PostgreSQL migration guide: supporting the decision

Socle Interministériel du Logiciel Libre,

PGGTIE PostgreSQL.fr

07/08/2021

1 Versions

History of this document versions:

Version	Date	Comments
0.1	18/12/2013	First draft
1.0	19/03/2015	Last corrections and publication
1.0.1	30/09/2015	Russian translation by I. Panchenko and O. Bartunov
2.0	10/12/2019	Update by the PGGTIE
2.0.1	08/06/2020	Removed DINSIC mention
3.0.0	XX/YY/2021	English version and licence change

2 Contributors

- Bruce BARDOU - DGFiP
- Barek BOUTGAYOUT - MASS
- Amina CHITOUR - MENMESR
- Alain DELIGNY - MEDDE
- Yohann MARTIN - Maif
- Alain MERLE - MEDDE
- Anthony NOWOCIEN – Société Générale
- Loïc PESSONNIER - MINDEF
- Marie-Claude QUIDOZ - CNRS
- Laurent RAYMONDEAU - MENESR

3 Introduction

This guide was produced within the framework of the « Ayrault » circular of September 19, 2012 on the use of free software in administration, which led to the creation of the inter-ministerial « Socle Interministériel du Logiciel Libre » (SILL). This guide presents the implementation of PostgreSQL, compared to commercial solutions. Its objective is to answer the questions of project owners and project managers for the implementation of PostgreSQL as a replacement for a commercial solution. This guide aims, without going into the details of the technical implementation, to demonstrate the benefits of PostgreSQL by describing the integration, security and robustness mechanisms.

This document was authored collaboratively by Mimprod[1] and PGGTIE[2].

[1] Mimprod: https://www.mim-libre.fr/mimprod-outils-de-production/
[2] PGGTIE: https://www.postgresql.fr/entreprises/english

4 Satisfying requirements

4.1 Use cases

PostgreSQL is an SQL compliant database management system, typically used in transactional applications to ensure data persistence, much like similar commercial products (Oracle, DB2, Informix, MS SQL, Sybase, etc).

There is a PostgreSQL extension for geographic objects (PostGIS[1]) that conforms to the Open Geospatial Consortium (OGC) Standards. It supports JSON format data, consisting of key-value pairs, allowing it to also satisfy the use cases of NoSQL-like solutions. PostgreSQL can also be used in the field of business intelligence as a data warehouse, in conjunction with reporting tools (BusinessObjects, Pentaho, etc). It also has functionality to support full-text search[2].

Many free software packages are natively based on PostgreSQL (document management systems (DMS), rules engines, collaboration software, supervision software, etc). PGGTIE has written an open letter[3] to vendors that do not yet support PostgreSQL.

[1] PostGIS: https://postgis.net/
[2] Full-text search: https://www.postgresql.org/docs/current/textsearch.html
[3] Open letter: https://www.postgresql.fr/entreprises/20171206_open_letter_to_software_vendors

PostgreSQL also supports background processing, such as batch or deferred processing.

4.2 Data access

PostgreSQL conforms[4] to the SQL:2016 standard, the common query language of many RDBMSs. Migration is therefore facilitated between RDBMSs respecting the standard.

4.3 Security requirements

PostgreSQL meets security needs in terms of availability, integrity, confidentiality and traceability (information security[5]).

Encryption is standard and several strong authentication methods (like SCRAM and Kerberos) are possible. Cryptographic requirements are covered by additional modules, in particular *pgcrypto* [6]. At present, it is not possible to encrypt an entire instance, but work is ongoing (see wiki on Transparent Data Encryption [7]).

PostgreSQL has a very responsive community that provides security patches and manages component obsolescence. It is recommended that you apply minor patches as soon as possible. A major version is released every year and supported [8] for 5 years.

[4] SQL standard compliance: https://www.postgresql.org/docs/current/features.html
[5] Information security: https://en.wikipedia.org/wiki/Information_security
[6] pgcrypto: https://www.postgresql.org/docs/current/pgcrypto.html
[7] Transparent Data Encryption: https://wiki.postgresql.org/wiki/Transparent_Data_Encryption
[8] Support and EOL: https://www.postgresql.org/support/versioning/

PostgreSQL guarantees the integrity of the data handled even in the event of an incident by respecting the properties of atomicity, consistency, isolation and durability (ACID [9]).

PostgreSQL natively offers the mechanisms to meet confidentiality and rights management needs through roles [10]. Very fine granularity can be achieved, even at the row level (Row Security Policies [11]).

4.4 Migration cases

The migration of a system represents a significant cost, consisting mainly of:

- data migration, even if there are a number of tools to perform this migration, depending on the source RDBMS;
- changes needed in the source code of the application. The need for changes depends on the use of any functionalities that are specific to the existing product (like stored procedures, triggers, and non-standard functions) and the use of an abstraction layer (like Hibernate);
- steps to guarantee equivalent functionalities, that no error was introduced during the migration (regression tests).

As such, migrations to PostgreSQL are often carried out during a functional evolution of the application. The cost of migration is integrated into the project as a whole, particularly in terms of technical and functional acceptance.

[9] ACID properties: https://en.wikipedia.org/wiki/ACID
[10] Roles: https://www.postgresql.org/docs/current/user-manag.html
[11] Row Security Policies: https://www.postgresql.org/docs/current/ddl-rowsecurity.html

However, an ambitious global migration plan can be arranged. PostgreSQL can become the preferred RDBMS for any new application. This, for example, is the case for many PGGTIE companies, for which the use of another RDBMS must be justified (exemption request).

5 Integration of PostgreSQL on technical platforms

5.1 Technical platforms

5.1.1 Processor architectures

PostgreSQL runs on the following processor architectures[1]: x86, x86_64, IA64, PowerPC, PowerPC 64, S/390, S/390x, Sparc, Sparc 64, ARM, MIPS, MIPSEL and PA-RISC.

5.1.2 Operating systems

PostgreSQL works on most operating systems. Binary versions exist for the Red Hat family (including CentOS / Fedora / Scientific), the Debian GNU / Linux family and its derivatives, the Ubuntu Linux family and its derivatives, SuSE, OpenSuSE, Solaris, Windows, macOS, FreeBSD, OpenBSD etc.

For distributions using .rpm files, it is possible to use the community repository https://yum.postgresql.org/. For those using .deb, we can refer to https://wiki.postgresql.org/wiki/Apt. Source code is also available online[2].

[1] Supported platforms: https://www.postgresql.org/docs/current/supported-platforms.html
[2] Source code: https://www.postgresql.org/ftp/source/

5.1.3 Compatibility with virtualization/containerization

PostgreSQL is compatible with VMWare and KVM in particular. Virtualization brings resiliency benefits. It is also possible to make PostgreSQL work with containers like Docker.

5.1.4 Compatibility with storage technologies

PostgreSQL works on storage arrays (SAN, NAS, etc) but like any DBMS, the attachment must be permanent and block level. Storage virtualization is completely transparent to the DBMS.

Communities provide recommendations on the types of filesystem to use (ext4, XFS, ZFS, etc). The use of NFS is not recommended.

5.1.5 Compatibility with backup technologies

PostgreSQL supports the usual backup methods:

- cold backup : file system level backup - the database must be offline ;
- hot backup : basic filesystem level backup supported. No possibility to perform an incremental backup natively. Tools like Barman[3] and pgBackRest[4] allow this, though ;
- continuous backup: file system level backup supported, allowing *Point In Time Recovery* (PITR). It is possible to use the tools mentioned above, or *pitrery* ;
- SQL dump : the database can be exported in SQL format, optionally compressed, in a format specific to PostgreSQL.

[3] Barman: https://www.pgbarman.org/
[4] pgBackRest: https://pgbackrest.org/

PostgreSQL is compatible with other backup tools as well (TINA, TSM, Veeam, Netbackup, Bacula, etc).

5.2 Security

5.2.1 User identification management

PostgreSQL provides authentication mechanisms and manages the attribution of privileges (GRANT, etc). It also allows separation through database schemas.

5.2.2 Confidentiality guarantees and use of encryption

Encryption is possible via the *pgcrypto* module, allowing column encryption by public key and private key.

5.2.3 Ensure traceability and logging

The traceability of events in PostgreSQL is ensured by transaction logs (called *WAL*, or Write-Ahead Logs):

- logging of engine operations (starting, stopping, etc) ;
- access logging (queries, user access, errors, etc).

5.2.4 Audit

While a significant amount of information can be recorded in the logs[5], it may be necessary in some cases to implement a more extensive audit policy. The most successful tool to do this is *PGAudit*[6]. In particular, it will make it possible to understand the context in which an operation was performed. An example implementation by the United States Defense Information Systems Agency (DISA) is available[7].

[5] Log configuration: https://www.postgresql.org/docs/current/runtime-config-logging.html
[6] PGAudit: https://www.pgaudit.org/
[7] DISA Security Technical Implementation Guide: https://www.crunchydata.com/disa-postgres-security-guide/disa-postgres-security-guide-v1.pdf

6 Scalability and resilience

6.1 Clustering and replication mechanisms

Definitions

Scalability and Elasticity

Scalability is the ability of PostgreSQL to adapt to an order of magnitude change in demand (up or down).

The elasticity of a system is the ability of a system to automatically increase or decrease its resources as needed.

Resilience and Robustness (or stability)

Resilience is the ability of a system or network architecture to continue to function in the event of failure.

Robustness is the quality of a system that does not crash, that functions well even in a hostile (penetration, DoS, etc) or abnormal (e.g. incorrect inputs) environment.

Cluster

A cluster is a group of servers (or « compute farm ») sharing common storage. A cluster provides high availability and load balancing functions.

Replication

Replication is a process of sharing information to ensure data consistency between multiple redundant data sources, to improve reliability, fault tolerance, or availability.

As with the vast majority of RDBMSs, it is easy to increase server resources (CPU, RAM, disk) to vertically scale the Postgres service. Certain solutions also allow you to perform sharding (distribute data over several instances, such as *Citus*, *PostgreSQL-XL*, etc).

Several clustering and replication modes are supported by PostgreSQL, with advantages and disadvantages to each. These considerations are independent of the choice of DBMS though; all products are affected :

- « Active-Passive » mode : a primary database is read / write, another database, the « standby », is synchronized in the background. Replication can be asynchronous (better performance) or synchronous (better security).

- « Partial Active-Active » or « Read/Write - Read » mode: a primary database is read / write, the « standby » databases are read-only.

Like other DBMSs, PostgreSQL does not allow transactions on several servers in « Active-Active » or « Read/Write - Read/Write » mode. You have to go through third-party contributions, such as 2ndQuadrant's BDR[1] (Bidirectional Replication), that offer this functionality. However, this is a proprietary development on a modified version of PostgreSQL.

[1] BDR: https://www.2ndquadrant.com/en/resources/postgres-bdr-2ndquadrant/

6.2 Resilience driven by the PostgreSQL engine

PostgreSQL provides replication mechanisms for setting up an « active-passive » or « partial active-active" cluster.

There are two main types of replication, namely physical replication (eg *streaming replication* [2] based on modification of data blocks) and logical replication (based on modification of database objects). For high availability requirements, we will focus on physical replication. Logical replication is intended for other use cases, like the selective supply of another database (e.g. datawarehouse), an upgrade with the aim of minimizing downtime, etc.

6.3 Resilience driven by the storage infrastructure

PostgreSQL is compatible with clustering driven by the platform, independently of the DBMS. Writing to a node is performed by PostgreSQL and replication to a second node is handled by the storage array.

To restart the engine on the second node, there must be an interruption of service.

6.4 Business continuity and disaster recovery plans

Business continuity and disaster recovery plans must be supported by the application. Recommendations for the choice of scenarios are made according to the recovery and continuity needs, independently of the DBMS.

[2] Streaming replication https://www.postgresql.org/docs/current/warm-standby.html#STREAMING-REPLICATION

PostgreSQL provides the tools for integration into recovery and continuity plans.

7 Development with PostgreSQL

7.1 Data types

PostgreSQL offers standard data types (alphanumeric, date, time, BLOB, etc) as well as more complex data types (geospatial, XML, JSON, etc). We refer to the documentation[1].

It is recommended to use ISO 8601[2] for dates (YYYY-MM-DD). For data including date and time, the *timestamp with time zone* data type should be used.

The *Binary Large OBject* (BLOB) data type allows storage of binary form content in the database (office files, PDFs, photos, audio, video, multimedia, etc).

PostgreSQL provides two ways of storing binary data :

- directly in the table using the *bytea* type;
- in a separate table with a special format that stores binary data and refers to it by a value of type *oid* in the main table using the Large Object function.

The *bytea* data type, which can hold up to 1 GB, is not suitable for storing very large amounts of binary data.

[1] Data types: https://www.postgresql.org/docs/current/datatype.html
[2] ISO 8601: https://en.wikipedia.org/wiki/ISO_8601

The Large Object function is best suited for storing very large volumes. However, it has its own limits :

- deleting a record does not delete the associated binary data. This requires a maintenance action on the database;
- any connected user can access binary data even if they do not have rights on the main database.

The use of BLOB fields is not recommended if there is no need to search the information.

7.2 Database schemas and structures

Table partitioning is implemented as standard[3]. It is worth noting a significant improvement with V10, which allows declarative partitioning. It was previously achieved through to the notion of inheritance and the writing of triggers. Version 11 continues the improvements, in particular on the possibility of having primary/foreign keys on the partitions, partition-pruning, the ability to define a default partition, etc.

In practice, one should avoid creating more than 500/1000 partitions for a single table. However, version 12 brings very appreciable performance gains in this area.

7.3 Development specifics

PostgreSQL complies with the SQL 2016 standard.

[3] Partitioning: https://www.postgresql.org/docs/current/ddl-partitioning.html

The documentation[4] provides links to see which functions are actually covered and which are not yet: overall the coverage of the standard is very good, although it is surprising that the `MERGE` statement is not supported. PostgreSQL does, however, have an `INSERT ... ON CONFLICT`[5] statement that meets a similar need.

As such, PostgreSQL does not require significant query syntax training. However, developers' habits regarding the use of specific functions and particular syntaxes for joins in their usual DBMS will require support. For example, the date functions are often specific to each DBMS (see Appendices by DBMS).

It is possible to create custom function libraries to simulate the specific functions of other DBMSs, facilitating both migration and training.

The choice of character set is done according to the applications. UTF-8 is recommended for the whole chain. ISO can also be used if needed.

Be careful with the default options when installing PostgreSQL: if no options are specified, *initdb* and `CREATE DATABASE` use the server's language configuration, either LATIN9 (ISO 8859-15) or, failing that, ASCII (https://www.postgresql.org/docs/current/app-initdb.html). To use UTF-8, you must specify it when creating the database.

PostgreSQL allows the use of materialized views. While they can be refreshed on demand (and with minimal blocking via the `CONCURRENTLY` clause), it is currently not natively possible to continuously update them.

[4] SQL conformance: https://www.postgresql.org/docs/current/features.html
[5] INSERT ON CONFLICT: https://www.postgresql.org/docs/current/sql-insert.html#SQL-ON-CONFLICT

7.4 API and access modes

7.4.1 Client interfaces

PostgreSQL provides several client interfaces for accessing data.

Name	Language	Comments	Address
DBD::Pg	Perl	Perl DBI driver	https://metacpan.org/release/DBD-Pg
JDBC	Java	Type 4 JDBC driver	https://jdbc.postgresql.org/
libpqxx	C++	New-style C++ interface	http://pqxx.org/development/libpqxx/
Npgsql	.NET	.NET data provider	https://www.npgsql.org/
pgtclng	Tcl	Tcl interface	https://sourceforge.net/projects/pgtclng
psqlODBC	ODBC	ODBC driver	https://odbc.postgresql.org/
psycopg	Python	DB API 2.0-compliant	https://initd.org/psycopg/

7.4.2 Abstraction layer

The use of an “ORM” type abstraction layer (object/relation mapping) is supported.

7.4.3 Connection pooling

A connection manager is strongly recommended for several hundred or more simultaneous connections. As PostgreSQL uses a process for each new session, establishing and closing a connection can become slow in relation

to query execution time. *PgBouncer*[6] is stable and performant – we recommend it. *Pgpool-II*[7] is more complex to use and is recommended for load-balancing.

7.5 Stored procedures, functions, and triggers

A trigger is an action (stored procedure or SQL query) executed on an event. Triggers have many uses, including implementation of traces related to the application (error management, activity, etc).

A stored procedure is a program stored in the database. This is usually a function, and version 11 allows you to actually define a procedure, invoked via the CALL command.

The use of stored procedures and triggers is discouraged in order to avoid adherence and facilitate portability. Furthermore, the business logic should not be embedded in the database.

Functions can be used to extend the use of pre-existing data types.

PostgreSQL allows you to write functions and procedures in languages other than SQL and C. These other languages are generically called procedural languages. There are currently four procedural languages in the standard PostgreSQL distribution:

- PL/pgSQL: SQL procedural language ;
- PL/Tcl: Tcl procedural language ;
- PL/Perl: Perl procedural language ;
- PL/Python: Python procedural language.

[6] PgBouncer: https://www.pgbouncer.org/
[7] Pgpool-II: https://www.pgpool.net/mediawiki/index.php/Main_Page

There are other procedural languages that are not included in the main distribution (PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme, PL/sh). Other languages can be defined by users, but the process can be a bit complex.

7.6 Foreign Data Wrappers

These are extensions that allow PostgreSQL to communicate with other data sources. The data sources can be relational databases (PostgreSQL, MySQL, Oracle, etc), NoSQL databases (CouchDB, MongoDB, etc), CSV files, or LDAP directories. The fact that the data comes from other sources is transparent to the end user.

Some FDWs, like Oracle and MySQL, have read/write capabilities; others only read.

For a complete list, see the wiki[8].

[8] FDW: https://wiki.postgresql.org/wiki/Foreign_data_wrappers

8 Administration

8.1 Monitoring and explotation tools for PostgreSQL

8.1.1 PgAdmin

PgAdmin is an client-server administration tools. It is an open source software under the PostgresSQL licence. It is available on all plateforms. It is included in Windows installation. You can download it there : https://www.pgadmin.org/download/.

This is a gui tool which make it usuable for every one. Here is the most important functionnality : grafic query tool, display customization, addons (for example PostGIS, Shapefile and DBF loader) and we can find a script langage. It can also display the query plan result (comman `EXPLAIN`).

We can also install it in server mode to avoid installation on a workstation.

This software is update frequently.

8.1.2 psql

Psql tool is a command line utility, with it we can write sql query, display database schema, import and export datas. It is include in all PostgreSQL distribution and we recommand it as the default client.

To have help for sql commands, we can use `\help`. From version 12, this command will also give the documentation link of the instruction. For meta-command's help, we have to use `\?`.

8.2 Monitring tools and logs analysis

Monitoring tools (Nagions, Munin, …) have plugins for PostgreSQL, they offer montoring of log and system tables . The Bucardo's perl script *check_postgres* is the most important plugin for this tools. They are some dedciated tools, for example *temBoard* ans *pgwatch2*.

Another modules can store database statistics evolution (*pg_stat_statements*[1] higly recommanded, *pgtop*, …) or analyze logs (*pgBadger*[2]).

Some modules (*pgFouine*, *pgstatpack*, …) are not maintained anymore. Before use this modules usaually packaged as plugins, we must check health and update …

8.3 Backup

Even if PostgreSQL can realize native hot backup, or make some Point In Time Recovery, … To industrialize it, it may be usefull to develop intern script or to use a backup dedicated tools.

Within the most advanced tools, we have *Barman* et *pgBackRest*. They allow:

- Make PITR easier ;

[1] pg_stat_statements: https://www.postgresql.org/docs/current/static/pgstatstatements.html
[2] pgBadger: https://pgbadger.darold.net/

- make incremental backup ;
- Have a centralized backup catalog, which can come from different servers ;
- Manage backup retention ;
- …

8.4 High-Availibility

Several tools exists to improve resilience and to make administration of an group of instances easier We will notice for example : - *repmgr*: Allow installation of complex architecture and make easiser some administratives actions as failover/switchover/add instance … - *Patroni*: A template to install a high Availibility solution. It is usually used with dockers and with a distributed configuration tools like *etcd*, *Consul* or *ZooKeeper*. - *PAF*: PostgreSQL Automatic Failover based on Pacemaker/Corosync to manage high availiblity and the possibilty to change role automatically.

8.5 Data migration tools to PostgreSQL

The tool *ora2pg*[3] can analyze an Oracle database. The DGFiP's extention ora2pg can evaluate migration costs. The tool can also realize the migration.

For Data transfer, we can use an ELT (Talend Open Studio, *pgloader*[4], …) which can transfrom data when transferring (adaptation of adata type, …).

[3] ora2pg: https://ora2pg.darold.net/
[4] pgLoader: https://pgloader.io/

For an initial load, in case of big volume, it is possible to use a tool like *pg_bulkload*.

The application code must be adapt and needs to be translate in sql instructions compatbible with PostgreSQL. The tool *code2pg*[5] can help by giving an estimation and help to developers.

It is highly recommanded to experimate migration in the same conditions than the production's environnment (production database, session application logs,…).

[5] code2pg: https://github.com/societe-generale/code2pg

9 Decision's Help

Even if PostgreSQL can answer to a lot of needs, some projects can use other technologies which can have a better answer to their needs. This Chapter will try to describe a decision process as it used in several enterprises.

PostgreSQL isn't under licence in it's communuty version. When we deploy it, there is no extra cost. If we want a warranty, we can choose to suscribe support on the community version or to buy a paid version which include support and advanced functionnality. The more often strategy used in enterprise is "PostgreSQL first" for all new projects except if PostgreSQL lacks a functionality or a vendor prerequisites.

Concerning migrations, their ROI may be difficult to justify and some times may be inexistant. It will depend of the project complexity, licences saving, … It may be simplier to migrate the database when the software is upgrading. A story about PostgreSQL, is that it is a good database on small volumes but some PGGTIE members have more than 10 To databases (transactional, Batch, scientific data, spatial…) without perfomance or stablity problems.

Will be list in the following board :

- Functionnalities
- Native support (from version number) or with plugins (from version number)
- Optional commentary - limitations

Functionnalities	Availibility	Commentary - Limits
Connection pooler	with modules pg-pool/pgbouncer …	see § 7.4.3
Master-standby replication without load-balance	9.0 +	see § 6.1
High availibility, automatic failover, connection routing	with modules Patroni/repmgr …	see § 8.4
Crypting data	with plugin Pgcrypto (8.3 +)	see § 4.3
Actvity audit	with plugin pgaudit (9.5 +)	see § 5.2.4
NoSQL	json (9.2+)	see § 4.1
Materialized views	9.3 +	see § 7.3
Tables partionning	10 +	see § 7.2/12.1.4
LOBs	7.2+	
Spatials datas	with module Postgis	see § 4.1/7.1/12.5
Monitoring and supervision	with module temboard/pgwatcl	
UPSERT	9.5 +	

Functionnalities	Availibility	Commentary - Limits
Autonomous transactions	with plugin pg_background	
Indexes		
Utilitaries (maintenance, Data load…)	7.2+ (Vacuum) 7.1+ (Copy)	
auto-explain	with plugin auto_explain	
Plain texte search	8.3 +	
Storage (local or SAN), Backup externalisation	Pg_basebackup or with module Pgbarman/Pgbackrest	see § 5.1.5/6.3/8.3

10 Limits or unsupported functionnalities

This chapter show unsupported functionnalities by the community version.
This is a non-exhaustive list.

- Whole database crypting ;
- Master-master clusters ;
- Compression ;

11 change

11.1 Participant's formation

Formation will be focus on three population :

- Developers : getting started and PostgreSQL migration (2 or 3 days);
- Architect and operators : Install, maintenance, backup, monitoring … (3 days);
- Advanced administrators (DBA) : operators formation + 3 days.

11.2 Support

There is many ways to make support :

- At ministery level : by the interdepartmental market hold by the French interior minsitry or by using the interdepartmental network (Mim-PROD, …);
- For entreprises or the administrations members, by communication with the PGGTIE members
- by the communauty (mailing list, forums,…).

A french professional support is also possible, we will refer to the support page[1].

11.3 Migration plan

Migration requires more or less significant adaptations of softwares for which a non regression test must be conducted. Its cost is significant.

If the migration is done when we have a fonctional evolution which need technical and operationnal test independently of the database change. this cost can be reduced. But we must challenged it with the risk to make technical and operationnal migration in the same time.

Support of project supervisor, project owner and productions services, for skills upgrades must be plan.

[1] Support: https://www.postgresql.org/support/professional_support/europe/

12 Return on investment

12.1 Database migration cost

Migration to a new DBMS should consider the following points :

- Entry cost (Actor's training, skill building,…);
- Application ajustement including test costs (technical, functional and non regression test);
- any renewal of the support market;
- The upgrade of the operating procedures.

12.2 Possesion cost

PostgreSQL is under an open source license[1] like BSD or MIT. So there is no pricing policy from an editor.

12.3 Control of trajectories

The roadmap of PostgreSQL is known and under control.

[1] Licence: https://www.postgresql.org/about/licence/

The PostgreSQL's ecosystem always grows which new plugins and new tools. The commercials software offers more and more interfaces and api to use PostgreSQL.

The french PostgreSQL communauty ist important and very active : check the mailing list[2] and the forum[3]. The inter enterprise work group (PGGTIE) was created in 2016 to answer this exchange need.

[2] Mailing lists: https://www.postgresql.org/list/
[3] Forum: https://forum.postgresql.fr/

13 Appendices

13.1 Migration from Oracle

Contribution of the Ministry of the Interior

You must be aware of the elements of the source servers to be able to define the appropriate target servers in terms of technical environment, software stack, storage system, data, criticality of the application and performance monitoring.

13.1.1 Technical environment

Characteristics of the Oracle servers :

Check the type of servers : dedicated, shared or virtualized?

Check the following technical elements :

- Types of processors ;
- Number of processors ;
- Processor clock speeds ;
- Register size ;
- RAM size.

13.1.2 Software stack

Identify the different software that make up the application stack.

The current OS, including which distribution of Linux, and which version?

The Oracle version, to which PostgreSQL version?

What are the source application servers?

The JVM implementation.

The virtualization system.

13.1.3 Storage

What is the storage system? In general, DBMS are hosted on SAN (Storage Area Network); there is no shortage of alternative solutions.

What is the access mode? Most often VMDK (Virtual Machine Disk).

13.1.4 Database migration

Migration tools.

There are various options: *Ora2Pg*, *ora_migrator*, *orafce*, *code2pg*, ETL or ELT, the development of custom programs, etc.

Process the metadata first before loading the data.

MIGRATION OF ORACLE STRUCTURES to POSTGRESQL: including the structures of tables and views with their appendices, the various procedures, functions, and triggers.

Check for the presence of partitioning and materialized views.

Partitioning is possible via triggers from PostgreSQL 9.1 onwards, and declaratively since version 10.

Materialized views

Since PostgreSQL 9.3, materialized views are a built-in feature.

The language of SQL procedures

Oracle uses PL/SQL and PostgreSQL uses PL/pgSQL ; they are quite similar but some adaptation is required.

Model for data type mapping

Strings can be replaced by VARCHAR or TEXT.

Oracle 7-byte dates by 8-byte TIMESTAMPTZ (don't use TIMESTAMP[1]).

CLOB strings are replaced by TEXT.

Integer values are replaced in ascending precision by SMALLINT, INTEGER, BIGINT, NUMERIC.

Decimal numeric values by NUMERIC.

BLOB data by BYTEA (be careful with the escape character when using Ora2Pg).

Migration of normal and stored procedures

List all PL/SQL procedures and functions and transform them into PL/pgSQL.

Generation of database migration scripts

The structure migration script should be separate from the data migration script; it will be used to initialize the PostgreSQL database.

Create the PostgreSQL database initialization script.

Include the data structures as well as the procedures and functions that have been re-adapted for PostgreSQL.

Data migration

[1] TIMESTAMP: https://wiki.postgresql.org/wiki/Don't_Do_This#Date.2FTime_storage

It can be achieved via a SQL script to be inserted into the new PostgreSQL database.

It is also possible to load the data directly from the Oracle database to PostgreSQL.

Application code

Code modification is necessary to integrate the PostgreSQL driver for managing transactions, opening and closing connections, replacing Oracle keywords with PostgreSQL keywords, external joins, pagination, etc. Depending on where the application logic is, this step can represent a significant part of the migration.

13.1.5 Criticality and backups

Consider the concepts of Disaster Recovery, Busines Continuity, RTO, RPO, and replication.

Which technical solution is used to ensure high availability: Oracle RAC, Oracle DATAGUARD, or something else?

There is no Oracle RAC equivalent in PostgreSQL; however, the DATAGUARD function is provided in PostgreSQL by replication.

The DRP (Disaster Recovery Plan) imposes on us a time limit for the recovery of the application. It takes into account the RTO (Recovery Time Objective) and the RPO (Recovery Point Objective). This will also inform, depending on the volume, the technology, the frequency and the type of backup to be implemented.

On Oracle, RMAN manages backups and restorations; there is no database equivalent under PostgreSQL. Several third-party tools exist, however, includ-

ing *pgBackRest* and *Barman*. Archive logging is well implemented, almost identically, in the two DBMSs.

Physical restores do not have a partial granularity in PostgreSQL (the whole instance is restored).

13.1.6 Monitoring, performance improvement and supervision

There is no global monitoring like Grid Control in Oracle. It is however possible to use Nagios with the plugin *check_postgres*[2]. Other tools are also available (*PoWA*, *PGObserver*, etc.).

[2] check_postgres: https://bucardo.org/check_postgres/

13.2 Migration from Db2

A migration leads to a review of the DDL from Db2: it is recommended to provide a tool for transforming (scripting) the DDL, taking into account the elements described below.

Regarding the DCL, it is preferable not to try to transpose the Db2 grants to pg, as the authorisation levels are very different.

Do not forget to modify the maintenance procedures: backups, history, defragmentation of the database, collection of statistics, cleaning of obsolete files, etc.

13.2.1 Some DDL specifics pg/Db2

- Character strings:
  PG: a character string is defined in a number of characters. Db2: the fixed or variable character strings are expressed in bytes with the consequence of a different number of possible characters depending on the encoding of the database: in UTF-8 some characters are coded on several bytes, in iso8859-15 each character is worth one byte. A varchar(5) string can therefore contain fewer than 5 characters in Db2 if the database is in UTF-8 and includes accented characters.

  Take this into account if you have to migrate a Db2 UTF-8 database to a PostgreSQL database.

- Creating a table in a tablespace:

pg	Db2
create table… tablespace <name_ts>	create table…IN <name_ts>

- Creating a table via the LIKE keyword: in pg, the use of ( ) is mandatory around the LIKE clause, in Db2 it is not necessary.

pg	Db2
create table eqos.statssov (LIKE eqos.stats INCLUDING ALL)	create table eqos.statssov LIKE eqos.stats

- Default value of a table column:

pg	Db2
create table…DEFAULT <value>	create table …WITH DEFAULT <value>

- Creating a table: Some keywords are not recognized by pg, so remove them from the Db2 DDL before migration. Example: *append*, *with restrict on drop*, and *long in*, all to do with table partioning, are defined differently in PostgreSQL.
- Automatic increment of a column counter on table creation: Db2 autoincrement (*[generated always | generated by default] as identity*) is now supported as of PostgreSQL v12. In earlier versions, you have to use sequences.

- The serial type in pg makes it possible to create a sequence easily, but you must not put the integer type (already carried by serial) nor « not null » as these will lead to syntax errors.
- In Db2, the creation of a sequence requires an obligatory « create sequence ».

Therefore, if the PostgreSQL version is lower than 12, all Db2 autoincrements must be converted. If the version is at least equal to 12, the keywords used must be checked because Db2 allows more options.

- Unlogged table: A Db2 table declared as « *not logged initially* » must be « *unlogged* » in pg.
- Temporary tables: The syntax and functionality of temporary tables differ.

Example:

- Db2 : `declare global temporary table tabtemp like eqos.stats not logged`
- pg:`create local temporary unlogged table tabtemp (like eqos.stats)`

Db2 temporary tables with inter-session data sharing are done via a « create global temporary table », they have no equivalence in pg because in pg although syntactically the keywords « local » and « global » are accepted, the behavior of the temporary table remains local in both cases (no sharing of temporary data).

- Timestamp column on row updates:

These columns, frequently used in Db2, do not exist in pg.

Example in Db2: a column with name TS_UPDATE is updated automatically as soon as the row is modified by UPDATE/INSERT sql:

```
CREATE TABLE...
TS_UPDATE TIMESTAMP NOT NULL GENERATED ALWAYS FOR EACH ROW ON
UPDATE
AS ROW CHANGE TIMESTAMP
```

When inserting or updating a row, Db2 automatically fills in the TS_UPDATE column. These columns are systematically deployed in certain Db2 databases, **when using pg, this action must be carried out by the application, so modify your programs accordingly**.

- ISO timestamp formats:

Not only is the precision of timestamp formats different, but the separator between date and time is different too. **Be careful when migrating timestamps from Db2 to pg!**

pg	Db2
2014-01-31 00:00:00	2014-01-31__-__00.00.00.__000000__

Note that the « NOT NULL DEFAULT CURRENT TIMESTAMP » format tolerated in Db2 is not allowed in pg. You must use CURRENT_TIMESTAMP (also recognized by Db2).

- Materialized views:
  pg: creation via « CREATE MATERIALIZED VIEW ».
  Db2: creation via « create table » followed by options specific to a materialized view (called MQT in Db2). **Therefore, the DDL from Db2 needs to be corrected**.

- Drop table and integrity constraints: pg : a « drop table… CASCADE » removes the integrity constraints.
  Db2 : a « drop table » is sufficient to remove the integrity constraints.
- Creating an index, schema: pg : an index name must not be preceded by a schema name, otherwise there is a syntax error. Db2 : it is recommended to include the schema name, and the Db2 tool for generating DDL generates this in front of the index name. If it is omitted in Db2, a schema with the same name as the current schema or connection authority will be used.
- Creating an index, options: Some keywords are not recognized by pg, and should therefore be removed from the Db2 DDL before migration. For example: cluster, allow reverse scan, pctfree, etc.
- Assigning indexes to a dedicated tablespace:
  pg : the assignment is made during table creation for the primary key and unique constraints. You can put an index in a dedicated tablespace when creating the index. Db2 : You can direct all indexes related to a table into a tablespace with the INDEX IN clause, for example:

  ```
  CREATE TABLE.. IN \<name\_ts\_table\> INDEX IN \<name\_ts\_ir
  ```

  Or when creating the index, like in pg.
- Page size:
  pg: only 8kB pages are allowed.
  Db2: we work with page sizes of 4, 8, 16 or 32kB. Modify any Db2 DDL which explicitly specifies page sizes.
- Bufferpool:

  The Db2 notion of bufferpool does not exist in pg, we do not create these resources so the Db2 DDL must be adapted accordingly by re-

moving all creation instructions and references to bufferpools in tablespaces.

- Tablespaces

  Tablespace creation options differ between Db2 and pg, so need to be fixed.

13.2.2 DCL

- Roles vs Unix groups:
  pg : it is mandatory to use grants given to roles.
  Db2 : grants are given either to roles or directly to Unix groups, which is the most frequent case at MEN.

- PUBLIC rights:

  The basic concept of RESTRICTIVE Db2, which eliminates so-called PUBLIC grants, does not exist in pg.

- GRANT ALTER TABLE:

  Used in Db2, but it does not exist as such in pg. In pg, it is necessary to make the account wishing to modify the table belong to a role which is the OWNER of the table.

- TRUNCATE TABLE:
  pg : a « grant truncate » must be done to be allowed to truncate.
  Db2 : a « grant delete » (or control) gives this privilege.

- GRANT CONNECT:

  To be authorized to connect to a database, you must:
  pg: grant connect on database <name_database>… Db2: grant con-

nect on database – do not specify the name of the database, the current database is used by default. If we specify the name of the database, we get a syntax error.

Other syntax differences may also emerge.

- GRANTS without Db2/pg equivalence:

  Some Db2 grants, those linked to functions not present or different in pg, do not exist in pg. This is the case with grant load, grant use of tablespace, grant usage on workflow, etc.

- System catalog

  This information is stored in upper case in Db2, but lower case in pg. Take this into account when searching for information, especially in scripts that use the catalog to collect information.

  Example: search for information in tables belonging to a schema with the name IDENTITY pg: `select * from pg\_tables where schemaname = 'identity'`
  Db2: `select * from syscat.tables where tabschema = 'IDENTITY'`

13.2.3 Other considerations

- Loading utility:
  A Db2 table can be loaded by *import*, *load*, *ingest* ou *db2move*.
  pg exclusively uses the *copy* utility which is much less feature rich than the Db2 utilities. Partial loading of a table is only possible since V12, with the introduction of a WHERE clause.

- Other Db2/pg utilities:
  reorg becomes *vacuum*
  runstats becomes *analyze*
  backup becomes *pg_dump*, *pg_dumpall* or *pg_basebackup*

- Double quote character:

During the migration, beware of the double quote character ": it is the default string delimiter in Db2 when exporting data so it is found in the unloaded (export) file, surrounding each string. By default, when loading to pg these double quote characters will be loaded into the table if they are present in the data file. To avoid this, files exported from Db2 should be in « DEL » (ASCII) mode and imported into pg as csv via the WITH CSV delimiter ‘,’ QUOTE ‘"’ parameters of copy.

- Transaction logs:
  Db2: each database has its own logs.
  pg: this is not the case, the logs are common to databases within the same instance, which can cause problems (logs dominated by one database, multi-database impact in case of corruption or disappearance of logs, etc.).

- LOBs:
  Db2 BLOBs can be replaced by columns in BYTEA format.
  Db2 CLOBs can be replaced by columns in TEXT format. Unlike Db2, which allows embedding LOBs in backups, pg does not backup LOBs via *pg_dumpall* (pg backs up LOBs via pg_dump). Db2 options linked to LOBs no longer apply in pg (logged/not logged, compact/not compact, inline length, etc.).

- Stored procedures: Written in Db2 in SQL/PL, they must be adapted to pg (PL/pgSQL).

- Source code (programs): Modify the call methods of the driver and the use of its properties. Change the SQL code which is specific to Db2/pg.

13.3 Migration from Informix

Contributed by the Ministry of Social Affairs.

13.3.1 Structure

Database schema (tables, indexes, constraints, etc.).

Scripts should allow the creation of different databases in PostgreSQL.

It will be necessary to ensure that all objects are created respecting the PostgreSQL syntax, as well as the recommended standards:

- Tables;
- Views;
- Triggers;
- Constraints;
- Indexes.

The following table describes some differences for the above objects between the Informix and PostgreSQL DBMSs:

Objects / DBMS	Informix	PostgreSQL
Data type	BLOB	BYTEA
Data type	DATETIME	TIMESTAMP

Objects / DBMS	Informix	PostgreSQL
Constraints (foreign keys)	alter table table_name add constraint (foreign key (column_name) references ref_table constraint constraint_name);	alter table table_name add constraint constraint_name foreign key references ref_table (column_name);
Indexes	create index index_name on table_name (column_name) using btree;	create index index_name on table_name using btree(column_name);

13.3.2 Plan to follow when migrating databases

The following steps can be followed for a migration:

- In the source (Informix) databases, first remove all stored procedures that are not used by applications, in order to avoid migrating processes to PostgreSQL that will never be executed;
- Create the databases;
- Load the data;
- Create the indexes and constraints. Adding the indexes and constraints after loading the data will result in a faster execution time of the process.

13.3.3 Integrating the Hibernate framework

If an application framework to manage the persistence of relational database objects in not being used, use the Hibernate framework.

This makes it not possible to directly write the SQL queries in the source code, the queries instead being generated by Hibernate.

The applications are then less dependent on a specific DBMS (with any stored procedures as an exception).

Applications that use the *Hibernate* framework manage the persistence of objects in relational databases. SQL queries are therefore not written in the code, but generated by *Hibernate*.

A migration from Informix to PostgreSQL will then require minimal changes to the source code of these applications.

However, testing is required to ensure there are no regressions..

13.3.4 Risks

Some risks that can be identified:

- Interruption of the use of applications during migration, during the data loading phase from Informix to PostgreSQL;
- A solution to reduce the execution time is to load the static data (such as the lookup tables) first, in order to load only the data likely to evolve (requests, files, etc.) on the day of migration to production. This solution should be considered if the loading time is too long and if time is saved (this depends on the proportion of data that can be qualified as static data);

- Certain processes, which had been optimized for the Informix IDS DBMS, may become slower.

13.4 Migration from MSSQL

Some elements:

Some tools that can help facilitate the data migration are:

- https://github.com/dalibo/sqlserver2pgsql;
- http://pgloader.io/.

For the migration of procedures, the code is very different and must be rewritten.

13.5 Some references

Some published examples of organisations deploying and managing PostgreSQL instances at scale:

GitLab https://about.gitlab.com/blog/2020/09/11/gitlab-pg-upgrade Data volume: > 6 TB;

OneSignal https://onesignal.com/blog/lessons-learned-from-5-years-of-scaling-postgresql 75 TB across 40 servers;

Discourse https://blog.discourse.org/2021/04/standing-on-the-shoulders-of-a-giant-elephant Upgrading to v13;

Some (French language) sites using PostgreSQL, from the testimonials on postgresql.fr:

Meteo France http://www.postgresql.fr/temoignages:meteo_france Data volume: 3.5 TB;

Le Bon Coin http://www.postgresql.fr/temoignages:le_bon_coin Data volume: > 6 TB;

IGN http://www.postgresql.fr/temoignages:ign Capacity to process over 100 million geometric objects;

Mappy http://www.oslandia.com/oslandia-et-mappy-vers-lopensource.html

13.6 Extensions and plugins for PostgreSQL

The table below shows some plugins mentioned in this document:

name	function
PostGIS	Spatial and geograghic
PGAudit	Audit
pg_stat_statements	Statistics
pgcrypto	Cryptography

13.7 Third-party tools for PostgreSQL

The table below shows some third-party tools mentioned in the document:

name	function
Barman	Backup / restore
check_postgres	Monitoring
code2pg	Migration tool
ora2pg	Migration tool
PAF	High availibility
Patroni	High availibility
PgAdmin	GUI
pgBackRest	Backup / restore
pgBadger	Log analysis
PgBouncer	Connection pool management
pgloader	Data loading
Pgpool-II	Connection pool management
pgtop	Statistics
pgwatch2	Monitoring
pitrery	Backup / restore
repmgr	High availibility
temBoard	Monitoring

14 References

PostgreSQL documentation:

- Manual: https://www.postgresql.org/docs/current/index.html
- Wiki: https://wiki.postgresql.org
- Mailing lists: https://www.postgresql.org/list/

Documentation in French:

- Manual: https://docs.postgresql.fr/current/
- Forums: https://forums.postgresql.fr/