Part 8: Disks and Caching

References:

- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Edition. Section 5.1–5.4.
- Ramakrishnan/Gehrke: Database Management Systems, 2nd Ed. Section 7.1, 7.2, 7.4.
- Garcia-Molina/Ullman/Widom: Database System Implementation. Chapter 2.
- Härder/Rahm: Datenbanksysteme, Konzepte und Techniken der Implementierung.
- Michael J. Corey, Michael Abbey, Daniel J. Dechichio, Ian Abramson: Oracle8 Tuning.
- Mark Gurry, Peter Corrigan: Oracle Performance Tuning, 2nd Edition (with disk).
- Jim Gray, Andreas Reuter: Transaction Processing: Concepts and Techniques.
- Oracle 8i Designing and Tuning for Performance, Release 2 (8.1.6), Oracle, 1999.
- Transtec Catalogue, Yellow Guide, Chap. 4: Mass Storage. [http://www.transtec.co.uk] German Version in [http://www.transtec.de].
- Seagate: [http://www.seagate.com/products/discsales/]
- Quantum/Maxtor: [http://www.maxtor.com/Maxtorhome.htm]
- IBM: [http://www.storage.ibm.com/]
- The PC Guide: Hard Disk Performance [http://www.pcguide.com/ref/hdd/perf/index.htm]
- Storage Review: [http://www.storagereview.com], [http://198.76.30.88/jive/sr/]
- Gray/Putzolu: The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time. Proc. of SIGMOD'87, Pages 395–398.
- J.N. Gray, G. Graefe: The Five Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb. ACM SIGMOD Record 26:4, 1997. pages 63–68.
- Patterson/Keeton: Hardware Technology Trends and Database Opportunities. SIGMOD'98. [http://www.cs.berkeley.edu/~pattrsn/talks/sigmod98-keynote-color.pdf]

Objectives

After completing this chapter, you should be able to:

- explain how disks work (list their main parts).
- evaluate disks, explain performance parameters.
- explain and evaluate different RAID configurations.
- create and use tablespaces in Oracle.
- explain the storage hierarchy and compare the characteristics of different storage media.
- explain how buffering (caching) works.
- find disk/buffer-related bottlenecks in Oracle.

Overview

1. Disks
2. RAID Storage
3. Tablespaces in Oracle
4. Storage Hierarchy, The Buffer Manager
5. Disk/Buffer Performance in Oracle

Disks (1)

- A disk consists of a stack of circular plates ("platters") each coated on one or both sides with magnetic recording material.

  ST318405LW (Seagate Cheetah 36XL, 18.4 GB, $260):
  2 platters of 3.5inch diameter, coated on both sides.
  This SCSI disk was current in 2001. I apologize for not updating the slides with a new disk.

- The platters are mounted to a rotating spindle.

  ST318405: platters rotate with 10000 rpm(revolutions/min).
  Other speeds are e.g. 3600, 5400, 7200, and 15000 rpm.

Disks (2)

- There is one read-write head for each magnetic surface, flying on an air cushion e.g. 3 microns above the surface.

  E.g. the ST318405 has 4 heads. Micron = 10^{-6} inch.

- The heads are mounted to an arm-assembly that looks like a comb and can move in and out.

  Only all heads together can be moved. Only one head can read or write at the same time.

Disks (3)

- The data is written on each surface in the form of concentric circles called tracks.

  ST318405: 19036 tracks per surface, 76144 in total.
  The track density is 24406 tracks/inch (TPI).

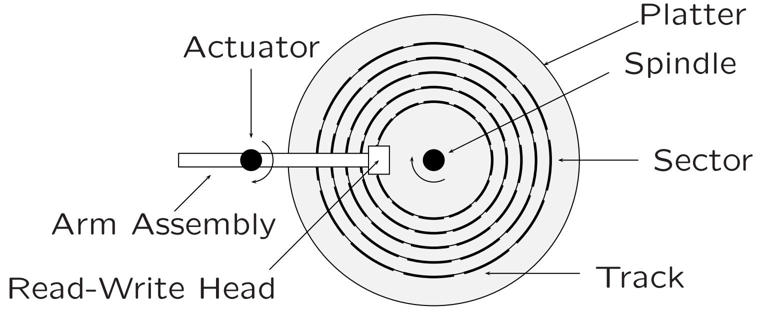
- The tracks with the same distance from the center on all surfaces together are called a cylinder.

- Each track is devided into sectors: This is the smallest unit of information that can be read/written.

  Sectors are small arcs of the circle. They often consist of 512 Bytes. Modern disks have more sectors on the outer tracks ("mutiple zone recording"), since the outer tracks are longer. The ST318405 has on average 471 sectors/track.

Disks (4)

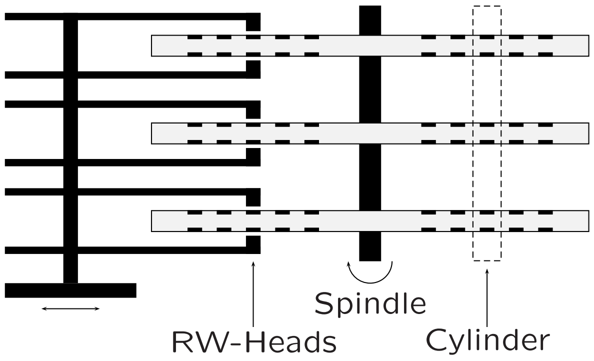
Top View:



IBM Ultrastar 36ZX:

Disks (6)

Side View:



Disks (7)

- Modern disk drives have a disk controller built-in. This is simply a small computer.
- It translates relatively high-level commands (such as read the sector with address defined by cylinder, surface, and sector number) into the commands for the real hardware (e.g. the motor for the arm-assembly).

Disks (8)

- The disk controller attaches checksums to the sectors.

  It also attaches address information to ensure that the head is really on the right track (embedded servo technology).

- It is not economically feasible to produce 100% defect-free media. Disks have a limited number of replacement sectors.

  During initialization of the disk, each sector is tested and bad sectors are found. The controller manages a defect map which uses one of the spare sectors in place of a bad sector.

Disks (9)

- Modern disks have a cache (RAM) for recently read sectors.

  If the sector is still in the cache, it can be sent immediately to the computer without the mechanical delay needed to read a sector from the disk.

- Read ahead: The controller usually reads sectors following a requested sector into the cache.

  They are anyway available while the disk continues to spin and often required next. If the program does not fast enough request the next sector, it might have passed already under the read/write-head. Without caching, one would have to wait an entire turn of the disk.

Disks (10)

- Except for the read-ahead, the disk cache is not very important for database systems, since the DBMS has a cache of its own.

  The ST318405 has 4 MB Buffer Cache. This is “multisegmented” which means that the cached data can be from different locations on the disk.

- Disks may be configured to cache write requests, too (i.e. the request is remembered for delayed execution). That is dangerous.

  If a write request is cached, it might be lost due to a power failure before it is really executed. But when the disk tells the DBMS that the data were written, the DBMS must be able to rely on that.

Disks (11)

- Blocks are a collection of consecutive sectors.

  The sector size is often 512 Byte, but it is more economical for the operating system to read/write larger units. The block size is often 8 KB under UNIX, and e.g. 2 KB under DOS (“cluster size”). Block size is usually determined during formatting as a multiple of the fixed sector size.

- In case of a power failure, it can happen that blocks are written only partially.

  This leaves one with neither the old version nor the new version, which is a problem for recovery. One technique to discover such destroyed blocks is to write the same bit pattern at the beginning and end of each block, and invert it whenever a new version is written. One can also write version numbers or checksums.

Disk Capacity

- Disk manufactureres define 1 MByte as $1000 * 1000$ Byte, otherwise 1 MByte is $1024 * 1024$ Byte.
- The operating system needs part of the disk space for control information (not available for user data).
- Thus, the disk will appear smaller than advertised.
- The required disk space will grow over time.

  A rule of thumb is that the needed disk space doubles every year. However, when planning a database, you should do a much more careful calculation. Since disk space becomes cheaper over time, it is also not a good idea to buy disk space for many years in advance. One author recommends to buy disk space for the next two years.

Reading a Sector (1)

- The operating system sends the read request over the disk interface to the disk controller.
- The arm is moved to the right cylinder (seek time).

  The average seek time is the time needed to move the arm one third of the maximal distance. For the ST318405, it is 5.4ms (read) / 6.2ms (write). The disk needs 0.8ms (read)/1.2ms (write) to position the head on the next track and 10.5/11ms for the full distance. The average seek time for write commands is slightly larger than for read commands, since for write commands the controller must be sure that the head is positioned on the right track (“it is locked into the track”). Sectors can be read earlier (when the position is not yet 100% sure) and then checked for the embedded address.

Reading a Sector (2)

- The disk then waits until the needed sector shows up under the read-write head (latency time).

  In average, half a turn is needed. So 10000rpm give an average latency of $60s/(10000 * 2) = 3ms$.

- The drive then reads the data.

  Modern disks can read an entire track in one revolution (interleave factor 1:1), i.e. the speed is (18352MB/76144)∗167(turns/s) = 40MB/s. The speed is higher on the outer tracks and lower on the inner tracks. Seagate specifies:
  Internal Transfer Rate 320–490 MBits/s,
  Internal Formatted Transfer Rate 31–50 MByte/s,
  Average Formatted Transfer Rate 43 MByte/s.

Disk Interfaces (1)

- After the disk controller has read the sector, it must transfer the data to the computer's main memory.
- The example disk has an Ultra160 SCSI (pronounced "scuzzy") interface which allows to transfer 160 MB/s.

  Previous versions: SCSI2 (Fast, Wide): 5/10/20 MB/s (1986), Ultra SCSI: 20/40 MB/s, Ultra-2 SCSI: 80 MB/s.

- Multiple disks (and other devices) can be connected over the same SCSI bus.

  Otherwise the high bandwidth would not be interesting, because it cannot be used up by a single disk (except possibly from the cache).

Disk Interfaces (2)

- PC interfaces:
  - ◇ IDE/ATA: 2–4MB/s,
  - ◇ Ultra ATA: 33MB/s,
  - ◇ Ultra ATA/66: 66MB/s.
  - ◇ Ultra ATA/100: 100MB/s.
- In future, there will probably be only one physical medium for networking and disk-computer connection, e.g. "Fibre Channel" (up to 400 MB/s).

Disk Performance (1)

- It is much faster to access consecutive blocks than blocks scattered randomly over the entire disk.

  Consecutive means first the following sectors on the same track, then another track (surface) in the same cylinder, and then an adjacent cylinder. Normally the sectors are ordered such that when we move to an neighbouring cylinder, no or only a minimal latency time is needed.

- E.g. reading 10 MB which are stored in one piece takes 0.3 seconds (307 ms).

  In the ST318405, a track contains on average 241128 Bytes, thus 44 tracks must be read. This requires one random seek (5.4 ms), latency time (3.0 ms), 10 seeks to the next cylinder ($10 * 0.8$ ms), and $11 * 3$ head switches (to adjust the head position on another surface, $33 * 0.8 = 26.4$ ms), and 44 revolutions for reading ($44 * 6 = 264$ ms).

Disk Performance (2)

- Textbooks say that 10 MB/s can be effectively read from a disk (if the data is stored in consecutive blocks).

  The above computation gives 30 MB/s.

- Reading the same amount of data from randomly scattered blocks needs about 100 times as much time.

  Assume that we need to read 5000 blocks of 2KByte.
  The time needed is $5000 * (5.4 + 3.0 + 0.05)\,\text{ms} = 44.5\,\text{s}$.

Disk Performance (3)

- Some authors say that if a disk has constantly more than 50 independent accesses per second, it becomes a bottleneck.

  This leaves 20ms for every access. The disk is faster, but when the requests are randomly distributed, and you keep the disk operating near its limits, a queue will build up. Probably today this amount has increased to 70–100 accesses per second.

- Thus, even if the entire database would fit on a single disk, it might be necessary to buy several disks if the system load is high.

  An alternative might be to increase the buffer cache (RAM), see below.

Disk Performance (4)

- To improve the performance, data that are needed together should be stored near to each other:
  - ◇ Ideally in the same block.
  - ◇ If several blocks are needed (e.g. full table scan), the data should be stored in consecutive blocks.
- Multiple disks can at least do the seek in parallel. So if the data cannot be stored together, it might be an option to store it on different disks.

  Depending on the maximal transfer rate of the interface, also the transfer can be done interleaved. Larger computers have several disk interfaces which can work in parallel.

Disk Performance (5)

- Expected future performance improvements:
  - ◇ Disk Capacity: 27–60% per year.

    Doubles every 1.5–2 years.
  - ◇ Transfer Rate: 22–40% per year.

    Doubles every 2–3.5 years.
  - ◇ Rotation/Seek Time: 8% per year.

    Halves every 7–10 years.
  - ◇ \$/MB: > 60% per year.

    Halves in less than 1.5 years.

Disk Performance (6)

- This shows that the difference between random and sequential accesses becomes greater and greater.
- In 1970, this factor was about 30, in 2000 it is about 140.

  Data of the IBM 3330 (1970): Capacity: 93.7 MB, average seek: 30 ms, next track: 10 ms, max. seek: 55 ms, 3600 rpm, 13 KB/track, 19 heads, 411 cylinders, 806 KB/s transfer rate.

Overview

RAID Storage (1)

- The speed and capacity of disks is limited.

  At least very large/fast disks become unproportionally more expensive.
- Especially the seek time is reduced very slowly compared to the processor speed.

  CPU speed: +60% per year, i.e. doubles every 1.5 years.
- Idea: Combine multiple disks to a larger storage component!
- RAID: Redundant Array of Independent (or Inexpensive) Disks.

RAID Storage (2)

- But: With 100 disks, one must expect one disk failure every 6 months.
- Therefore, RAID systems must include measures in order to prevent the loss of data when a disk fails.

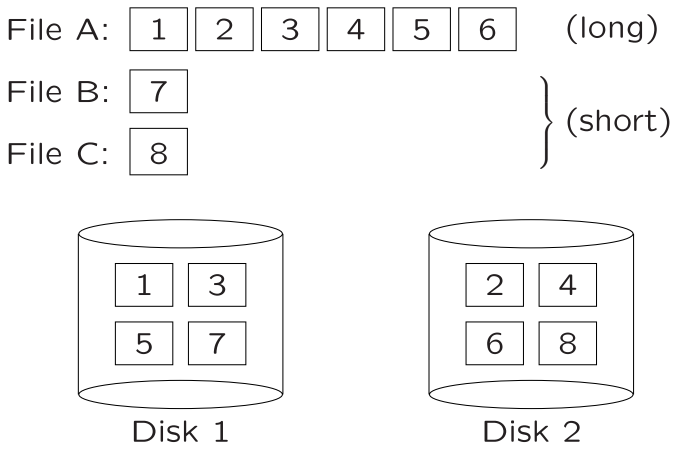
  In addition, good RAID systems allow to exchange faulty disks during the operation ("hot swap"), automatically manage a built-in spare disk ("hot spare"), and have also redundant controllers, power suplies, etc.
- There are different ways to couple the disks (RAID levels).

RAID 0: Striping (1)

- With two disks, the first block is written to disk 1, the second to disk 2, the third to disk 1, and so on.
- The speed of read/write accesses for long files scales up nearly linearly with the number of disk drives.

  With two drives, the time required for reading/writing a long file is nearly half of the time required with one disk.
- Random accesses to single blocks do not become faster, but they are distributed among multiple drives (more requests per second can be processed).
- A single disk failure makes all data unusable.

RAID 0: Striping (2)



RAID 1: Mirroring (1)

- Here disk 2 keeps a copy of disk 1.
- The time needed for write accesses is not improved.

  It required time might actually increase: Some controllers first write a block on disk 1, and only after that succeeded, also on disk 2 (to leave at least one in a consistent state).
- The time for read accesses is slightly impoved.

  A read access can be scheduled for the disk where the head is nearer. For very large files, both disks could work in parallel.
- Read accesses can be distributed among the disks (reads/sec doubles), write commands must be done on both disks.

RAID 1: Mirroring (2)

- If one of the two disks fails, the controller can continue to work without interruption with the other disk.

  If the system has a "hot spare" disk, the controller can immediately start to copy the contents of the remaining disk on the spare disk, so that the data is again kept redundantly on two disks.
- Even with mirroring, backups must be done.

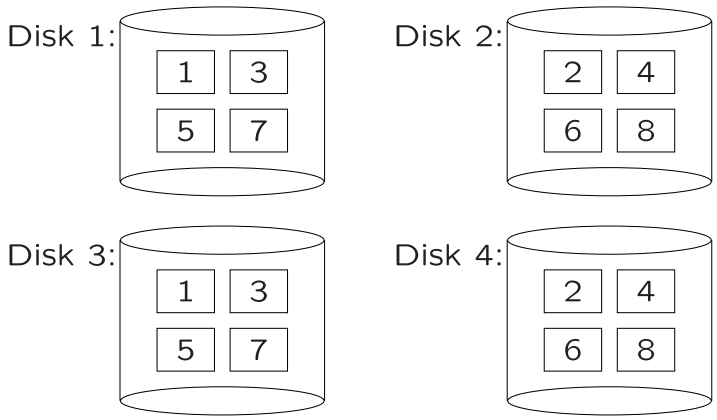
  There is no protection against a fire or errors of the DBA (`DROP TABLE ...`), software bugs, viruses etc.

RAID 10: Stripe+Mirror (1)

- RAID Level 0+1 together, i.e. striping and mirroring, is sometimes called RAID Level 10.
- E.g. the first block is stored on disk 1 and 3, the second on disk 2 and 4, the third on disk 1 and 3, etc.
- This probably gives the best performance, but the mirroring is quite expensive.

  As explained below, one can get basically the same security (no data is lost if a single disk fails) for less money, but then performance not as good as in this RAID level.

RAID 10: Stripe+Mirror (2)



RAID 4: Parity Disk (1)

- Mirroring doubles the required disk capacity (space overhead 100%).
- RAID levels 3–5 support striping of the data among n disks and add only a single disk with parity information.

  E.g. the XOR (exclusive or) of the data stored on the n disks is stored on the parity disk (i.e. the number of "1" in the corresponding bits on the $n+1$ disks is always even).
- So e.g. with four data disks, the space overhead is only 25%.

RAID 4: Parity Disk (2)

- If one disk fails, its information can be reconstructed from the other disks.

  E.g. if the number of "1" in the other n disks is even, the corresponding bit on the failed disk must have been "0". Two disk failures are always fatal. In RAID level $0+1$, one might be lucky that not both copies of the mirrored disk are hit. But for planning a safe system, it is probably more important what can be guaranteed.
- The performance for reading is similar to RAID 0:
  - ◇ The speed is improved only for long reads,
  - ◇ for small reads load-balancing is done (number of reads/second scales up lineraly with data disks).

RAID 4: Parity Disk (3)

- Writing becomes slower than with a single disk, since the old version of the block and the old version of the parity block must be read first to recompute the parity.

  NewParity = (OldData XOR NewData) XOR OldParity.
  So on each of the two disks, one needs to read a block, wait one rotation of the disk, and write it. On the ST318405 the read-modify-write cycle takes 56% more time than a simple write access. Caching might help.

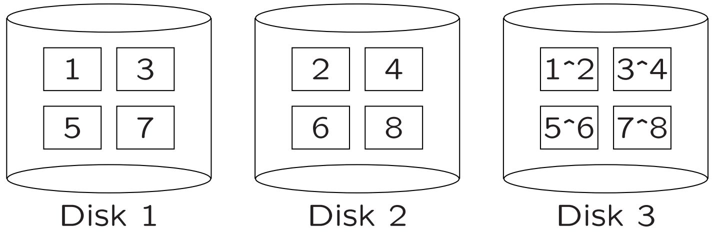
RAID 4: Parity Disk (4)

- If a disk fails, the performance goes down quite drastically (it halves although only one out of many disks failed).

  E.g. the array consists of 10 disks plus one parity disk, each disk can process 50 reads/second. Thus, the performance of whole array is 500 reads/second (assuming optimal distribution). Suppose one of the 10 data disks fails. In one second, one can read 25 times from each of the 9 working disks plus 25 times from all disks to reconstruct the blocks on the failed disk. So the performance is only 250 reads/second. If the parity disk fails, performance is not reduced.

- Larger RAID systems partition the data disks into groups, and have one parity disk for every group.

RAID 4: Parity Disk (5)



Disk 1	0	0	1	1
Disk 2	0	1	0	1
Disk 3	0	1	1	0

RAID 5: Distributed Parity (1)

- In RAID Level 4, the parity disk becomes a bottleneck for write operations.

  Each write operation, no matter on which disk, requires in addition a read and a write on the parity disk.

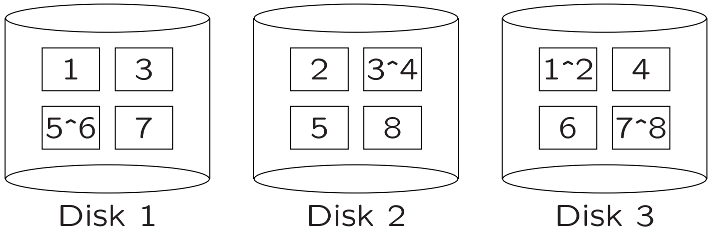
- RAID 5 distributes the parity information evenly among the disks. Otherwise, it is like RAID 4.

  Actually, reads can be distributed now over all $n+1$ disks, instead of only the n data disks.

- Some write operations can now be done in parallel.

  In RAID Level 4, writes are serialized by the necessity to access the parity disk.

RAID 5: Distributed Parity (2)



Other RAID Levels

- RAID 2 uses error detecting and correcting codes.

  This is not used in practice, since each disk normally knows that it is faulty.

- RAID 3 is similar to RAID 4, but does the striping bit-wise instead of block-wise.

  So instead of putting e.g. the first 4 KB on disk 1, and the second 4 KB on disk 2, the bits of a block of 8 KB are distributed between both disks. But then it is always necessary to access all disks (no increase in accesses/second).

- RAID 6 uses two disks with error correcting codes: It can recover from two simultaneous disk failures.

RAID Storage: Evaluation (1)

- RAID systems are easy to administer: They look like one big disk.
- But better performance can be reached if database objects are explicitly distributed over multiple disks.

  This is an important part of physical database design.

- RAID Level 0+1 might be ok for databases.

  For redo log files, the stripe size should be small. But redo log files are much better kept on disks of their own, and not mixed with the data files. They must be mirrored or otherwise protected, but the DBMS software can do that.

RAID Storage: Evaluation (2)

- If one uses RAID Level 3–5, one should benchmark the system also with a simulated disk failure.

  Will the performance still be sufficient if one disk should fail?

- RAID Level 3–5 is not good for redo log files, since these are only written.

Overview

1. Disks
2. RAID Storage
3. Tablespaces in Oracle
4. Storage Hierarchy, The Buffer Manager
5. Disk/Buffer Performance in Oracle

Tablespaces (1)

- In Oracle, one can use tablespaces to control on which disk(s) a table is stored.

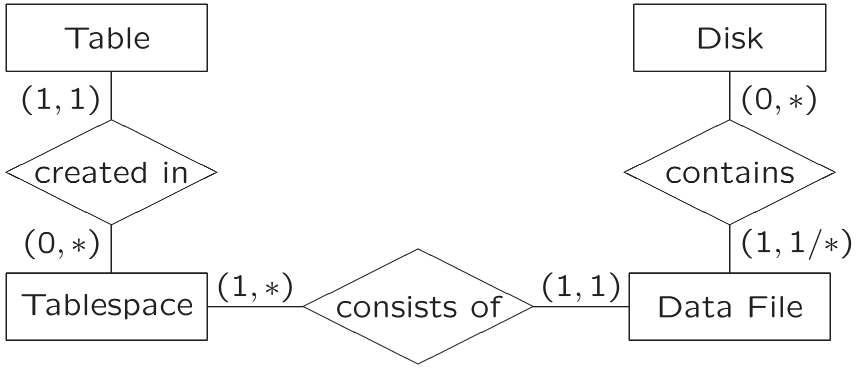
  Tablespaces are physical containers for tables. When a table is created, the tablespace in which the table is stored can be defined.

- Tablespaces are groups of data files. The files can be on the same disk or spread across serveral disks.

  A tablespace is something like a logical disk.

- Every data file can belong to only one tablespace.
- It is possible to have data files from different tablespaces on the same physical disk.

Tablespaces (2)



A data file is spread across several disks only in case of RAID systems.

Tablespaces (3)

- Example for defining the tablespace for a table:

```
CREATE TABLE STUDENTS(STUD_ID NUMERIC(5), ...)
             TABLESPACE USER_DATA;
```

- Clauses setting physical attributes are specific to a DBMS (in this case Oracle), they are not contained in the SQL standard.
- The data file cannot be specified.

  Actually, the data file cannot be specified only for the first extent (piece of storage) allocated for the table. One can manually allocate additional extents in specified data files in order to stripe a table between different disks.

Tablespaces (4)

- If the tablespace consists of more than one file, Oracle may store part of the table in one data file, and part in the other.

  For this reason, most DBAs prefer to have only one data file per tablespace, if possible.

- Every Oracle database has a tablespace "`SYSTEM`", which contains e.g. the data dictionary.

  Simple DBs have only this tablespace. However, it is recommended to store user data in a different tablespace.

Tablespaces (5)

- It is normally easiest to have only one data file per tablespace.

  If a single disk is not large enough, it might be better to explicitly distribute the data among different tablespaces.

- It increases the flexibility if not too many tables (at least no unrelated ones) are put into the same tablespace.

  Tablespaces can be separately taken online or offline, separately exported and recovered. Backup copies are taken of datafiles. Performance statistics are available for datafiles.

Managing Tablespaces (1)

- A tablespace is created with a command like the following:

```
CREATE TABLESPACE USER_DATA
DATAFILE 'D:\User1.ora' SIZE 20M;
```

- Oracle will automatically create the datafile.

  Use "`SIZE 20M REUSE`" if the file exists and can be overwritten (the size is optional in this case).

- A data file can be added to a tablespaces with the following command:

```
ALTER TABLESPACE USER_DATA
ADD DATAFILE 'D:\User2.ora' SIZE 20M;
```


Managing Tablespaces (2)

- It is possible to let Oracle extend the datafile whenever the tablespace becomes full:

```
CREATE TABLESPACE USER_DATA
DATAFILE 'D:\User1.ora' SIZE 20M
         AUTOEXTEND ON NEXT 5M MAXSIZE 50M;
```

  The file is created with 20 MB size. When it is full, it is increased to 25 MB, 30 MB, and so on until 50 MB. When the 50 MB are used up, further commands that need additional storage (e.g. insertions) will fail. Without `MAXSIZE`, the entire disk is filled.

- The data file size can also be manually increased:

```
ALTER DATABASE
DATAFILE 'D:\User2.ora' RESIZE 100M
```

Managing Tablespaces (3)

- Tablespaces can be taken offline (i.e. made not available):

```
ALTER TABLESPACE USER_DATA OFFLINE;
```

The SYSTEM tablespace cannot be taken offline.

- The following command deletes a tablespace with all data in it:

```
DROP TABLESPACE USER_DATA
INCLUDING CONTENTS;
```


Managing Tablespaces (4)

- The following command is needed when data files are renamed or moved to another disk:

```
ALTER DATABASE
RENAME FILE 'C:\User2.ora' TO 'D:\User2.ora'
```

The file cannot be currently in use. E.g. the tablespace is offline or the DBMS server is in the MOUNT state, but not OPEN. The command only changes the file name that Oracle uses to access the file (stored in Oracle's control file), one must use OS commands to actually move the data file.

- See the Oracle SQL Reference for more options.

TS in the Data Dictionary (1)

- DBA_TABLESPACES is list of all tablespaces. Columns are, e.g.
  - ◇ TABLESPACE_NAME: Name of the tablespace.
  - ◇ INITIAL_EXTENT, NEXT_EXTENT, MIN_EXTENTS, MAX_EXTENTS, PCT_INCREASE:
    Default storage parameters for tables created in this tablespace (see next chapter).
  - ◇ MIN_EXTLEN: Minimal size for storage pieces allocated in this tablespace.

TS in the Data Dictionary (2)

- Selected columns of DBA_TABLESPACES, continued:
  - ◇ STATUS: ONLINE, OFFLINE, READ ONLY.
  - ◇ CONTENTS: PERMANENT or TEMPORARY.

    TEMPORARY: For sorting during query evaluation.
  - ◇ LOGGING: LOGGING or NOLOGGING.

    If changes are not logged, they cannot be recovered.
  - ◇ EXTENT_MANAGEMENT: DICTIONARY or LOCAL.

    Extends are pieces of storage allocated for tables or other database objects. Originally, free space was managed by entries in the data dictionary. Oracle 8i has introduced local extend management (probably a bitmap inside the datafile) that is supposed to be more efficient.

TS in the Data Dictionary (3)

- `USER_TABLESPACES` lists all tablespaces for which the current user has write permission.
- `USER_TS_QUOTAS` lists the current file space usage and the allowed maximum for every tablespace writeable by the user.
- `USER_FREE_SPACE`: Pieces of free space in tablespaces.
- `TABS` (= `USER_TABLES`) contains the tablespace in which a table is stored:

```
SELECT TABLE_NAME, TABLESPACE_NAME
FROM   TABS
```


TS in the Data Dictionary (4)

- See also:
  - ◇ `DBA_DATA_FILES`,
  - ◇ `V$DATAFILE`,
  - ◇ `V$DATAFILE_HEADER`,
  - ◇ `DBA_FREE_SPACE`,
  - ◇ `DBA_FREE_SPACE_COALESCED`,
  - ◇ `DBA_TS_QUOTAS`,
  - ◇ `V$FILESTAT`,
  - ◇ `V$TABLESPACE`.

Performance Aspects (1)

- One should try to distribute the load (i.e. accesses per second) evenly between the disks.

  It is ok if a disk containing an often accessed file remains half empty.

- Distribute the tables among relatively many tablespaces to increase the flexibility for later changes.

  E.g. one can move a tablespace with all its tables to a different disk. If one has only a single tablespace with multiple files, it is not predictable, which table is contained in which file.

- One can manually stripe a table over multiple disks.

  See `ALTER TABLE ... ALLOCATE EXTENT ....`
  Necessary if the table requires more accesses/sec. than a single disk can deliver and one does not use a RAID system.

Performance Aspects (2)

- Before a `COMMIT` operation can be completed, a disk block must be written to the redo log files.

  This is actually the only case in which a disk block must be written synchronously no matter how much memory one has.

- If some disks are faster than others, think carefully what to put on them.

  If there are many transcations per minute, it would make sense to place the redo log on the fastest disks. But it is even more important not to place anything else on the disks with the redo log: The redo log is sequentially written, so the read/write head can remain at the current track and does not have to move around.

Performance Aspects (3)

- It is not recommended to store indexes on the same disk as their tables.

  This is advantageous only if there is a series of several index lookups, either because of many queries of this type or because of a join evaluated with the index. In that case the disk head does not have to move back and forth between index and table. This also gives load distribution: Both disks can work in parallel. Also, whenever the table is modified, the index must also be updated, and this can be done parallel on the two disks.

- In the same way, tables that are often joined together should be on different disks.

  Unless they can be put into a cluster, i.e. stored jointly in the same disk blocks, see below.

Performance Aspects (4)

- The data files should be stored in consecutive disk blocks.

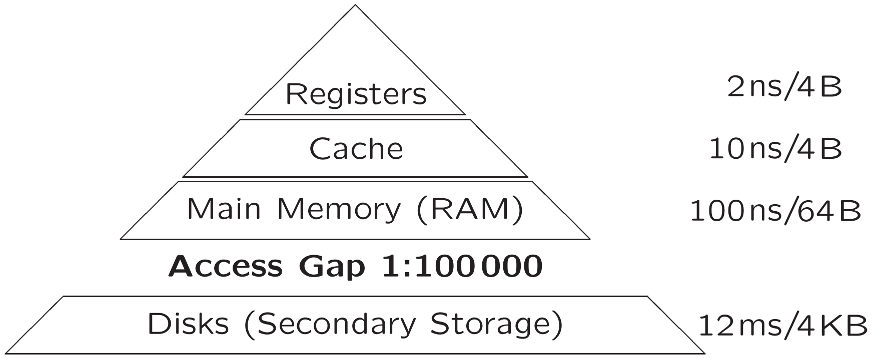
  This automatically happens if the filesystem on the disk was created directly before the DB files were created. Otherwise there might be OS defragmentation tools.

- The number of DB blocks read in one I/O operation for full table scans is determined by the initialization parameter `DB_FILE_MULTIBLOCK_READ_COUNT`.

  It should be set relatively high, e.g. 8, 16, 32.

Overview

Storage Hierarchy



Nearline External Memory (Tape Robot, Jukebox)
Offline External Memory (Tapes: Tertiary Storage)

Storage Characteristics (1)

- **Size**: Disks are usually much bigger than the main memory.

  PC (2001): 256 MB RAM, 36 GB Disk.
  AltaVista (1997): 6 GB RAM, 210 GB disks
  32Bit computers cannot have more than 4 GB RAM.

- **Cost**:
  - ◇ RAM is currently priced at approx. \$0.15–\$1.00 per MB,
  - ◇ Disk: \$0.002–0.01/MB (\$2–15/GB).
  - ◇ Tape (DLT Cartridge): approx. \$0.001/MB.

Storage Characteristics (2)

- **Persistence**: The contents of main memory is lost in case of a power failure or system crash.

  The disk contents is only lost in case of a headcrash etc. The mean time between failure (MTBF) is today typically 500000–1Mio h (57–114 years). But this measures only the probability of a failure when the drives are still young.

- **Operations**: In order to work with the data, they have to be brought into main memory.

  Disks allow random access, tapes only sequential access, CDs only read access, etc.

Storage Characteristics (3)

- **Granularity**:
  - ◇ In main memory, every bit can be accessed.
  - ◇ On disks, one has to read/write entire blocks (e.g. 2KByte).
- **Speed**:
  - ◇ Accessing a word in main memory costs e.g. 100 ns ($1\,\text{ns} = 10^{-9}\text{s}$).
  - ◇ Reading a block on a disk needs e.g. 12 ms.

    The CPU can execute e.g. 500000 instructions during one disk access. In main memory, the time needed to access a word is constant. On disks, the time depends on the distance.

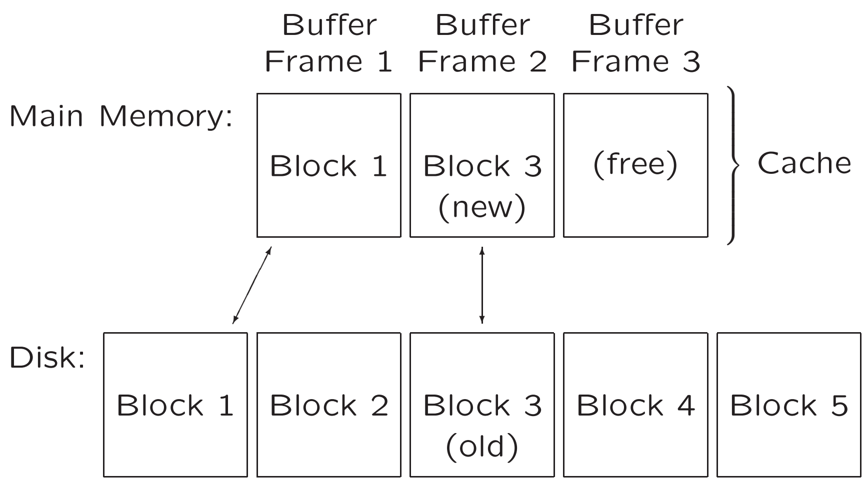
Buffering/Caching (1)

- Database blocks must be brought into main memory in order to work with them.
- The idea of buffering/caching is to keep the contents of the block for some time in main memory after the current operation on the block is done.

  Of course, if the block was modified, it might be necessary to write it back to disk. This can be delayed if other measures protect the data.

- If this same block is later requested again, it is much faster to use the copy in main memory instead of loading it again from the disk.

Buffering/Caching (2)



Buffering/Caching (3)

- The part of main memory that is used to keep copies of disk blocks is called the cache, the (disk) buffer, or the buffer cache.
- The cache is organized into pieces that can contain exactly one disk block, called (block) buffers or buffer frames.

  E.g. the block size might be 8 KB. Then each buffer frame is 8 KB large. When the cache consists of 1000 buffer frames, the cache size is 8 MB (plus some overhead for managing the cache, e.g. a table that states which disk block is contained in which buffer frame).

Buffering/Caching (4)

- If every second block request can be satisfied by using an already cached version (i.e. from main memory), the execution speed approximately doubles.

  This assumes a CPU and main memory access cost of 0, which is of course a simplification. However, because disk access is so much slower than main memory access, the result is already a relatively good approximation. If one fetched every block from the disk, the bottleneck would certainly be the disk, and the CPU would be idle for most of the time.

- In a well-tuned system, only 10% or less of the block requests really lead to a disk access.

  The remaining 90% can be satisfied from the buffer. This of course depends on the kind of queries that executed.

Buffering/Caching (5)

- One module of the DBMS software is the buffer manager (or cache manager). It gets "logical block accesses" from the upper layers of the DBMS:
  - ◇ Some of the requested blocks are contained in the cache ("cache hit"): No disk access needed.
  - ◇ Otherwise ("cache miss"), a real "physical block access" is required.
- The percentage of disk block accesses that can be satisfied from the cache is called the hit ratio. I.e. hit ratio = cache hits/(cache hits + cache misses).

Buffering/Caching (6)

- Of course, when the DBMS has just been started, the hit ratio is 0%, because the cache is still empty: Every logical block access leads to a physical block access.
- However, after some time there might have been 1000 logical block accesses and only 200 physical ones. Then the hit ratio is 80%.
- Depending on the author, good hit ratios are 80%, 90%, 95% (for normal OLTP databases).

Buffering/Caching (7)

- Normally the database is much bigger than the main memory. Therefore not all blocks can be kept in the buffer cache.
- E.g. suppose that the DB consists of 1 million blocks (8 GB), and the buffer cache consists of only 10 000 blocks (80 MB).
- If the block accesses were randomly distributed, the hit ratio would be 1%. Then the possible speedup would be 1% or less, which is not worth the effort.

Buffering/Caching (8)

- But normally, a small part of the database is accessed very often, and a large part of the database only seldom.

  An 80-20 rule applies to many things in the real world (Pareto principle). For database block accesses, it would mean that 80% of the block accesses go to 20% of the blocks. However, this would still not allow effective caching. Often, the distribution is much more uneven.

- When benchmarking a DBMS or measuring query runtimes, one must respect the cache: When the same query is executed for a second time, it usually runs much faster.

A Typical Buffer Manager (1)

- The following slides explain how the "Buffer Manager" module inside a DBMS might work.

  This is a hypothetical textbook DBMS. I believe that also Oracle basically works this way, but Oracle of course does not publish such internal details.

- This is mainly interesting for people who need to implement a DBMS.

  E.g. the Oracle development staff or students who want to work in my deductive database project.

A Typical Buffer Manager (2)

- The procedures explained on the following slides are called by the DBMS layers above the buffer manager.
- They are internal to the DBMS. The DBA or database user does not (and cannot) directly call them.

  Of course, when queries are executed, the DBMS software calls these procedures on behalf of the user.
- However, in order to do performance tuning, it is good (or even necessary) to have some understanding how the DBMS works internally.

A Typical Buffer Manager (3)

Procedure "Pin Block x":

- Determine whether block x is already in a buffer frame (i.e. in the cache).

  There probably is a hash table to quickly find the block.
- If yes (cache hit):
  - ◇ Return the memory address of this buffer frame.
  - ◇ Make sure that the block will remain in the buffer frame.

    I.e. is "pinned" there. The caller wants to work with this block in memory, and it would be fatal if suddenly the buffer frame is overwritten with another block. The caller will tell the buffer manager with "unpin" that he/she is finished.

A Typical Buffer Manager (4)

Procedure "Pin Block x", Continued:

- If the block is not in the cache (cache miss):
  - ◇ Get an empty buffer frame.

    Normally there is no empty buffer frame. Then an occupied one must be selected and made free, see "Replacement Strategy" below.
  - ◇ Call the disk manager to read the block into this buffer frame.
  - ◇ Return the memory address of the buffer frame.

    Again, the block must be pinned in the buffer frame.

A Typical Buffer Manager (5)

Procedure "Unpin Frame x" (block was not changed):

- The caller is done with this block (for the moment).
- Therefore, its buffer frame can be used for another block.

  Of course, since it is possible that the same block is requested again, it should not be immediately removed from the buffer frame — only when space is needed.
- Since the buffer frame was not changed, the disk still contains the same version.
- Thus, when space is needed, the buffer frame may simply be overwritten with another block.

A Typical Buffer Manager (6)

Procedure "Unpin Frame x" (block was changed):

- If the caller has changed the block, the version in the buffer frame is newer than the version stored on the disk.
- So before this buffer frame is reused, its contents must be written back to the corresponding block on the disk.

  The buffer is called "dirty" in this case.

A Typical Buffer Manager (7)

Delayed Writing:

- Writing a modified block back to the disk does not have to happen immediately.
- The persistence of the transaction is normally ensured via a different mechanism (log file).

  The log file is a transcript of all changes. Actually, before the buffer manager can write the block back to the disk it must ensure that the undo information was written to the log file (in Oracle via the rollback segments). This is called the WAL principle (write ahead log). If the system should crash, it must be possible to undo all changes by transactions that were not yet finished (committed). The easiest way to do this is by not writing back modified blocks that contain changes by unfinished transactions, but this has other problems.

A Typical Buffer Manager (8)

Multiple Pins for One Block:

- It is possible that multiple clients request the same block at the same time.

  In this case, one can use a "pin counter" which is incremented for every pin and decremented for every unpin. Only when this counter becomes 0, the block can be removed from the buffer frame.

- Of course, it must be avoided (by means of locks) that other processes access the block while one process changes it.

  Note that these locks should be held only for a very short time. They are something different than the locks which a transaction holds on a changed row until the commit.

Replacement Strategy (1)

- The first n requested blocks are loaded into the n available buffer frames. After that, all buffer frames are always "full".

  There is no advantage removing a block from the buffer without need (unless we know that it is not used again).

- Thus, when a new block must be loaded, a victim is selected among the blocks already in the buffer (by the "replacement strategy").
- This block is removed from the cache, and the new block is loaded into the same buffer frame.

Replacement Strategy (2)

- Note that if the block to be removed from the buffer was changed, it must be saved first.
- The system should save modified blocks from time to time so that there are sufficiently many buffer frames available which can simply be overwritten when needed.

  Oracle has one or more background processes "DB Writer" (DBW0, DBW1, etc.) for this purpose.

Replacement Strategy (3)

- Normally, a "least recently used" (LRU) strategy is used:
  - ◇ Whenever a block becomes unpinned, its buffer frame is entered into a queue (at the rear).
  - ◇ When a buffer frame is needed, the one at the front is taken.
  - ◇ If a block is pinned again while it is in the queue, it is removed from the queue.
- If all buffers are pinned, the caller must wait.

Exercise

- Suppose there are 3 buffer frames and 10 disk blocks. What actions does the buffer manager perform for these requests:
  - ◇ Pin block 1, unpin block 1 (not changed)
  - ◇ Pin block 2, unpin block 2 (changed)
  - ◇ Pin block 1, unpin block 1 (not changed)
  - ◇ Pin block 5, unpin block 5 (not changed)
  - ◇ Pin block 8, unpin block 8 (changed)
  - ◇ Pin block 1, unpin block 1 (not changed)
  - ◇ Pin block 8, unpin block 8 (changed)

Sequential Flooding (1)

- Suppose that the DBMS has n buffer frames and needs to read a table stored in $n + 1$ data blocks multiple times.

  E.g. for a nested loop join.
- Then the LRU strategy makes the buffer useless: A block is forced out of the buffer immediately before it is needed again.

  This problem is called "sequential flooding of the buffer". LRU is one of the worst possible replacement strategies here: Although we have n buffers, no block is buffered long enough to be accessed again from the buffer. If the table had $\leq n$ blocks, it would be read only once, and all following requests could be answered out of the buffer.

Sequential Flooding (2)

- One possibility to avoid this behaviour is zig-zag reading (used by some DBMS):
  - The first pass through the table is done forward,
  - the second pass backward,
  - the third pass again forward, etc.
- Oracle puts blocks read in long full table scans normally at the front of the LRU queue, so they are immediately reused.

DBMS vs. OS (1)

- Modern operating systems have virtual memory, which works quite similar to the decribed buffering scheme.
- So why repeat parts of the operating system in the DBMS?
- Operating systems are not very good in supporting the specific needs of DBMS, although they often do *nearly* the same thing.
- In the future there will be combined OS/DBMS.

DBMS vs. OS (2)

- In the OS, the file used for paging is initialized during every startup — it cannot be used it as the persistent database.

  One could of course request enough main memory to read every block from the DB into memory. But this would store most blocks two times on the disk: In the DB file and the swap file.
- On a 32bit-machine, virtual memory is limited to 4 GB. Databases can be terrabytes large.
- Operating system calls take normally quite long. But pin/unpin are called very often.

  Since a block should be kept pinned only for a short time.

DBMS vs. OS (3)

- The DBMS might have information about future references to a block, which can be utilized in the replacement strategy.

  Also prefetching of blocks, e.g. in a sequential scan, is very effective.
- The buffer frames of the DBMS should be in real memory.

  If it should happen often that buffer frames are paged out ("double paging"), the best replacement strategy becomes useless. Choose a smaller number of buffer frames and do not run other memory-intensive processes on this machine.

Overview

1. Disks
2. RAID Storage
3. Tablespaces in Oracle
4. Storage Hierarchy, The Buffer Manager
5. Disk/Buffer Performance in Oracle

Performance Monitoring (1)

- For performance tuning, the bottlenecks of the system must be found (i.e. the performance problems must be located).

  E.g. it is useless to increase the cache if it performs well.

- In Oracle, a lot of statistical information is available in the `V$*`-tables, e.g. `V$SYSSTAT`.

  The `V$*`-tables are called the "Dynamic Performance Views". They give access to data structures inside the Oracle server. They are not stored tables. Of course, the `V$*`-tables can only be accessed by the DBA.

Performance Monitoring (2)

- `V$SYSSTAT` contains 226 different performance related numbers (counters, average times, etc.). Its columns are:
  - `STATISTIC#`: Identifying number of the statistic.
  - `NAME`: Symbolic name of the statistic.
  - `CLASS`: Bit pattern to classify the statistic.

    E.g. all cache-related statistics have the third bit (8) set.
  - `VALUE`: The value of the statistic.

Performance Monitoring (3)

- E.g., this query prints the number of data blocks that were physically read since system startup:

```
SELECT VALUE
FROM   V$SYSSTAT
WHERE  NAME = 'physical reads'
```

- The Oracle server maintains a counter that is initialized to 0 when the system is started and incremented each time a block is read from disk.
- There are many different such counters.

  Some statistics are available only when the initialization parameter `TIMED_STATISTICS` is set to `TRUE` (because they cause some overhead).

Performance Monitoring (4)

- In addition, there is a table `V$SESSTAT` that contains statistics for each session. Columns are:
  - ◇ `SID`: Session identifier (more info in `V$SESSION`).
  - ◇ `STATISTIC#`: Identifying number of the statistic.
  - ◇ `VALUE`: The value of the statistic.
- Here, a join with `V$STATNAME` is necessary in order to decode the statistic numbers.

  `V$STATNAME` lists all available statistics, it has the columns `STATISTIC#`, `NAME`, `CLASS`. Some of the statistics are only meaningful in `V$SYSSTAT`, others only in `V$SESSTAT`.

Performance Monitoring (5)

- Two scripts in "`$ORACLE_HOME/rdbms/admin`" can be used to print a report containing many statistics:
  - ◇ `utlbstat.sql` (begin statistics) records the current values of the statistics counters.

    The scripts are executed with SQL*Plus. They log in as `INTERNAL`. It might be necessary to belong to the OS user group "`dba`".
  - ◇ Then there should be normal production usage of the DBMS for some time.
  - ◇ `utlestat.sql` (end statistics) computes the differences of the then current values with the stored ones and generates a report (in `report.txt`).

Performance Monitoring (6)

- SQL*Plus shows a few statistics for each executed query after

  `SET AUTOTRACE ON`
- In addition, the query execution plan is shown.

  `SET AUTOTRACE ON STATISTICS` shows only the statistics, `SET AUTOTRACE ON EXPLAIN` only the execution plan. Try also `SET TIMING ON`. If a user has rights on a view, but not the base tables, the execution plan is not shown. Before one can see the execution plan, a table for storing information about that plan must be created by executing the script `$ORACLE_HOME/rdbms/admin/utlxplan.sql`. Before a user can see the statistics, the DBA must grant the role `PLUSTRACE` to that user. The role is created with the script `$ORACLE_HOME/sqlplus/admin/plustrce.sql`.

Buffer Performance (1)

- In Oracle, the number of cache misses is the value of the counter "`physical reads`".
- The total number of requests are the sum of two statistics values (this sum is called "logical reads"):
  - ◇ `consistent gets`: Requests for block versions that contain only changes that were committed before the query started.
  - ◇ `db block gets`: Requests for the current version of a block.

Buffer Performance (2)

- In Oracle, the hit ratio is computed as:

$$\frac{\texttt{consistent gets} + \texttt{db block gets} - \texttt{physical reads}}{\texttt{consistent gets} + \texttt{db block gets}}$$

- Exercise: Write an SQL query for this.
- The hit ratio should be above 90% or 95%.

  At least for OLTP (online transaction processing) applications.

- If the hit ratio is below 60%, 70% or 80%, the buffering is not working well and something should be done.

Buffer Performance (3)

- E.g., in order to improve the hit ratio, it might be possible to increase the initialization parameter DB_BLOCK_BUFFERS (number of buffer frames).

  Total buffer memory: DB_BLOCK_BUFFERS * DB_BLOCK_SIZE.

- It is important that the entire SGA (system global area, includes the cache) remains in real memory.

  If the increase of the number of buffer frames leads to paging on the operating system level (i.e. "virtual memory" is used), the situation is worse than before ("double paging").

- If necessary, more memory must be bought.

Buffer Performance (4)

- However, before one tunes the buffer cache, there are many other things to check and improve.
- E.g., indexes might reduce the number of accessed disk blocks. Then the hit ratio will improve without adding more buffer frames.

  Oracle therefore recommends a specific sequence for tuning: Business rules, data design, application design, logical DB structure, DB operations, access paths, memory allocation, I/O and physical structure, resource contention, OS/hardware.

Buffer Performance (5)

- The hit ratio can also be improved by caching only blocks from certain tables.

  E.g., if blocks from a very large table are accessed at random, they do not profit from the valuable buffer space, but push other blocks out of the cache.

- Besides the DEFAULT buffer pool, Oracle can manage two other buffer pools: KEEP and RECYCLE.
- One can distribute the available buffer frames between these three buffer pools and assign database objects to a specific buffer pool:

```
CREATE TABLE ...(...) STORAGE(BUFFER_POOL KEEP)
```

Buffer Performance (6)

- Oracle normally places blocks read during a full table scan at the front of the LRU queue, so that the buffer frames are immediately reused.
- For small lookup tables one should request that they are even if read in a full table scan:

```
CREATE TABLE EXERCISES (CAT CHAR(1), ...)
                TABLESPACE USER_DATA
                CACHE
```

Small tables are nearly always read in full table scans.
See also: V$BUFFER_POOL, V$BUFFER_POOL_STATISTICS, V$BH.

The Five Minute Rule (1)

- Sometimes disks must be bought not because more disk space is needed, but because more accesses per second are required.
- In such a situation, caching can save not only time, but also money (see next slide).
- This rule was originally known as the five minute rule: If a block is accessed every five minutes, it should remain in the cache.
- However, as technology advanced, it is now really the ten minute rule.

The Five Minute Rule (2)

- Simplified/Naive calculation:
  - ◇ A disk costs about \$200 and allows e.g. 70 accesses per second. Suppose that each access for 8KB (in average).
  - ◇ If the data is accessed only every ten minutes from the disk, it costs $\$200/(70*600) = \$0.005$.
  - ◇ So it is still cheaper to buy 8KB more buffer (8KB RAM cost \$0.004 if price/MB is \$0.50).

Disk Performance (1)

- V$FILESTAT contains performance statistics for each file. It has the following columns (continued below):
  - ◇ FILENO#: File number.

    V$DATAFILE relates FILENO# and NAME.
  - ◇ PHYRDS: Number of read operations for this file.

    Physical reads, i.e. real reads (not from buffer).
  - ◇ PHYWRTS: Number of write operations for this file.
  - ◇ PHYBLKRD: Number of blocks read.

    PHYBLKRD can be larger than PHYRDS since sometimes a chunk of several consecutive blocks is read in one call.
  - ◇ PHYBLKWRT: Number of blocks written.

Disk Performance (2)

- Columns of V$FILESTAT, continued:
  - ◇ READTIM: Time spent in reading (in 1/100s).

    Timing information is collected only when the initialization parameter TIMED_STATISTICS is TRUE.
  - ◇ WRITETIM: Time spent in writing.
  - ◇ AVGIOTIM: Average time for an I/O operation.
  - ◇ LSTIOTIM: Time for last I/O operation.
  - ◇ MINIOTIM: Minimum time for an I/O operation.
  - ◇ MAXIOWTM: Maximum time for a write operation.
  - ◇ MAXIORTM: Maximum time for a read operation.