zip
postgres@pg$ sudo yum localinstall oracle-instantclient12.1-basic-
12.1.0.2.0-1.x86_64.rpm oracle-instantclient12.1-devel-12.1.0.2.0-
1.x86_64.rpm
postgres@pg$ cd oracle_fdw-1.4.0
postgres@pg$ make
postgres@pg$ make install
```

Sample setups
oracle_fdw

Create the extension

```
postgres@pg$ psql postgres
psql (9.5.2 dbi services build)
Type "help" for help.

(postgres@[local]:4445) [postgres] > create extension oracle_fdw;
ERROR:  could not load library
"/u01/app/postgres/product/95/db_2/lib/oracle_fdw.so":
libclntsh.so.12.1: cannot open shared object file: No such file or
directory
Time: 148.873 ms
```

Why this?

```
postgres@pg$ ldd /u01/app/postgres/product/95/db_2/lib/oracle_fdw.so
        linux-vdso.so.1 =>  (0x00007fff085f2000)
        libclntsh.so.12.1 => not found
        libc.so.6 => /lib64/libc.so.6 (0x00007f59aa17f000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f59aa76b000)
```

Sample setups
oracle_fdw

libclntsh is the Oracle Client Code Library

This library must be in LD_LIBRARY_PATH for oracle_fdw to work

```
postgres@pg$ export
LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib/:$LD_LIBRARY_PATH
postgres@pg$ pg_ctl -D /u02/pgdata/PGREP/ restart -m fast
postgres@pg$ psql postgres
psql (9.5.2 dbi services build)
Type "help" for help.
(postgres@[local]:4445) [postgres] > create extension oracle_fdw;
CREATE EXTENSION
```

Remember to adjust the LD_LIBRARY_PATH in your PostgreSQL startup script

Sample setups
oracle_fdw

Setting up a connection to Oracle from here on is just a matter of a few commands

```
postgres=# create server oracle_srv foreign data wrapper oracle_fdw
options (dbserver '//192.168.22.42/DELLDVD' );
CREATE SERVER
postgres=# create user mapping for postgres server oracle_srv options
(user 'DS2', password 'ds2');
CREATE USER MAPPING
```

Create a new schema and import all foreign tables

```
postgres=# create schema ds2_oracle;
CREATE SCHEMA
postgres=# import foreign schema "DS2" from server oracle_srv into
ds2_oracle;
IMPORT FOREIGN SCHEMA
```

Sample setups
oracle_fdw

Everything is there

```
(postgres@[local]:4445) [postgres] > set search_path='ds2_oracle';
(postgres@[local]:4445) [postgres] > \d
                       List of relations
   Schema    |         Name          |     Type      |  Owner
-------------+-----------------------+---------------+----------
 ds2_oracle | categories             | foreign table | postgres
 ds2_oracle | cust_hist              | foreign table | postgres
 ds2_oracle | customers              | foreign table | postgres
 ds2_oracle | derivedtable1          | foreign table | postgres
 ds2_oracle | inventory              | foreign table | postgres
 ds2_oracle | orderlines             | foreign table | postgres
 ds2_oracle | orders                 | foreign table | postgres
 ds2_oracle | products               | foreign table | postgres
 ds2_oracle | reorder                | foreign table | postgres
(17 rows)
```

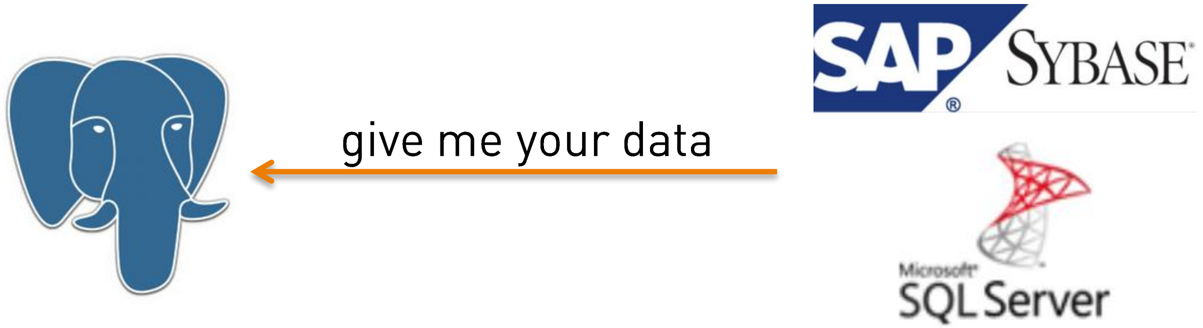
Sample setups
oracle_fdw

All the data is available (remotely)

```
(postgres@[local]:4445) [postgres] > select count(*)
                                      from orders;
  count
---------
 1200000
(1 row)

(postgres@[local]:4445) [postgres] > select count(*)
                                      from orderlines;
  count
---------
 5997620
(1 row)
```

Sample setups
tds_fdw – MS SQL Server & Sybase



Sample setups
tds_fdw – MS SQL Server & Sybase

The tds_fdw can be obtained from GitHub

> https://github.com/tds-fdw/tds_fdw

In addition you'll need a library that implements the "Tabular Data Stream" protocol

For Unix/Linux based systems there is FreeTDS

> http://www.freetds.org

Sample setups
tds_fdw – MS SQL Server & Sybase

As a first step install FreeTDS (either from your distribution's repository or from source)

```
postgres@pg$ wget ftp://ftp.freetds.org/pub/freetds/stable/freetds-patched.tar.gz
postgres@pg$ ls
freetds-patched.tar.gz
postgres@pg$ tar -axf freetds-patched.tar.gz
postgres@pg$ ./configure
postgres@pg$ make
postgres@pg$ sudo make install
```

Sample setups
tds_fdw – MS SQL Server & Sybase

In a second step verify that you are able to connect to a MS SQL Server instance by using tsql

There is a sample configuration where you can add your remote database instance

```
postgres@pg$ cat /usr/local/etc/freetds.conf
# A typical Microsoft server
[mssql]
    host = 192.168.22.102
    port = 1433
    database = ds2
    tds version = 7.3
```

Sample setups
tds_fdw – MS SQL Server & Sybase

Now you can test using plain FreeTDS

```
postgres@pg$ tsql -I /usr/local/etc/freetds.conf -S mssql -U ds2user -
P xxxxx -o v
locale is "LC_CTYPE=en_US.UTF-8;LC_NUMERIC=de_CH.UTF-
8;LC_TIME=en_US.UTF-8;LC_COLLATE=en_US.UTF-8;LC_MONETARY=de_CH.UTF-
8;LC_MESSAGES=en_US.UTF-8;LC_PAPER=de_CH.UTF-8;LC_NAME=de_CH.UTF-
8;LC_ADDRESS=de_CH.UTF-8;LC_TELEPHONE=de_CH.UTF-
8;LC_MEASUREMENT=de_CH.UTF-8;LC_IDENTIFICATION=de_CH.UTF-8"
locale charset is "UTF-8"
using default charset "UTF-8"
1> select count(*) from sys.databases;
2> go
using TDS version 7.3
5
(1 row affected)
using TDS version 7.3
```

Sample setups
tds_fdw – MS SQL Server & Sybase

Ready for the fdw stuff

```
postgres@pg$ git clone https://github.com/tds-fdw/tds_fdw.git
postgres@pg$ cd tds_fdw
postgres@pg$ PATH=$PGHOME:$PATH make USE_PGXS=1
postgres@pg$ sudo PATH=$PGHOME/bin:$PATH make USE_PGXS=1 install
postgres@pg$ psql postgres
postgres= create extension tds_fdw;
CREATE EXTENSION
postgres= create server mssql_svr foreign data wrapper tds_fdw options
( servername '192.168.22.102', port '1433',  database 'ds2',
tds_version '7.3');
postgres= CREATE USER MAPPING FOR postgres SERVER mssql_svr  OPTIONS
(username 'ds2user', password 'xxxxx');
postgres= CREATE schema ds2_mssql;
```

Sample setups
tds_fdw – MS SQL Server & Sybase

Import the schema

```
postgres= IMPORT FOREIGN SCHEMA ds2 FROM SERVER mssql_svr into
ds2_mssql;
ERROR:  foreign-data wrapper "tds_fdw" does not support IMPORT FOREIGN
SCHEMA
```

Not so good. This is not implemented right now, but it is tracked:

https://github.com/tds-fdw/tds_fdw/issues/75

No other choice than to create the tables one by another

Sample setups
tds_fdw – MS SQL Server & Sybase

Manually create the foreign tables

```
create foreign table ds2_mssql.orders
  ( orderid numeric not null
  , orderdate timestamp(0) without time zone not null
  , customerid numeric
  , netamount numeric(12,2) not null
  , tax numeric(12,2) not null
  , totalamount numeric(12,2) not null
  )
SERVER mssql_svr
OPTIONS (table 'dbo.orders', row_estimate_method 'showplan_all');
```

Sample setups
tds_fdw – MS SQL Server & Sybase

Manually create the foreign tables

```
create foreign table ds2_mssql.orderlines
  ( orderlineid numeric not null
  , orderid numeric not null
  , prod_id numeric not null
  , quantity numeric not null
  , orderdate timestamp(0) without time zone not null
  )
SERVER mssql_svr
OPTIONS (table 'dbo.orderlines', row_estimate_method 'showplan_all');
```

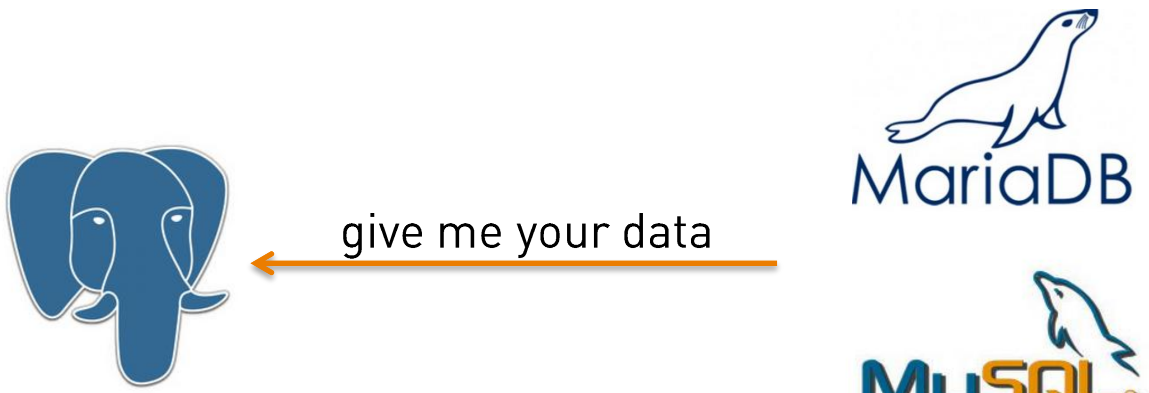
Sample setups
tds_fdw – MS SQL Server & Sybase

Check that you can access the remote data

```
postgres=# select count(*) from ds2_mssql.orders;
 count
--------
 200000
(1 row)

postgres=# select count(*) from ds2_mssql.orderlines;
 count
--------
 200000
(1 row)
```

Sample setups
mysql_fdw – MySQL/MariaDB



Sample setups
mysql_fdw – MySQL/MariaDB

The mysql_fdw can be obtained from GitHub

> https://github.com/EnterpriseDB/mysql_fdw

In addition you'll need the MySQL/MariaDB client and development libraries

For Unix/Linux based systems they should be available in the repository of your distribution

Sample setups
mysql_fdw – sample setup

As a first step install MySQL or MariaDB client and development libraries

```
postgres@pg$ sudo yum install -y mariadb-libs mariadb-devel
```

Make sure pg_config and mysql_config are in your $PATH

```
postgres@pg$ which pg_config
/u01/app/postgres/product/95/db_2/bin/pg_config
postgres@pg$ which mysql_config
/usr/bin/mysql_config
```

Sample setups
mysql_fdw – sample setup

Build and install the mysql_fdw

```
postgres@pg$ wget
https://github.com/EnterpriseDB/mysql_fdw/archive/master.zip
postgres@pg$ unzip mysql_fdw-master.zip
postgres@pg$ cd mysql_fdw-master
postgres@pg$ make USE_PGXS=1
postgres@pg$ make USE_PGXS=1 install
```

The usual PostgreSQL fdw stuff

```
postgres= create extension mysql_fdw;
CREATE EXTENSION
CREATE SERVER mysql_srv FOREIGN DATA WRAPPER mysql_fdw  OPTIONS (host
'192.168.22.41', port '3306');
CREATE USER MAPPING FOR postgres SERVER mysql_srv OPTIONS (username
'web', password 'web');
CREATE schema ds2_mysql;
```

Sample setups
mysql_fdw – sample setup

Adjust your MySQL configuration to allow connections from the PostgreSQL instance

```
MariaDB [DS2]> grant all on *.* TO 'web'@'192.168.22.39' identified by
"web";
Query OK, 0 rows affected (0.00 sec)

MariaDB [DS2]> SHOW GLOBAL VARIABLES LIKE 'PORT';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| port          | 3306  |
+---------------+-------+
1 row in set (0.00 sec)
```

Sample setups
mysql_fdw – sample setup

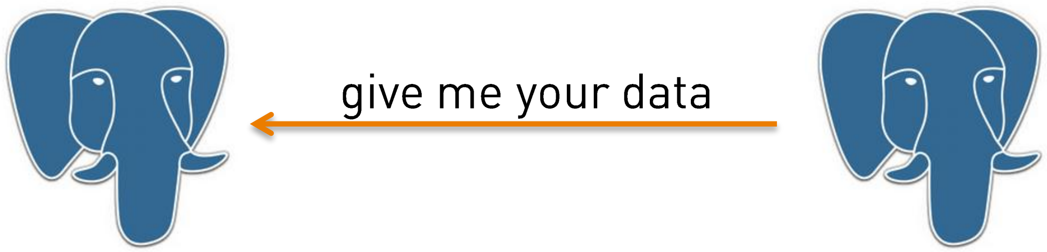
Import the schema

```
postgres= import foreign schema "DS2" from server mysql_srv into
ds2_mysql;
IMPORT FOREIGN SCHEMA
```

Check the remote data

```
postgres= select count(*) from ds2_mysql."ORDERS";
 count
--------
 120000
(1 row)
postgres= select count(*) from ds2_mysql."ORDERLINES";
 count
--------
 120000
(1 row)
```

Sample setups
postgresql_fdw



Sample setups
postgresql_fdw – sample setup

Easy, this one is available by default

```
postgres= create extension postgres_fdw;
CREATE EXTENSION
postgres= CREATE SERVER postgres_srv FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host '192.168.22.40', port '5432', dbname 'ds2');
CREATE SERVER
postgres= CREATE USER MAPPING FOR postgres SERVER postgres_srv OPTIONS
(user 'ds2', password 'ds2');
postgres= create schema ds2_postgresql;
postgres= import foreign schema "public" from server postgres_srv into
ds2_postgresql;
IMPORT FOREIGN SCHEMA
```

Sample setups
postgresql_fdw – sample setup

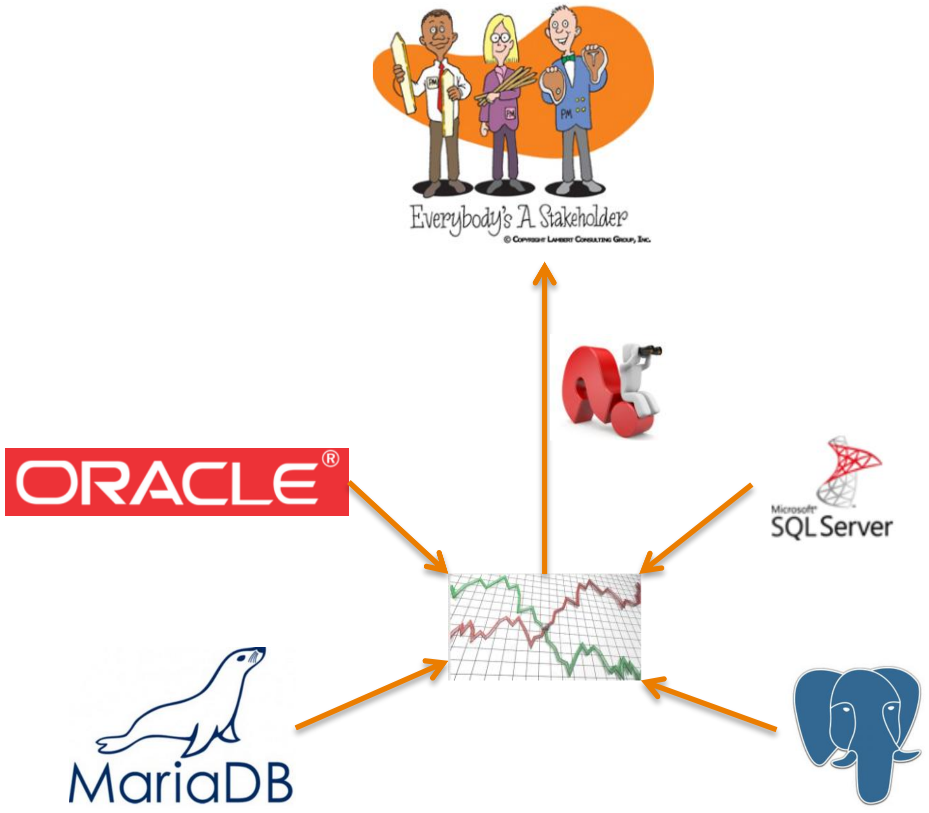
Check the remote data

```
postgres= select count(*) from ds2_postgresql.orders;
  count
---------
 1200000
(1 row)

postgres= select count(*) from ds2_postgresql.orderlines;
  count
---------
 5997620
(1 row)
```

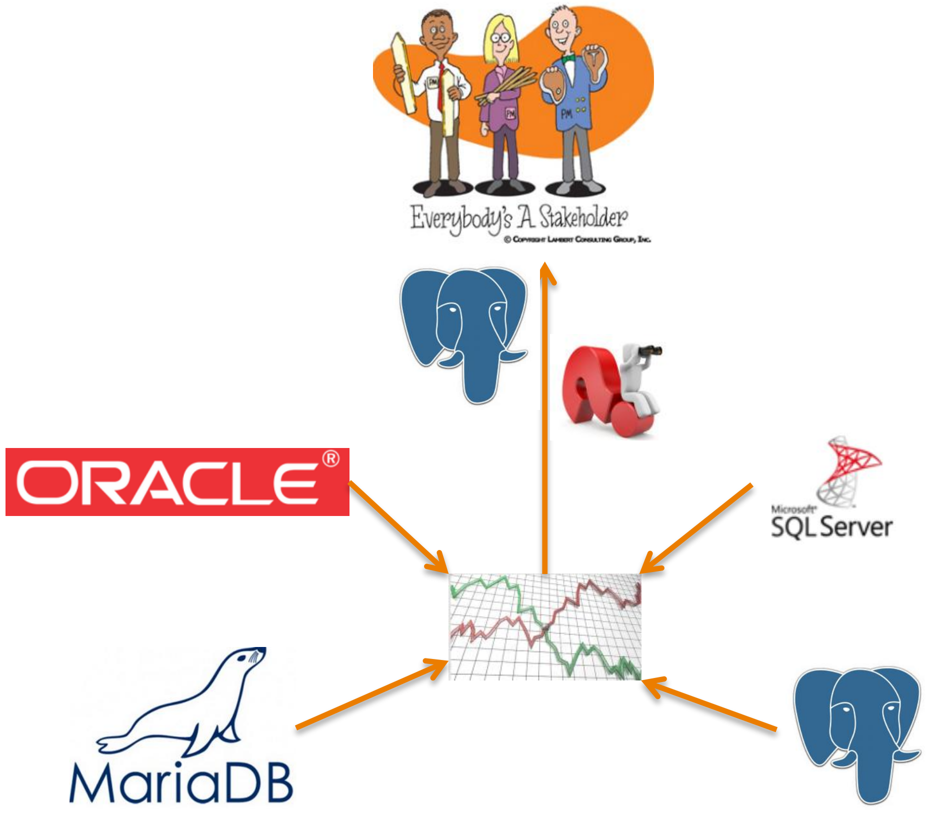
Sample setups
Complete picture

The test setup is exactly this


Everybody's A Stakeholder
© Copyright Lambert Consulting Group, Inc.
ORACLE
Microsoft SQL Server
MariaDB

Sample setups
Complete picture

+ this


Everybody's A Stakeholder
© Copyright Lambert Consulting Group, Inc.
ORACLE®
Microsoft SQL Server
MariaDB

Sample setups
cstore_fdw

Wasn't there something else?

You want columnar storage for PostgreSQL?

https://github.com/citusdata/cstore_fdw

It is implemented as a FDW

Does currently not support DELETE, UPDATE and single row inserts

Supports compression

Sample setups
cstore_fdw

Build and install

```
postgres@pg$ sudo yum install protobuf-c-devel
postgres@pg$ wget
https://github.com/citusdata/cstore_fdw/archive/master.zip
postgres@pg$ unzip master.zip
postgres@pg$ cd cstore*
postgres@pg$ which pg_config
postgres@pg$ make
postgres@pg$ sudo make install
```

Sample setups
cstore_fdw

Create the extension

```
postgres@pg$ psql postgres
postgres= alter system set shared_preload_libraries='cstore_fdw';
ALTER SYSTEM
(postgres@[local]:4445) [postgres] > \q
postgres@pg$ pg_ctl -D $PGDATA restart -m fast
postgres@pg$ psql postgres
(postgres@[local]:4445) [postgres] > create extension cstore_fdw;
CREATE EXTENSION
(postgres@[local]:4445) [postgres] > create server cstore_server
foreign data wrapper cstore_fdw;
```

More on the extension in the demo

Demo

Demo
The Dell DVD store sample schema

All databases attached to the PostgreSQL instance hold the Dell DVD store sample schema with different amounts of data

http://linux.dell.com/dvdstore

"The Dell DVD Store is an open source simulation of an online ecommerce site with implementations in Microsoft SQL Server, Oracle, MySQL and PostgreSQL along with driver programs and web applications"

I will only use the ORDERS and ORDERLINES tables for this short demo

Demo
The list of extensions

These are the extensions installed

```
(postgres@[local]:4445) [postgres] > \dx
     Name      |     Version
--------------+----------------
 cstore_fdw    | 1.4
 mysql_fdw     | 1.0
 oracle_fdw    | 1.1
 plpgsql       | 1.0
 postgres_fdw | 1.0
 tds_fdw       | 2.0.0-alpha.1
(6 rows)
```

Demo
The list schemas

The four schemas holding the foreign tables

```
(postgres@[local]:4445) [postgres] > \dn ds2*
      List of schemas
      Name        |  Owner
-----------------+----------
 ds2_mssql        | postgres
 ds2_oracle       | postgres
 ds2_postgresql | postgres
 ds2_mysql        | postgres
```

Demo
One additional schema to hold all data locally

The combined orders and order lines will be in a separate schema

```
(postgres@[local]:4445) [postgres] > create schema ds2_combined;
CREATE SCHEMA
```

So what can we do now?

Demo
Local tables to hold the data

We could create combined big "order" and "orderlines" tables

```
(postgres@[local]:4445) [postgres] > create table ds2_combined.orders
(like ds2_postgresql.orders);
(postgres@[local]:4445) [postgres] > create table
ds2_combined.orderlines (like ds2_postgresql.orderlines);
```

Demo
Loading the foreign data

… and then get all the data in one (or two) batch(es):

```
BEGIN;
  insert into ds2_combined.orders select * from ds2_mssql.orders;
  insert into ds2_combined.orders select * from ds2_mysql."ORDERS";
  insert into ds2_combined.orders select * from ds2_oracle.orders;
  insert into ds2_combined.orders select * from ds2_postgresql.orders;
END;
```

Demo
Loading the foreign data

... and then get all the data in one (or two) batch(es):

```
BEGIN;
  insert into ds2_combined.orderlines
         select * from ds2_mssql.orderlines;
  insert into ds2_combined.orderlines
         select * from ds2_mysql."ORDERLINES";
  insert into ds2_combined.orderlines
         select * from ds2_oracle.orderlines;
  insert into ds2_combined.orderlines
         select * from ds2_postgresql.orderlines;
END;
```

Demo
Loading the foreign data

How much data do we have now?

```
select *   from
pg_size_pretty(pg_total_relation_size('ds2_combined.orders'));


 pg_size_pretty
----------------
 171 MB
(1 row)


select *   from
pg_size_pretty(pg_total_relation_size('ds2_combined.orderlines'));
 pg_size_pretty
----------------
 657 MB
(1 row)
```

Demo
Working with the former foreign data

How much data do we have now?

```
select count(*)
  from ds2_combined.orders;

  count
---------
 2640000
(1 row)

select count(*)
  from ds2_combined.orderlines;

  count
----------
 13196328
(1 row)
```

Demo
Working with the former foreign data

Now that the data is locally available you can do whatever you want with it, eg:

```
select sum(netamount),avg(netamount)
  from ds2_combined.orders;
     sum       |          avg
--------------+----------------------
 528109733.63 | 200.0415657689393939
(1 row)


722.655 ms
```

Or transform first, then report on it

Demo
Working with the former foreign data

Now the data is locally available and you can do what ever you want with it, eg:

```
select sum(quantity)
     , avg(quantity)
     , max(quantity)
  from ds2_combined.orderlines;

Time: 1678.482 ms
```

Demo
Using cstore_fdw

To speed up processing we can use cstore_fdw

```
create foreign table ds2_combined.cs_orders
    ( orderid int, orderdate date
    , customerid int
    , netamount numeric
    , tax numeric
    , totalamount numeric)
SERVER cstore_server OPTIONS(compression 'pglz');

create foreign table ds2_combined.cs_orderlines
    ( orderlineid int
    , orderid integer
    , prod_id integer
    , quantity smallint
    , orderdate date)
SERVER cstore_server OPTIONS(compression 'pglz');
```

Demo
Using cstore_fdw

Load the cstore tables (or only some for which you know you'll benefit from cstore)

```
insert
  into ds2_combined.cs_orders
select *
  from ds2_combined.orders;

insert
  into ds2_combined.cs_orderlines
select *
  from ds2_combined.orderlines;
```

Demo
Using cstore_fdw

What about performance?

```
select sum(netamount),avg(netamount)
  from ds2_combined.cs_orders;

     sum       |          avg
--------------+----------------------
 528109733.63 | 200.0415657689393939
(1 row)

Time: 566.816 ms
```

Demo
Using cstore_fdw

What about performance?

```
select sum(quantity)
     , avg(quantity)
     , max(quantity)
  from ds2_combined.cs_orderlines;

   sum    |         avg          | max
----------+--------------------+-----
 26398439 | 2.0004382279676589 |   3
(1 row)

Time: 1488.931 ms
```

Demo
Using cstore_fdw

What about performance?

```
select sum(quantity)
     , avg(quantity)
     , max(quantity)
  from ds2_combined.cs_orderlines;

   sum    |        avg         | max
----------+--------------------+-----
 26398439 | 2.0004382279676589 |   3
(1 row)

Time: 1488.931 ms
```

Demo
Using cstore_fdw

What about size?

```
select * from
pg_size_pretty(pg_total_relation_size('ds2_combined.cs_orders'));


 pg_size_pretty
----------------
 0
(1 row)


select *   from
pg_size_pretty(pg_total_relation_size('ds2_combined.cs_orderlines'));
 pg_size_pretty
----------------
 0
(1 row)
```

Cool, we do not require any on disk storage :)

Demo
Using cstore_fdw

What about size?

```
…/PGREP/cstore_fdw/13294/ [PGREP] ls -lha
total 134M
drwx------. 2 postgres postgres   72 Jun 15 08:39 .
drwx------. 3 postgres postgres   18 Jun 10 16:27 ..
-rw-------. 1 postgres postgres  51M Jun 15 08:39 139452
-rw-------. 1 postgres postgres  329 Jun 15 08:39 139452.footer
-rw-------. 1 postgres postgres  84M Jun 15 08:39 139455
-rw-------. 1 postgres postgres 1.4K Jun 15 08:39 139455.footer
```

The traditional tables are 171M and 657 MB

Compression is quite well

Demo
Materialized views

Materialized views can also be used to load the data

```
create materialized view ds2_combined.mv_orders as
  select * from ds2_mssql.orders
  union all
  select * from ds2_mysql."ORDERS"
  union all
  select * from ds2_oracle.orders
  union all
  select * from ds2_postgresql.orders
  with no data;
refresh materialized view ds2_combined.mv_orders with data;
```

Core message

Core message

PostgreSQL is very well prepared to integrate data from various sources using Foreign Data Wrappers

Many FDWs exist for many data sources

Tweaking the configuration to speed up data loading is highly recommended

Use cstore_fdw if you are short on disk space and can either reload or append the data regularly without the requirement for updating

Core message

But wait

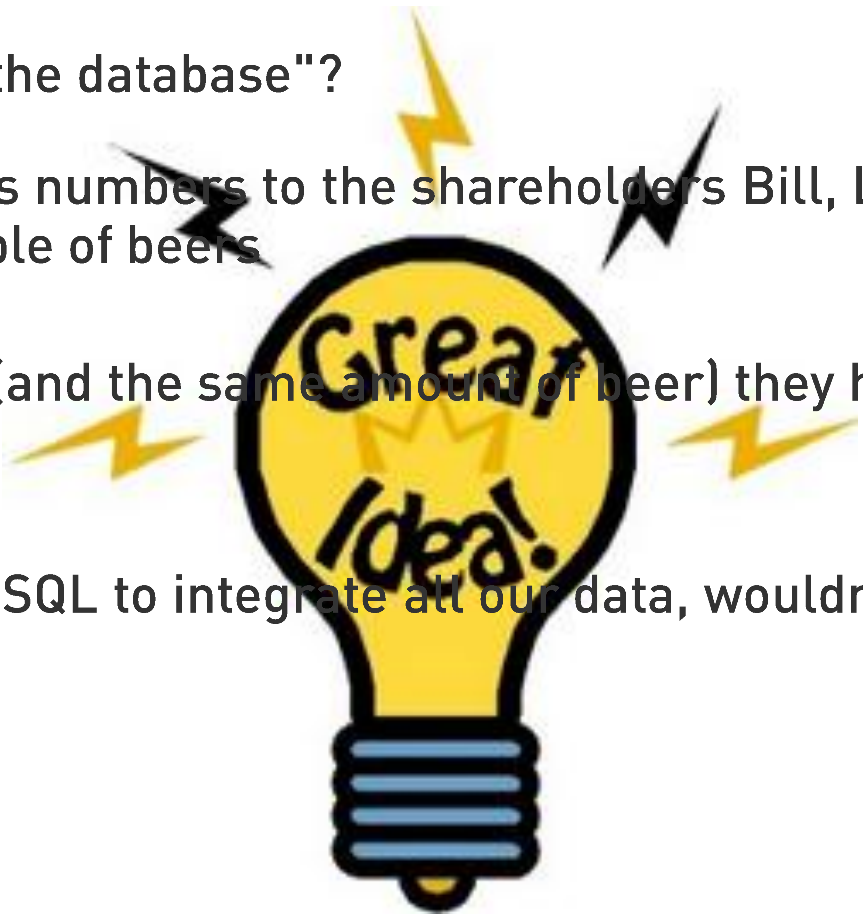
...what happened to the "the database"?

Having delivered the sales numbers to the shareholders Bill, Larry and Maria went for a couple of beers

Almost at the same time (and the same amount of beer) they had exactly the same idea:

When we can use PostgreSQL to integrate all our data, wouldn't it make sense

...



Any questions? Please do ask

Daniel Westermann
Senior Consultant
Open Infrastructure Technology Leader

+41 79 927 24 46
daniel.westermann@dbi-services.com

We look forward to working with you!