course 2 Storage

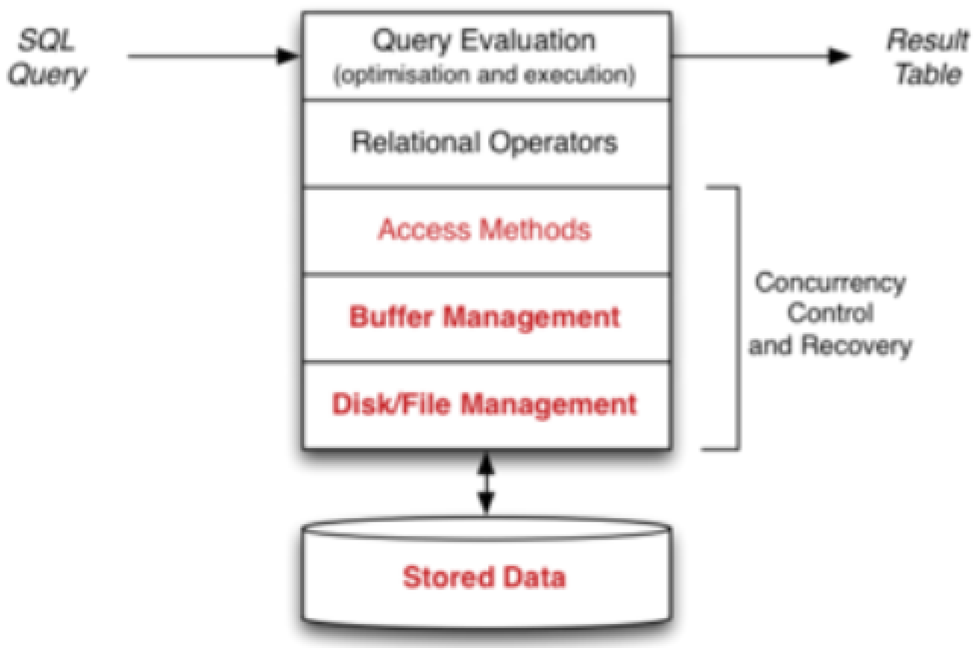
this section is about storage in postgreSQL like: devices, files, pages, tuples, buffers, catalogs

1. Storage Management

aims of storage management in DBMS are shown below:

- map from database objects (e.g. tables) to disk files
- use buffer to minimise disk/memory data transfers
    - and also manage transfer of data to/from disk storage
- provide view of data as collection of pages/tuples

levels of DBMS related to storage management:

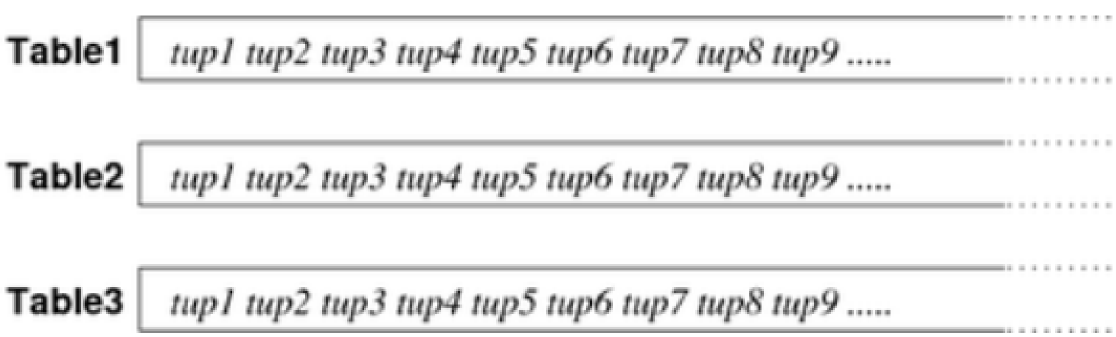


there are severals topics to be considered in storage management:

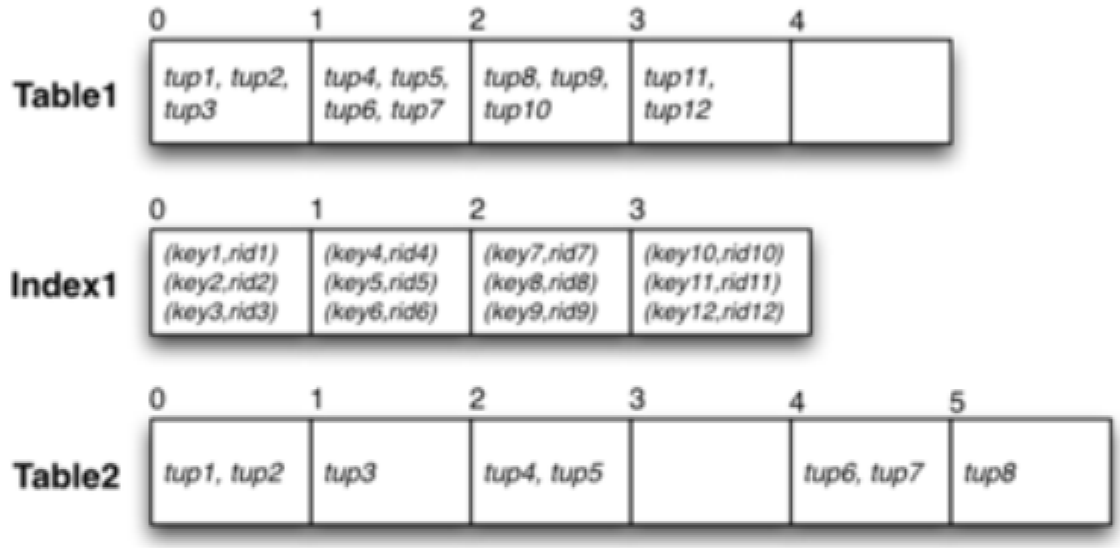
- Disk and files
    - performance issue and organisation of disk files
- Buffer management
    - cache & page replacement strategies
- Tuple/Page management
    - how tuples are represented within disk pages
- DB object management (Catalog)
    - how tables/views/functions/types, etc are represented

1.1 views of data

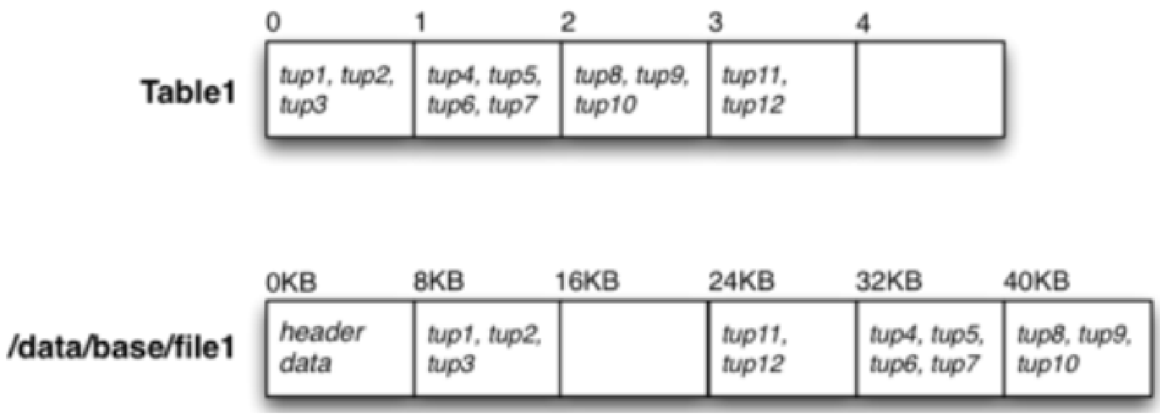
- Users and top-level query see data as:
    - a collection a tables, each table contains a set of tuples



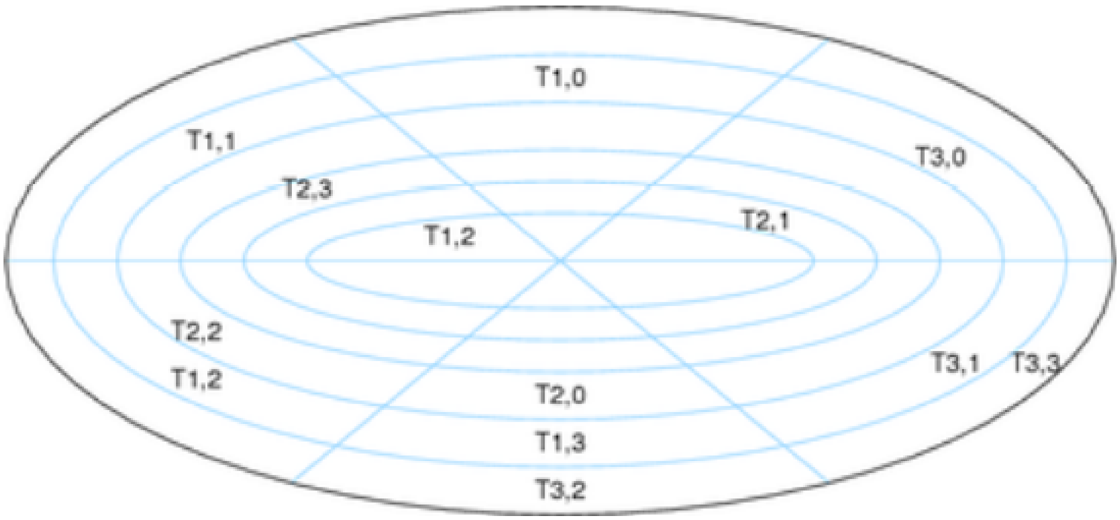
- Relational operators and access methods see data as:
    - sequence of **fixed-size pages**, typically 1KB to 8KB, **each page contains tuple or index data**



- File manager see data as:
  - maps table name + page index to file + offset



- Disk manager see data as:
  - fixed-size sectors of bytes, typically 512B
  - sectors are scattered across a disk device



1.2 storage manager interface

look how storage manager conduct a query (how it work):

Example: simple scan of a relation:

```
select name from Employee
```

is implemented as something like

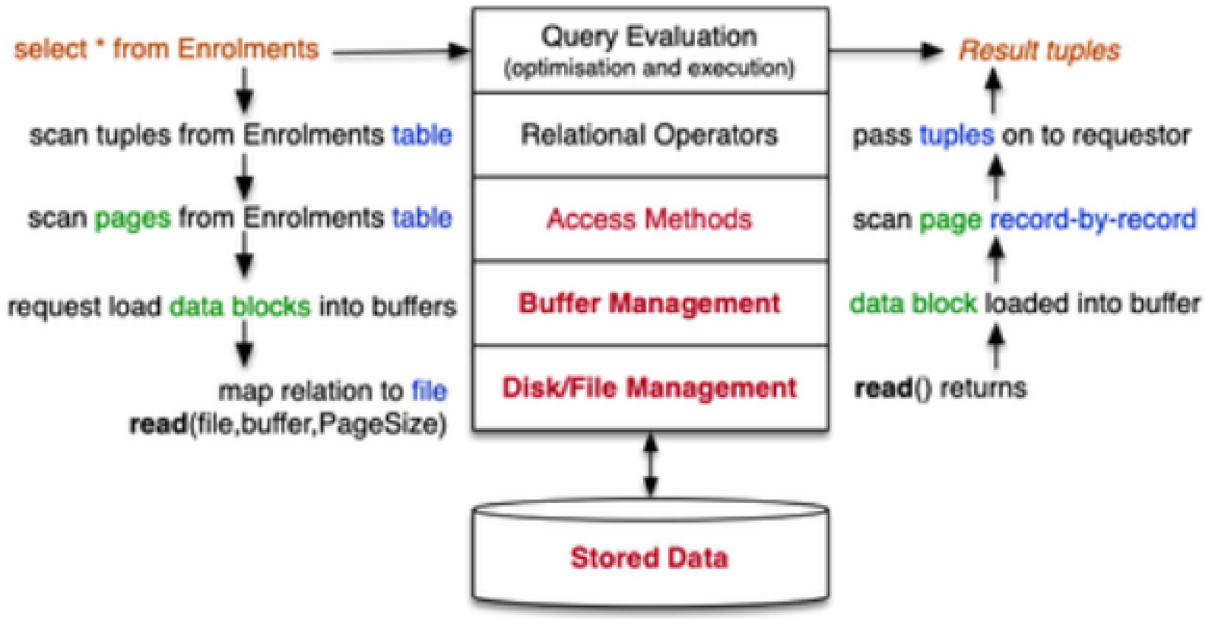
```
DB db = openDatabase("myDB");
Rel r = openRel(db,"Employee");
Scan s = startScan(r);
Tuple t;
while ((t = nextTuple(s)) != NULL)
{
   char *name = getField(t,"name");
   printf("%s\n", name);
}
```

the above shows several kinds of operations/mappings

- using database-name to access meta-data
- mapping a relation-name to a file

- performing page-by-page scans of files
- extracting tuples from pages
- extracting fields from tuples

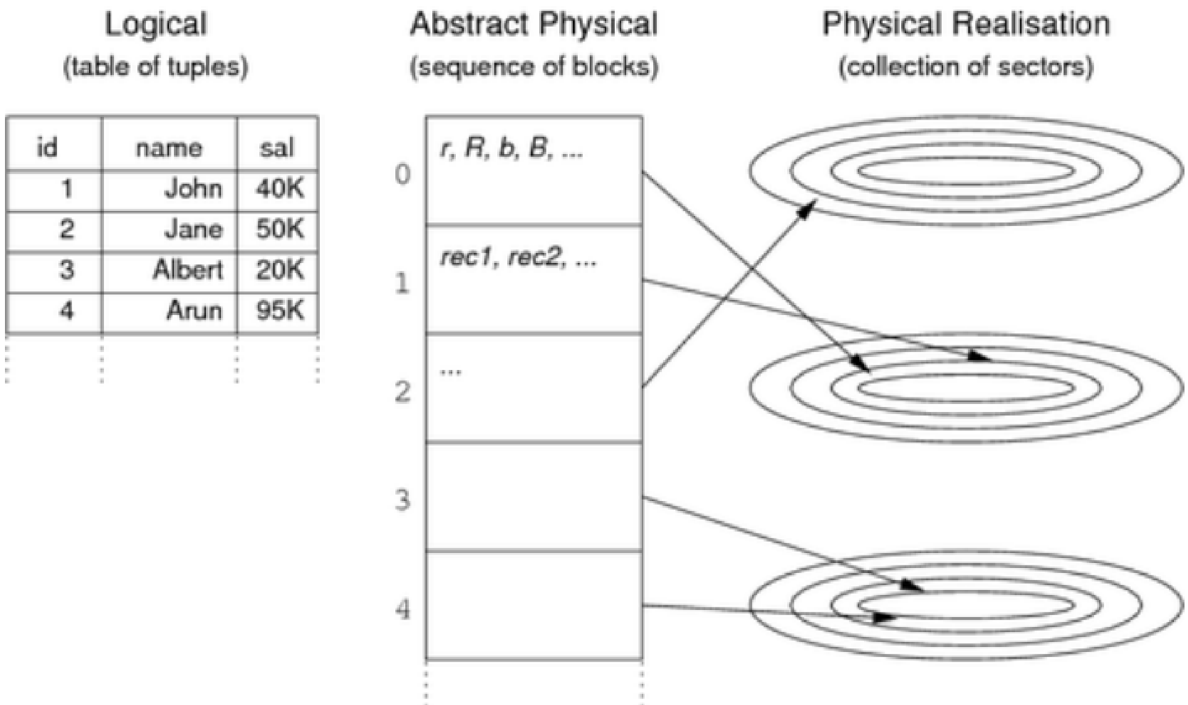
1.3 data flow in query evaluation



files in DBMS

Data sets (file) can be viewed at several levels of abstraction in DBMS:

- logical view:
- abstract physical: a file a sequence of fixed-size data blocks
- physical realisation:



1.4 storage technology

there are two methods of storage technology:

- computational storage: based on RAM
- bulk data storage: based on Disk

Computational storage: mainly use main memory(RAM)

bulk data storage: there are several kinds of bulk storage technology currently exitst:

- magnetic disks, optical disks, flash memory(SSD)

comaring HDD and SDD:

	HDD	SDD
Cost/byte	~ 4c / GB	~ 13c / GB
Read latency	~ 10ms	~ 50μs
Write latency	~ 10ms	~ 900μs
Read unit	block (e.g. 1KB)	byte
Writing	write a block	write on empty block

2. Disk Management

Aim:

- handles mapping from **database ID to disk address**(filesystem)
- tranfer blocks of data **between buffer pool and disk**
- also attempts to handle disk access error problem

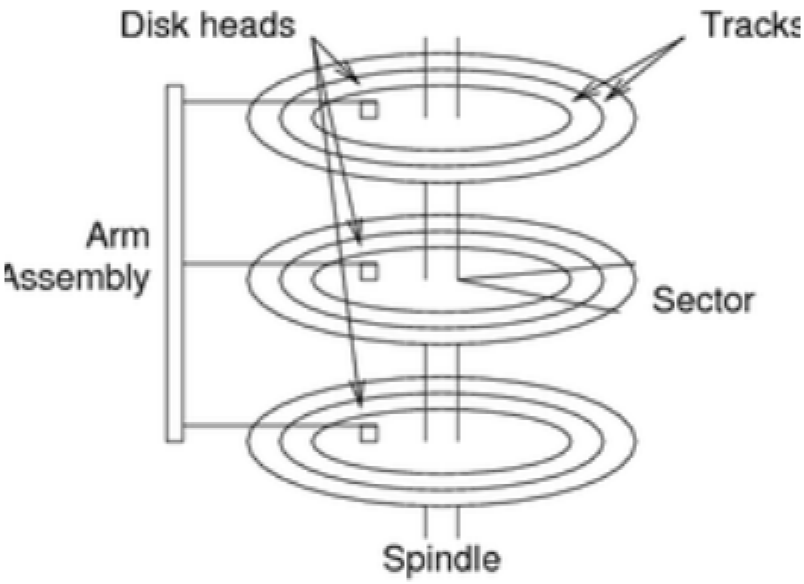
2.1 Disk Manager

there are severl interface about basic disk management:

- get_page: read disk block corresponding to PageId into buffer Page
- put_page: write block Page to disk block identified by PageId
- allocate_page: allocate a group of n disk blocks, optimised for sequential access
- deallocate_page: deallocate ….(just a reversal option of allocate)

2.2 Disk Technology

illustration of disk architecture:



characteristics of disk:

- platters → tracks → sectors(blocks)
- transfer unit: 1block(e.g. 512B, 1KB, 2KB)

2.3 Disk access costs

- access time = seek time + rotational delay + tranfer time
- cost to write a block is similar to access time
- verify data on disk cost: add full rotation delay + block tranfer time

examples-1:

- 3.5 inches (8cm) diameter, 3600RPM, 1 surface (platter)
- 16MB usable capacity ($16 \times 2^{20} = 2^{24}$)
- 128 tracks, 1KB blocks (sectors), 10% gap between blocks
- #bytes/track = $2^{24}/128 = 2^{24}/2^{7} = 128KB$
- #blocks/track = $(0.9*128KB)/1KB = 115$
- seek time: min: 5ms (adjacent cyls), avg: 25ms max: 50ms

Time T_r to read one random block on disk #1:

- 3600 RPM = 60 revs per sec, rev time = 16.7 ms
- Time over blocks = 16.7 $\times$ 0.9 = 15 ms
- Time over gaps = 16.7 $\times$ 0.1 = 1.7 ms
- Transfer time for 1 block = 15/115 = 0.13 ms
- Time for skipping over gap = 1.7/115 = 0.01 ms

T_r = seek + rotation + transfer

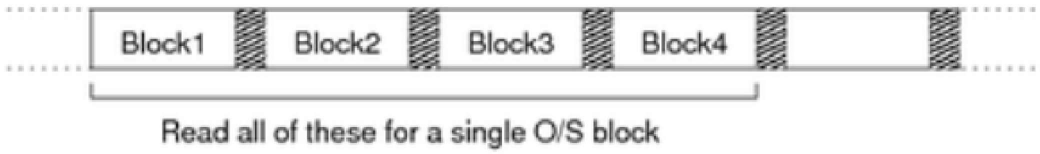
Minimum T_r = 0 + 0 + 0.13 = 0.13 ms

Maximum T_r = 50 + 16.7 + 0.13 = 66.8 ms

Average $T_r = 25 + (16.7/2) + 0.13 = 33.5\ ms$

examples-2:

if OS deals in 4KB blocks



$$T_r(4\text{–blocks}) = 25 + (16.7/2) + 4 \times 0.13 + 3 \times 0.01 = 33.9 \text{ ms}$$

$$T_r(1\text{–block}) = 25 + (16.7/2) + 0.13 = 33.5 \text{ ms}$$

compared with 1KB blocks, 4KB blocks accessing takes a bit more time

Note: sequential access reduces average block read cost significantly, but

- is limited to 115 blocks sequences
- is only used if blocks need to be sequentially scanned

- 3.5 inches (8cm) diameter, 3600RPM, 8 surfaces (platters)
- 8GB usable capacity ($8 \times 2^{30} = 2^{33}$ bytes)
- 8K (2^{13}) cylinders = 8k tracks per surface
- 256 sectors/track, 512 (2^9) bytes/sector

disk blocks addressing: 3bits(surface) + 13bits(cylinder) + 8bits(sector)

so if you use 32-bit OS, there will be 8bit left

2.4 Disk characteristics

disk has three most importantly characteristics:

- capacity
- access time
- reliability

so how to improve access time?

- minimise block tranfers: clustering, buffering, scheduled access

    - clustering: if there are two blocks are frequently access together, then we put them in the same block
- reduce seek: scheduling algorithm
- reduce latency:
- layout of data on disk (file organisation) can also assist

2.5 improve disk access

scheduled disk access

~~you can learn it from Operate System~~

2.5.1 disk layout

if data set is going to be acccessd in a pre-determined manner, arrange data on disk to minimise access time

E.g. if we want to conduct sequential scan, there are severl ways to reduce access time(by change disk layout)

- place subsequent blocks in same cylinder, different platters
- stagger so that as soon as block i was readed, block i+1 can be read
- once cylinder exhasuted, move to adjacent cylinder

2.5.2 improving writes

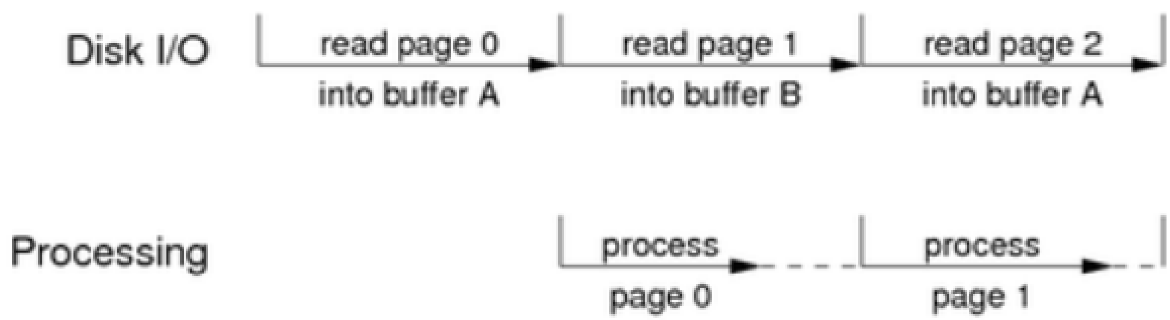
there are two mainly approaches to improving write option in disk:

- Nonvolatile write buffer
    - write all blocks to mem buffers in nonvolatile RAM
    - transfer to disk when idle
- Log disk
    - write all blocks to a special sequential access file-system
    - transfer to disk when idle

2.5.3 double buffering

double buffering exploits potential concurrency between disk and memory, whle reads/writes to disk are underway, other processing can be done.

Note: it just like the **I/O buffer management in OS-course**



let's compare single buffer and double buffer with a examples:

```
select sum(salary) from Employee
```

Note: we know that realtion = file = a sequence of n blocks A, B, C, D

- with a single buffer

```
read A into buffer then process buffer content
read B into buffer then process buffer content
read C into buffer then process buffer content
...
```

Costs:

- cost of reading a block = T_r
- cost of processing a block = T_p
- total elapsed time = $b.(T_r+T_p) = bT_r + bT_p$

Typically, $T_p < T_r$ (depends on kind of processing)

- with a double buffer

```
read A into buffer1
process A in buffer1
  and concurrently read B into buffer2
process B in buffer2
  and concurrently read C into buffer1
...
```

Costs:

- overall cost depends on relative sizes of T_r and T_p
- if $T_p \cong T_r$, total elapsed time = $T_r + bT_p$ (cf. $bT_r + bT_p$)

2.6 Multiple Disk Systems

essentially, multiple disks allow；

- improved reliablity by redundant storage of data
- reduced access cost by exploting parallelism

Note: we know that capacity increase naturally by adding multiple disks

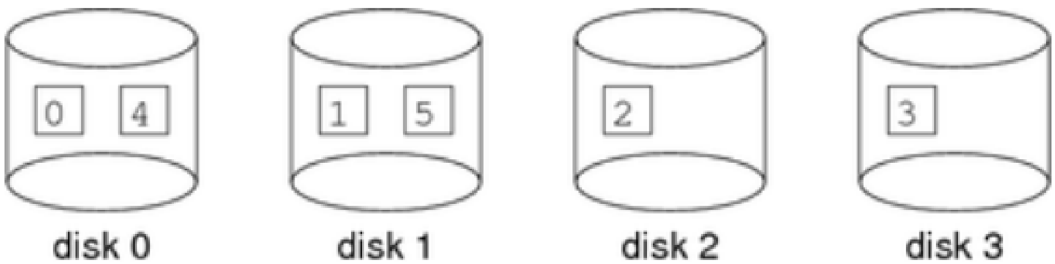
RAID: redundant arrays on independent disks, whose main purpose is to improve reading speed

2.6.1 RAID Level 0

uses striping to partition data for one file over several disks

E.g. for *n* disks, block *i* in the file is written to disk *(i mod n)*

Example: file with 6 data blocks striped onto 4 disks using *(pid mod 4)*



Increases capacity, improves data transfer rates, reduces reliability.

the operation will change accordingly：

```
writePage(PageId)
```

to

```
disk = diskOf(PageId,ndisks)
cyln = cylinderOf(PageId)
plat = platterOf(PageId)
sect = sectorOf(PageId)
writeDiskPage(disk, cyln, plat, sect)
```

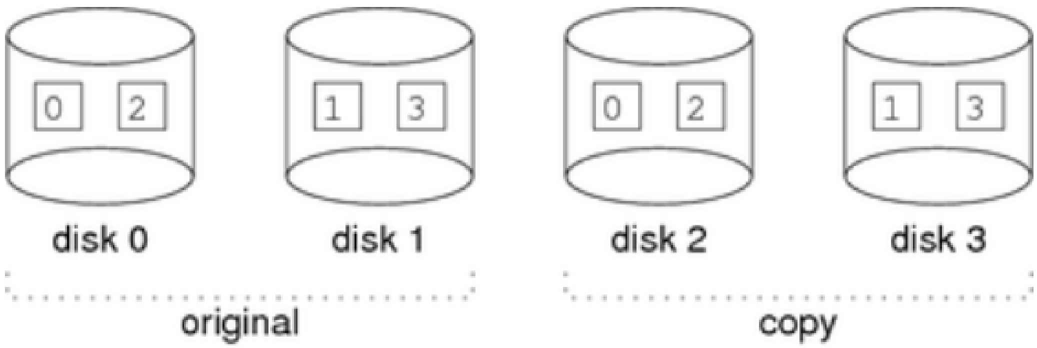
2.6.2 RALD Level 1

uses mirroring to store multiple copies of each blocks

Since disks can be read/written in parallel, transfer cost unchanged.

Multiple copies allows for single-disk failure with no data loss.

Example: file with 4 data blocks mirrored on two 2-disk partitions



Reduces capacity, improves reliability, no effect on data transfer rates.

the operation will change accordingly：

```
writePage(PageId)
```

to

```
n = ndisksInPartition
disk = diskOf(PageId,n)
cyln = cylinderOf(PageId)
plat = platterOf(PageId)
sect = sectorOf(PageId)
writeDiskPage(disk, cyln, plat, sect)
writeDiskPage(disk+n, cyln, plat, sect)
```

2.6.3 RAID levels 2-6

the higher levels of raid incorporate various combinations of:

- block/bits-level striping, mirroring, and error correcting codes（奇偶校验位）

the differences are primarily in:

- the kind of error correcting codes that are used

- where the ECC parity bits(奇偶校验位) are stored

RAID levels 2-5 can recover from failure in a single disk

RAID levels 6 can recover from smultaneous failures in two disks

3. Database Objects

the most important concept that we can learn from recent course is that:

- **how DB object are mapped to file system by Disk Manager?**

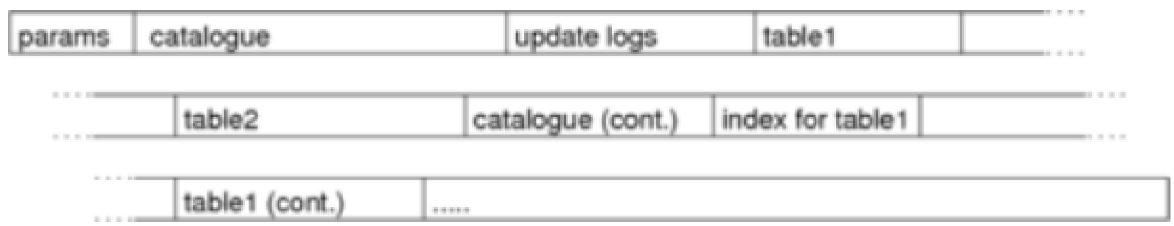
there are several DB objects:

- database
- parameters: global configuration information
- catalogue: meta-information describing database contents
- tables
- tuples
- indexes: access methods for efficient searching
- update logs: for handling rollback/recovery
- procedures: active elements

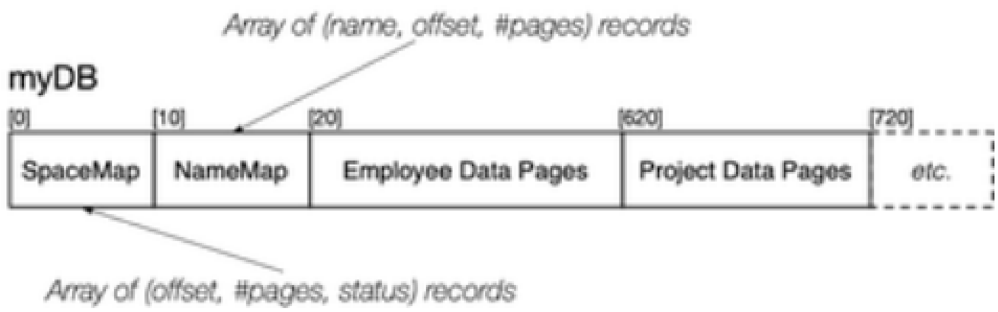
4. Storage Manager

4.1 Single-File storage manager

in single-file storage manager, objects are allocated to regions(segments) of the file



examples:



E.g.

SpaceMap = [(0,10,U), (10,10,U), (20,600,U), (620,100,U), (720,20,F)]

TableMap = [("employee",20,500), ("project",620,40)]

storage manager data structures for Database and tables:

```
typedef struct DBrec {
   char *dbname;     // copy of database name
   int fd;           // the database file
   SpaceMap map;     // map of free/used areas
   NameTable names; // map names to areas + sizes
} *DB;

typedef struct Relrec {
   char *relname;    // copy of table name
   int   start;      // page index of start of table data
   int   npages;     // number of pages of table data
   ...
} *Rel;
```

examples:

```
select name from Employee
```

```
select name from Employee
```

might be implemented as something like

```
DB db = openDatabase("myDB");
Rel r = openRelation(db,"Employee");
Page buffer = malloc(PAGESIZE*sizeof(char));
for (int i = 0; i < r->npages; i++) {
   PageId pid = r->start+i;
   get_page(db, pid, buffer);
   for each tuple in buffer {
      get tuple data and extract name
      add (name) to result tuples
   }
}
```

```
// start using DB, buffer meta-data
DB openDatabase(char *name) {
   DB db = new(struct DBrec);
   db->dbname = strdup(name);
   db->fd = open(name,O_RDWR);
   db->map = readSpaceTable(db->fd);
   db->names = readNameTable(db->fd);
   return db;
}
// stop using DB and update all meta-data
void closeDatabase(DB db) {
   writeSpaceTable(db->fd,db->map);
   writeNameTable(db->fd,db->map);
   fsync(db->fd);
   close(db->fd);
   free(db->dbname);
   free(db);
}
```

```
// set up struct describing relation
Rel openRelation(DB db, char *rname) {
   Rel r = new(struct Relrec);
   r->relname = strdup(rname);
   // get relation data from map tables
   r->start = ...;
   r->npages = ...;
   return r;
}

// stop using a relation
void closeRelation(Rel r) {
   free(r->relname);
   free(r);
}
```

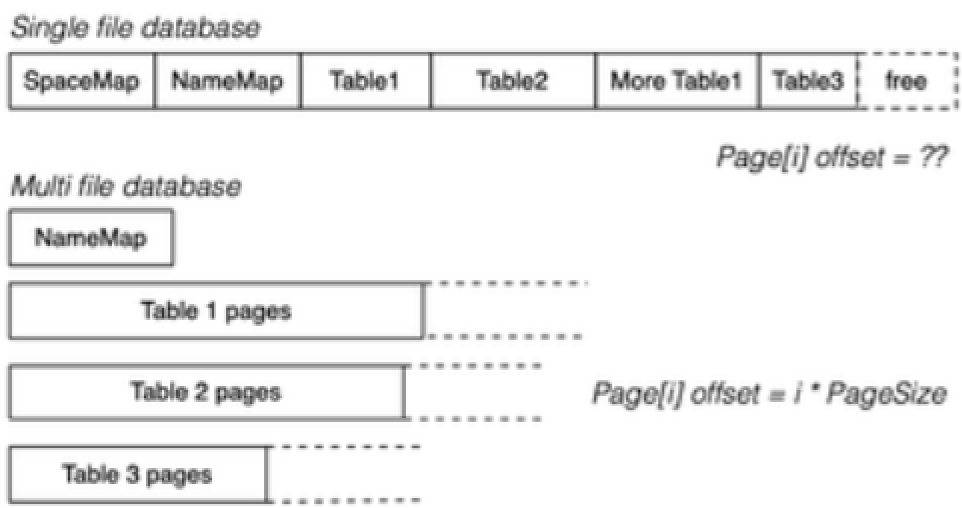
```
// assume that Page = byte[PageSize]
// assume that PageId = block number in file

// read page from file into memory buffer
void get_page(DB db, PageId p, Page buf) {
   lseek(db->fd, p*PAGESIZE, SEEK_SET);
   read(db->fd, buf, PAGESIZE);
}

// write page from memory buffer to file
void put_page(Db db, PageId p, Page buf) {
   lseek(db->fd, p*PAGESIZE, SEEK_SET);
   write(db->fd, buf, PAGESIZE);
}
```

```
// assume an array of (offset,length,status) records

// allocate n new pages
PageId allocate_pages(int n) {
   if (no existing free chunks are large enough) {
      int endfile = lseek(db->fd, 0, SEEK_END);
      addNewEntry(db->map, endfile, n);
   } else {
```

```
      grab "worst fit" chunk
      split off unused section as new chunk
   }
   // note that file itself is not changed
}
```

```
// drop n pages starting from p
void deallocate_pages(PageId p, int n) {
   if (no adjacent free chunks) {
      markUnused(db->map, p, n);
   } else {
      merge adjacent free chunks
      compress mapping table
   }
   // note that file itself is not changed

}
```

4.2 Multiple-File Disk Manager



if system use several files per table, PageId contains:

- relation identifier
- file identifier
- page number

4.3 Oracle File Structures

oracle uses five different kinds of files:

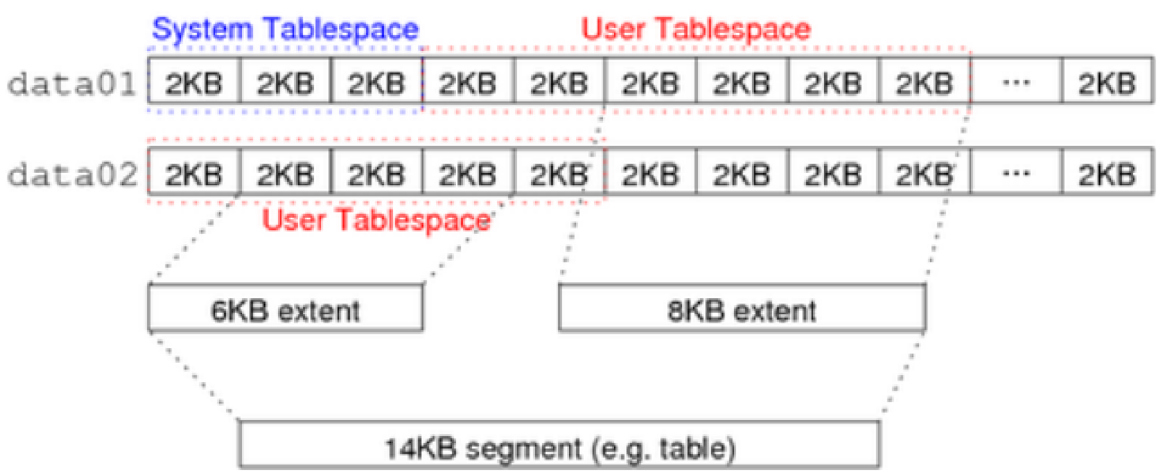
- data files: catalogue, tables, proceudures
- redo log files: update logs
- alert log files: record system events
- control files: configuration info
- archive files: off-line collected updates

layout of data within oracle file storage:

every database object resides in exactly one tablespace

Units of storage within a tablespace:

- data block:
- extent
- segment



5. PostgreSQL Storage Manager