CUBRID

42 | SEOUL

42 Seoul & CUBRID

Phase 2

Disk Manager

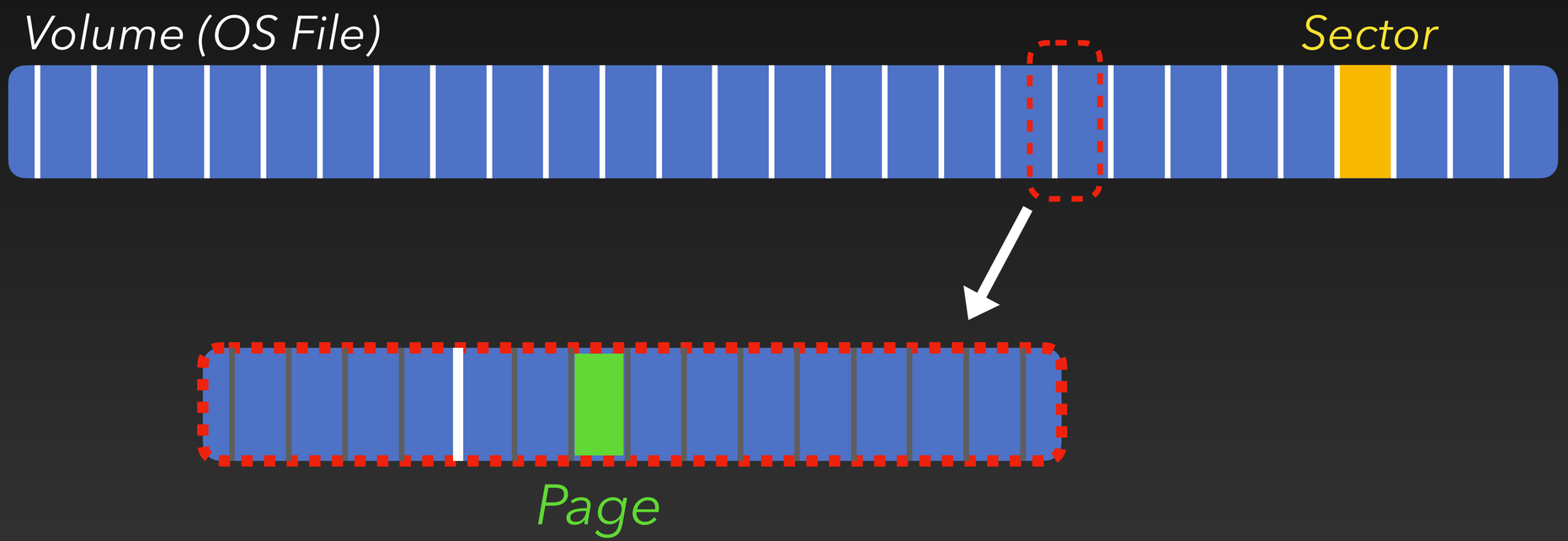
Team A

bypark jolim seubaek jseo jinbekim samin hyeonkki

CUBRID

42 | SEOUL

Volume

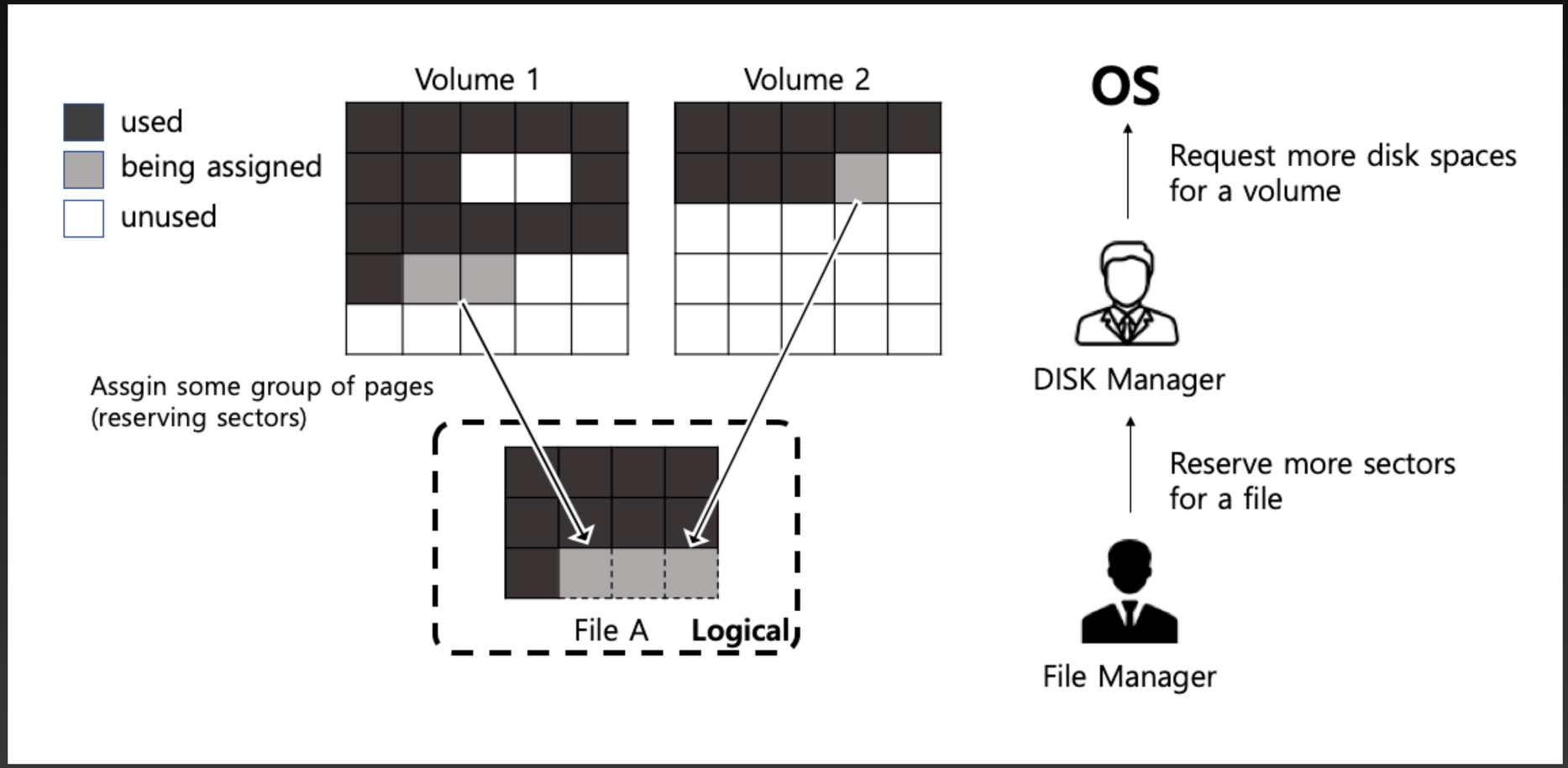


1 Sector = 64 Pages

CUBRID

42 | SEOUL

Disk / File Manager



CUBRID

42 | SEOUL

Disk Manager → 볼륨 공간 전체 관리, 섹터들의 예약여부 트래킹

`disk_*prefix`

File Manager → 파일 내에서의 페이지 할당 여부 트래킹

`file_*prefix`

CUBRID

42 | SEOUL

Volume & File Type

Volume Type

영구 볼륨 : 테이블, 인덱스, 시스템데이터 등이 담기는 볼륨 (영구 존재)

임시 볼륨 : 임시데이터들을 담는 볼륨 (데이터베이스 종료와 재시작시 모두 삭제)

File Type

영구 파일 : 인덱스, 힙데이터, 파일트래커 등의 특정한 목적을 갖음
변경이 일어나면 로그로 기록 (리커버리의 대상)

임시 파일 : 쿼리나 정렬의 중간결과들이 일시적으로 쓰여지는 파일

Volume

Sector Management

Expansion of Volume

Purpose of Volume

어떤 목적으로 *Sector*를 사용할 것이냐에 따라서
데이터의 관리 방법과 라이프사이클이 달라지기 때문

CUBRID

42 SEOUL

Volume - Purpose

Permanent Purpose Data

영구 목적의 데이터는 한 번 쓰여지면
영구적으로 보존

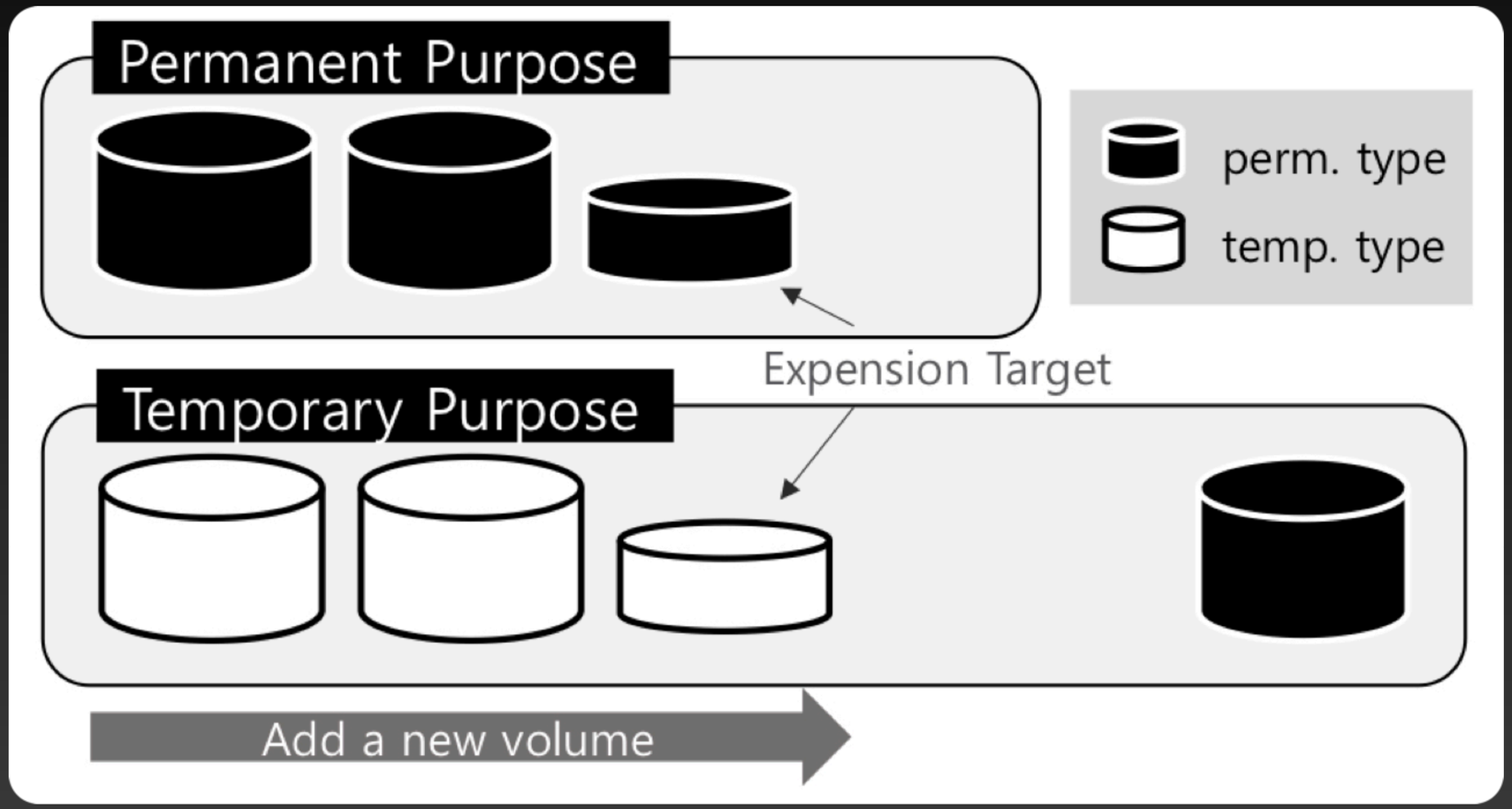
영구 타입 볼륨에만 영구 목적 데이터 저장

사용자가 임의로 생성하거나 공간이 부족
해질 경우 추가적인 영구타입볼륨을 생성

Temporary Purpose Data

임시 목적의 데이터는 트랜잭션이 실행되는
동안 필요한 일시적인 데이터가 저장

만약 사용자가 영구타입볼륨으로 임시목적
데이터를 위한 공간을 미리 확보해두었다면
그 볼륨을 먼저 사용

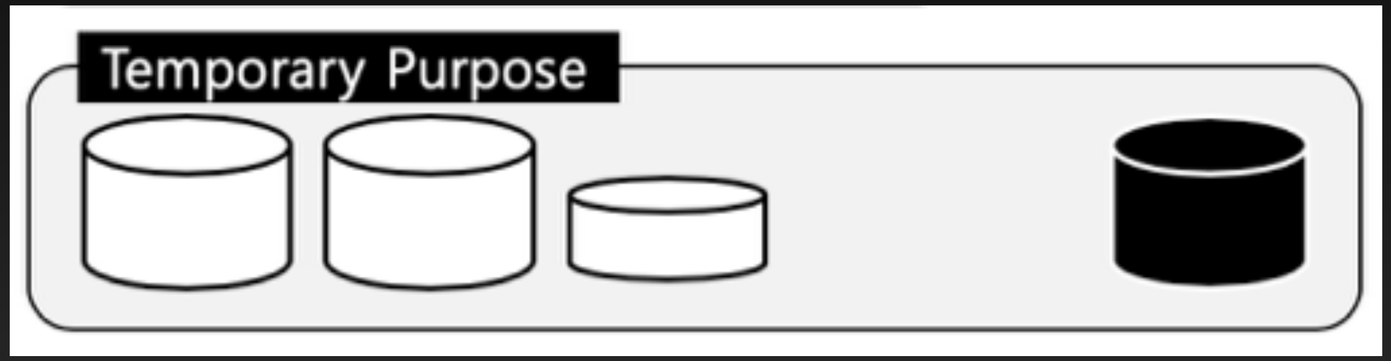


CUBRID

42 SEOUL

Volume - Purpose

Temporary Purpose Data



Logging ❌

Page 반환 ❌

파일 재사용

한 종류 Sector Table 사용

CUBRID

42 | SEOUL

Disk Manager

CUBRID

Disk Manager

42 | SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
int
disk_manager_init (THREAD_ENTRY * thread_p, bool load_from_disk)
{
  int error_code = NO_ERROR;
 * thread_p (in)       : thread entry
 * load_from_disk (in) : true to also populate disk cache wi
 *
 *
 * th volume info
 */
int
disk_manager_init (THREAD_ENTRY * thread_p, bool load_from_disk)
{
  int error_code = NO_ERROR;

  disk_Temp_max_sects = (DKNSECTS) prm_get_integer_value (PRM_ID_BOSR_MAXTMP_PAGES);
  if (disk_Temp_max_sects < 0)
    {
      disk_Temp_max_sects = SECTID_MAX; /* infinite */
    }
  else
    {
      disk_Temp_max_sects = disk_Temp_max_sects / DISK_SECTOR_NPAGES;
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
int
disk_manager_init (THREAD_ENTRY * thread_p, bool load_from_disk)
{
  int error_code = NO_ERROR;

  disk_Temp_max_sects = (DKNSECTS) prm_get_integer_value (PRM_ID_BOSR_MAXTMP_PAGES);
  if (disk_Temp_max_sects < 0)
    {
      disk_Temp_max_sects = SECTID_MAX; /* infinite */
    }
  else
    {
      disk_Temp_max_sects = disk_Temp_max_sects / DISK_SECTOR_NPAGES;
    }

  disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

  if (disk_Cache != NULL)
    {
      disk_log ("disk_manager_init", "%s", "reload disk cache");
      disk_cache_final ();
    }
  error_code = disk_cache_init ();
```

src/base/system_parameter.h

```
enum param_id
{
  PRM_FIRST_ID = 0,
  PRM_ID_ER_LOG_DEBUG = 0,
  PRM_ID_ER_BTREE_DEBUG,
  PRM_ID_ER_LOG_LEVEL,
  PRM_ID_ER_LOG_WARNING,
  PRM_ID_ER_EXIT_ASK,
  PRM_ID_ER_LOG_SIZE,
  PRM_ID_ER_LOG_FILE,
  PRM_ID_ACCESS_IP_CONTROL,
  PRM_ID_ACCESS_IP_CONTROL_FILE,
  PRM_ID_IO_LOCKF_ENABLE,
  PRM_ID_SR_NBUFFERS,
  PRM_ID_SORT_BUFFER_SIZE,
  PRM_ID_PB_BUFFER_FLUSH_RATIO,
  PRM_ID_PB_NBUFFERS,
  PRM_ID_PAGE_BUFFER_SIZE,
  PRM_ID_HF_UNFILL_FACTOR,
  PRM_ID_HF_MAX_BESTSPACE_ENTRIES,
  PRM_ID_BT_UNFILL_FACTOR,
  PRM_ID_BT_OID_NBUFFERS,
  PRM_ID_BT_OID_BUFFER_SIZE,
  PRM_ID_BT_INDEX_SCAN_OID_ORDER,
  PRM_ID_BOSR_MAXTMP_PAGES,
  PRM_ID_LK_TIMEOUT_MESSAGE_DUMP_LEVEL,
  PRM_ID_LK_ESCALATION_AT,
  PRM_ID_LK_ROLLBACK_ON_LOCK_ESCALATION,
  PRM_ID_LK_TIMEOUT_SECS,
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
int
disk_manager_init (THREAD_ENTRY * thread_p, bool load_from_dis
{
  int error_code = NO_ERROR;

  disk_Temp_max_sects = (DKNSECTS) prm_get_integer_value (PRM_
  if (disk_Temp_max_sects < 0)
    {
      static SYSPRM_PARAM prm_Def[] = {
```

```
struct sysprm_param
{
  PARAM_ID id;      /* parameter ID */
  const char *name;   /* the keyword expected */
  unsigned int static_flag; /* bitmask flag representing status words */
  SYSPRM_DATATYPE datatype; /* value data type */
  unsigned int *dynamic_flag; /* shared by both original and duplicated */
  void *default_value;    /* address of (pointer to) default value */
  void *value;      /* address of (pointer to) current value */
  void *upper_limit;    /* highest allowable value */
  void *lower_limit;    /* lowest allowable value */
  char *force_value;    /* address of (pointer to) force value string */
  DUP_PRM_FUNC set_dup;   /* set duplicated value to original value */
  DUP_PRM_FUNC get_dup;   /* get duplicated value from original value */
};
typedef struct sysprm_param SYSPRM_PARAM;
```

```
  }
  error_code = disk_cache_init ();
```

```
void *
prm_get_value (PARAM_ID prm_id)
{
#if defined (SERVER_MODE)
  THREAD_ENTRY *thread_p;

  assert (prm_id <= PRM_LAST_ID);

  if (PRM_SERVER_SESSION (prm_id) && BO_IS_SERVER_RESTARTED ())
    {
      SESSION_PARAM *sprm;
      thread_p = thread_get_thread_entry_info ();
      sprm = session_get_session_parameter (thread_p, prm_id);
      if (sprm)
  {
    return &(sprm->value);
  }
    }

  return prm_Def[prm_id].value;
#else /* SERVER_MODE */
  assert (prm_id <= PRM_LAST_ID);

  return prm_Def[prm_id].value;
#endif /* SERVER_MODE */
}
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
int
disk_manager_init (THREAD_ENTRY * thread_p, bool load_from_disk)
{
  int error_code
```

```
typedef INT32 SECTID;
typedef SECTID DKNSECTS;
```

Disks Number of Sectors : 볼륨들의 섹터 수

```
  disk_Temp_max_sects = (DKNSECTS) prm_get_integer_value (PRM_ID_BOSR_MAXTMP_PAGES);
  if (disk_Temp_max_sects < 0)
    {
      disk_Temp_max_sects = SECTID_MAX; /* infinite */
    }
  else
    {
      disk_Temp_max_sects = disk_Temp_max_sects / DISK_SECTOR_NPAGES;
    }

  disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

  if (disk_Cache != NULL)
    {
      disk_log ("disk_manager_init", "%s", "reload disk cache");
      disk_cache_final ();
    }
  error_code = disk_cache_init ();
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
int
disk_manager_init (THREAD_ENTRY * thread_p, bool load_from_disk)
{
  int error_code = NO_ERROR;

  disk_Temp_max_sects = (DKNSECTS) prm_get_integer_value (PRM_ID_BOSR_MAXTMP_PAGES);
  if (disk_Temp_max_sects < 0)
    {
      disk_Temp_max_sects = SECTID_MAX; /* infinite */
    }
  else
    {
      disk_Temp_max_sects = disk_Temp_max_sects / DISK_SECTOR_NPAGES;
    }

  disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

  if (disk_Cache != NULL)
    {
      disk_log ("disk_manager_init", "%s", "reload disk cache");
      disk_cache_final ();
    }
  error_code = disk_cache_init ();
```

임시 목적 데이터의 최대 섹터 수 ?

CUBRID

Disk Manager

42 | SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
int
disk_manager_init (THREAD_ENTRY * thread_p, bool load_from_disk)
{
  int error_code = NO_ERROR;

  disk_Temp_max_sects = (DKNSECTS) prm_get_integer_value (PRM_ID_BOSR_MAXTMP_PAGES);
  if (disk_Temp_max_sects < 0)
    {
      disk_Temp_max_sects = SECTID_MAX; /* infinite */
    }
  else
    {
      disk_Temp_max_sects = disk_Temp_max_sects / DISK_SECTOR_NPAGES;
    }

  disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

  if (disk_Cache != NULL)
    {
      disk_log ("disk_manager_init", "%s", "reload disk cache");
      disk_cache_final ();
    }
  error_code = disk_cache_init ();
```

= INT_MAX

```
static DKNSECTS disk_Temp_max_sects = -2;
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
int
disk_manager_init (THREAD_ENTRY * thread_p, bool load_from_disk)
{
  int error_code = NO_ERROR;

  disk_Temp_max_sects = (DKNSECTS) prm_get_integer_value (PRM_ID_BOSR_MAXTMP_PAGES);
  if (disk_Temp_max_sects < 0)
    {
      disk_Temp_max_sects = SECTID_MAX; /* infinite */
    }
  else
    {
      disk_Temp_max_sects = disk_Temp_max_sects / DISK_SECTOR_NPAGES;
    }

  disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

  if (disk_Cache != NULL)
    {
      disk_log ("disk_manager_init", "%s", "reload disk cache");
      disk_cache_final ();
    }
  error_code = disk_cache_init ();
```

단위 섹터 페이지 수 *(=64)*

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

if (disk_Cache != NULL)
  {
    disk_log ("disk_manager_init", "%s", "reload disk cache");
    disk_cache_final ();
  }
  {
    disk_Temp_max_sects = SECTID_MAX; /* infinite */
  }
else
  {
    disk_Temp_max_sects = disk_Temp_max_sects / DISK_SECTOR_NPAGES;
  }

disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

if (disk_Cache != NULL)
  {
    disk_log ("disk_manager_init", "%s", "reload disk cache");
    disk_cache_final ();
  }
error_code = disk_cache_init ();
```

system parameter

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

if (disk_Cache != NULL)
  {
    disk_log ("disk_manager_init", "%s", "reload disk cache");
    disk_cache_final ();
  }
```

```
bool
prm_get_bool_value (PARAM_ID prm_id)
{
  assert (prm_id <= PRM_LAST_ID);
  assert (PRM_IS_BOOLEAN (&prm_Def[prm_id]));

  return PRM_GET_BOOL (prm_get_value (prm_id));
}
```

```
#define PRM_GET_BOOL(x)      (*((bool *) (x)))
```

```
(thread_p))
  {
    ASSERT_ERROR_AND_SET (error_code);
    disk_manager_final ();
    return error_code;
  }

#if defined (SERVER_MODE)
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

if (disk_Cache != NULL)
  {
    disk_log ("disk_manager_init", "%s", "reload disk cache");
    disk_cache_final ();
  }
error_code = disk_cache_init ();
```

```
static bool disk_Logging = false;
#define disk_log(func, msg, ...) \
  if (disk_Logging) \
    _er_log_debug (ARG_FILE_LINE, "DISK " func " " LOG_THREAD_TRAN_MSG ": \n\t" msg "\n", \
                   LOG_THREAD_TRAN_ARGS (thread_get_thread_entry_info ()), __VA_ARGS__)
```

```
if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
  {
    ASSERT_ERROR_AND_SET (error_code);
    disk_manager_final ();
    return error_code;
  }

#if defined (SERVER_MODE)
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

if (disk_Cache != NULL)
  {
    disk_log ("disk_manager_init", "%s", "reload disk cache");
    disk_cache_final ();
  }
error_code = disk_cache_init ()
```

```
static bool disk_Logging = false;
#define disk_log(func, msg, ...) \
  if (disk_Logging) \
    _er_log_debug (ARG_FILE_LINE, "DISK " func " " LOG_THREAD_TRAN_MSG ": \n\t" msg "\n", \
                   LOG_THREAD_TRAN_ARGS (thread_get_thread_entry_info ()), __VA_ARGS__)
```

```
typedef struct disk_cache DISK_CACHE;
...
struct disk_cache
{
  ...
  pthread_mutex_t mutex_extend; /* note: never get expand mutex while keeping reserve mutexes */
#if !defined (NDEBUG)
  volatile int owner_extend;
#endif        /* !NDEBUG */
};

static DISK_CACHE *disk_Cache = NULL;
```

```
if (load_from_disk && !disk_cac
  {
    ASSERT_ERROR_AND_SET (error
    disk_manager_final ();
    return error_code;
  }

#if defined (SERVER_MODE)
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

if (disk_Cache != NULL)
  {
    disk_log ("disk_manager_
    disk_cache_final ();
  }
error_code = disk_cache_init
if (error_code != NO_ERROR)
  {
    ASSERT_ERROR ();
    return error_code;
  }
assert (disk_Cache != NULL);

if (load_from_disk && !disk_
  {
    ASSERT_ERROR_AND_SET (er
    disk_manager_final ();
    return error_code;
  }

#if defined (SERVER_MODE)
```

```
static void
disk_cache_final (void)
{
  if (disk_Cache == NULL)
    {
      /* not initialized */
      return;
    }

  assert (disk_Cache->perm_purpose_info.extend_info.owner_reserve == -1);
  assert (disk_Cache->temp_purpose_info.extend_info.owner_reserve == -1);
  assert (disk_Cache->owner_extend == -1);

  pthread_mutex_destroy (&disk_Cache->perm_purpose_info.extend_info.mutex_reserve);
  pthread_mutex_destroy (&disk_Cache->temp_purpose_info.extend_info.mutex_reserve);
  pthread_mutex_destroy (&disk_Cache->mutex_extend);

  free_and_init (disk_Cache);
}
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  disk_Logging = prm_get_bool_value

  if (disk_Cache != NULL)
    {
      disk_log ("disk_manager_init"
      disk_cache_final ();
    }
  error_code = disk_cache_init ();
  if (error_code != NO_ERROR)
    {
      ASSERT_ERROR ();
      return error_code;
    }
  assert (disk_Cache != NULL);

  if (load_from_disk && !disk_cache
    {
      ASSERT_ERROR_AND_SET (error_
      disk_manager_final ();
      return error_code;
    }

#if defined (SERVER_MODE)
```

```
static int
disk_cache_init (void)
{
  int i;

  assert (disk_Cache == NULL);

  disk_Cache = (DISK_CACHE *) malloc (sizeof (DISK_CACHE));
  if (disk_Cache == NULL)
    {
      er_set (ER_ERROR_SEVERITY, ARG_FILE_LINE, ER_OUT_OF_VIRTUAL_MEMORY, 1, sizeof (DISK_CACHE));
      return ER_OUT_OF_VIRTUAL_MEMORY;
    }

  disk_Cache->nvols_perm = 0;
  disk_Cache->nvols_temp = 0;

  disk_Cache->perm_purpose_info.extend_info.nsect_vol_max =
    DISK_SECTS_ROUND_UP ((DKNSECTS) (prm_get_bigint_value (PRM_ID_DB_VOLUME_SIZE) / IO_SECTORSIZE));
  disk_Cache->perm_purpose_info.extend_info.nsect_free = 0;
  disk_Cache->perm_purpose_info.extend_info.nsect_total = 0;
  disk_Cache->perm_purpose_info.extend_info.nsect_max = 0;
  disk_Cache->perm_purpose_info.extend_info.nsect_intention = 0;
  disk_Cache->perm_purpose_info.extend_info.voltype = DB_PERMANENT_VOLTYPE;
  disk_Cache->perm_purpose_info.extend_info.volid_extend = NULL_VOLID;
  pthread_mutex_init (&disk_Cache->perm_purpose_info.extend_info.mutex_reserve, NULL);
#if !defined (NDEBUG)
  disk_Cache->perm_purpose_info.extend_info.owner_reserve = -1;
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  disk_Logging = prm_get_bool_value

  if (disk_Cache != NULL)
    {
      disk_log ("disk_manager_init"
      disk_cache_final ();
    }
  error_code = disk_cache_init ();
  if (error_code != NO_ERROR)
    {
      ASSERT_ERROR ();
      return error_code;
    }
  assert (disk_Cache != NULL);

  if (load_from_disk && !disk_cache
    {
      ASSERT_ERROR_AND_SET (error_
      disk_manager_final ();
      return error_code;
    }

#if defined (SERVER_MODE)
```

```
static int
disk_cache_init (void)
{
  int i;

  assert (disk_Cache == NULL);

  disk_Cache = (DISK_CACHE *) malloc (sizeof (DISK_CACHE));
  if (disk_Cache == NULL)
    {
      er_set (ER_ERROR_SEVERITY, ARG_FILE_LINE, ER_OUT_OF_VIRTUAL_MEMORY, 1, sizeof (DISK_CACHE));
      return ER_OUT_OF_VIRTUAL_MEMORY;
    }

  disk_Cache->nvols_perm = 0;
  disk_Cache->nvols_temp = 0;

  disk_Cache->perm_purpose_info.extend_info.nsect_vol_max =
    DISK_SECTS_ROUND_UP ((DKNSECTS) (prm_get_bigint_value (PRM_ID_DB_VOLUME_SIZE) / IO_SECTORSIZE));
  disk_Cache->perm_purpose_info.extend_info.nsect_free = 0;
  disk_Cache->perm_purpose_info.extend_info.nsect_total = 0;
  disk_Cache->perm_purpose_info.extend_info.nsect_max = 0;
  disk_Cache->perm_purpose_info.extend_info.nsect_intention = 0;
  disk_Cache->perm_purpose_info.extend_info.voltype = DB_PERMANENT_VOLTYPE;
  disk_Cache->perm_purpose_info.extend_info.volid_extend = NULL_VOLID;
  pthread_mutex_init (&disk_Cache->perm_purpose_info.extend_info.mutex_reserve, NULL);
#if !defined (NDEBUG)
  disk_Cache->perm_purpose_info.extend_info.owner_reserve = -1;
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  disk_Logging = prm_get_bool_value (PRM_ID_DISK_LOGGING);

  if (disk_Cache != NULL)
    {
      disk_log ("disk_manager_init", "%s", "reload disk cache");
      disk_cache_final ();
    }
  error_code = disk_cache_init ();
  if (error_code != NO_ERROR)
    {
      ASSERT_ERROR ();
      return error_code;
    }
  assert (disk_Cache != NULL);

  if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
    {
      ASSERT_ERROR_AND_SET (error_code);
      disk_manager_final ();
      return error_code;
    }

#if defined (SERVER_MODE)
```

CUBRID

Disk Manager

42 | SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
    {
      ASSERT_ERROR_AND_SET (error_code);
      disk_manager_final ();
      return error_code;
    }
    }
  error_code = disk_cache_init ();
  if (error_code != NO_ERROR)
    {
      ASSERT_ERROR ();
      return error_code;
    }
  assert (disk_Cache != NULL);

  if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
    {
      ASSERT_ERROR_AND_SET (error_code);
      disk_manager_final ();
      return error_code;
    }

#if defined (SERVER_MODE)
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
    {
      ASSERT_ERROR_AND_SET (error_code);
      disk_manager_final ();
      return error_code;
    }
```

```
#if defined (SERVER_MODE)
```

```
static bool
disk_cache_load_all_volumes (THREAD_ENTRY * thread_p)
{
  /* Cache every single volume */
  assert (disk_Cache != NULL);
  return fileio_map_mounted (thread_p, disk_cache_load_volume, NULL);
}
```

```
static bool
disk_cache_load_volume (THREAD_ENTRY * thread_p, INT16 volid, void *ignore)
{
  DB_VOLPURPOSE vol_purpose;
  DB_VOLTYPE vol_type;
  DISK_VOLUME_SPACE_INFO space_info = DISK_VOLUME_SPACE_INFO_INITIALIZER;

  if (disk_volume_boot (thread_p, volid, &vol_purpose, &vol_type, &space_info) != NO_ERROR)
    {
      ASSERT_ERROR ();
      return false;
    }

  if (vol_type != DB_PERMANENT_VOLTYPE)
    {
      /* don't save temporary volumes... they will be dropped anyway */
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
  {
    ASSERT_ERROR_AND_S
    disk_manager_final
    return error_code;
  }

#if defined (SERVER_MODE
```

```
static bool
disk_cache_load_volume (TH
{
  DB_VOLPURPOSE vol_purpos
  DB_VOLTYPE vol_type;
  DISK_VOLUME_SPACE_INFO s

  if (disk_volume_boot (th
    {
      ASSERT_ERROR ();
      return false;
    }

  if (vol_type != DB_PERMA
    {
      /* don't save tempo
```

```
static int
disk_volume_boot (THREAD_ENTRY * thread_p, VOLID volid, DB_VOLPURPOSE * purpose_out, DB_VOLTYPE * voltype_out,
                  DISK_VOLUME_SPACE_INFO * space_out)
{
  PAGE_PTR page_volheader = NULL;
  DISK_VOLUME_HEADER *volheader;
  int error_code = NO_ERROR;

  assert (volid != NULL_VOLID);

  error_code = disk_get_volheader (thread_p, volid, PGBUF_LATCH_WRITE, &page_volheader, &volheader);
  if (error_code != NO_ERROR)
    {
      ASSERT_ERROR ();
      return error_code;
    }

  *purpose_out = volheader->purpose;
  *voltype_out = volheader->type;

  if (*voltype_out == DB_TEMPORARY_VOLTYPE)
    {
      /* don't load temporary volumes */
      pgbuf_unfix_and_init (thread_p, page_volheader);
      return NO_ERROR;
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  if (load_from_disk &&
      {
        ASSERT_ERROR_AND_S
        disk_manager_final
        return error_code;
      }

#if defined (SERVER_MODE
static bool volume_expan
disk_cache_load_volume (TH
{
  DB_VOLPURPOSE vol_purpos
  DB_VOLTYPE vol_type;
  DISK_VOLUME_SPACE_INFO s

  if (disk_volume_boot (th
      {
        ASSERT_ERROR ();
        return false;
      }

  if (vol_type != DB_PERMA
      {
        /* don't save tempo
```

```
if (volheader->purpose == DB_TEMPORARY_DATA_PURPOSE)
  {
    /* reset volume */
    assert (volheader->nsect_max == volheader->nsect_total);
    /* set back sectors used by system */
    error_code = disk_stab_init (thread_p, volheader);
    if (error_code != NO_ERROR)
{
  ASSERT_ERROR ();
  goto exit;
}
    space_out->n_free_sects = space_out->n_total_sects - SECTOR_FROM_PAGEID (volheader->sys_lastpage) - 1;
  }
else
  {
    space_out->n_free_sects = 0;
    error_code =
disk_stab_iterate_units_all (thread_p, volheader, PGBUF_LATCH_READ, disk_stab_count_free,
                             &space_out->n_free_sects);
    if (error_code != NO_ERROR)
{
  ASSERT_ERROR ();
  goto exit;
}
  }
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
    {
      ASSERT_ERROR_AND_SET (error_code);
      disk_manager_final ();
      return error_code;
    }
```

```
#if defined (SERVER_MODE)
```

```
static bool
disk_cache_load_all_volumes (THREAD_ENTRY * thread_p)
{
  /* Cache every single volume */
  assert (disk_Cache != NULL);
  return fileio_map_mounted (thread_p, disk_cache_load_volume, NULL);
}
```

```
static bool
disk_cache_load_volume (THREAD_ENTRY * thread_p, INT16 volid, void *ignore)
{
  DB_VOLPURPOSE vol_purpose;
  DB_VOLTYPE vol_type;
  DISK_VOLUME_SPACE_INFO space_info = DISK_VOLUME_SPACE_INFO_INITIALIZER;

  if (disk_volume_boot (thread_p, volid, &vol_purpose, &vol_type, &space_info) != NO_ERROR)
    {
      ASSERT_ERROR ();
      return false;
    }

  if (vol_type != DB_PERMANENT_VOLTYPE)
    {
      /* don't save temporary volumes... they will be dropped anyway */
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
static bool
disk_cache_load_volume (THREAD_ENTRY * thread_p, INT16 volid, void *ignore)
{
  DB_VOLPURPOSE vol_purpose;
  DB_VOLTYPE vol_type;
  DISK_VOLUME_SPACE_INFO space_info = DISK_VOLUME_SPACE_INFO_INITIALIZER;

  if (disk_volume_boot (thread_p, volid, &vol_purpose, &vol_type, &space_info) != NO_ERROR)
    {
      ASSERT_ERROR ();
      return false;
    }

  if (vol_type != DB_PERMANENT_VOLTYPE)
    {
      /* don't save temporary volumes... they will be dropped anyway */
      return true;
    }

  /* called during boot, no sync required */
  if (vol_purpose == DB_PERMANENT_DATA_PURPOSE)
    {
      disk_Cache->perm_purpose_info.extend_info.nsect_free += space_info.n_free_sects;
      disk_Cache->perm_purpose_info.extend_info.nsect_total += space_info.n_total_sects;
      disk_Cache->perm_purpose_info.extend_info.nsect_max += space_info.n_max_sects;
```

```
ad_p)

e_load_volume, NULL);
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
    /* don't save temporary volumes... they will be dropped anyway */
    return true;
  }

/* called during boot, no sync required */
if (vol_purpose == DB_PERMANENT_DATA_PURPOSE)
  {
    disk_Cache->perm_purpose_info.extend_info.nsect_free += space_info.n_free_sects;
    disk_Cache->perm_purpose_info.extend_info.nsect_total += space_info.n_total_sects;
    disk_Cache->perm_purpose_info.extend_info.nsect_max += space_info.n_max_sects;

    assert (disk_Cache->perm_purpose_info.extend_info.nsect_free
      <= disk_Cache->perm_purpose_info.extend_info.nsect_total);
    assert (disk_Cache->perm_purpose_info.extend_info.nsect_total
      <= disk_Cache->perm_purpose_info.extend_info.nsect_total);

    if (space_info.n_total_sects < space_info.n_max_sects)
{
  assert (disk_Cache->perm_purpose_info.extend_info.volid_extend == NULL_VOLID);
  disk_Cache->perm_purpose_info.extend_info.volid_extend = volid;
}
  }
else
  {
    assert (space_info.n_total_sects == space_info.n_max_sects);
```

```
ad_p)

e_load_volume, NULL);
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
    /* don't save temporary volumes... they will be dropped anyway */
    return true;
  }

/* called during boot, no sync required */
if (vol_purpose == DB_PERMANENT_DATA_PURPOSE)
  {
    disk_Cache->perm_purpose_info.extend_info.nsect_free += space_info.n_free_sects;
    disk_Cache->perm_purpose_info.extend_info.nsect_total += space_info.n_total_sects;
    disk_Cache->perm_purpose_info.extend_info.nsect_max += space_info.n_max_sects;

    assert (disk_Cache->perm_purpose_info.extend_info.nsect_free
      <= disk_Cache->perm_purpose_info.extend_info.nsect_total);
    assert (disk_Cache->perm_purpose_info.extend_info.nsect_total
      <= disk_Cache->perm_purpose_info.extend_info.nsect_total);

    if (space_info.n_total_sects < space_info.n_max_sects)
{
  assert (disk_Cache->perm_purpose_info.extend_info.volid_extend == NULL_VOLID);
  disk_Cache->perm_purpose_info.extend_info.volid_extend = volid;
}
  }
else
  {
    assert (space_info.n_total_sects == space_info.n_max_sects);
```

```
ad_p)

e_load_volume, NULL);
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
    if (space_info.n_total_sects < space_info.n_max_sects)
  {
    assert (disk_Cache->perm_purpose_info.extend_info.volid_extend == NULL_VOLID);
    disk_Cache->perm_purpose_info.extend_info.volid_extend = volid;
  }
    }
  else
    {
      assert (space_info.n_total_sects == space_info.n_max_sects);

      disk_Cache->temp_purpose_info.nsect_perm_free += space_info.n_free_sects;
      disk_Cache->temp_purpose_info.nsect_perm_total += space_info.n_total_sects;

      assert (disk_Cache->temp_purpose_info.nsect_perm_free <= disk_Cache->temp_purpose_info.nsect_perm_total);
    }

  disk_Cache->vols[volid].nsect_free = space_info.n_free_sects;
  disk_Cache->vols[volid].purpose = vol_purpose;

  disk_Cache->nvols_perm++;
  return true;
}
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
    {
      ASSERT_ERROR_AND_SET (error_code);
      disk_manager_final ();
      return error_code;
    }

#if defined (SERVER_MODE)
```

```
static bool
disk_cache_load_all_volumes (THREAD_ENTRY * thread_p)
{
  /* Cache every single volume */
  assert (disk_Cache != NULL);
  return fileio_map_mounted (thread_p, disk_cache_load_volume, NULL);
}
```

```
bool
fileio_map_mounted (THREAD_ENTRY * thread_p, bool (*fun) (THREAD_ENTRY * thread_p, VOLID vol_id, void *args),
                    void *args)
{
  FILEIO_VOLUME_INFO *vol_info_p;
  FILEIO_VOLUME_HEADER *header_p;
  int i, j, max_j, min_j, num_temp_vols;

  FILEIO_CHECK_AND_INITIALIZE_VOLUME_HEADER_CACHE (false);

  header_p = &fileio_Vol_info_header;
  for (i = 0; i <= (header_p->next_perm_volid - 1) / FILEIO_VOLINFO_INCREMENT; i++)
    {
      max_j = fileio_max_permanent_volumes (i, header_p->next_perm_volid);
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
    {
      ASSERT_ERROR_AND_SET (error_code);
      disk_manager_final ();
      return error_code;
    }

#if defined (SERVER_MODE)
```

```
static bool
disk_cache_load_all_volumes (THREAD_ENTRY * thread_p)
{
  /* Cache every single volume */
  assert (disk_Cache != NULL);
  return fileio_map_mounted (thread_p, disk_cache_load_volume, NULL);
}
```

```
bool
fileio_map_mounted (THREAD_ENTRY * thread_p, bool (*fun) (THREAD_ENTRY * thread_p, VOLID vol_id, void *args),
                    void *args)
{
  FILEIO_VOLUME_INFO *vol_info_p;
  FILEIO_VOLUME_HEADER *header_p;
  int i, j, max_j, min_j, num_temp_vols;

  FILEIO_CHECK_AND_INITIALIZE_VOLUME_HEADER_CACHE (false);

  header_p = &fileio_Vol_info_header;
  for (i = 0; i <= (header_p->next_perm_volid - 1) / FILEIO_VOLINFO_INCREMENT; i++)
    {
      max_j = fileio_max_permanent_volumes (i, header_p->next_perm_volid);
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
for (i = 0; i <= (header_p->next_perm_volid - 1) / FILEIO_VOLINFO_INCREMENT; i++)
  {
    max_j = fileio_max_permanent_volumes (i, header_p->next_perm_volid);

    for (j = 0; j <= max_j; j++)
{
  vol_info_p = &header_p->volinfo[i][j];
  if (vol_info_p->vdes != NULL_VOLDES)
    {
      if (((*fun) (thread_p, vol_info_p->volid, args)) == false)
  {
    return false;
  }
    }
}
  }

num_temp_vols = LOG_MAX_DBVOLID - header_p->next_temp_volid;
for (i = header_p->num_volinfo_array - 1;
     i > (header_p->num_volinfo_array - 1
    - (num_temp_vols + FILEIO_VOLINFO_INCREMENT - 1) / FILEIO_VOLINFO_INCREMENT); i--)
  {
    min_j = fileio_min_temporary_volumes (i, num_temp_vols, header_p->num_volinfo_array);
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
  {
    ASSERT_ERROR_AND_SET (error_code);
    disk_manager_final ();
    return error_code;
  }

#if defined (SERVER_MODE)
  disk_auto_volume_expansion_daemon_init ();
#endif /* SERVER_MODE */
  return NO_ERROR;
}
```

```
static bool
disk_cache_load_all_volumes (THREAD_ENTRY * thread_p)
{
  /* Cache every single volume */
  assert (disk_Cache != NULL);
  return fileio_map_mounted (thread_p, disk_cache_load_volume, NULL);
}
```

`fileio_map_mounted()`

Mount

`disk_cache_load_volume()`

Cache에 정보 저장

`disk_volume_boot()`

volume_header 정의

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
  {
    ASSERT_ERROR_AND_SET (error_code);
    disk_manager_final ();
    return error_code;
  }
```

```
void
disk_manager_final (void)
{
#if defined (SERVER_MODE)
  disk_auto_volume_expansion_daemo
#endif /* SERVER_MODE */

  disk_cache_final ();
}
```

```
static void
disk_cache_final (void)
{
  if (disk_Cache == NULL)
    {
      /* not initialized */
      return;
    }

  assert (disk_Cache->perm_purpose_info.extend_info.owner_reserve == -1);
  assert (disk_Cache->temp_purpose_info.extend_info.owner_reserve == -1);
  assert (disk_Cache->owner_extend == -1);

  pthread_mutex_destroy (&disk_Cache->perm_purpose_info.extend_info.mutex_reserve);
  pthread_mutex_destroy (&disk_Cache->temp_purpose_info.extend_info.mutex_reserve);
  pthread_mutex_destroy (&disk_Cache->mutex_extend);

  free_and_init (disk_Cache);
}
```

CUBRID

Disk Manager

42 | SEOUL

src/storage/disk_manager.c

disk_manager_init()

```
  if (load_from_disk && !disk_cache_load_all_volumes (thread_p))
    {
      ASSERT_ERROR_AND_SET (error_code);
      disk_manager_final ();
      return error_code;
    }

#if defined (SERVER_MODE)
  disk_auto_volume_expansion_daemon_init ();
#endif /* SERVER_MODE */

  return NO_ERROR;
}
```

CUBRID

Disk Manager

42 SEOUL

src/storage/disk_manager.c

disk_manager_init()

Disk Cache → Sector Status

Concurrency

```
/*
 * disk_manager_init () - load disk manager and allocate all required resources
 *
 * return                : error code
 * thread_p (in)         : thread entry
 * load_from_disk (in) : true to also populate disk cache wi
 *
 *
 * th
 */
int
disk_manager_init (THREAD_ENTRY * thread_p, bool load_from_disk)
{
  int error_code = NO_ERROR;

  disk_Temp_max_sects = (DKNSECTS) prm_get_integer_value (PRM_ID_BOSR_MAXTMP_PAGES);
  if (disk_Temp_max_sects < 0)
    {
      disk_Temp_max_sects = SECTID_MAX; /* infinite */
    }
  else
    {
      disk_Temp_max_sects = disk_Temp_max_sects / DISK_SECTOR_NPAGES;
```

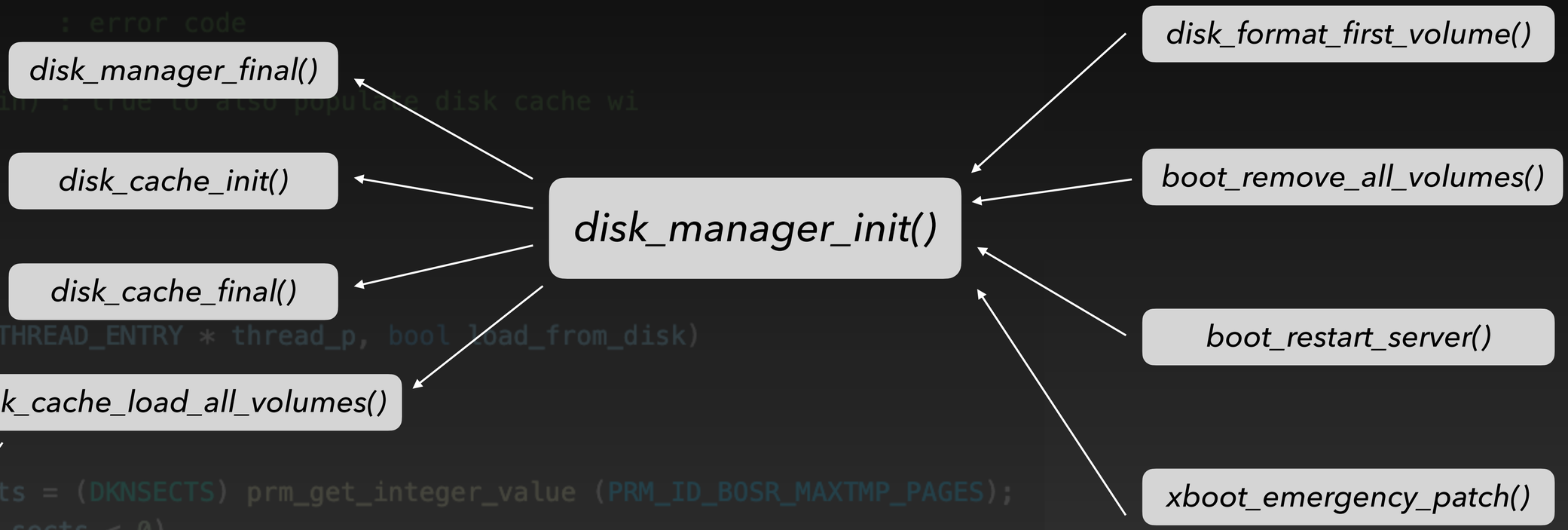
CUBRID

Disk Manager

42 | SEOUL

src/storage/disk_manager.c

disk_manager_init()



CUBRID

42 | SEOUL

Disk Manager

Team A

bypark jolim seubaek jseo jinbekim samin hyeonkki

by bypark