Although disk I/O is central to the operation of relational database systems, databases go to considerable lengths to avoid it. The reason has to do with disk access times. Access to CPU cache is significantly faster than access to main memory, which in turn is significantly faster than access to disk. So databases work more effectively with data that is a memory access away from the CPU, rather than a disk access away.

When a database operation needs a database block that is located on disk, the operation typically stalls while the page is read into memory. When data is written to the log file, the logger process stalls until the write is completed. The only database disk activity that does not routinely result in stalls is writing to database tables, since multiple pages are written asynchronously.

Overview of the Buffer Cache

One of the main tools used by databases to reduce disk I/O is the database buffer cache. The database acquires a segment of shared memory and typically sets aside the largest proportion of it to hold database blocks (also referred to as database pages). When a transaction requires a block, it reads the block from disk and stores it in the buffer cache; subsequent transactions requesting the same block can then retrieve it from memory rather than from disk.

In practice, there is rarely enough memory in the buffer cache to accommodate every database block required by transactions, so blocks cannot be held in memory indefinitely. Databases use a least recently used algorithm to
retain the most frequently accessed blocks and replace blocks that have fewer accesses.

Some databases also provide alternative strategies to discriminate against or on behalf of blocks in selected tables. For example, there is little point in caching blocks from tables that are rarely reused; conversely, some small tables are so heavily accessed that they benefit from being cached permanently.

Although effective database buffer caches are crucial for efficient functioning of OLTP applications, not all database operations use the buffer cache. In DSS environments, for example, Oracle and Inforin XPS bypass the cache for parallel queries on large tables. The reasoning is that table sizes are often so large compared to the cache size that a table scan would overwhelm the cache without any benefit. DB2 for Solaris does use the buffer cache for DSS table scans, although buffer caches do not need to be large since blocks are discarded quickly.

The buffer cache is most effective when data access is skewed. Access is skewed if some data is accessed more frequently than other data. Let me illustrate skewed access with a practical example. On a normal Saturday, my wife and I might complete a number of financial transactions. Between us we purchase food, gas, and other items, transfer money between accounts, and withdraw cash from an ATM. Since we are such busy shoppers, our account details will appear several times in the buffer cache of the accounts database at our credit union. Our credit card account will be accessed many times, our checking account might be accessed a couple of times, and our savings account probably won’t be accessed at all. The account most likely to remain in the database buffer cache is the credit card account, since it is the most heavily accessed. In practice, of course, the volume of financial transactions on a Saturday is such that no individual’s account is likely to be cached for long, but the example does illustrate skewed or nonuniform access to our three accounts.

In the same way, businesses often find that 80% of their business is done with 20% of their customers, with the result that some database rows are much more frequently accessed than others. The same principle has application in many database environments.

When access to data is uniform (that is, all rows are equally likely to be referred to), caching offers fewer benefits. When access to data is skewed, caching can greatly reduce the proportion of reads that require physical disk I/Os. Data skew is typical for OLTP applications and less typical for DSS applications.

**Monitoring the Buffer Cache**

Given that the buffer cache has a significant impact on the performance of OLTP applications, it should be no surprise that monitoring the effectiveness of the buffer cache is an important priority. The buffer cache hit rate is one of the main database metrics that you should monitor for OLTP workloads. The
Monitoring the Buffer Cache

A cache hit rate measures how many database blocks were found in the buffer cache rather than read from disk. Blocks read from disk are referred to as physical reads. Blocks retrieved either from disk or the buffer cache are referred to as logical reads (so physical disk reads are included in both metrics). The buffer cache hit rate is calculated in the following way:

\[ \text{BufferCacheHitRate} = \left(1 - \frac{\text{PhysicalReads}}{\text{LogicalReads}}\right) \times 100 \]

Not all databases supply the buffer cache hit rate in a convenient form; you might need to calculate it yourself. The chapters focusing on database tuning for Oracle, Sybase, Informix, and DB2 (Chapters 22 to 25) explain how to calculate this metric for each database.

Note that the buffer cache stores more than just table data. Index blocks are held in the buffer cache and tend to enjoy high cache hit rates since they are relatively compact and can store many index nodes per block. Since index blocks are frequently used, they are often given a more favorable weighting than data blocks so they will stay in the cache longer.

The data dictionary, which holds information about tables, columns, indexes, database files, and similar data, is held in database tables and therefore also finds its way into the buffer cache. Once the database has been running for a while, accesses to data dictionary blocks can usually be satisfied from the cache, leading to a high cache hit rate for these blocks (usually higher than for data table blocks).

**An Acceptable Cache Hit Rate**

What represents an acceptable buffer cache hit rate? Unfortunately, this question is not easy to answer.

I recently asked a database administrator about the cache hit rates on his Oracle instances. He told me they were within Oracle guidelines and added that they were over 90%.

If his answer was fuzzy, it certainly wasn’t without foundation. I recently reviewed three Oracle8 tuning textbooks and discovered three different values for the minimum recommended buffer cache hit rate for OLTP applications: 70%, 95%, and 98%.

Unfortunately, achieving an acceptable buffer cache hit rate isn’t as simple as setting an arbitrary value to aim for. As we will see, though, there are objective measures that you can use to evaluate your current hit rate. But first we need to consider the meaning and implications of the cache hit rate statistic.
The Role of the Buffer Cache

The Cache Hit Rate Confusion

During training I have sometimes asked the following question: "If you were able to improve your cache hit rate from 90% to 95%, how much of a reduction will you see in your physical reads?"

Answers have ranged from "There will be a 5% reduction" to "You can't be sure." In fact you can be sure: the physical reads will be halved.

The confusion stems from the fact that the hit rate obscures the metric that we really care about: the miss rate. The miss rate is the number of I/O requests that cannot be satisfied from the cache and therefore result in a physical read. The miss rate is simple to compute: it is 100 minus the hit rate. So a 90% cache hit rate means a 10% cache miss rate, and a 95% hit rate means a 5% miss rate.

When the earlier question is expressed in terms of the miss rate, it becomes easier to answer: "If you were able to reduce your cache miss rate from 10% to 5%, how much of a reduction will you see in your physical reads?" Reducing the miss rate from 10% to 5% means that only half as many physical reads are required.

So the number of physical disk reads is halved when the buffer cache hit rate improves from 90% to 95%, and halved again from 95% to 97.5%.

With the implications of the cache hit rate clarified, which recommendation—70%, 95%, or 98%—is the optimal cache hit rate to aim for?

Cache Hit Rate Guidelines

There is no magic figure that represents the acceptable buffer cache hit rate for a particular database application or database system. There are, however, factors that you must consider in deciding whether your current hit rate is acceptable. The following list identifies the most important factors.

- **Available memory.** You typically increase the cache hit rate by expanding the size of the buffer cache. Such an expansion only makes sense if you have enough main memory to support it, though. If you put too much memory into the buffer cache and starve the applications as a result, you will cause application paging. Application paging is likely to lead to much worse performance problems than those you solved by increasing the size of the buffer cache.

  Occasionally you might actually find yourself decreasing the size of the buffer cache. For example, if the number of active users connected to the system increases, the demands on main memory will increase also. By decreasing the size of the buffer cache, you can expect to suffer an increase in the number of physical database reads. The lower cache hit rate will be worthwhile, though, if it helps you avoid application paging.

- **Disk utilization.** An increased cache hit rate means fewer physical reads, which means fewer I/Os on the data disks. Use the iostat, sar, or statit utilities to check the utilization of the relevant disks. If the
utilization is high, reducing the number of disk reads might reduce the load on the disks sufficiently to reduce the disk response time. Reduced response times can translate into savings in transaction response times.

Remember that changes in cache hit rates have little effect on data disk writes. If, for example, you observe that half the disk I/Os are reads and half are writes, then halving the cache miss rate will only reduce disk I/Os by 25% since disk writes will stay constant.

If you are using UFS database files, then each physical read you eliminate means one less page the Solaris Operating Environment will need in the file system page cache (unless you are using Direct I/O, in which case UFS pages are not cached). The result is less kernel activity.

• Application response times. The transaction path length (that is, the number of machine code instructions executed) is shorter if a block is found in the buffer cache rather than if it is read from disk. Shorter path lengths mean lower CPU consumption, and freeing up CPU cycles can help improve transaction response times. If transaction response times are an issue, then a higher cache hit rate may help. Note, though, that many other factors contribute to transaction response times, so improvements resulting from caching effects may not prove to be significant.

A Worked Example

To illustrate the principles described above, let's consider a practical example. The following statistics, based on a real workload, were reported by the Oracle8 utlbstat and utlestat utilities (described in “The utlbstat and utlestat Scripts” on page 313):

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>consistent gets</td>
<td>52180045</td>
</tr>
<tr>
<td>db block gets</td>
<td>1675582</td>
</tr>
<tr>
<td>physical reads</td>
<td>1217906</td>
</tr>
<tr>
<td>physical writes</td>
<td>200263</td>
</tr>
</tbody>
</table>

We will flesh out the process by asking a number of questions.

• What is the current cache hit rate?

As we see from “Calculating the Buffer Cache Hit Rate” on page 317, logical reads for Oracle8 are the total of consistent gets and db block gets, so the cache hit rate is:

\[
(1 - \frac{1217906}{1675582 + 52180045}) \times 100 = 97.7\%
\]

According to two of the three Oracle8 tuning text books I referred to earlier, it's time to kick back and relax! We’re almost at the 98% target suggested in the third book, too. So all appears to be well. However, there's still a lot we don't know, so we won't take a vacation just yet.
• **What is the physical read rate?**

We can see that we are doing more than a million physical reads, which sounds like a lot of I/O. Before we become too alarmed, though, we need to find out the duration of the measurement period. At the end of the report the measurement interval was reported as being one hour, from 17:53:06 to 18:53:06. So 1,217,906 physical reads in one hour means the rate was 338 reads per second.

• **What is the physical write rate?**

The report showed that in the same interval, 200,263 physical writes were completed, or 56 writes per second. The total I/Os per second to the Oracle data disks amounted to 394 (the sum of 338 and 56).

• **How many disks, and of what type, are used to store the data tables?**

In this case, I happen to know that the data tables were striped across eight old 5,400 rpm disks, each with a capacity of 2.1 Gbytes. These disks can reasonably be expected to sustain a peak of around 60 I/Os per second. Assuming the I/Os were evenly spread across the eight disks, we were averaging 49 I/Os per second, approximately 80% of the expected I/O capacity of the disks.

Most workloads are not uniform in the load they place on the system over the course of a one-hour period; peaks and troughs are more usual. So it is likely that I/O during peak periods will exceed 60 I/Os per second per disk. Increasing the cache hit rate could prove beneficial in this case, especially if the one-hour measurement interval does not represent a heavy workload period. If it proved impossible to improve the cache hit rate for some reason, it would be wise to use more than eight disks in the stripe.

• **How much free memory is available to expand the buffer cache?**

Before we decide to increase the size of the buffer cache to improve the cache hit rate, we need to be sure that we have enough free memory in the system to support an increase.

In this case, the buffer cache was tiny—only 40 Mbytes in size—and plenty of memory was available to increase it.

• **By what proportion can we increase the buffer cache?**

We also need to determine the percentage by which we can increase the buffer cache. In this case, the size of the buffer cache could have been quadrupled, and since the load on the disk was heavy, an expanded buffer cache might have been resulted in worthwhile benefits.

Unfortunately, the higher the cache hit rate, the more memory it takes to increase it further. Eventually a point is reached beyond which there is little improvement in the cache hit rate, no matter how much memory is added.
Suppose, for example, your cache is 512 Mbytes in size and your hit rate is 80%. You double the size of the cache by adding another 512 Mbytes and discover that your cache hit rate has increased to 90%. If you are hoping that a further 512 Mbytes will improve your cache hit rate by the same amount, bringing it close to 100%, you are likely to be sadly disappointed. It is more probable that you will need to double the cache again to achieve a significant improvement. The law of diminishing returns applies to cache sizing; we discuss this effect in more detail in the next section.

With a cache hit rate of 97.7%, we might have almost reached the upper limit. On the other hand, there could be further benefits to be enjoyed. The miss rate is 2.3%, and reducing it to 1.15% (representing a cache hit rate of 98.85%) would halve physical read I/Os. Only testing can reveal the possibilities.

In summary, the objective of monitoring the buffer cache hit rate is not simply to achieve some arbitrary number; it is to optimize database performance by finding a balance between memory consumption and disk I/O.

If your cache hit rate is 70%, you will probably find that even a modest increase in buffer cache size will reduce the rate of physical reads for an OLTP workload. If, however, you have little or no spare memory, your disk subsystem is comfortably coping with the load and your users are happy, you may not need to make any changes at all.

If your cache hit rate is 95%, you will probably need a substantial increase in buffer cache size to make much impact on the rate of physical reads. If your disks are overloaded and you have plenty of spare memory, though, it is worth the attempt.

The appropriate balance in your environment will depend on data skew, application response times, and, especially, the availability of memory and disk I/O capacity.

### Sizing the Buffer Cache

Data skew and access patterns vary considerably between workloads, making it difficult to predict caching behavior. Consequently, sizing buffer caches is a challenging exercise. As a simple rule of thumb, some people suggest sizing the buffer cache as a proportion of the size of the database. I have read recommendations ranging from 1% to 20% of total database size; the variation in these recommendations highlights the degree of difficulty involved in such a sizing.

Some time ago, Performance and Availability Engineering conducted a study to better understand the relationship between cache sizes and hit rates and the impact of both on performance. As we will see, the optimal buffer cache size was between 10% and 15% of the database size for the workload used in the study.
The TPC-C schema and workload provided the test environment, and DB2 for Solaris was used as the test database.

The investigation was divided into two phases. In the first phase, the number of CPUs was held constant while the database size and the buffer cache size were varied. In the second phase, the number of CPUs was varied while the buffer cache size and database size were held constant.

Measurements were taken with buffer cache sizes ranging from 199 Mbytes to 3.1 Gbytes, and database scales of 100 (equivalent to 8.7 Gbytes), 250, 375, 425, and 525 (equivalent to 45.7 Gbytes).

The findings of the study included the observations outlined in the following sections.

**Influence of Buffer Cache Size on Throughput**

Transaction throughput is dependent on the ratio of the database size to the buffer cache size. When the buffer cache is small relative to the size of the database, throughput is far below the peak value achievable for the configuration. Figure 7.1 shows the relationship between the throughput and the size of the buffer cache.

**Figure 7.1  Throughput versus buffer cache size**
As the buffer cache is gradually increased in size, throughput rises steeply at first, then reaches a “knee point,” after which it continues to increase, but at a greatly reduced rate. The 100 scale curve clearly shows this phenomenon: as the buffer cache increases from 50,000 to 300,000 pages, the throughput increases rapidly. Beyond that point, large increases in the size of the buffer cache only yield small gains in throughput. Larger databases (scale 250 and above) also exhibit this behavior.

The 525 scale result highlights another key characteristic: as buffer cache sizes grow, it takes more memory to achieve the same throughput increase. In this case, the throughput at 200,000 buffers doubled as the cache increased by another 200,000 buffers, but beyond 400,000 buffers, it took twice as much memory (400,000 buffers) to increase the throughput by the same amount again.

Figure 7.1 also clearly demonstrates that smaller databases benefit more than larger databases when a small buffer cache is increased in size. This behavior may seem counterintuitive, but it results from the fact that a high proportion of a small database can be cached quickly, whereas the same memory increase caches a small proportion of a large database. At the same time, the growth in the throughput slows much more quickly for small databases, whereas the benefits continue for larger databases. In every case, a point of diminishing returns is eventually reached, beyond which further increases in buffer cache size yield diminishing benefits.

The study demonstrated that, for this workload, the optimal buffer cache size is between 10% and 15% of the database size. Table 7-1 shows the buffer cache sizes required to reach the “knee point,” the point beyond which further cache size increases result in diminishing returns.

<table>
<thead>
<tr>
<th>Scale</th>
<th>Buffer Size (Gbytes)</th>
<th>% of Database Size</th>
<th>Data Hit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.24</td>
<td>14.2</td>
<td>95.7%</td>
</tr>
<tr>
<td>150</td>
<td>1.59</td>
<td>12.2</td>
<td>96.7%</td>
</tr>
<tr>
<td>200</td>
<td>1.97</td>
<td>11.3</td>
<td>96.3%</td>
</tr>
<tr>
<td>250</td>
<td>2.19</td>
<td>10.1</td>
<td>96.3%</td>
</tr>
</tbody>
</table>

Results for larger database scales are not reported since the knee point was not reached (larger buffer caches would have been required). Note, too, that the cache hit rate shown in the table applies only to data pages (DB2 for Solaris separately reports cache hit rate statistics for data pages, index pages, and all pages).

As Table 7-1 shows, the smaller databases used slightly larger proportions of buffer cache memory to reach their knee points.

A more recent study on a different hardware platform with a different database suggested an optimal buffer cache size of between 5% and 10% of the database size. The difference between these results and those from the earlier study demonstrates that no single answer fits all situations.
Influence of Buffer Cache Size on Data Cache Hit Rate

Figure 7.2 shows the relationship between the data cache hit rate and the buffer cache size.

Figure 7.2 Data cache hit rate versus buffer cache size

As with throughput, the data cache hit rate initially climbs steeply as the buffer cache increases in size, then reaches a knee point and gradually settles onto a plateau. As before, the 100 scale test illustrates the behavior most clearly, whereas the larger database sizes needed more memory to reach the cache hit rate upper limit.

The practical limit to the data cache hit rate for this workload was around 98%. The overall cache hit rate was higher, though; including the index cache hit rate in this figure would have increased the upper limit to around 99% since the index cache hit rate was consistently higher than the data cache hit rate.

Further tests confirmed that the behavior described for 12 CPUs also applied to 4 and 8 CPUs.

In concluding this discussion, it is worth noting that the memory capacities of Sun servers have increased significantly in recent years, a trend that is likely to continue. Although systems with 1 Gbyte of memory per CPU
were typical in the past, current generation systems ship with 4 or 8 Gbytes per CPU. Larger memory configurations, combined with 64-bit databases on 64-bit Solaris, should allow buffer caches to be sized generously.

**Influence of Page Size on Buffer Cache Effectiveness**

Choosing the database page size (or block size) has important implications for performance of the database. Larger page sizes are usually chosen for DSS applications since they reduce overhead when large volumes of data are sequentially processed. Smaller page sizes are usually chosen for OLTP databases since I/O access tends to be random and row sizes relatively small.

One of the main reasons why the database page size impacts performance is due to its influence on the effectiveness of the database buffer cache. The page size determines both the size of pages in the buffer cache and the size of I/Os to and from the database tables on disk.

Consider a table with an average row length of 500 bytes: up to four rows can be accommodated in a 2-Kbyte page. The unit of database I/O is the page size, so every time a row is read into the buffer cache on behalf of an application, up to three other rows will accompany it. If the other three rows are required at the same time by that application or by other users, the buffer space is being used effectively and future I/Os will be saved. If the I/O access is random, though, then the other three rows are unlikely to be used.

The key issue here is access locality: locality of data access is high if data that is physically contiguous is accessed at or around the same time. Remember that in most cases, data is not stored in any particular order within a database table, so locality of access is hard to predict (although for some types of clustered index, data is stored in key sequence). When an entire table is scanned, such as for batch jobs or DSS queries, effective access locality is high since all rows will be processed within each page.

By contrast, up to 64 of the 500-byte rows can be accommodated in a 32-Kbyte page. If access locality is high and most or all of the other 63 rows can be used at the same time, many I/Os have been saved.

Large pages are the worst-case scenario, though, where data access is random and access locality is low. With 32-K byte pages, for example, each time a 500-byte row is read or written, a 32-Kbyte I/O must be completed. Worse still, for a buffer cache 2 Gbytes in size, the number of available 32-Kbyte buffers is only 65,536, compared to 1,048,576 buffers with 2-Kbyte pages. So fewer pages can be cached, while little benefit is derived from the large number of rows carried along with each page.

Very large buffer caches with 64-bit databases introduce other considerations. A 32-Gbyte buffer cache, for example, will support 16,777,216 pages 2 Kbytes in size and 1,048,576 pages 32 Kbytes in size. The overhead associated with managing the buffer cache increases as the number of pages increases. Consequently, a point will probably be reached beyond which it is more efficient to deal with fewer large pages.
The Role of the Buffer Cache

To summarize, use smaller page sizes (2 to 8 Kbytes) for OLTP applications to maximize the effectiveness of the cache for randomly accessed data. Use larger page sizes for DSS applications to reduce the number of pages involved during access to large tables.