Import & Export Utilities

ExtenDB Parallel Server

Version 1.1

October 2006

Table of Contents

1 Introduction

The ExtenDB Parallel Server offers two different tools for importing and exporting data.

XDBLoader is designed to load data as quickly as possible. It should be used especially when initially populating the database.

The xdbimpex utility is for both importing and exporting data to and from the database. It is not as fast as XDBLoader when importing, so using XDBLoader is recommended.

1.1 Performance Considerations

In populating the database as fast as possible, there are some things to consider.

1. After creating the tables, it is best to load data before creating any indexes or primary or foreign key constraints. The entire process will complete sooner.

2. Modifying the parameters of the underlying database. You may want to change the database configuration temporarily to speed up the loading or data. For example, if using PostgreSQL 8.1:

    a. From the PostgreSQL manual: "Temporarily increasing the `checkpoint_segments` variable can also make large data loads faster. This is because loading a large amount of data into PostgreSQL can cause checkpoints to occur more often than the normal checkpoint frequency (specified by the `checkpoint_timeout` configuration variable). Whenever a checkpoint occurs, all dirty pages must be flushed to disk. By increasing `checkpoint_segments` temporarily during bulk data loads, the number of checkpoints that are required can be reduced."

    b. From the PostgreSQL manual: "Increase `maintenance_work_mem`. Temporarily increasing the `maintenance_work_mem` configuration variable when loading large amounts of data can lead to improved performance. This is because when a B-tree index is created from scratch, the existing content of the table needs to be sorted. Allowing the merge sort to use more memory means that fewer merge passes will be required. A larger setting for `maintenance_work_mem` may also speed up validation of foreign-key constraints."

    c. Fsync. Setting `fsync` in the `pg_hba.conf` file to false is generally not a good idea since it does not guarantee writes to disk have occurred, but

can be considered to disable temporarily when doing initial loading of the database. We recommend leaving it set to the default, true, but wanted to point out this option nonetheless.

2 XDBLoader

Syntax:

```
XDBLoader -d database -u username [-p password]
          -t table [-h host] [-s port]
          -i inputfilename [-f separator] [-c configfilename]
         [-b bad_file [-r comment_prefix]]
         [-k chunk_count[,autoreducing_rate[,min_interval]]
             -y bad_chunk_directory [-x xrowid]]
         [-a] [-n] [-r comment] [-q quotechars [-e quoteescape]]
         [-z NULL] [-v] [-g]
         [-o connection_options]
```

The XDBLoader utility is designed to load up data quickly.

One of the best ways to do this is to leverage any loading facility of the underlying database. PostgreSQL, for example, includes the COPY command for fast loading. With the xdb.config file properly configured, the XDBLoader will read in data, determine the destination node destination for the target table, and then pipe the data into a process running the COPY command for the target node. By default, the xdb.config file is properly configured for PostgreSQL.

XDBLoader should be executed under the ExtenDB user.

Options:

-a	Added ending delimiter. By default, a field delimiter is required only between the fields, not after the final field. Including -a indicates that a trailing final delimiter is present.
-b bad_file	If specified, some basic checks will be done on the lines of the input file, checking if the correct number of delimiters exist and that columns defined as not null have values. The bad lines are written to bad_file, but the load will continue. See also option -r, in order to see the rejection causes.
-c	Full path to the xdb.config file that corresponds to server.
-d database	The ExtenDB database to connect to.
-e char	Quote escape character
-f separator	Separator. The field delimiter.
-g	Generate serial ids. This means that one of your columns is defined as a serial id, that your input file does not contain any values for

	it, and that you would like ExtenDB to generate serial values for it.
-h host	Host to connect to
-i inputfile	Input file to load from. If not specified, data is loaded from stdin.
-j jdbcurl	The JDBC url to use to connect to the ExtenDB Server
-k chunk_count	This instructs the loader to break up committing the bulk load operations into "chunks", every chunk_count rows. This is useful because normally if even a single insert fails on the back end, the entire load will fail. Instead, -k will still allow good segments of data to be committed, and just flag bad ones that contain problematic input. The bad chunks are created as new files at the path location specified by -o. It is recommended to try and use a fairly high chunk count if possible, like 100000, for performance reasons when loading a lot of data.
-n	No transformations; pass through. Just read the file and try and import as quickly as possible, without performing any transformations that may slow down performance.
-o	Connection options, available to most command line utilities. Available options are mode and charset. If mode is set to P, the connection to the database is persisted. The charset option refers to the character set used. Example: -o mode=P charset=utf8
-p	The password to use when connecting. If not included, the user will be prompted
-q	Quote character.
-r string	Remark (comment) string. Lines that start with this will be ignored. If used in conjunction with -b, all bad input lines will be written out to the bad file, preceeded by a comment line starting with the string here, explaining the reason for the rejection.
-s port	The socket port to connect to. By default it is 6453.
-t table	Target table
-u username	The username to use when connecting
-v	Verbose mode
-x	Used in conjunction with -k and -o, it indicates that the loader should xrowid (internal row identifier) in the rejected chunk files.
-y bad_chunk_directory	This is used in conjunction with -k, and instructs the loader where to create bad chunk files.
-z nullvalue	Value that indicates null

2.1 Handling Bad Input Lines

The loader contains additional options for handling input files that may cause errors when loading. This will allow you to try and continue loading as much data as possible, even if you encounter an error.

These command line parameters are optional by default, because it does involve some extra overhead and may slow down your load process. If you know you have clean data, you can leave these using these options off, otherwise you should find these to be helpful.

The first of these is by including the –b and, optionally, the –r options. You can specify a file name following –b. This instructs the loader that you wish to perform some basic checks before the data is sent to the backends. The basic checks include checking that the number of delimiters is correct and that the columns defined as "NOT NULL" have a value. It will not perform more complicated checks like verify foreign key constraints.

If an input line fails any of the basic checks, it is written to the file name specified by the –b option. You can also include –r followed by a comment string like 'REM' or '#'. This will precede each bad line in the bad file with the reason why it failed to load, to make it easier for the user to clean up and try again.

Note that during the load if you leave off –b and a critical error occurs, like not being able to find a value for the partitioning column, the process will abort, even if you included the –k option. If you wish to always continue to try and load data, please include –b.

The second set of options to use is –k and –y. With –k, the input is broken out into the "chunk" row count specified. This allows smaller discrete segments of the input file to be committed if there are not any errors. Should an error occur on one of the backends, a new file will be created in the directory specified by –y. This allows the user to try and clean up any problems and reload the data, potentially in turn processing it in smaller and smaller chunks until the data is clean.

The bad chunk files are created in the format:

```
<database_host>_<node_database>_table_<internal_id>.dump
```

There is one file per target underlying database. Including the –x option with –k and –y indicates that the generated xrowid identifier should be included in the bad chunk files, if being used in the table. While this is not recommended, this gives the user the flexibility to try and apply the bad chunk file directly to the underlying database, bypassing XDBLoader.

The –k option also allows you to specify a auto-reduce rate and minimum row amount, in addition to the chunk size, separated by commas, without any spaces. The advantage of this is if a chunk is bad, the loader will automatically break it out into "line count /auto-reduce rate" separate sub-chunks and to retry loading the rows and narrow down the particular problematic lines. This process is repeatedly recursively up until the minimum amount of specified rows.

The exact options to use with –k depend on how clean you think your data is. For performance, if few errors are expected, a large count number should be used.

Example: -k 100000,10,1.

This will result in a chunk size of 100,000 being used. If a chunk fails, that is broken out into 10 sub-chunks, resulting in chunks of 10,000 lines being used. Those that fail will be broken out to 1,000, then 100, then 10, and finally 1. The loader will have loaded up all of the lines that it could; the only remaining lines in the bad chunk files are the ones that it could not load up.

Both of these methods of handling bad input lines can be used together: -b to catch basic input errors on a line-by-line basis, and -k for ones that the underlying database catches.

2.2 Example Usage

```
XDBLoader.sh –d BIGDB –u myuser –p mypassword –h localhost
  –i /home/extendb/mig/order_fact.tbl –t orders –f '|'
  –b /home/extendb/mig/bad/order_fact.bad –r '#'
  –k 100000,20,1 –y /home/extendb/mig/bad
```

3 xdbimpex

Like XDBLoader, xdbimpex can also be used to import data. It offers a little more flexibility at the cost of slower load speeds. In addition, xdbimpex includes the ability to export data from tables as well.

Modes

There are 2 operating modes, import and export, the modes of which are mutually exclusive. Import is invoked with the "-i" and export with –x, where in either case it is followed by the source or target file.

An optional format file may be used with the "-f" option to allow more complex mapping information to appear. If the import is relatively simple, the user can also just enter the desired options on the command line.

3.1 Format File and Command Line options

Importing

[INFILE=file_name]
[TARGET=table_name]
[OVERWRITING=[0|1] (default is 0)
| IGNORE=[0|1]] (default is 0) (at most only one of these two can be set)
[[DELIMITER=delimiter]
|[column_name delimited_position, [n…]]]
[ADD_TRAILING_DELIMITER=[0|1]] (default is 0)
[TERMINATOR=terminator]
[LOCK=[0|1]] (default is 0)
[SILENT=[0|1]] (default is 0)
[START_LINE=line_num]
[END_LINE=line_num]
[POSITION_FORMATTED { column_name start:stop, [n...] }]
[QUOTED=quote_character]
[COMMIT_INTERVAL=integer]
[MAX_ERRORS=integer]
[DATA_ERROR_FILE=filename]

[DRIVERCLASS=driverclass] (default to extendb.connect.XDriver)]

[JDBC_URL=jdbc_url of target database]

Exporting

[EXTRACT=query_string]

[OUTFILE=file_name]

[TRIM_TRAILING_SPACES=[0|1] (default is 0)

A table appears below that describes both the command line options and the format file parameters, depending on the preferred mode of usage.

Format File Value	Command Line Option	Description
	-f	Specifies a format file to use to allow more complex mapping information to appear. Followed by the file name for the formatting. Command line option only.
INFILE	-i	Import (-i), followed by the source file. If no source file specified, data is read from stdin. Required for command line operation
TARGET	-t	The target table, if importing
OUTFILE	-x	Export (-x), followed by the query sting and output or target file name. Required for command line operation
EXTRACT (query string)	-y	The SQL query to run to get the data. If it is just a single word, it is assumed to be the name of the table and will do a “SELECT * FROM <table>”.
OVERWRITING or IGNORE	-w, -g	Used for handling input records that duplicate existing records on primary key values. If OVERWRITING is specified, rows will get overwritten with the new data, provided they have the same value for primary or unique index as the row to be replaced. If IGNORE is specified, rows will be ignored with the new data if they have the same value for primary or unique index as the row to be replaced. If neither option is present, it will always try and insert the row (default). These are mutually exclusive.
DELIMITER	-d	Default delimiter is pipe (|). Optionally, in the format file, it can be followed by matching column names with the positional delimited items, to allow the data to be mapped. Command line option for mapping column names is not available.
ADD_TRAILING_DELIMITER	-a	Indicates that a final delimiter follows the last field.
TERMINATOR	-z	Default is carriage return

LOCK	-l	Whether or not to lock the entire table
SILENT	-h	If Omitted, the number of rows processed will be displayed every 10,000 rows. Default is verbose
START_LINE	-s	Default will begin at 1. This is useful if importing from a large file and something goes wrong after 210,000 records for example. The import can be restarted with the same import file, but told to start on line number 210,001.
END_LINE	-e	Default will be the end of file
POSITION_FORMATTED	-p	Used to match column_name with start and stop character positions of data in the row, for non-delimited, fixed format import files.
QUOTED	-q	Used if data is quoted, surrounded by " or '.
COMMIT_INTERVAL	-c	Default is to commit after each insert. Otherwise, batches will be used, and the batch will be committed after every COMMIT_INTERVAL number of rows. **It is important to use this for faster loads.** A default of 1000 is a good value to start with.
MAX_ERRORS	-m	Default is 1. Set to any positive integer to instruct the loader to continue processing up until at least that many errors occur. Setting to 0 (zero) will ignore all errors, and always continue to load the next line from the file.
DATA_ERROR_FILE	-r	Specifies target file for rows that could not be loaded up successfully. This way, the user can first try and load entire file, then just work with problematic data in a separate file that could not be loaded up, and try again.
JDBC_URL	-j	The JDBC URL for connecting to the server. For example: jdbc:xdb:BIGDB:myuser/mypassword@extendbhost
DRIVERCLASS	-C	The driver class name, if exporting from other databases, for example, like: org.postgresql.Driver.
TRIM_TRAILING_SPACES	-T	If set, strings that are read from the source that have trailing spaces in them will be trimmed when writing to the output file. That is useful for saving disk space for large files, but it can impact your data- if you were expecting a column to contain a single space, for example, it will now be empty.

3.2 Importing

A command line option should be available for use with all the commands unless there are mapping columns used, as available POSITION_FORMATTED. If the column order differs in the source file from the target table, the user must use a format file to describe the mapping and cannot do this via the command line.

Example:

```
xdbimpex -c 1000 -d '|'  -i customer.dat -t customer
```

```
-j jdbc:xdb:BIGDB:myuser/mypassword@extendbhost
```

This will import the customer.data file into customer, with a pipe delimiter and a batch size of 1000, using the specified jdbc string.

3.3 Exporting

Examples:

```
xdbimpex -x orders.out -t orders
      -j jdbc:xdb:BIGDB:myuser/mypassword@extendbhost

xdbimpex -x orders.out -y orders
      -j jdbc:xdb:BIGDB:myuser/mypassword@extendbhost

xdbimpex -x orders.out -y "select * from orders"
      -j jdbc:xdb:BIGDB:myuser/mypassword@extendbhost
```

The following example demonstrates using a format file and exporting from a PostgreSQL database:

```
xdbimpex -f format.txt
```

where format.txt is:

```
EXTRACT=select * from atable
DRIVERCLASS=org.postgresql.Driver
JDBC_URL=jdbc:postgresql://localhost/mydb?user=myuser&passw
ord=mypassword
OUTFILE=/tmp/atable.txt
```