NAME

ss_m::concurrency_t, ss_m::store_property_t − Enumerations for Class ss_m

SYNOPSIS

```
// In use, each of the values must be qualified by ss_m:
enum ss_m::concurrency_t {
    t_cc_bad,
    t_cc_none,
    t_cc_record,
    t_cc_page,
    t_cc_file,
    t_cc_vol,
    t_cc_kvl,
    t_cc_im,
    t_cc_modkvl
};

// In use, each of the values must be qualified by ss_m:
enum ss_m::store_property_t {
    t_regular,
    t_temporary,
    t_load_file,
    t_insert_file,
    t_bad_storeproperty
};
```

DESCRIPTION

enum ss_m::concurrency_t

The enumeration, **ss_m::concurrency_t** is used for specifying locking granularity. Lock granularity comes in several flavors, depending on the context in which it is used. In any case, it is based on a lock hierarchy. The lock hierarchy for files is: volume, file, page, record. The hierarchy for indexes is: volume, index, key-value. The values for the enumeration have the following general meanings (more specific meanings are described in the manual pages for the context in which they are used):

ss_m::t_cc_none:
: No locking is to be done. The responsibility for consistency lies with the user of the SSM service being invoked.

ss_m::t_cc_volume:
: Acquire only volume locks.

ss_m::t_cc_file:
: Acquire file or index locks (and volume locks resulting from the hierarchy).

ss_m::t_cc_page:
: Acquire page locks, along with the file, and volume locks determined by the hierarchy. (Page locks are never acquired for indexes.)

ss_m::t_cc_record:
: Acquire record locks, and acquire the page, file, and volume locks determined by the hierarchy.

ss_m::t_cc_kvl:
Acquire key-value (predicate) locks, along with the index and volume locks determined by the hierarchy. This is the default locking protocol for B+ trees, and when it is used, next-key locks are acquired for phantom protection, giving full degree-3 transaction isolation. See *Gray, J., Reuter, A. Transaction Processing: concepts and techniques, 1993.* for to learn about predicate locking and transaction isolation.

ss_m::t_cc_modkvl:
This is a simplistic form of key-value locking, in which there is no phantom protection. When used, range scans on indexes are not permitted.

ss_m::t_cc_im:
This locking protocol can be used with B+ tree indexes that are used as access methods. It works ONLY for indexes whose values are record identifiers. When used, the index manager acquires record locks, along with the page, file, and volume locks determined by the hierarchy, for records identified by the index entries.

enum ss_m::store_property_t

The enumeration **ss_m::store_property_t** identifies special properties that a store (a set of pages; a file or an index) may have. A store can be given one of these properties when the store is created, and subsequently it can be given a different property. This is useful for bulk-loading.

The enumerated values have the following meanings:

ss_m::t_regular:
Updates to the store (file or index) are logged, providing the Durability property in ACID transactions.

ss_m::t_temporary:
File or index is temporary. This means that no logging will be done on the data in the store, and the effect of rollback on updates is undefined. (The store's structural integrity is guaranteed by logging, but the user's data within the store are not. In the case of B+trees and R*trees, the integrity of the tree is NOT guaranteed; only the integrity of the underlying store is guaranteed. In event of abort, the index must be destroyed.) Temporary stores are removed when the volumes containing them are dismounted or mounted; consequently, temporary stores do not survive crashes.

ss_m::t_load_file:
A store that is created with this property starts out as a temporary store, and is converted to a regular store at commit time.

ss_m::t_insert_file:
Updates to existing pages of the store are fully-logged (as if the store had the property *ss_m::t_regular*), but pages allocated (while the store has this property) are not logged. This makes sense in the context in which a store is bulk-loaded in one transaction (with property *ss_m::t_load_file*), and more data are appended in a subsequent transaction. To avoid the cost of logging, the latter transaction would convert the store's property to *ss_m::t_insert_file,* append its data, and convert the store's property back to *ss_m::t_regular.*

CAVEAT

The numeric values of these enumeration constants may change from release to release. Use only the symbolic forms. Do not use these constants as bit-mask values.

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

intro(ssm), lock(ssm), Gray, J., Reuter, A. Transaction Processing: concepts and techniques, 1993.