Database - Feature #4723

make it significantly easier to run database import

07/01/2020 01:21 PM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #4722: data import should be able to run with o...	**Test**
Related to Database - Support #3871: determine how to change codepages/locale...	**Closed**
Related to Database - Feature #1664: data import improvements	**New**

History

#1 - 07/01/2020 01:21 PM - Greg Shah

- Related to Feature #4722: data import should be able to run with only the converted application jar file (and FWD) added

#2 - 07/01/2020 01:23 PM - Greg Shah

- Related to Support #3871: determine how to change codepages/locales during import added

#3 - 07/01/2020 01:31 PM - Greg Shah

The database import should be very simple to run, but today it has some "sharp edges" which tend to trip people up. We need it to be foolproof. Sadly, it is currently not even "smartpersonproof". :)

Issues to be resolved:

- Too many sources of configuration, 3rd party properties files, hard coded directories and other similar requirements that together makes this messy and fragile.
- Better handling the differences between databases is needed. The tool itself should hide this as much as possible.
- It should handle the needed inputs for #3871.
- We have multiple approaches to scripting. In a perfect world, the command line driver itself would be easy to execute directly. Any scripting on top of that would be trivial.

I know I'm missing some things, but hopefully the idea is clear.

#4 - 12/04/2020 02:14 PM - Eric Faulhaber

- Related to Feature #1664: data import improvements added

#5 - 08/10/2021 04:39 PM - Eric Faulhaber

I recently noticed that upon importing test data into a PostgreSQL database, I got errors about bytes that could not be encoded with UTF8 on their way into the database. The database was created in a cluster built on a custom locale, the code page of which supported the encoding of the exported data. These bytes required "safe" mappings in p2j.cfg.xml or at the import command line, and I had not provided such mappings at the time I attempted the first import. The problematic records were dropped. At the end of the import process, I was left with a database which could not be used, because no p2j_id_generator_sequence sequence had been created, and the persistence runtime relies upon this sequence to allocate new primary keys.

It can be argued that a database with such import errors should rightly be considered "broken", and thus the sequence should not be created after some records are dropped due to import errors, because otherwise customers will think the database is usable and ok, but it will not truly represent the exported data without those dropped records. Currently, the sequence intentionally is not created, unless there are no errors at all during the import.

If there are errors, the current practice to fix the database requires that safe byte mappings be defined for the problematic bytes. Then, the "broken" tables are ~~dropped~~ cleared (manually), and the import is restarted just for those tables (i.e., only the *.d files for those tables are made available for the import). Only once the import of all records in the *.d files completes successfully is the p2j_id_generator_sequence sequence created. However, I think currently there is a technical flaw in this approach, in that the sequence probably will be created with the wrong starting value, since only the re-imported tables will have been considered in the allocation of sequential primary key values, and the sequence's starting value may be lower than other primary key values already allocated for other tables on the original import attempt. It will also result in duplicate primary keys being assigned during import (albeit for different tables), while in the 4GL it is my understanding that rowid values are unique, database-wide.

Aside from this technical flaw, I think the process itself is flawed, in that, unless the exported *.d files are scanned for bad bytes (i.e., with the DatabaseDumpChecker) before import, they will not be discovered until after waiting for the initial import process to finish. At that point, customers will potentially have waited a very long time, only to have an unusable database with missing records and a missing primary key generator sequence at the end. Furthermore, they will face an error-prone corrective process (i.e., dropping the right tables, managing the correct subset of export files to re-import) in order to fix it, and still potentially end up with the wrong primary key sequence starting value at the end. Thus, any unmapped problem bytes will lead to wasted time and user frustration.

I think the import process should default to performing a scan of the exported data (i.e., using the DatabaseDumpChecker), *before* attempting to import any data. If that fails, the import should abort, so the user can provide the necessary byte mappings, or fix the exported data manually, or take other corrective actions. Thus, the user will know relatively quickly (potentially minutes instead of hours) whether any corrective actions need to be taken before actual import begins. Only once the byte scan passes without error would the actual import of records begin. At that point, only unexpected errors (e.g., database connection problems, power outage, etc.) should be able to impact the import process.

For expert users or for use after it is known that all problem bytes have been mapped to "safe" bytes, an option could be provided to skip the scan before import, but otherwise, I think it should be the default.

#6 - 08/10/2021 05:35 PM - Ovidiu Maxiniuc

Yes, I think it is possible to launch DatabaseDumpChecker during import process and passing the current byte mapping (if any). If the utility returns with success, the import will continue as usual, knowing no character conversion errors will occur. OTOH, if there are no definitions for byte mapping or they do not cover all invalid bytes from the source data, the process will stop and the user is responsible to update the mapping in p2j.cfg.xml and restart the process.

#7 - 08/10/2021 05:35 PM - Greg Shah

Even if some records are dropped, the system should be left in a workable state. In my opinion, the p2j_id_generator_sequence should always be created.

#8 - 08/10/2021 05:53 PM - Eric Faulhaber

Greg Shah wrote:

> Even if some records are dropped, the system should be left in a workable state. In my opinion, the p2j_id_generator_sequence should always be created.

In fact, I think it *has* to be created if any data was imported. We would need it to determine the starting primary key value for any follow-up import restart, thus preventing the technical flaw I mentioned above.

Furthermore, the sequence's existence would indicate that this is not a "fresh" import, but rather a restart. We could use this information to be a little smarter on the second/subsequent pass. For instance, we might disallow the import of any records into a table which already has data in it, since this would mean that either:

- a broken table had not been cleared (I mistakenly said dropped previously) in preparation for a follow-up attempt; or
- a .d file for a table successfully imported on a previous pass had not been removed for the follow-up attempt.