NAME

db_archive − the DB database archiver

SYNOPSIS

db_archive [**-alsv**] [**-h home**]

DESCRIPTION

The *db_archive* utility writes the pathnames of log files that are no longer in use (e.g., no longer involved in active transactions), to the standard output, one pathname per line. These log files should be written to backup media to provide for recovery in the case of catastrophic failure (which also requires a snapshot of the database files), but they may then be deleted from the system to reclaim disk space.

The options are as follows:

−a Write all pathnames as absolute pathnames, instead of relative to the database home directories.

−h Specify a home directory for the database.

−l Write out the pathnames of all of the database log files, whether or not they are involved in active transactions.

−s Write the pathnames of all of the database files that need to be archived in order to recover the database from catastrophic failure. If any of the database files have not been accessed during the lifetime of the current log files, *db_archive* will not include them in this output.

It is possible that some of the files referenced in the log have since been deleted from the system. In this case, *db_archive* will ignore them. When *db_recover*(1) is run, any files referenced in the log that are not present during recovery are assumed to have been deleted and will not be recovered.

−v Run in verbose mode, listing the checkpoints in the log files as they are reviewed.

The *db_archive* utility attaches to DB shared memory regions. In order to avoid region corruption, it should always be given the chance to detach and exit gracefully. To cause *db_archive* to clean up after itself and exit, send it an interrupt signal (SIGINT).

The *db_archive* utility exits 0 on success, and >0 if an error occurs.

DB ARCHIVAL PROCEDURES

There are two aspects to managing the recoverability and disk consumption of your DB databases. First, you may want to periodically create snapshots of your databases to make it possible to recover them from catastrophic failure. Second, you'll want to periodically remove log files in order to conserve on disk space. The two procedures are distinct from each other, and you cannot remove the current log files simply because you have created a database snapshot.

To create a snapshot of your database that can be used to recover from catastrophic failure, the following steps should be taken:

1. Run *db_archive* −s to identify all of the database data files that must be saved, and copy them to a backup device, (e.g., tape). If the database files are stored in a separate directory from the other database files, it may be simpler to archive the directory itself instead of the individual files.

   More importantly, if any of the database files have not been accessed during the lifetime of the current log files, db_archive will not list them in its output! For this reason, it may be important to use a separate database file directory, archiving it instead of the files listed by *db_archive*.

2. If your database is currently active, i.e., you are reading and writing to the database files while the snapshot is being taken, run *db_archive* −l to identify the database log files, and copy them to a backup device, (e.g., tape). If the database log files are stored in a separate directory from the other database files, it may be simpler to archive the directory itself instead of the individual files.

Note that the order of these operations is important, and that the database files **must** be archived before the log files.

The DB library supports on-line backups, and it is not necessary to stop reading or writing your databases during the time when you create this snapshot. Note however, that the snapshot of an active database will be consistent as of some unspecified time between the start of the archival and when archival is completed. To create a snapshot as of a specific time, you must stop reading and writing your databases for the entire time of the archival, in which case the log files do not need to be archived.

Once these steps are completed, your database can be recovered from catastrophic failure to its state as of the time the archival was done. To update your snapshot so that recovery from catastrophic failure is possible up to a new point in time, repeat step #2, copying all existing log files to a backup device.

Each time that a complete snapshot is made, i.e. all database and log files are copied to backup media, you may discard all previous snapshots and saved log files.

The time to restore from catastrophic failure is a function of the number of log records that have been written since the snapshot was originally created. Perhaps more importantly, the more separate pieces of backup media you use, the more likely that you will have a problem reading from one of them. For these reasons, it is often best to make snapshots on a regular basis.

For archival safety remember to ensure that you have multiple copies of your database backups, that you verify that your archival media is error-free, and that copies of your backups are stored off-site!

To restore your database after catastrophic failure, the following steps should be taken:

1. Restore the copies of the database files from the backup media.
2. Restore the copies of the log files from the backup media, **in the order in which they were written**. (It's possible that the same log file appears on multiple backups, and you only want the most recent version of that log file!)
3. Run *db_recover* −c to recover the database.

It is possible to recreate the database in a location different than the original, by specifying appropriate pathnames to the −h option of the *db_recover* utility.

To remove log files, the following steps should be taken:

1. If you are concerned with catastrophic failure, first copy them to backup media (e.g., tape), as described above. This is because log files are necessary for recovery from catastrophic failure.
2. Run *db_archive*, without options, to identify all of the log files that are no longer in use (e.g., involved in an active transaction).
3. Remove those log files from the system.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *db_archive*:

DB_HOME

If the **−h** option is not specified and the environment variable *DB_HOME* is set, it is used as the path of the database home, as described in *db_appinit*(3).

SEE ALSO

The DB library is a family of groups of functions that provides a modular programming interface to transactions and record-oriented file access. The library includes support for transactions, locking, logging and file page caching, as well as various indexed access methods. Many of the functional groups (e.g., the file page caching functions) are useful independent of the other DB functions, although some functional groups are explicitly based on other functional groups (e.g., transactions and logging). For a general description of the DB package, see *db_intro*(3).

db_archive(1), *db_checkpoint*(1), *db_deadlock*(1), *db_dump*(1), *db_load*(1), *db_recover*(1), *db_stat*(1), *db_intro*(3), *db_appinit*(3), *db_cursor*(3), *db_dbm*(3), *db_internal*(3), *db_lock*(3), *db_log*(3), *db_mpool*(3), *db_open*(3), *db_thread*(3), *db_txn*(3)