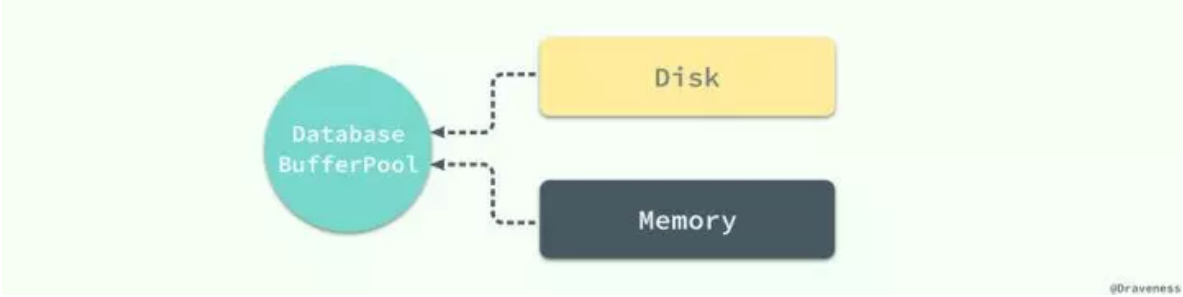
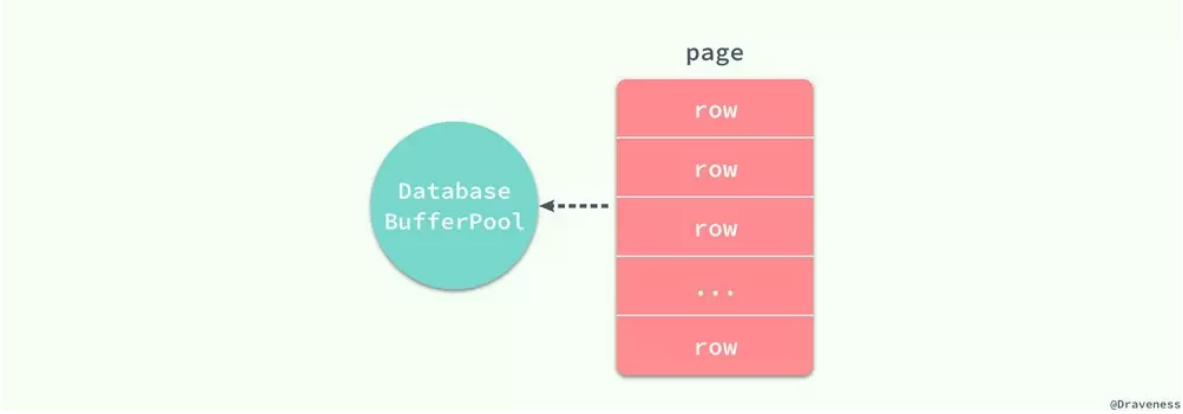
数据库和磁盘数据访问

MySQL中所有数据以文件的形式保存在磁盘上，而访问磁盘是一个非常耗时的操作。所以数据库和OS提供了缓冲池和内存来提高数据的访问速度。



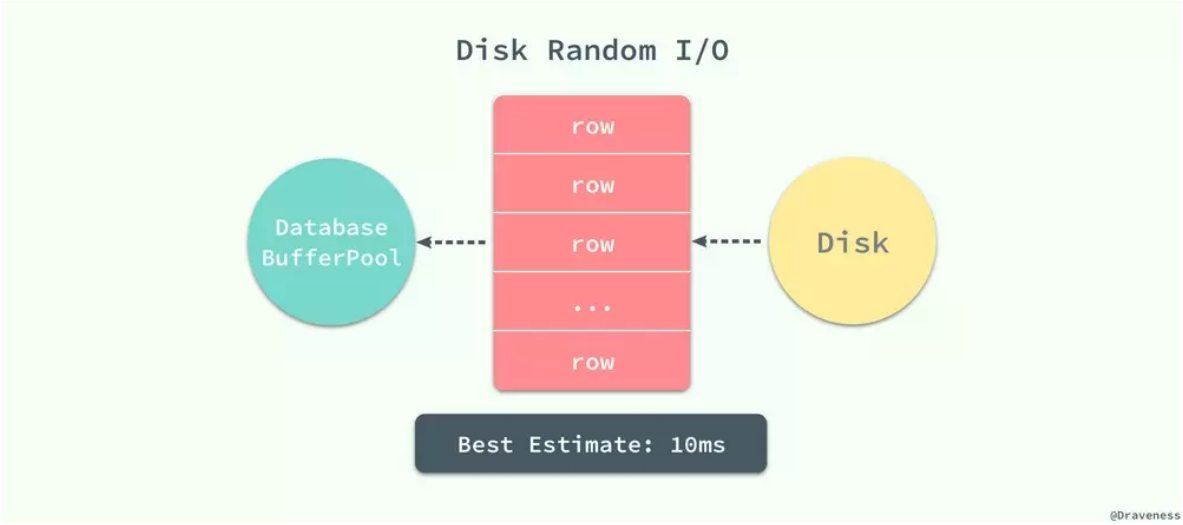
数据库对数据的访问不是以行为单位的，无论读取一行还是多行，都是将该行所在的页全部记载到内存中，然后在读取对应的数据记录。也可以从另外一个角度理解，读取数据所消耗的时间与行数无关，而是与页数有关。 这种数据访问，让我想起对磁盘数据的访问，每次不会读取一个扇区，而是读取一个或者多个页。同样都是预读取操作。

注： 数据块是数据库中数据在磁盘中存储的最小单位，也是一次IO访问的最小单位，一个数据块通常可以存储多条记录，数据块大小是DBA在创建数据库或表空间时指定。

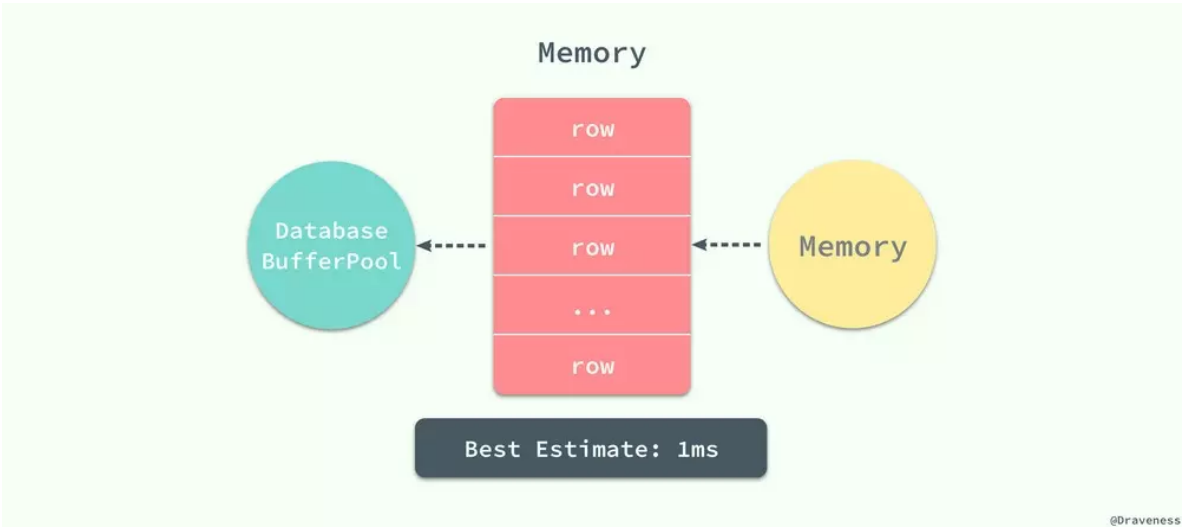


MySQL中，页的大小一般是16KB，也有可能是其他值，这与存储引擎相关。索引或者行记录是否在缓存池中极大地影响了访问索引或者数据的成本。

数据库等待一个页从磁盘读取到缓冲池需要的时间大约是10ms（巨大的开销）。



等待时间为 3ms、磁盘的实际繁忙时间约为 6ms，最终数据页从磁盘传输到缓冲池的时间为 1ms 左右。 如果在数据库的缓冲池中没有找到相关的数据页，那么回去内存中寻找数据页，时间约是1ms.



从磁盘读取数据并不是都要付出很大的代价，当数据库管理程序时，读取的速度会异常的快，大概在40MB/s 左右。

如果一个页面的大小为 4KB，那么 1s 的时间就可以读取 10000 个页，读取一个页面所花费的平均时间就是 0.1ms，相比随机读取的 10ms 已经降低了两个数量级，甚至比内存中读取数据还要快。

数据页面的顺序读取有两个非常重要的优势：

1）同时读取多个界面意味着总时间的消耗会大幅度减少，磁盘的吞吐量可以达到 40MB/s； 2）数据库管理程序会对一些即将使用的界面进行预读，以减少查询请求的等待和响应时间；

数据库缓存

缓存数据库查询结果，加快访问速度，缓解数据库压力

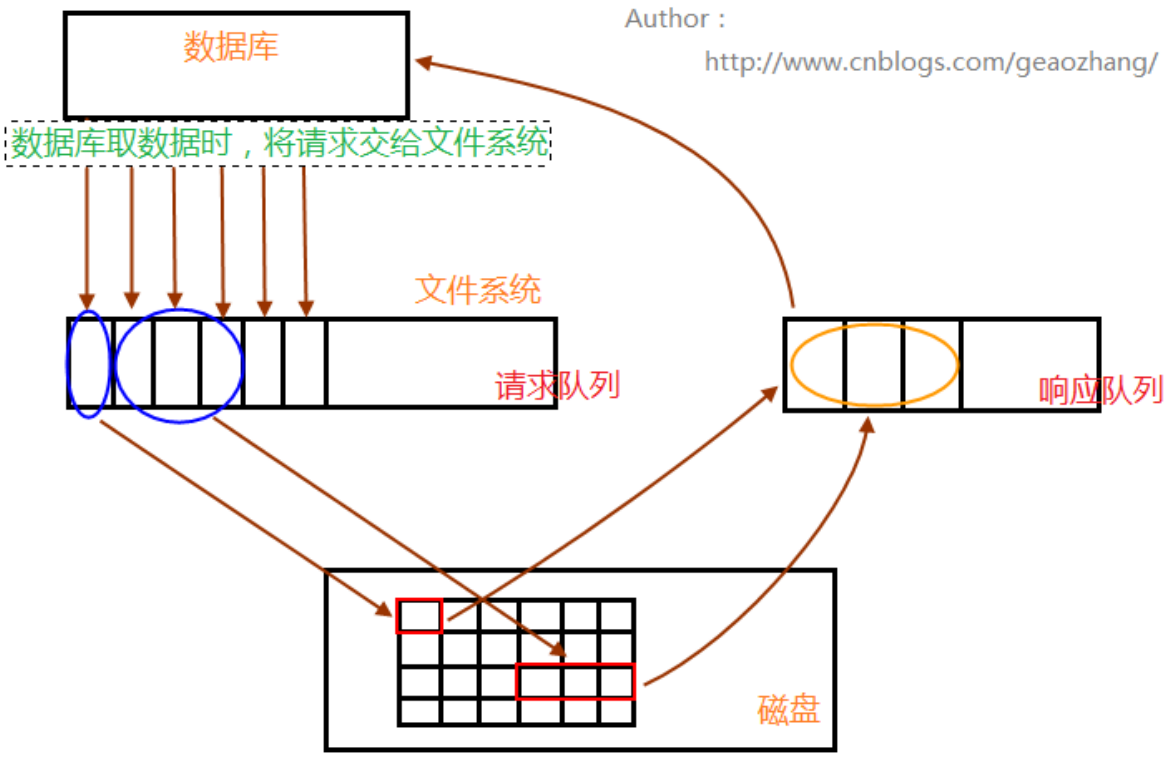
缓存原理

1. 检查用户请求的数据时缓存中是否存在，若存在直接返回，无需查询数据库。
2. 若请求数据在缓存中查询不到，去查询数据库，返回数据，同时把数据存储到缓存中一份。
3. 保持缓存的“新鲜性”，每当数据发生变化的时候，要同步更新缓存的信息，确保用户不会在缓存取到旧的数据。

关于MySQL buffer pool的预读机制

一、预读机制**

InnoDB在I/O的优化上有个比较重要的特性为预读，预读请求是一个i/o请求，它会异步地在缓冲池中预先回迁多个页面，预计很快就会需要这些页面，这些请求在一个范围内引入所有页面。InnoDB以64个page为一个extent，那么InnoDB的预读是以page为单位还是以extent?



数据库请求数据的时候，会将读请求交给文件系统，放入请求队列中；相关进程从请求队列中将读请求取出，根据需求到相关数据区(内存、磁盘)读取数据；取出的数据，放入响应队列中，最后数据库就会从响应队列中将数据取走，完成一次数据读操作过程。

接着进程继续处理请求队列，(如果数据库是全表扫描的话，数据读请求将会占满请求队列)，判断后面几个数据读请求的数据是否相邻，再根据自身系统IO带宽处理量，进行预读，进行读请求的合并处理，一次性读取多块数据放入响应队列中，再被数据库取走。(如此，一次物理读操作，实现多页数据读取，rrqm>0（# iostat -x），假设是4个读请求合并，则rrqm参数显示的就是4)

二、两种预读算法

InnoDB使用两种预读算法来提高I/O性能：线性预读（linear read-ahead）和随机预读（randomread-ahead）

为了区分这两种预读的方式，我们可以把线性预读放到以extent为单位，而随机预读放到以extent中的page为单位。线性预读着眼于将下一个extent提前读取到buffer pool中，而随机预读着眼于将当前extent中的剩余的page提前读取到buffer pool中。

1、线性预读（linear read-ahead）

线性预读方式有一个很重要的变量控制是否将下一个extent预读到buffer pool中，通过使用配置参数innodb_read_ahead_threshold，控制触发innodb执行预读操作的时间。

如果一个extent中的被顺序读取的page超过或者等于该参数变量时，Innodb将会异步的将下一个extent读取到buffer pool中，innodb_read_ahead_threshold可以设置为0-64的任何值(因为一个extent中也就只有64页)，默认值为56，值越高，访问模式检查越严格。

```
mysql> show variables like 'innodb_read_ahead_threshold';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| innodb_read_ahead_threshold | 56    |
+-----------------------------+-------+
```

例如，如果将值设置为48，则InnoDB只有在顺序访问当前extent中的48个pages时才触发线性预读请求，将下一个extent读到内存中。如果值为8，InnoDB触发异步预读，即使程序段中只有8页被顺序访问。

可以在MySQL配置文件中设置此参数的值，或者使用SET GLOBAL需要该SUPER权限的命令动态更改该参数。

在没有该变量之前，当访问到extent的最后一个page的时候，innodb会决定是否将下一个extent放入到buffer pool中。

2、随机预读（randomread-ahead）

随机预读方式则是表示当同一个extent中的一些page在buffer pool中发现时，Innodb会将该extent中的剩余page一并读到buffer pool中。

```
mysql> show variables like 'innodb_random_read_ahead';
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| innodb_random_read_ahead | OFF   |
+---------------------------+-------+
```

由于随机预读方式给innodb code带来了一些不必要的复杂性，同时在性能也存在不稳定性，在5.5中已经将这种预读方式废弃，默认是OFF。若要启用此功能，即将配置变量设置innodb_random_read_ahead为ON。

三、监控Innodb的预读

1、可以通过show engine innodb status\G显示统计信息

```
mysql> show engine innodb status\G
----------------------
BUFFER POOL AND MEMORY
----------------------
……
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
……
```

1、Pages read ahead：表示每秒读入的pages；

2、evicted without access：表示每秒读出的pages；

3、一般随机预读都是关闭的，也就是0。

2、通过两个状态值，评估预读算法的有效性

```
mysql> show global status like '%read_ahead%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Innodb_buffer_pool_read_ahead_rnd     | 0     |
| Innodb_buffer_pool_read_ahead         | 2303  |
| Innodb_buffer_pool_read_ahead_evicted | 0     |
+---------------------------------------+-------+
3 rows in set (0.01 sec)
```

1、Innodb_buffer_pool_read_ahead：通过预读(后台线程)读入innodb buffer pool中数据页数

2、Innodb_buffer_pool_read_ahead_evicted：通过预读来的数据页没有被查询访问就被清理的pages，无效预读页数