**1.内存**

**1.1缓冲池**

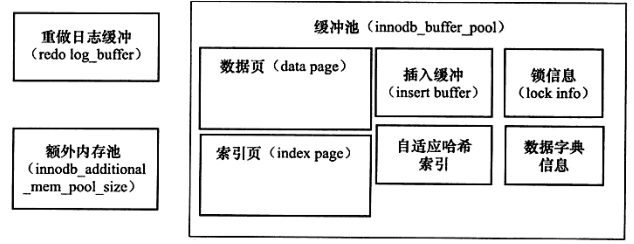
在数据库系统中，CPU与磁盘速度之间存在极大速度差异，基于磁盘的数据库系统通常使用缓冲池技术来提高数据库的整体性能。缓冲池是一块内存区域，通过内存速度弥补磁盘速度较慢带来的影响。

数据库读取页操作，首先将从磁盘读到的页存放到缓冲池中，称为将页fix到缓冲池中。下次读取相同页时，首先判断缓冲池中是否有该夜，如果有，则命中，直接读取该页，如果没有，在进行fix的过程。

数据库中页的修改操作，先修改缓冲池中的页，然后再以一定的频率刷到磁盘上。从缓冲池刷回磁盘操作并不是在每次页发生更改后进行，而是通过一种checkPoint机制刷回磁盘。

缓冲池的大小直接影响着数据库的整体性能。InnoDB存储引擎而言，缓冲池的配置通过参数innodb\_buffer\_pool\_size来设置。

缓冲池中的数据也类型有：索引页，数据页，插入缓冲，自适应哈希索引，锁信息，数据字典信息。



**1.2LRU list，Free list, Flush list**

通常来说，数据库中的缓冲池是通过LRU（最少最近使用）算法来进行管理的。最近最频繁使用的页在LRU列表的前端，最少使用的在LRU列表的末端，当缓冲池不能存放新读取的页时，将首先释放LRU列表中尾端的页。

InnoDB中，缓冲页大小默认为16KB，同样使用LRU算法对缓冲池进行管理，并对其进行了优化，，LRU列表中加入了midPoint位置，新读取到的页，虽然是最新访问的页，但并不是直接放入LRU列表的首部，而是放入到LRU列表的midPoint位置，称为中间点插入策略。默认midPoint为5/8位置，可用innodb\_old\_blocks\_pct控制（百分比）表示后面百分比为old列表。把midPoint之后的列表称为old列表，之前的列表称为new列表，new列表中的数据为热点数据。

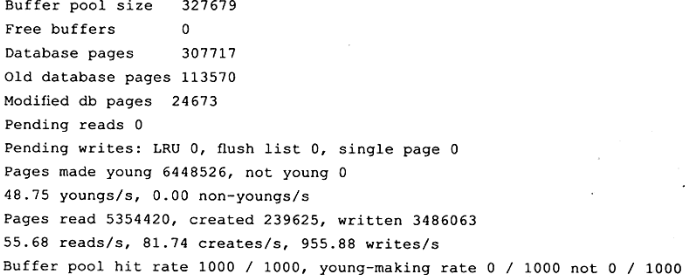
为什么要进行这样的优化呢？

假设数据50%为热点数据，在进行索引或者数据扫描操作的时候，需要访问表中许多页甚至是全部页，而这些页只是在此次查询中需要的，不是活跃的热点数据，此时如果把这些页放入到LRU的首部，这样很有可能把50%的热点数据一起清除，下次在需要读取的时候，又需要从磁盘读取。

InnoDB还引入了一个参数进一步管理LRU列表，innodb\_old\_blocks\_time，用于表示页读取到mid位置后需要等待多久才会被插入mid位置。

Page made young：页从LRU列表中的old移到new；

Page no made young：由于innodb\_old\_blocks\_time设置导致页没有从old移到new



Buffer pool size共有327679个页，即3267679\*16k，总共5G的缓冲池。

Free Buffer表示free列表的页的数量，Database pages表示LRU列表中页的数量。可能Free buffers和Database pages数量之和不等于Buffer pool size，因为缓冲池中的页还有可能分配给哈希索引，锁信息和插入缓冲等，这部分不需要LRU算法维护，不在LRU列表中

Buffer pool hit rate：缓冲池命中率，该值若小于95%的情况，用户需要观察是否由于全盘扫描引起LRU列表被污染的问题。

**Free List**

数据库刚启动的时候，LRU 列表为空，此时需要用到的时候直接将Free列表中的页删除，在LRU列表中增加相应的页，维持页数守恒。

**Flush List**

脏页：LRU列表中的页被修改，缓冲池中的页与磁盘中的页产生了数据不一致。数据库通过checkpoint机制将脏页刷新回磁盘，flush list中的页即为脏页列表。脏页既存在于LRU中，也存在于Flush中。LRU list用于管理缓冲池中页的可用性，Flush list用于将页刷新回磁盘。

**1.3重做日志缓冲**

InnoDB存储引擎首先将重做日志信息先放入到重做日志缓冲区，然后按照一定频率将其刷新道磁盘的重做日志文件中，三种情况下会将重做日志缓冲区中的内容刷新到外部磁盘重做日志文件中。

①Master Thread每一秒将重做日志缓冲刷新到重做日志文件中

②每个事务提交时会将重做日志缓冲刷新到重做日志文件中。

③当重做日志缓冲池剩余空间小于1/2时，重做日志缓冲刷新到重做日志文件中。

**1.4checkPoint技术**

在新页版本刷新到磁盘的过程中，如果发生宕机，那么数据就不能恢复了，为了避免这个发生，数据库系统普通都采用了write ahead log策略，即当事务提交时，先写重做日志缓冲区，再刷新磁盘页，如果发生数据丢失，通过重做日志来完成数据恢复。

Checkpoint技术的目的是解决以下几个问题：

①缩短数据库的恢复时间；

②缓冲池不够用时，将脏页刷新到磁盘

③重做日志不可用时，刷新脏页

当数据库发生宕机时，checkpoint之前的页都已经刷新会磁盘，只需对checkpoint后面的重做日志进行恢复。

当缓冲池不够用时，根据LRU算法会溢出最近最少使用的页，如果该页为脏页，那么需要强制执行checkPoint，将脏页刷新回磁盘。

如果数据库宕机，部分重做日志的值已经刷新回主存，那么该部分重做日志就可以被覆盖重用，如果此时重做部分还需要使用，那么必强制产生checkPoint，将缓冲池中的页至少刷新到当前重做日志位置。

InnoDB内部，有两种checkPoint，sharp checkpoint发生在数据库关闭时，将所有脏页都刷新回磁盘，属于默认工作方式，innodb\_fast\_shutdown=1，fuzzy checkPoint数据库在运行时使用，只刷新一部分脏页回磁盘。

**1.5InnoDB关键特性**

插入缓冲、两次写、自适应哈希素银、异步IO、刷新邻接页

**插入缓冲insert buffer**？

为什么需要插入缓冲？不可能每张表都只有一个聚集索引，更多情况下还有非聚集索引，例如t，有属性id，name，id为聚集索引，name为非聚集索引，但是进行插入操作的时候，数据页的存放依然是按照主键id的顺序存放的，但是对于非聚集索引，例如数据（1，b）、（2，d）、（3，a）、（4，c）这个插入b,d,a,c就显然不是顺序了，这时就需要随机访问非聚集索引页，因此会导致**插入操作性能下降**。这时需要**insert buffer**。

**Insert buffer**：对于非聚集类索引的插入和更新操作，不是每一次都直接插入到非聚集索引页中，而是先判断插入的非聚集索引页是否在缓冲池中，若在，则直接插入；若不在，则先将插入的记录放到insert buffer中，然后在进行相关操作merge插入缓冲池和辅助索引页。这样做的好处是：（1）减少磁盘的离散读取；（2）将多次插入合并为一次操作。

例如：(1,'Maria',10), (2,'David',7), (3,'Tim', 11), (4,'Jim', 7), (5,'Monty', 10), (6,'Herry', 7), (7,'Heikki', 7) 第三个参数是page\_no，按照page\_no进行排序，变成了

(2,'David',7), (4,'Jim', 7), (6,'Herry', 7), (7,'Heikki', 7) , (1,'Maria',10), (5,'Monty', 10), (4,'Tim', 11)然后合并的时候将page\_no相同的记录一次性插入到辅助索引中，并按照name进行排序，此时就可大大降低随机读取和插入次数。

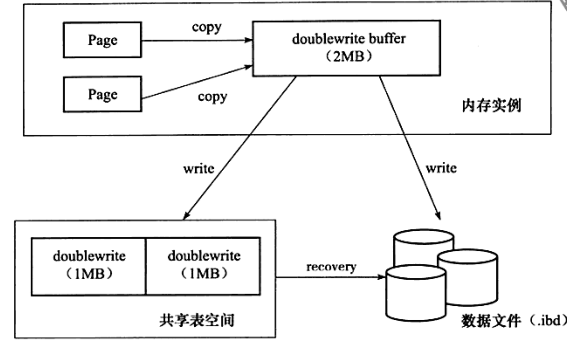
前提：

①索引是辅助索引

②索引是非唯一：如果是唯一，那么在插入的时候还需遍历已插入的记录，判断是否唯一，还需要随机读取，这样就失去了insert buffer的意义了

**两次写**？

为什么要两次写？如果正在写入某个页到表中，而这个页只写了一部分，比如16kb写了，4kb，之后发生宕机，这种情况称为部分写失效，此时不能用重做日志，因为它是对页做物理操作，前提是页必须是好的，但是此时这个页本身已经损坏，重做没有意义，只能在写入数据前，做一个**页的副本**，如果**写入失效**发生，从页的副本中还原页，这就是**两次写**。



需要额外添加两个部分：

①内存中的两次写缓冲（double writer buffer）大小为2MB

②磁盘上共享表空间中连续128页，大小也为2MB

原理：

①当刷新缓冲池脏页时，并不直接写到数据文件中，而是先拷贝到两次写缓冲区

②接着从两次写缓冲区写入共享表空间中，每次写入1MB

③再将两次写缓冲区的数据写到ibd数据文件中

④如果此时系统发生崩溃，就在共享表空间中找到该页的副本，将其拷贝覆盖数据文件，再应用到重做日志页。

**自适应哈希索引**？

InnoDB存储引擎会监控对表上各索引页的查询，如果建立哈希索引可以带来速度提升，则建立哈希索引。Adaptive hash index，AHI是通过缓冲池的B+树构造而来，建立速度非常快，不需要程序员手动干预，InnoDB引擎会自动根据访问的频率和模式来为某些热点数据建立哈希索引。

**异步IO**？

为了提高磁盘操作性能，数据库系统采用异步IO的方法来处理磁盘操作。

Sync IO，同步IO，没进行一次IO操作，需要等待此次操作结束才能继续接下来的操作，，如果出现，一次查询，需要扫描多个页，同步IO就会出现每扫描一个页并等待其完成再去下一次扫描，这样对操作性能会有影响。异步IO，用户可以在发出一个请求后立即发出另一个IO请求，全部请求发送完毕后，等待所有IO操作的完成。

AIO还能进行IO merge操作，当多个页是连续的时候，发一个IO请求，从第一页开始，往后读取多个页的大小。

**刷新邻接页**？

当刷新一个脏页时，InnoDB存储引擎会检测该页所在区的所有页，如果是脏页，那么一起进行刷新。这样做的好处是，同AIO可以将多个IO合并成一个IO，该工作机制在传统机械磁盘下有着显著的优势。