Application systems migration techniques from Oracle to PostgreSQL

Peter Petrov, Database Engineer,

September 09, 2021

postgrespro.com

Speaker bio

1. Started working with Oracle 10g Release 2 in 2013 and PostgreSQL 8.4 in 2015.
2. Wrote procedures to transfer data from Oracle to PostgreSQL, database sizes varied from 1TB to 5TB.
3. Consulted customers on correct business logic design and development using pl/pgSQL programming language.
4. Participated in optimization of various database queries.
5. Participated in troubleshooting various situations that occurred during PostgreSQL maintenance.
6. Designed some business logic by using PL/SQL and Java programming languages.

Agenda

1. Determining the source RDBMS features and assessment of migration feasibility.
2. Estimating costs for the migration process.
3. Data migration.
4. Choosing data conversion tools.
5. Stored code migration.
6. Interconnection testing between related systems.
7. Preparing the system for the real-world workload.

Determining the source RDBMS features and assessment of migration feasibility (1)

First, a schema analysis is required for finding out:

1. Incompatible data types usage in the source database.
2. Specific data types usage such as BLOB, RAW, ROWID, URI etc.
3. Virtual columns usage.
4. External tables for storing the data outside the source database.
5. Partitioning schemas and their keys.

Determining the source RDBMS features and assessment of migration feasibility (2)

The data schema can be migrated “as is” or with the following modifications:

- It is possible to migrate **just a part of the schema** since some objects may be used in some deprecated code.
- There can also be changes related to the **schema’s normalization and denormalization**.
- Replacing column data types with the **PostgreSQL compatible counterparts**.
- Dividing some objects into **smaller partitions** for reducing maintenance operations time.
- Creating additional data structures for storing some **intermediate computation results**.

The source RDBMS features in the application (1)

The application itself can use the source RDBMS features, therefore, it must be rewritten while migrating to PostgreSQL:

- **Old join** syntax in user queries.
- **Hierarchical queries** usage in the application code.
- Presence of **user-defined** data types.
- Specific functions, procedures or modules for communication with some external systems.

The source RDBMS features in the application (2)

- **Parallel techniques** usage for speeding up stored procedures execution.
- Queries in which filtration and join clauses are computed during **its execution time**.

The source RDBMS features in the application (3)

Using third-party libraries to generate SQL queries:

- **Multiple access** to the same tables during a query execution.
- **Aggregate** and **window** functions **are not used**.
- **No Common Table Expressions** for reducing SQL-code duplication.
- Many filtering and join conditions are computed **during a query execution** which may lead to an **inaccurate execution plan**.

Estimating costs for the migration process

The following things should be assessed:

1. Equipment for developing and testing the migration process and emulating the real-world workload.
2. The system conversion and testing, tuning communication with different DBMS.
3. Licenses for an alternative DBMS and its technical support.

Brief data migration description (1)

1. Creating an environment with **a copy of the real-world database** from the source DBMS and a server for PostgreSQL.
2. Choosing, installing and configuring schema and data **conversion tools**.
3. Choosing a **migration strategy** (full, incremental, migration's frequency, automatization techniques).
4. Creating the schema in PostgreSQL **without indexes and constraints**.

Brief data migration description (2)

5. Validating tables structure in PostgreSQL after migration.
6. Transferring data to the destination DBMS.
7. Validating the tables data after migration.
8. Transferring indexes, foreign and primary keys, other constraints and triggers.

Brief data migration description (3)

9. Validating indexes, foreign and primary keys, constraints and triggers after transferring them to PostgreSQL.
10. Gathering statistics for the PostgreSQL planner.
11. Capturing changes from the source DBMS and transferring them to PostgreSQL.

Data conversion tips (1)

1. It’s required to establish a **data type mapping** between **the source** and **the destination DBMS**.
2. If the required data type **is absent** in PostgreSQL, then a **transformation procedure** should be used to **convert it into the existing type** in PostgreSQL.
3. Some **additional PostgreSQL modules** provide additional data types for the application’s needs.
4. Consider the possibility of storing **various document formats**, as well as a sequence of **nested data structures** inside tables columns.

Data conversion tips (2)

4. Take into account the presence of **money values** inside tables' columns.
5. Take into account the presence of **large binary data** and **geometry objects** while transferring the data.

ora2pg as a tool for schema and data migration

ora2pg is the open-source utility for work with Oracle DBMS providing the following features:

1. A migration project creation.
2. Scanning the source RDBMS and extracting its schema definition and data.
3. Generating commands to create PostgreSQL-compatible structures.
4. Saving data of the source DBMS in intermediate files, if necessary.

Benefits of ora2pg

1. An ability to **customize the list of migrated objects**.
2. An opportunity to specify **multiple schemas** for transferring.
3. A possibility to customize parameters through the **configuration file**.
4. An ability to **customize the mapping of data types** between the source and destination RDBMS.
5. A possibility to transfer data **directly** to PostgreSQL.

Disadvantages of ora2pg

1. **Computed columns** conversion may be incorrect.
2. The utility may fail while working with **large binary objects**.
3. Objects with **large rows number** may not be transferred correctly.
4. **Lack of flexibility** in the mechanism of **parallel data reading**.

pgloader as a tool for schema and data migration

pgloader is the open-source utility for work with MS SQL Server, MySQL and PostgreSQL DBMS. **pgloader** provides the following features:

- Scanning the source RDBMS and extracting its schema definition and data.
- Schema objects can be created **directly in PostgreSQL** without generating intermediate SQL files.
- Generate a **file containing row data that encountered an error during processing** in the destination DBMS. In this case, the **remaining lines** are successfully saved.

Pentaho kettle as an ETL tool for data migration

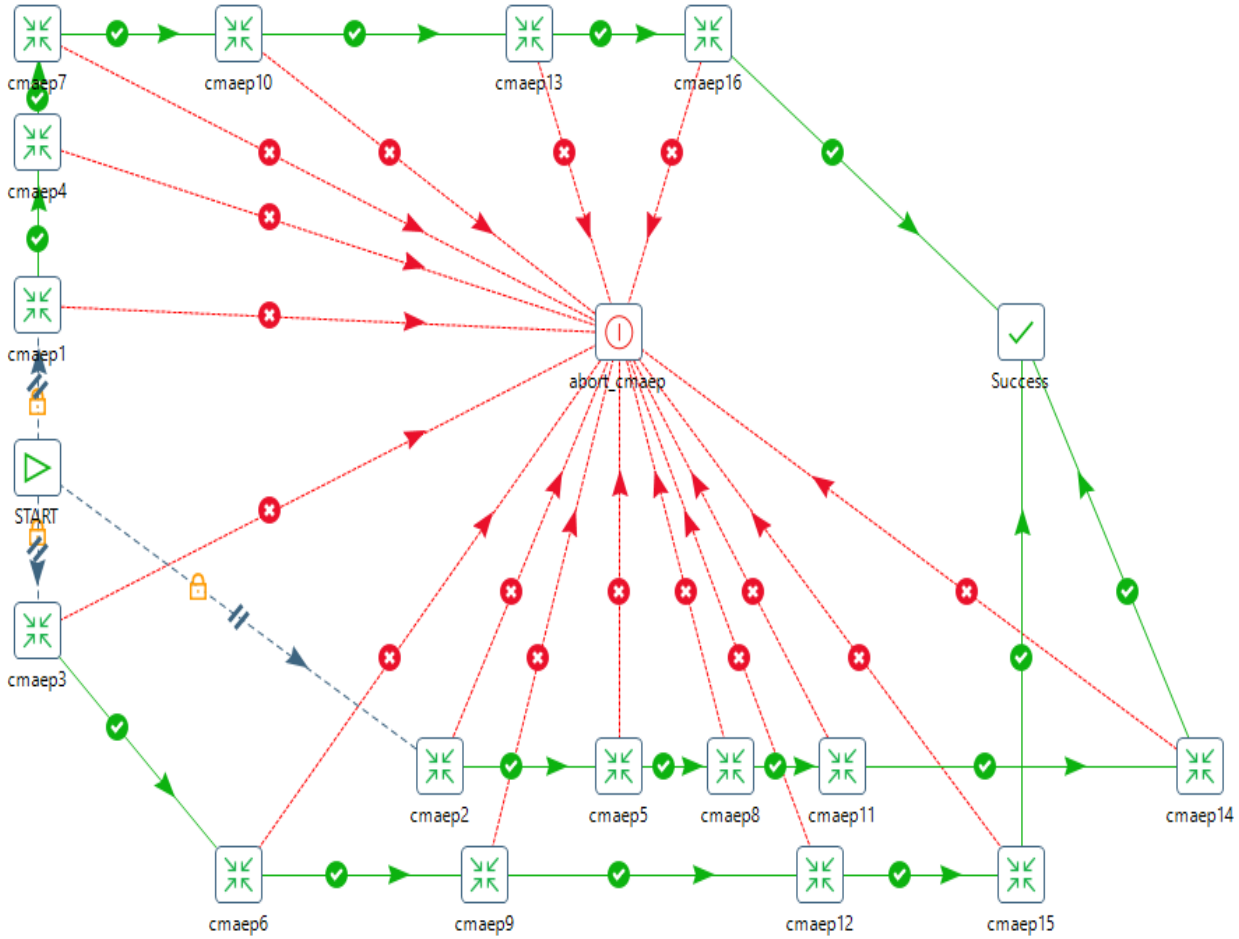
pentaho kettle is an ETL software providing the following features:

1. Developing various cases of data extraction, transformation and loading.
2. Embedding logic **implemented in Java programming language** into an ETL-process.
3. Running tasks on **a schedule**.
4. Working with **various DBMS** during jobs or transformations execution.

Benefits of Pentaho kettle

1. An opportunity to read the data from **various sources**.
2. An ability of **developing and debugging various scenarios** by using **GUI**.
3. A possibility of designing **repetitive actions in a form of subtask**.
4. An opportunity of **data storing in log tables when an error occurs** during migration.
5. **Data transfer verification** by storing **various metrics** during data migration.

An example of a data transfer job

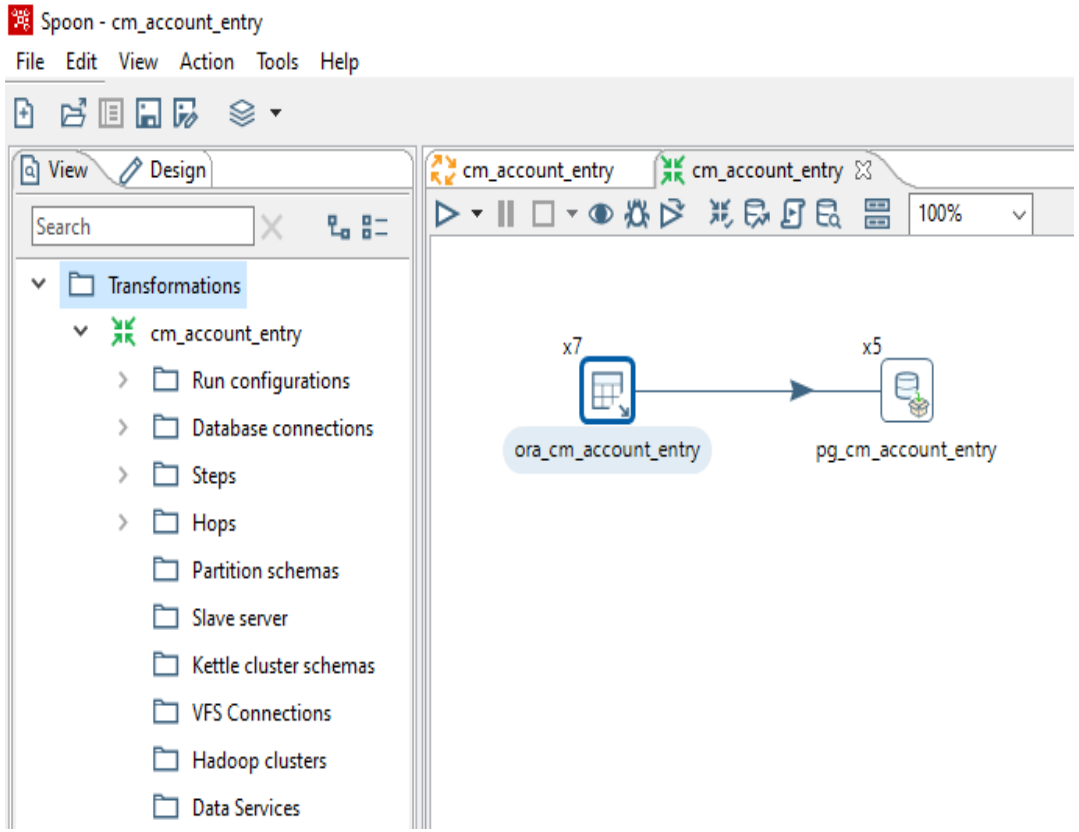


The task allows to transfer data from the partitioned table **cmaep**. The process begins with the execution of the **Start** component and is carried out by three threads:

- Start – cmaep1
- Start – cmaep2
- Start – cmaep3

Each transformation is responsible for transferring a separate section. If the data transfer is successful, control is transferred to cmaep4, cmaep5, cmaep6 etc. If an error occurs, the execution is transferred to the **abort_cmaep** component, the entire job ends with an error.

Pentaho kettle transformation example using multiple read and write threads



The **ora_cm_account_entry** component is responsible for reading data from the source DBMS, it is assigned to a corresponding request and the number of threads for simultaneous reading.

The **pg_cm_account_entry** component is responsible for writing data to the PostgreSQL and uses the COPY command for this, which can significantly reduce the transfer time.

It's also possible to specify thread count to increase the writing speed.

Brief description of the stored code migration (1)

1. Rough code migration.
2. Finding code that can’t be translated to PL/pgSQL due to the source DBMS specific modules.
3. Designing workarounds for code detected on the step 2.
4. Functional and load testing for translated code fragments.

Brief description of the stored code migration (2)

5. Performance optimization of code snippets.
6. Functional and load testing as part of the application.
7. The application optimization based on the results of the functional and stress testing.

Available converters for the stored code migration

1. **ora2pg** for partial translation of Oracle and MySQL stored procedure code.
2. **ANTLR4** and its grammar files for PL/SQL and T-SQL code.
3. Third party online conversion services such as **sqlines.com**.

Examples of incorrect code conversion when using ora2pg(1)

1. Conversion of **old Oracle syntax** for table joining is not supported.
2. Incorrect translation of the **DECODE** statement.
3. Oracle **implicit type conversion** is ignored. For example, during a string conversion to a number Oracle RDBMS erases leading zeros. In PostgreSQL it should be done by the application developer.
4. **Recursive queries syntax** is not supported.

Examples of incorrect code conversion when using ora2pg(2)

5. If a query has columns with **double quotes**, it may not be translated correctly. Sometimes, ora2pg can put column names into **lower register**. As a result, a query can't be compiled due to the syntax error.
6. The construction **KEEP (DENSE_RANK LAST ORDER BY) OVER(PARTITION BY)** is not supported, the external module **first_last_agg** is required.
7. **Dynamic queries** are not always converted properly (**EXECUTE IMMEDIATE**).
8. The code with the usage of **dbms_xmldom**, **dbms_lob** should be rewritten manually.

Finding the source DBMS specific features (1)

1. Usage of the **advanced queue**.
2. Technologies for building applications **based on Oracle DBMS** (Apex, Formspider).
3. **Hierarchical** queries with the **old syntax usage**.
4. Usage of the **old Oracle syntax for table joining**.

Finding the source DBMS specific features (2)

5. Interaction with some **external services at the DBMS level** (UTL_HTTP, UTL_SMTP).
6. PL/SQL collections usage.
7. Usage of **BLOB, XMLType, JSON data types as well as various packages** for their manipulation.
8. Parallel data processing based on the **dbms_parallel_execute package**.

Finding the source DBMS specific features (3)

9. Frameworks for **Unit testing** of the stored code.
10. Autonomous transactions.
11. Using global data structures at the package level.
12. Using RLS.

Finding the source DBMS specific features (4)

13. Queries with **filter** and **join conditions** that can't be calculated during a **query execution time**.
14. Executing schedules tasks by using the **dbms_scheduler** package.

The source DBMS specific features workarounds(1)

1. The solution related to queue mechanisms can be implemented on the application level by using various technologies such as **Apache Kafka** and **ActiveMQ**. The external PostgreSQL module **pgq** can be used as well.
2. There is **no Apex counterpart** in PostgreSQL.
3. Hierarchical queries are replaced on **recursive CTE**.
4. The old join syntax for tables is rewritten by using **JOIN, LEFT JOIN, RIGHT JOIN** constructs.

The source DBMS specific features workarounds(2)

5. Communication with some external services could be implemented by using **non-trusted programming languages** such as **plpython3u**.
6. **Nested tables** could be replaced by using **object arrays**, **associative arrays** – **hstore** or **jsonb**.
7. **blob** data type could be replaced by **bytea**, **XMLType – xml**, **JSON – jsonb** or **json**.
8. **dbms_parallel_execute** functionality could be implemented on the application level. It could also be designed by using **pg_cron**, **pgAgent**, **pg_timetable** or **pgpro_scheduler** in case that Postgres Pro Enterprise distribution is available.

The source DBMS specific features workarounds(3)

9. For the **stored code unit-testing pg_tap** and **pg_prove** modules should be used.
10. **Autonomous transactions** are available in **Postgres Pro Enterprise** distribution. **dblink** or **pg_background** modules could also be used for that.
11. **pg_variables** could be used for **storing various data structures during user sessions**.
12. PostgreSQL provides the **RLS mechanism** for the implementation of **additional row access rules**.

The source DBMS specific features workarounds(4)

13. For **profiling pl/pgSQL procedures and functions**, there is the **plprofiler** extension, which **builds reports after their execution**.

14. To create a task execution schedule, the Postgres Pro Enterprise DBMS has the **pgpro_scheduler** extension, which replaces the **dbms_scheduler** functionality. It could also be replaced by using **pg_cron**, **pgAgent**, **pg_timetable**.

Interconnection testing between related systems

1. Creation of a stand with the PostgreSQL DBMS.
2. Communication channels preparation for data exchange between PostgreSQL and external systems.
3. Implementing part of the integration mechanism at the application level, if necessary.
4. Workload generation for data transferring from PostgreSQL to the external systems, measuring response times.
5. Testing the reception of information from the external systems to PostgreSQL, measuring execution times.
6. Integration mechanisms performance tuning based on the results of the test scenarios.

Preparing the system for the real-world workload

1. Preparation and testing a code and data conversion solution, acceptable downtime should be taken into account here.
2. Choosing and customizing a backup/recovery solution.
3. Choosing monitoring tools.
5. Adapting a solution to ensure fault tolerance and disaster recovery.
6. Creating a plan for system real-world usage.

Preparation a code and data conversion solution (1)

The main goals are **downtime reducing** and **avoiding code reconversion**.

The possible methods are presented below:

1. Capturing **changes** in the **schema definition and its data**.
2. Detecting **unchanged data** and **its conversion**.
3. Using triggers for capturing data changes.
4. Testing data transferring procedure and measuring its execution time.

Preparation a code and data conversion solution (2)

5. Freezing the application code development.
6. Detecting stored code changes by the database's system dictionaries.
7. Using ANSI-compatible syntax for writing new queries on the source DBMS.

System downtime minimization(1)

1. MATERIALIZED VIEW LOG and its data transferring to a message queue such as **Apache Kafka**:

   - https://github.com/averemee-si/oracdc. This solution is tailored to the business reporting schemas as well as Oracle business suite

2. Log Miner for redo и archive journals analysis:

   - https://github.com/erdemcer/kafka-connect-oracle
   - https://debezium.io/documentation/reference/connectors/oracle.html. Log Miner usage for analyzing changes in redo-logs and transferring them to the message queue.

System downtime minimization(2)

3. **debezium + xstream** (Oracle Golden Gate license is required):

   - https://github.com/debezium/debezium

4. **symmetricDS** uses triggers to write changes to a service table, then data packages are formed and transferred for the exchange between the source and target nodes:

   - https://www.symmetricds.org/doc/3.7/html/user-guide.html#_architecture

System downtime minimization(3)

5. **Debezium connectors** for retrieving data changes from various DBMS with subsequent transfer to **Apache Kafka**:

   - https://github.com/debezium/debezium/tree/master/debezium-connector-sqlserver
   - https://debezium.io/documentation/reference/connectors/sqlserver.html

6. **PowerExchange CDC Data sources**:

   - https://docs.informatica.com/data-integration/powerexchange-for-cdc-and-mainframe/10-0/_cdc-guide-for-linux-unix-and-windows_powerexchange-for-cdc-and-mainframe_100_ditamap/powerexchange_cdc_data_sources_1/db2_for_linux_unix_and_windows_cdc.html

pg_probackup as a tool for a database cluster backup and recovery

pg_probackup is a utility to manage backup and recovery of PostgreSQL database clusters.
It offers the following benefits:

- Incremental backup and restore
- Validation and verification
- Multiple threads usage while making a backup or restoring from it.
- Backup from standby

For more information, please, click the following link:

https://postgrespro.github.io/pg_probackup/#pbk-overview

PostgreSQL workload monitoring tools (1)

Mamonsu is a monitoring agent for collecting PostgreSQL and system metrics and sending them to the **Zabbix** server:

- Works with various operating systems / OSs
- 1 agent = 1 database instance
- Works with PostgreSQL version >= 9.5
- Provides various metrics related to PostgreSQL activity

PostgreSQL workload monitoring tools (2)

Zabbix Agent 2 is another tool for collecting various metrics which is available from **Zabbix Server version 5.0**:

- 1 agent can collect more than 95 metrics from multiple PostgreSQL instances.
- Available from Zabbix standard repository.
- Can work with PostgreSQL version 10 and higher.
- An opportunity to write custom plugins in Golang.

An example of transferring a corporate document management system to the Postgres Pro Standard (1)

System's summary:

1. Total database size is **600GB**.
2. The number of concurrent user sessions on the application servers: **600-800**.
3. **All business logic** was implemented **on the application side**.

An example of transferring a corporate document management system to the Postgres Pro Standard (2)

Goals:

1. Develop an automated procedure for transferring data to the Postgres Pro Standard DBMS. Determine the objects list to be transferred.
2. Suggest some solutions for improving the database performance.
3. Convert heavy queries for work with the Postgres Pro Standard DBMS.

An example of transferring a corporate document management system to the Postgres Pro Standard (3)

Actions performed:

1. Scripts for automatic invocation of the **ora2pg** and **pentaho kettle** utilities for schema conversion and data transfer have been designed. **The user could specify lists of transferred objects as well as the number of simultaneously reading and writing threads**.
2. Heavy queries have been converted, and recommendations for their optimization have been developed.
3. Schema changes have been included in the migration procedures.
4. Consultations for the DBMS and OS tuning have been provided.

An example of transferring a department system to Postgres Pro Enterprise (1)

System's summary:

1. Total database size is **5TB**.
2. The number of concurrent user sessions on the application servers : **2000-5000**.
3. **95%** of business logic was implemented **on the application side**, **5%** was implemented in **a form of Oracle views** for reporting.
4. There was a table **with large binary data with the total size of 4.5TB** and tables with **the number of rows > 1 billion**.

An example of transferring a department system to Postgres Pro Enterprise (2)

Goals:

1. Develop a procedure for transferring data to the Postgres Pro Enterprise DBMS. Determine the objects list to be transferred.
2. Convert views for work with the Postgres Pro Enterprise.
3. Optimize Postgres Pro Enterprise DBMS to handle **hundreds of user sessions** on a server **with many cores**.

An example of transferring a department system to Postgres Pro Enterprise (3)

Actions performed:

1. Scripts for automatic invocation of the **ora2pg** and **pentaho kettle** utilities for schema conversion and data transfer have been implemented. **The user could specify lists of transferred objects as well as the number of simultaneously reading and writing threads**.
2. Heavy views have been converted and recommendations for their optimization have been developed, consultations for the DBMS and OS tuning have been provided.
3. Methods for tracking changes in tables with binary data and a large number of rows have been proposed.
4. **Patches of the DBMS kernel have been developed** to optimize working with a large number of user sessions.

PostgreSQL features from version 12 (1)

1. The **partition_pruning** mechanism has been improved and it now allows to extract data from **the necessary partitions** not only at **the planning stage** but also **during a query execution**.
2. The **partition_wise** mechanism has been improved and now allowing to join the corresponding section with smaller size.
3. **btree-index** sizes have been reduced.
4. **Extended statistics** can be used with **various filter clauses** as well as with the **IN operator**.

PostgreSQL features from version 12 (2)

5. The ability to use **multiple extended statistics in one query**.
6. The opportunity to **process multiple indexes concurrently while performing vacuum operations**.
7. The ability to **control CTE behaviour**.
8. Improving **pl/pgSQL internals in terms of performance**.
9. The ability to create **extended statistics based on column expressions**.

Useful links (1)

- ora2pg: https://github.com/darold/ora2pg
- pentaho kettle: https://help.pentaho.com/Documentation/9.1
- pgloader: https://github.com/dimitri/pgloader
- ANTLR4: https://github.com/antlr/antlr4
- sqlines: https://github.com/dmtolpeko/sqlines
- orafce: https://github.com/orafce/orafce
- pg_cron: https://github.com/citusdata/pg_cron

Useful links (2)

- pg_timetable: https://github.com/cybertec-postgresql/pg_timetable
- plprofiler: https://github.com/bigsql/plprofiler
- multicorn: https://github.com/Segfault-Inc/Multicorn
- oracdc: https://github.com/averemee-si/oracdc
- kafka-connect-oracle: https://github.com/erdemcer/kafka-connect-oracle
- debezium + log miner: https://github.com/debezium/debezium-incubator/pull/185
- debezium + xstream: https://github.com/debezium/debezium
- mamonsu: https://github.com/postgrespro/mamonsu

Useful links (3)

- debezium + sql server: https://debezium.io/documentation/reference/connectors/sqlserver.html
- SymmetricDS: https://www.symmetricds.org/doc/3.7/html/user-guide.html#_architecture
- pg_probackup: https://postgrespro.github.io/pg_probackup
- zabbix agent 2: https://github.com/zabbix/zabbix/tree/master/src/go/cmd/zabbix_agent2
- PowerExchange CDC Data Sources: https://docs.informatica.com/data-integration/powerexchange-for-cdc-and-mainframe/10-0/_cdc-guide-for-linux-unix-and-windows_powerexchange-for-cdc-and-mainframe_10-0_ditamap/powerexchange_cdc_data_sources_1/db2_for_linux_unix_and_windows_cdc.html

Postgres Professional

http://postgrespro.com/
p.petrov@postgrespro.com
info@postgrespro.com