NAME

db_cursor − database sequential access functions

SYNOPSIS

#include <db.h>

int
DBcursor->c_close(DBC *cursor);

int
DBcursor->c_del(DBC *cursor, int flags);

int
DBcursor->c_get(DBC *cursor, DBT *key, DBT *data, int flags);

int
DBcursor->c_put(DBC *, DBT *key, DBT *data, int flags);

DESCRIPTION

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).

This manual page describes the specific details of the cursor support for the DB access methods.

The *db_cursor* functions are the library interface supporting sequential access to the records stored by the access methods of the DB library. Cursors are created by calling the *cursor* function of a DB structure returned by *db_open*(3), which returns a pointer to a DBC structure.

Each cursor maintains positioning information within a set of key/data pairs. In the presence of transactions, cursors are only valid within the context of a single transaction, the one specified during the *cursor* call described in *db_open*(3). All cursor operations will be executed in the context of that transaction. Before aborting or committing a transaction, all cursors used within that transaction must be closed. In the presence of transactions, the application must call *txn_abort* if any of the cursor operations returns that a deadlock (EAGAIN) or system failure occurred.

When locking is enabled, page locks are retained between consecutive cursor calls. For this reason, in the presence of locking, applications should discard cursors as soon as they are done with them. Calling the db *close* function (see *db_open*(3)) discards any cursors opened in the context of a particular DB structure returned by the *db_open* call.

The cursor has an associated set of functions to access elements in the database, accessed through the DBC structure. The elements of the DBC structure are defined as follows:

int (*c_close)(DBC *cursor);

> A pointer to a function that discards the cursor. No further references to the cursor should be made.
>
> The *c_close* function returns the value of *errno* on failure and 0 on success.

int (*c_del)(DBC *cursor, int flags);

> A pointer to a function that deletes the key/data pair currently referenced by the cursor.
>
> The *flags* parameter is currently unused, and must be set to 0.
>
> The cursor position is unchanged after a delete and subsequent calls to cursor functions expecting the cursor to reference an existing key will fail.
>
> The *c_del* function returns the value of *errno* on failure, 0 on success, and DB_KEYEMPTY if the element has already been deleted.

int (*c_get)(DBC *cursor, DBT *key, DBT *data, int flags);

> A pointer to a function that retrieves key/data pairs from the database. The address and length of the key are returned in the structure referenced by *key* (except for the case of the DB_SET flag where the *key* structure is unchanged), and the address and length of the data are returned in the structure

referenced by *data*.

Modifications to the database during a sequential scan will be reflected in the scan, i.e. records inserted behind a cursor will not be returned while records inserted in front of a cursor will be returned.

In recno databases, missing entries (i.e., entries that were never explicitly created or that were created and then deleted), will be skipped during a sequential scan.

If multiple threads or processes insert items into the same database file without using locking, the results are undefined. For more detail, see the section below on cursor stability.

The parameter *flags* must be set to exactly one of the following values:

DB_FIRST
: The cursor is set to reference the first key/data pair of the database, and that pair is returned. In the presence of duplicate key values, the first data item in the set of duplicates is returned.

  If the database is empty, the *c_get* function will return DB_NOTFOUND.

DB_LAST
: The cursor is set to reference the last key/data pair of the database, and that pair is returned. In the presence of duplicate key values, the last data item in the set of duplicates is returned.

  If the database is empty, the *c_get* function will return DB_NOTFOUND.

DB_NEXT
: If the cursor is not yet initialized, DB_NEXT is identical to DB_FIRST.

  Otherwise, move the cursor to the next key/data pair of the database, and that pair is returned. In the presence of duplicate key values, the value of the key may not change.

  If the cursor is already on the last record in the database, the *c_get* function will return DB_NOTFOUND.

DB_PREV
: If the cursor is not yet initialized, DB_PREV is identical to DB_LAST.

  Otherwise, move the cursor to the previous key/data pair of the database, and that pair is returned. In the presence of duplicate key values, the value of the key may not change.

  If the cursor is already on the first record in the database, the *c_get* function will return DB_NOTFOUND.

DB_CURRENT
: Return the key/data pair currently referenced by the cursor.

  If the cursor key/data pair has been deleted, the *c_get* function will return DB_KEYEMPTY.

  If the cursor is not yet initialized, the *c_get* function will return EINVAL.

DB_SET
: Move the cursor to the specified key/data pair of the database, and return the datum associated with the given key.

  In the presence of duplicate key values, *c_get* will return the first data item for the given key.

  If the database is a recno database and the requested key exists, but was never explicitly created by the application or was later deleted, the *c_get* function returns DB_KEYEMPTY.

  If no matching keys are found, the *c_get* function will return DB_NOTFOUND.

DB_SET_RANGE
: The DB_SET_RANGE flag is identical to the DB_SET flag, except that the key is returned as well as the data item, and, in the case of the btree access method, the returned key/data pair is the smallest key greater than or equal to the specified key, permitting partial key matches and

range searches.

DB_SET_RECNO

Move the cursor to the specific numbered record of the database, and return the associated key/data pair. The *data* field of the specified *key* must be a pointer to a memory location from which a *db_recno_t* may be read, as described in *db_dbt*(3). This memory location will be read to determine the record to be retrieved.

For DB_SET_RECNO to be specified, the underlying database must be of type btree and it must have been created with the DB_RECNUM flag (see *db_open*(3)).

DB_GET_RECNO

Return the record number associated with the cursor. The record number will be returned in the data DBT as described in *db_dbt*(3). The *key* parameter is ignored.

For DB_GET_RECNO to be specified, the underlying database must be of type btree and it must have been created with the DB_RECNUM flag (see *db_open*(3)).

Otherwise, the *c_get* function returns the value of *errno* on failure and 0 on success.

If *c_get* fails for any reason, the state of the cursor will be unchanged.

int (*c_put)(DBC *, DBT *key, DBT *data, int flags);

A pointer to a function that stores key/data pairs into the database.

The *flags* parameter must be set to exactly one of the following values:

DB_AFTER

In the case of the btree and hash access methods, insert the data element as a duplicate element of the key referenced by the cursor. The new element appears immediately after the current cursor position. It is an error to specify DB_AFTER if the underlying btree or hash database was not created with the DB_DUP flag. The *key* parameter is ignored.

In the case of the recno access method, it is an error to specify DB_AFTER if the underlying recno database was not created with the DB_RENUMBER flag. If the DB_RENUMBER flag was specified, a new key is created, all records after the inserted item are automatically renumbered, and the key of the new record is returned in the structure referenced by the parameter *key*. The initial value of the *key* parameter is ignored. See *db_open*(3) for more information.

If the cursor is not yet initialized, the *c_put* function will return EINVAL.

DB_BEFORE

In the case of the btree and hash access methods, insert the data element as a duplicate element of the key referenced by the cursor. The new element appears immediately before the current cursor position. It is an error to specify DB_BEFORE if the underlying btree or hash database was not created with the DB_DUP flag. The *key* parameter is ignored.

In the case of the recno access method, it is an error to specify DB_BEFORE if the underlying recno database was not created with the DB_RENUMBER flag. If the DB_RENUMBER flag was specified, a new key is created, the current record and all records after it are automatically renumbered, and the key of the new record is returned in the structure referenced by the parameter *key*. The initial value of the *key* parameter is ignored. See *db_open*(3) for more information.

If the cursor is not yet initialized, the *c_put* function will return EINVAL.

DB_CURRENT

Overwrite the data of the key/data pair referenced by the cursor with the specified data item.

The *key* parameter is ignored.

If the cursor is not yet initialized, the *c_put* function will return EINVAL.

DB_KEYFIRST

In the case of the btree and hash access methods, insert the specified key/data pair into the database. If the key already exists in the database, the inserted data item is added as the first of the data items for that key.

The DB_KEYFIRST flag may not be specified to the recno access method.

DB_KEYLAST

Insert the specified key/data pair into the database. If the key already exists in the database, the inserted data item is added as the last of the data items for that key.

The DB_KEYLAST flag may not be specified to the recno access method.

If the cursor record has been deleted, the *c_put* function will return DB_KEYEMPTY.

Otherwise, the *c_put* function returns the value of *errno* on failure and 0 on success.

If *c_put* fails for any reason, the state of the cursor will be unchanged. If *c_put* succeeds and an item is inserted into the database, the cursor is always positioned to reference the newly inserted item.

CURSOR STABILITY

In the absence of locking, no guarantees are made about the stability of cursors in different processes or threads. However, the btree and recno access methods guarantee that cursor operations, interspersed with other cursor or non-cursor operations in the same thread of control (i.e., thread or single-threaded process), will always return keys in order and will return each non-deleted key/data pair exactly once. Because the hash access method uses a dynamic hashing algorithm, it cannot guarantee any form of stability in the presence of inserts and deletes unless locking is performed.

If locking was specified when the DB file was opened, but transactions are not in effect, the access methods provide repeatable reads with respect to the cursor. That is, a DB_CURRENT call on the cursor is guaranteed to return the same record as was returned on the last call to the cursor.

In the presence of transactions, the access method calls between *txn_begin* and *txn_abort* or *txn_commit* provide degree 3 consistency. For all access methods, a cursor scan of the database performed within the context of a transaction is guaranteed to return each key/data pair once and only once, except in the following case. If, while performing a cursor scan using the hash access method, the transaction performing the scan inserts a new pair into the database, it is possible that duplicate key/data pairs will be returned.

ERRORS

The *DBcursor->c_close* function may fail and return *errno* for any of the errors specified for the following DB and library functions: calloc(3), fcntl(2), fflush(3), lock_get(3), lock_id(3), lock_put(3), lock_vec(3), log_put(3), malloc(3), memcpy(3), memmove(3), memp_fget(3), memp_fput(3), memp_fset(3), memset(3), realloc(3), and strerror(3).

In addition, the *DBcursor->c_close* function may fail and return *errno* for the following conditions:

[EAGAIN]

A lock was unavailable.

[EPERM]

Database corruption was detected. All subsequent database calls (other than *DB->close*) will return EPERM.

The *DBcursor->c_del* function may fail and return *errno* for any of the errors specified for the following DB and library functions: DB->del(3), calloc(3), fcntl(2), fflush(3), lock_get(3), lock_id(3), lock_put(3), lock_vec(3), log_put(3), malloc(3), memcpy(3), memmove(3), memp_fget(3), memp_fput(3), memp_fset(3), memset(3), realloc(3), and strerror(3).

In addition, the *DBcursor->c_del* function may fail and return *errno* for the following conditions:

[EAGAIN]

A lock was unavailable.

[EINVAL]
An invalid flag value or parameter was specified.

[EPERM]
Database corruption was detected. All subsequent database calls (other than *DB->close*) will return EPERM.

The *DBcursor->c_get* function may fail and return *errno* for any of the errors specified for the following DB and library functions: DB->get(3), calloc(3), fcntl(2), fflush(3), lock_get(3), lock_id(3), lock_put(3), lock_vec(3), log_put(3), malloc(3), memcmp(3), memcpy(3), memmove(3), memp_fget(3), memp_fput(3), memp_fset(3), memset(3), realloc(3), and strerror(3).

In addition, the *DBcursor->c_get* function may fail and return *errno* for the following conditions:

[EAGAIN]
A lock was unavailable.

[EINVAL]
An invalid flag value or parameter was specified.

The DB_THREAD flag was specified to the *db_open*(3) function and neither the DB_DBT_MALLOC or DB_DBT_USERMEM flags were set in the DBT.

[EPERM]
Database corruption was detected. All subsequent database calls (other than *DB->close*) will return EPERM.

The *DBcursor->c_put* function may fail and return *errno* for any of the errors specified for the following DB and library functions: calloc(3), fcntl(2), fflush(3), lock_get(3), lock_id(3), lock_put(3), lock_vec(3), log_put(3), malloc(3), memcmp(3), memcpy(3), memmove(3), memp_fget(3), memp_fput(3), memp_fset(3), memset(3), realloc(3), and strerror(3).

In addition, the *DBcursor->c_put* function may fail and return *errno* for the following conditions:

[EACCES]
An attempt was made to modify a read-only database.

[EAGAIN]
A lock was unavailable.

[EINVAL]
An invalid flag value or parameter was specified.

[EPERM]
Database corruption was detected. All subsequent database calls (other than *DB->close*) will return EPERM.

SEE ALSO

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)