NAME

db_recover − the DB database recovery utility

SYNOPSIS

db_recover [**-cv**] [**-h home**]

DESCRIPTION

The *db_recover* utility must be run after an unexpected application, DB, or system failure to restore the database to a consistent state. All committed transactions are guaranteed to appear after *db_recover* has run, and all uncommitted transactions will be completely undone.

The options are as follows:

−c Failure was catastrophic.

−h Specify a home directory for the database.

−v Run in verbose mode.

In the case of catastrophic failure, an archival copy, or ''snapshot'' of all database files must be restored along with all of the log files written since the database file snapshot was made. (If disk space is a problem, log files may be referenced by symbolic links). For further information on creating a database snapshot, see ''DB ARCHIVAL PROCEDURES'' in *db_archive*(1).

If the failure was not catastrophic, the files present on the system at the time of failure are sufficient to perform recovery.

If log files are missing, *db_recover* will identify the missing log file(s) and fail, in which case the missing log files need to be restored and recovery performed again.

The *db_recover* 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_recover* to clean up after itself and exit, send it an interrupt signal (SIGINT).

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

FILESYSTEM OPERATIONS

Filesystem operations, e.g., moving the database environment to a different machine or file creation, deletion or renaming, cannot be transaction protected. For this reason, *db_recover* cannot re-create, delete or rename files as part of recovery.

If *db_recover* cannot find a database file referenced in the log, it will output a warning message that it was unable to locate a file it expected to find. This message is only a warning, as the file may have subsequently been deleted as part of normal database operations before the failure occurred. **Note that committed transactions that involved these missing files are rolled forward, even though the files were not found. If the files were not intentionally deleted (e.g., they were created after the last database snapshot, but were lost during a failure), they must be manually created (using** *db_open*(3)), and *db_recover* must be rerun.

Generally, it is simplest to perform filesystem operations at the same time as making a snapshot of the database. To perform filesystem operations:

1. Cleanly shutdown database operations.
2. Rename, create or delete files.
3. Make a snapshot of the database.
4. Restart database applications.

To cleanly shutdown database operations, all applications accessing the database environment must be shutdown. If the applications are not implemented such that they can be shutdown gracefully (i.e., closing all references to the database environment), *db_recover* must be run after all applications have been killed to ensure that the underlying database is consistent and flushed to disk.

RECOVERY

After an application or system failure, there are two possible approaches to database recovery. If there is no need to retain state across the failure, and all databases can be started anew, the database home directory can simply be removed and recreated. If it is necessary to retain persistent state across failures, then the *db_recover*(1) utility should be run for each DB application environment, i.e., each database home directory.

The *db_recover* utility will remove all the shared regions (which may have been corrupted by the failure), establish the end of the log by identifying the last record written to the log, and then perform transaction recovery. Database applications must not be restarted until recovery completes. During transaction recovery, all changes made by aborted transactions are undone and all changes made by committed transactions are redone, as necessary. After recovery runs, the environment is properly initialized so that applications may be restarted. Any time an application crashes or the system fails, *db_recover* should be run on any exiting database environments.

Additionally, there are two forms of recovery: normal recovery and catastrophic recovery. The DB package defines catastrophic failure to be failure where either the database or log files have been destroyed or corrupted from the point of view of the filesystem. For example, catastrophic failure includes the case where the disk drive on which either the database or logs are stored has crashed, or when filesystem recovery is unable to bring the database and log files to a consistent state with respect to the filesystem. If the failure is non-catastrophic, i.e., the database files and log are accessible on a filesystem that has recovered cleanly, *db_recover* will review the logs and database files to ensure that all committed transactions appear and that all uncommitted transactions are undone. If the failure is catastrophic, a snapshot of the database files and the archived log files must be restored onto the system. Then *db_recover* will review the logs and database files to bring the database to a consistent state as of the date of the last archived log file. Only transactions committed before that date will appear in the database. See *db_archive (1)* for specific information about archiving and recovering databases after catastrophic failure.

ENVIRONMENT VARIABLES

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

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).

BUGS

Future versions of DB are expected to remove the restriction that database files must be manually created before recovery is performed.

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)