cdbs_sql_gen

Rick Whitman

November 27, 1996

Usage

The tool is invoked by entering *cdbs_sql_gen* on the command line followed by one or more arguments which are file specifications for the load files you want to process. These may consist of any legal file specification including wildcards. For example:

```
cdbs_sql_gen a3d1145dy.lod d9g1458cu.lod
```

There are also two environment variables that must be defined in order for the tool to perform some database actions using OpenSTDB. These environment variables are:

- CDBS_SERVER - set to the database server name (in uppercase)
- CDBS_DB - set to the CDBS database name (in lowercase)

The output will be a single SQL file which, when executed will load the appropriate database tables with the information contained in the load file(s). The file will be named cdbs_delivery<number>.sql where <number> will be determined by incrementing a delivery number found in the database. For example:

```
cdbs_delivery123.sql
```

The tool will also output a line indicating the input file currently being processed to stdout.

```
Processing a3d1145dy.lod ...
Processing d9g1458cu.lod ...
```

It will also write messages indicating errors, if they occur, to stdout. Upon detection of an error the task is terminated.

Algorithm/Module Description

The tool consists of a single file with several functions. It also calls functions from the loadfile library and the OpenSTDB library. First the tool determines the delivery number to use by querying the database. Then for each load file, the following steps occur:

- The load file is read into an internal structure using the *read_loadfile* function from the loadfile library. This allows for the load file data to subsequently be referenced using the loadfile library functions.
- Next it is determined whether the file is an unexpanded or expanded file based on the presence of the `EXPANSION_NUMBER` keyword in the file. If it is an unexpanded file the `load_unexpanded` function is called to process the file, otherwise the `load_expanded` function is called. These functions will produce the SQL for loading the database with the load file values. The processing for each type of file is described below:

Unexpanded Load Files

For each unexpanded load file, one or more row-level statements is generated, followed by a single file-level INSERT statement. For <instrument> = wfpc, hrs, foc, hsp, wfpc2, fos, nic, and stis: The row level insert statement is of the form:

```
INSERT  <instrument>_row (file_name,
                          expansion_number,
                          <mode_keyword1>,
                          <mode_keyword2>,
                          ...
                          <mode_keywordN>,
                          equivalence_class_severe,
                          equivalence_class_moderate,
                          pedigree,
                          observation_begin_date,
                          observation_end_date,
                          comment)
                   VALUES (<file_name>,
                           0,
                           <mode_value1>,
                           <mode_value2>,
                           ...
                           <mode_valueN>,
                           <equivalence_class_severe>,
                           <equivalence_class_moderate>,
                           <pedigree>,
                           <observation_begin_date>,
                           <observation_end_date>,
                           <comment>)
```

The <file_name> is read from the header of the load file. <pedigree>, <observation_begin_date>, <observation_end_date>, <comment> and the <mode_keyword> values are read from the rows of the detailed section of the load file.
The equivalence class information is determined by first calling the *get_old_equiv_class* function which issues the following query to get the equivalence class data for the comparison file (the old euivalence class data):

```
SELECT old_equiv_class_severe = equivalence_class_severe,
       old_equiv_class_moderate = equivalence_class_moderate
FROM <instrument>_row
WHERE
file_name = <comparison_file_name>
AND
<mode_keyword1> = <mode_value1>
AND
<mode_keyword2> = <mode_value2>
...
AND
<mode_keywordN> = <mode_valueN>
```

And then calling the *get_new_equiv_class* function which issues the the following query to get the new equivalence class data:

```
SELECT new_equiv_class_severe = value + 1,
       new_equiv_class_moderate = value + 2
FROM equiv_class_value

UPDATE equiv_class_value SET value = value + 2
```

where quantities <comparison_file_name>, <mode_keywordN>, and <mode_valueN> are in the detailed section of the load file. Again, <instrument> is in the header. Once the old and new values have been determined, the values to be inserted into the database are determined based on the value <change_level> (found in the detailed section of the load file) as follows:

if <change_level> = SEVERE then

- <equivalence_class_severe> = <new_equiv_class_severe>
- <equivalence_class_moderate> = <new_equiv_class_moderate>

if <change_level> = MODERATE then

- <equivalence_class_severe> = <old_equiv_class_severe> and

- <equivalence_class_moderate> = <new_equiv_class_moderate>,

if <change_level> = TRIVIAL then

- <equivalence_class_severe> = <old_equiv_class_severe> and
- <equivalence_class_moderate> = <old_equiv_class_moderate>.

For each entry in the detailed section, one such row-level insert statement needs to be constructed.

For <instrument> = synphot, if <reference_file_type> = COMPTAB, then all equivalence class data in the INSERT statements is omitted.

If <reference_file_type> is not equal to THROUGHPUT, then all mode information in the INSERT statements is omitted. All other fields are dealt with as described above.

For <instrument> = multi the equivalence_class data is omitted and the opus_flag field is always set to "N". The useafter_date information is optional but will be included in the multi_file INSERT statement if present in the load file. The pedigree, observation_begin_date, observation_end_date, and comment information are also optional but will be included in the multi_row INSERT, if present in the load file. All other fields are dealt with as described above.

The file-level INSERT statement is:

```
INSERT  <instrument>_file (delivery_number,
                           reference_file_type,
                           file_name,
                           useafter_date,
                           comparison_file_name,
                           opus_flag,
                           comment,
                           reject_flag)
                   VALUES (delivery_num,
                           <reference_file_type>,
                           <file_name>,
                           <useafter_date>,
                           <comparison_file_name>,
                           <opus_flag>,
                           <comment>,
                           'N')
```

where all values denoted by < > are obtained from the header of the load file and `delivery_num` value is retrieved from the database as described earlier.

Expanded load files

For expanded load files, there is an expansion_number = 1, etc. in the detailed section of the load file. Expanded load files are only possible for <instrument> = wfpc, hrs, foc, hsp, wfpc2, fos, nic, and stis. (<instrument> is in the header of the load file.)

For each expansion number, there is a row-level INSERT statement. The row level insert statement is of the form:

```
INSERT  <instrument>_row (file_name,
                          expansion_number,
                          <mode_keyword1>,
                          <mode_keyword2>,
                          ...
                          <mode_keywordN>,
                          equivalence_class_severe,
                          equivalence_class_moderate,
                          pedigree,
                          observation_begin_date,
                          observation_end_date,
                          comment)
                   VALUES (<file_name>,
                           <expansion_number>,
                           <mode_value1>,
                           <mode_value2>,
                           ...
                           <mode_valueN>,
                           <equivalence_class_severe>,
                           <equivalence_class_moderate>,
                           <pedigree>,
                           <observation_begin_date>,
                           <observation_end_date>,
                           <comment>)
```

The <file_name> is in the header, <mode_keywordN> and <mode_valueN> are in the detailed section of the load file (found as <mode_keywordN> = <mode_valueN>). (<mode_keywordN>). The <pedigree>, <observation_begin_date>, <observation_end_date>, <expansion_number>, and <comment> values are found in the detailed section. The <equivalence_class_severe> and <equivalence_class_moderate>

values are obtained as described above for unexpanded load files. The determination of the <equivalence_class_severe> and <equivalence_class_moderate> need only be done once per expanded load file. So all expanded row-level records for a given load file have the same values for <equivalence_class_severe> and the same values for <equivalence_class_moderate>.

There will also be a file-level insert statement generated for each expansion number of the form:

```
INSERT  <instrument>_file (delivery_number,
                           reference_file_type,
                           file_name,
                           expansion_number,
                           useafter_date,
                           comparison_file_name,
                           opus_flag,
                           comment,
                           reject_flag)
                    VALUES (delivery_num,
                           <reference_file_type>,
                           <file_name>,
                           <expansion_number>,
                           <useafter_date>,
                           <comparison_file_name>,
                           <opus_flag>,
                           <comment>,
                           'N')
```

where all values (denoted by < >) are obtained from the load file header except <expansion_number> which is obtained from the detailed section of the load file and . The delivery_num value is retrieved from the database as described earlier.

Testing

The test script *cbds_sql_gen.sh* is available to test the *cdbs_sql_gen* tool. The test script is invoked by entering its name. It is assumed that you are running the script from the cdbs/test/script directory and that your data is in the cdbs/test/data directory and that a cdbs/test/data/tmp directory exists. The script will execute *cdbs_sql_gen* on a number of test load files and populate the CDBS test database using the SQL files generated. The test data consists of the following files:

- unex-hrs-init.lod - this file is an example of an initial unexpanded load file which doesn't reference a comparison file already in the database. The comparison_file value in the load file = (INITIAL). Note that the equivalence class values generated will all be based on the new values since there is no comparison file.

- unex-hrs-next.lod - this is an example of an unexpanded load file which does reference a comparison file already in the database. This example shows how equivalence class values are determined for the various change levels.

- ex-hrs-init.lod - this file is an example of an initial expanded load file which doesn't reference a comparison file already in the database. This just shows the output structure for an expanded file

- ex-hrs-next.lod - this is an example of an expanded load file which does reference a comparison file already in the database. This demostrates that all equivalence class values are based on the change_level of the first detailed row entry.

- synphot-init.lod - this is an example of an initial unexpanded load file for instrument synphot. It demonstrates that equivalence class and mode information are output when reference_file_type is THROUHPUT.

- synphot-comptab.load - this is an example of an unexpanded load file for instrument synphot. It demonstrates that equivalence class and mode information are not output when reference_file_type is COMPTAB.

- synphot-other.load - this is an example of an unexpanded load file for instrument synphot. It demonstrates that equivalence class info is output and mode information is not output when reference_file_type is neither COMPTAB or THROUGHPUT.

- multi.lod - this is an example of an unexpanded load file for instrument multi. Note that equivalence class data is omitted.

Note that these files are not intended to be accurate examples of CDBS load files but are only intended to demonstrate the various program paths.