NAME

scan_file_i – Classes for Scanning Files in the Shore Storage Manager

SYNOPSIS

```
#include <sm_vas.h>  // which includes scan.h

class scan_file_i {
public:
    /* Logical-ID version */
    NORET               scan_file_i(const lvid_t&     lvid,
          const serial_t&                fid,
          concurrency_t             cc = t_cc_file,
          bool                      prefetch = false);

    /* Physical-ID version */
    NORET               scan_file_i(const stid_t&     stid,
          concurrency_t             cc = t_cc_file,
          bool                      prefetch = false);


    /* Logical-ID version */
    NORET               scan_file_i(const lvid_t&     lvid,
          const serial_t&                fid,
          const serial_t&                start_rid,
          concurrency_t             cc = t_cc_file,
          bool                      prefetch = false);

    /* Physical-ID version */
    NORET               scan_file_i(const stid_t&     stid,
          const rid_t&              start_rid,
          concurrency_t             cc = t_cc_file,
          bool                      prefetch = false);


    NORET               ~scan_file_i();

    rc_t                next(pin_i*&pin_ptr,
          smsize_t                  start_offset,
          bool&                     eof);

    rc_t                next_page(pin_i*& pin_ptr,
          smsize_t                  start_offset,
          bool&                     eof);

    // logical serial # and volume ID of the file if created that way
    const serial_t&           lfid() const;
    const lvid_t&       lvid() const;

    bool                is_logical() const;
    tid_t               xid() const;

    void                finish();
    bool                eof();
    rc_t                error_code();
```

```
};

class append_file_i : public scan_file_i {
public:
    /* Logical-ID version */
    NORET               append_file_i(const lvid_t&   lvid,
          const serial_t&                fid;

    /* Physical-ID version */
    NORET               append_file_i(const stid_t&   stid);



    NORET               ~append_file_i();

    /* Logical-ID version */
    rc_t                        create_rec(const vec_t& hdr,
          smsize_t                    len_hint,
          const vec_t&                data,
          lrid_t&               lrid); // logical id

    /* Physical-ID version */
    rc_t                        create_rec(const vec_t& hdr,
          smsize_t                    len_hint,
          const vec_t&                data,
          rid_t&                rid); // physical id

};
```

DESCRIPTION

Class *scan_file_i* supports iterating over the records in a file. The scan is controlled by a *scan_file_i* object. Multiple scans can be open at the same time.

Class *append_file_i* allows a VAS to appending of records to a file in rapid succession. The location of the end of the file is maintained by the *append_file_i object.* The constructor of the *append_file_i* exclusively locks the file. The effect of using more than one *append_file_i* on a single file within a single transaction is undefined.

Each instance of *scan_file_i* and *append_file_i* keeps a record pinned (and therefore, a page fixed) throughout its existence. You must carefully control the use of these classes to avoid fixing all the pages in the buffer pool.

The order in which records are visited by a scan is called the *scan order* . There are two guarantees about scan order.

The first guarantee is that if two scans are performed on a file, the scan orders will be identical as long as none of the following operations occur between the two scans: **creating a** record in the file **destroying** a record in the file **changing the size** of a record in the file.

The second guarantee is that if a file is created using the **append_file_i** class, and no updates are performed on the file or any of its records, the scan order is identical to the order of record creation.

Constructors and Destructors

scan_file_i(lvid, fid, cc)

scan_file_i(lvid, fid, start_rid, cc)

The **scan_file_i** constructors have a number of parameters in common. The first two parameters, *lvid* and *fid* specify the **logical ID** of the file to be scanned. The *cc* parameter specifies the granularity of locks acquired for concurrency control. See **enum(ssm)** for a description of the values. Here are the effects of the values for file scan:

t_cc_none: The file is IS (intention shared) locked, but no locks are obtained on any pages or records in the file.

t_cc_page: t_cc_record: Pages are SH locked; records are not locked. Files are IS locked.

t_cc_file: The file is SH locked, so no finer granularity locks are obtained.

The starting location (record) of the scan can be controlled using the optional *start_rid* parameter.

append_file_i(lvid, fid)

The arguments to **append_file_i()** are as described above, for the **scan_file_i constructors.**

˜scan_file_i() ˜append_file_i()

These destructors unpin the current record, if any, and destroy the object. To unpin the record before destroying the object, you may explicitly call **finish().**

Scanning

next(pin_ptr, start_offset, eof)

The **next** method is used to retrieve records from the scan and (including the first). A handle to the retrieved record made available through the *pin_ptr* parameter. See **pin_i(ssm)** for information on using this handle. The *start_offset* parameter controls what part of the record to retrieve. This parameter is passed directly to the **pin_i** constructor. The *eof* parameter will be set to *true* only when no value can be retrieved. So, if a file contains two records and *next* has been called twice, *eof* will return *false* on the second call, and *true* on the third.

next_page(pin_ptr, start_offset, eof)

The **next_page** method advances the scan to the next disk page in the file and returns a handle to the first record in the page. Its parameters are identical to those of **next.**

CAVEAT

Do not unpin the record explicitly during a scan.

Other Member Functions

The **lfid** and **lvid** methods return the logical ID of the file being scanned.

The **finish** method unpins the current record and closes the scan.

The **eof** method returns *true* if the end of the file has been reached.

The **error_code** method returns any error code generated by the the scan member methods. See the **ERRORS section for more information.**

Updates While Scanning

A common question is what is the effect of changes to a file (or its records) made by a transaction that is also scanning the file. In general, it is safest not to change anything in the file while scanning. Instead, a list of changes should be made during the scan and only performed after the scan is complete.

However, there are a number of changes that can safely be made to a file while scanning. It is safe to:

Update any record in the file using **update_rec** or **update_rec_hdr.**

Destroy any record in the file using **destroy_rec,** including the current one (although the **pin_i** for the current record will no longer be valid.

It is also safe to change the size of records using **truncate_rec** or **append_rec** and to create new records. However, this may cause records to be moved and therefore revisited or never visited during the scan.

Appending to a file

Create_rec in class **append_file_i** appends a record to a file. It is used much the same way that the static function **ss_m::create_rec** is used, but appending to a file with **append_file_i::create_rec** efficient way to populate a file if many creations are to be performed without other intervening operations on the file, and it guarantees that new records are placed at the end of the file.

ERRORS

A scan_file_i object remembers if an error has occured while constructing the scan or while scanning. An error that occurs in constructing the scan (such as having a bad file ID), can be detected by calling **error_code.** Alternatively, the error can be detected on the first call to **next** which will return the remembered error code. Therefore, if an error occurs while constructing or scanning, repeated calls to next will all return the first error code and no progress will be made on the scan.

EXAMPLES

To Do.

VERSION

This manual page applies to Version 2.0 of the Shore Storage Manager.

SPONSORSHIP

The Shore project is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-91-C-Q518. Further funding for this work was provided by DARPA through Rome Research Laboratory Contract No. F30602-97-2-0247.

COPYRIGHT

SEE ALSO

pin_i(ssm), file(ssm), scan_index_i(ssm), intro(ssm),