RAIMA

Raima Database Manager 11.0

dbRepair Guide

Trademarks

Contents

Introduction

DBREPAIR is a suite utilities that implement a number of functions to support database maintenance in critical situations of database corruption or in cases of performance degradation. The functions of the module perform the following operations:

Utility Program	Description
`dbcheck`	Consistency check of the database records, keys and sets and diagnostics of database corruption cases. Creates database repair list for **dbrepair**.
`dbrepair`	Complete or partial repair of corrupted database structure. Fixes problems encountered by **dbcheck** with the exception of key files (use **keybuild** instead).
`keybuild`	Rebuild of key files.
`dbdefrag`	Compression of fragmented database
`dchain`	Rebuilding of delete chains (linked list of delete records)
`dbcluster`	

There are two types of database corruptions: physical and structural. Physical database corruption is caused by writing invalid data directly into a database file or through a corrupted cache page. The former can happen through an occasional direct modification of the database files from other applications. The latter can only be caused by an error in the direct-linked application or by a bug in the runtime engine. As a result, some fields in the database records contain garbage data.

In general, the DBREPAIR utility cannot restore the value of the corrupted data fields. But if the fields are the database pointers used for a set relationship, the utility can attempt to restore these values in those fields using redundant pointers in doubly-linked lists. We say that a database pointer, which is a value of the DB_ADDR type, is physically corrupted if it is not valid for the particular database. A DB_ADDR value is valid as a database pointer if its `fileno` component is a valid file number and its `slotno` component is a valid slot number for this file. If a pointer is a set pointer the `fileno` component additionally must be consistent with the set definition in the database schema.

Structural corruption is caused by inconsistency in several associated pointers that represent owner-member relationships between database records.

Utility Descriptions

This section contains descriptions of the Db Repair utilities in alphabetical order. Each page header contains the name of the utility for easy reference. The Prototype section contains the command-line syntax showing all possible parameters and options. The Description section explains the purpose of the utility and any command-line parameters. The Options section explains each command-line option.

dbcheck

Database consistency check utility

Prototype

```
dbcheck [-docroot path] [-s] [-k] [-dk] [-kd] [-v] [-dv] [-vd] [-b] [-db] [-a]
    [-openmode mode] [-nk] [-nv] [-nb] [-qd] [-qk] [-r #] [-p #] [-f #] [-t]
    [-c] [-l] [-locale locale] [-stderr filename] [-q|-stdout filename]
    dbname [dbfile[ dbfile]...]
```

Description

This utility checks the consistency of all database files in database dbname. If any database files (dbfile) are specified, only those files are checked. Otherwise, all files in the database are checked.

The **dbcheck** utility checks database consistency by validating the position of each record occurrence and checking the integrity of the delete chains. Options perform more complete consistency checks. Use of these options slows down **dbcheck** execution.

With the -s option, the **dbcheck** utility can perform set consistency checking. Set membership consistency verifies the following:

1. Member and owner record types are valid
2. Membership count is correct
3. Doubly linked lists are properly formed

The -k option causes **dbcheck** to verify the B-tree structure of the key files. This ensures that each node (except the root node) is at least half full, that the number of filled slots is correct, that the key slots on each node are properly sorted, and that each leaf node is at the correct level in the B-tree. Note that if the -dk option does not result in any errors, it is very unlikely that the -k option will result in any inconsistencies.

Key file structure consistency checks performed (optionally) verify that:

1. All B-tree pages are either in the B-tree or are deleted
2. All B-tree null child pointers are at the same tree level
3. B-tree key order is correct
4. Duplicate keys are correct

The -dk option validates the existence of the key values associated with each record and key field in the data files. Optional keys are only checked if the key has been stored. The -kd option will read, for each key in a key file, the associated record and check to ensure that the key's data field contents matches that stored in the key file. If the key is optional, the "key stored" bit in the record header is also checked.

Inconsistencies are reported with a message indicating the nature of the inconsistency and the file and location of the offending record or key. If the -t option is specified, a trace back of the B-tree is printed when a key file inconsistency is detected.

The -v option causes **dbcheck** to verify the structure of varchar files. This ensures that each varchar is less than the maximum size, that the varchar pointers point to valid locations, and varchar file chunk points to a valid varchar field.

Options

`-q`	Quiet - suppresses all output (except errors)
`-docroot` *`path`*	[non-RPC configuration only] Specifies the document root when in Direct-Link or Standalone configuration.
`-s`	Performs a complete consistency check of sets.
`-k`	Performs B-tree structure consistency check. This checks to ensure that the key file adheres to all of the rules for a B-tree. If -dk is also specified, this check is unnecessary.
`-dk`	Checks the existence of each key from each record occurrence. While scanning a data file, a `d_keyfind` is performed on each key to ensure that it exists.
`-kd`	Checks the existence of each record from each key. While scanning a key file, a `d_recread` function call is performed and the associated key field value is checked against the key to ensure that it matches.
`-v`	Perform checks on varchar files.
`-dv`	Checks the existence of each varchar chunk referenced from a varchar field while scanning a data file.
`-vd`	Checks the validity of the owner pointer for each varchar chunk in a varchar file.
`-b`	Perform BLOB file structure consistency check.
`-db`	Checks the existence of each BLOB from each record occurrence.
`-a`	All. Performs all of the above consistency checks.
`-openmode` *`mode`*	Spedify the open mode to use for the database (default 's' using ROT).
`-nk`	Disables key file checking, -k, -dk and -kd.
`-nv`	Disables varchar checking, -v, -dv and -vd.
`-nb`	Disables blob checking, -b and -db.
`-qd`	Quick dchain checks (only report one dchain error per file).
`-qk`	Quick key checks (only report one error per key file).
`-r #`	Reports every # percentage completed to stderr.
`-p #`	Sets to # the number of pages in the RDM cache for use by **dbcheck**. For example, `-p 128` specifies that **dbcheck** is to allocate 128 pages for the cache. The default is 64 pages. If an insufficient memory error (S_NOMEMORY) occurs when starting dbcheck, use the `-p` option to specify a value less than 64.
`-f #`	Maximum number of open files allowed to have open at once.
`-t`	Prints a trace back of the B-tree when a key file inconsistency is detected.
`-c`	Prints a count of objects scanned in the check.
`-l`	Log errors to repair database.
`-locale` *`locale`*	Specify a *locale* to use for ordering string values.
`-stderr` *`filename`*	Print stderr to *filename*.
`-stdout` *`filename`*	Print stdout to *filename*

dbcluster

Database set optimization utility

Prototype

```
dbcluster [-stderr filename] [-q|-stdout filename]
          [-docroot path] [-locale locale] [-v]
             dbname [@datfile|setname(s)]
```

Description

The **dbcluster** utility relocates all members of the selected sets contiguously in the database storage files.

A database contains one or more data files. Each data file contains database pages of a specific size (size can be set by the user), with each page containing as many records (slots) as will fit on a page. The data files can contain multiple owner and member record types. It is easy to conceive that during normal database activity many different types of records could be intermixed during inserts. As this occurs and the database grows large, performance can degrade due to set navigation having to traverse a set through multiple pages. Dbcluster can help by placing related set members into contiguous record slots.

It is not always possible to optimize every set defined in a database. If a record is a member in multiple sets it is only possible to optimize one of the sets. You can either explicitly specify the set to optimize or the utility will choose the first set defined in the database.

Procedures

1. Run dbcheck on your database and ensure that the database is valid.
2. Backup your database.
3. Run the dbcluster utility.

Options

`-q`	Quiet - suppresses all output (except errors)
`-stderr filename`	Specify redirection of stderr into *filename*.
`-stdout filename`	Specify redirection of stdout into *filename*. Cannot use -v with this option.
`-docroot path`	[non-RPC configuration only] Specifies the document root when in Direct-Link or Standalone configuration.
`-locale locale`	Specify a locale to use for collating string values
`-v`	Verbose - shows detailed output of the operations performed on the database.
`dbname`	Specifies the database to optimize
`datfile`	A file that contains a list of set names to optimize
`setname(s)`	A list of sets to optimize

See Also

d_dbcluster

dbdefrag

Database defragmentation utility

Prototype

```
dbdefrag [-docroot path] [-v] [-stderr filename] [-q|-stdout filename]
         [-locale locale]  dbname
```

Description

Each database consists of one or more data files. Each of these data files contains database pages which in turn consist of database slots (records).

Normally, whenever a record is deleted from the database, it is marked for reuse and put on the delete chain, but not physically removed from the file. These free (deleted) records can occupy a substantial amount of disk space, especially if a database has had numerous records inserted and subsequently many of them marked as deleted.

The **dbdefrag** utility moves all valid records to the front of a data file and all deleted slots to the end of the data file. The data file is then truncated in order to reclaim disk space.

The defragmentation takes place in the following operations.

1. A temporary database is created by **dbdefrag** to store the addresses of the deleted slots
2. The records are moved to the front of the database and the files are truncated if warranted.
3. The references to set owners, set members and database address fields, if used, are updated.
4. New key files are created based upon the new data files.

Options

`-q`	Quiet - suppresses all output (except errors)
`-stderr` *`filename`*	Specify redirection of stderr into *filename*.
`-stdout` *`filename`*	Specify redirection of stdout into *filename*. Cannot use -v with this option.
`-docroot` *`path`*	[non-RPC configuration only] Specifies the document root when in Direct-Link or Standalone configuration.
`-locale` *`locale`*	Specify a locale to use for collating string values
`-v`	Verbose - shows detailed output of the operations performed on the database.

See Also

d_dbdefrag

dchain

Delete chain sort utility

Prototype

```
dchain [-docroot path] [-q] [-v] dbname [dbfile]...
```

Description

The **dchain** utility sorts the deleted record slots on the delete chains of the listed data files ([*dbfile*]...) from database dbname in database address order. If no files are listed, all data files in database dbname will be processed. This utility does not sort key file delete chains.

The purpose of this utility is to increase the probability that newly created records in the same file will be placed close together. The extent of any performance improvement, however, will depend upon the application. The most likely time to use this utility is after a periodic purge of a particular set of record types, many of which were entered together; thus there would be many contiguous record slots on the delete chain.

Options

`-docroot` *`path`*	[non-RPC configuration only] Specifies the document root when in Direct-Link or Standalone configuration.
`-q`	Only display error messages (to stderr).
`-v`	Display detailed output about all operations (to stdout).

See Also

d_dchain

keybuild

Key file build utility

Prototype

```
keybuild [-docroot path] [-f N] [-p N] [-locale locale] [-q]
    [-stderr filename] [-q|-stdout filename] dbname
```

Description

The keybuild utility rebuilds all key files for the database `dbname`. Rebuilding key files is a two-step process. First, the file is reinitialized. Then, each record is sequentially read from each data file record, and each key file is re-created from the record contents.

This utility can be used to re-create the key files when dbcheck reports a database inconsistency. The utility can also construct new key files after you have added or removed key attributes from fields in your DDL specification. For example, if you make an existing key field a non-key, and change a non-key to a key field in your DDL, you can run `keybuild` to rebuild the key files for the new schema. You can also use `keybuild` to reassign key fields to different key files.

Options

`-q`	Quiet - suppresses all output (except errors)
`-stderr` *`filename`*	Specify redirection of stderr into *filename*.
`-stdout` *`filename`*	Specify redirection of stdout into *filename*. Cannot use -v with this option.
`-docroot` *`path`*	[non-RPC configuration only] Specifies the document root when in Direct-Link or Standalone configuration.
`-locale` *`locale`*	Specify a locale to use for sorting string values.
`-f` *`N`*	Report progress to stdout every N records. Default is 1000. Set to 0 for no report.
`-p` *`N`*	The number of cache pages to use. Default is 1000.