MySQL——数据页

一、简介

- 为了避免一条一条读取磁盘数据，InnoDB采取**页**的方式，作为**磁盘**和**内存**之间**交互**的**基本单位**。一个页的大小一般是**16KB**。
- InnoDB为了不同的目的而设计了多种**不同类型的页**。

> 比如：存放**表空间头部信息**的页、存放**undo日志信息**的页等等。我们把存放表中**数据记录**的页，称为**索引页**or**数据页**。

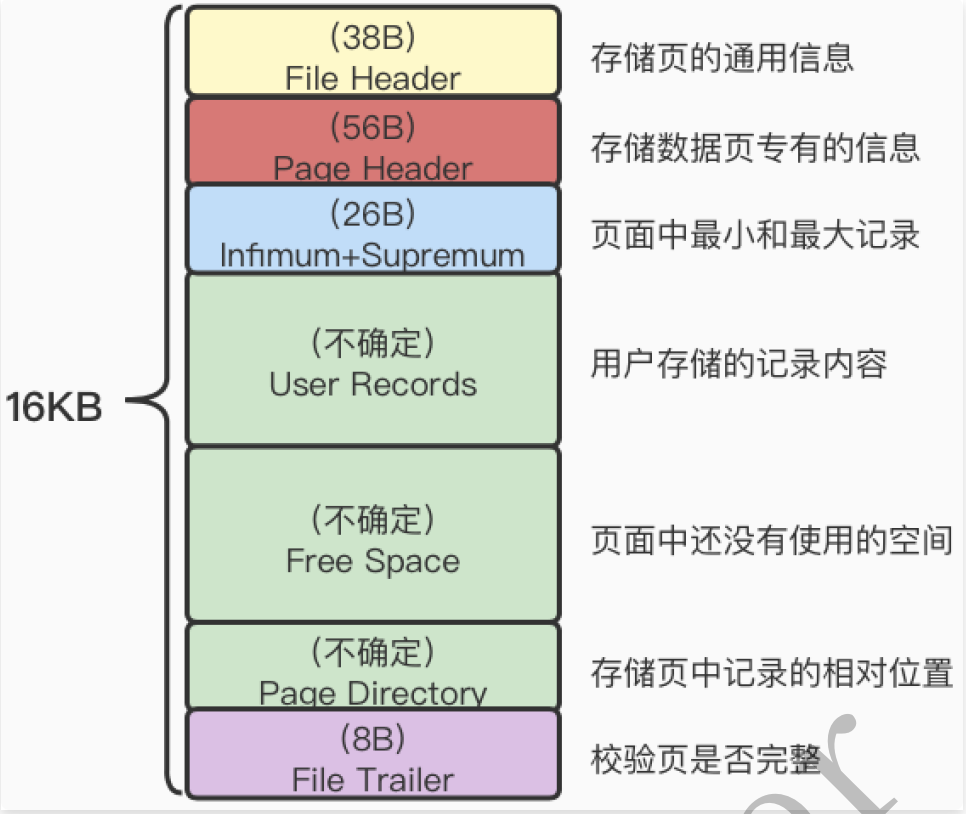
- 创建一张学生信息表

tb_student			
id	number	name	age
1	100	muse	16
2	200	bob	17
3	300	tom	16
4	400	john	18

学生信息表
id：int类型的主键
number：int类型的学生学号
name：varchar(30) 学生姓名
age：int类型的学生年龄

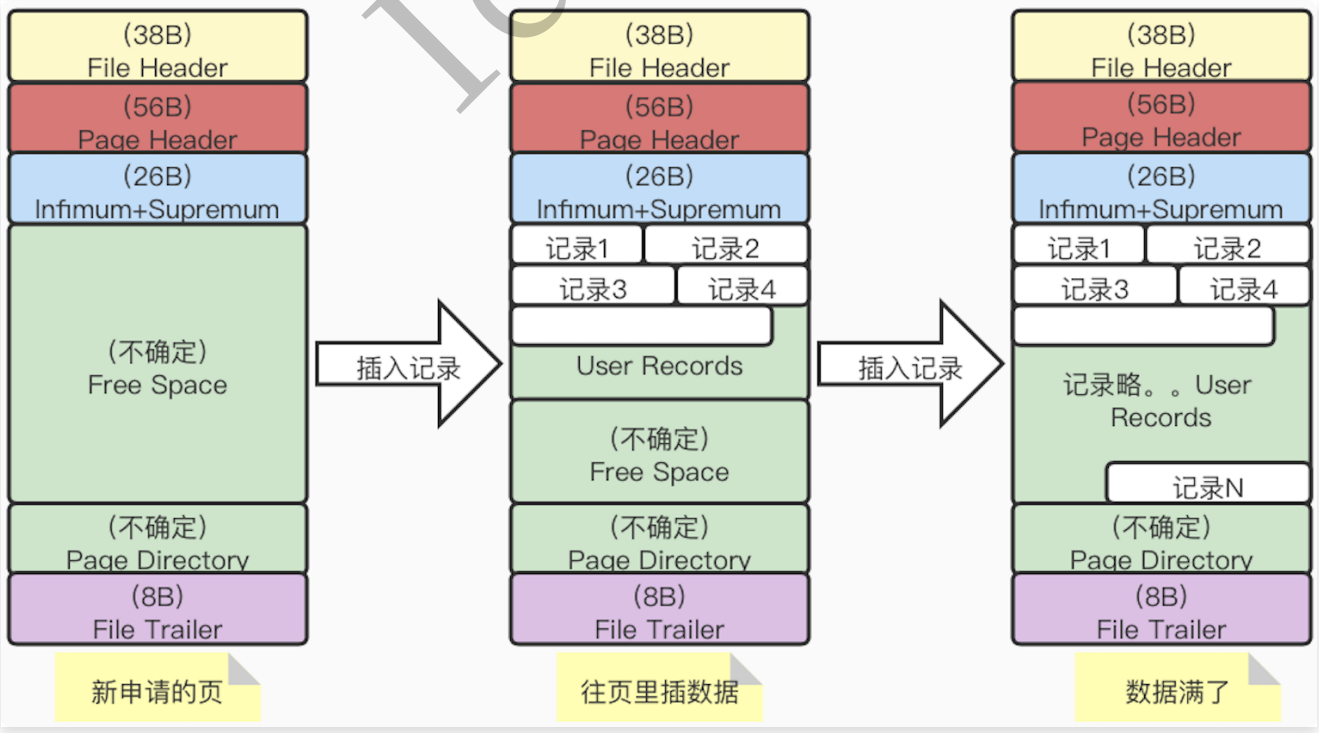
二、数据页结构概览

- InnoDB数据页结构示意图



三、记录在页中的存储

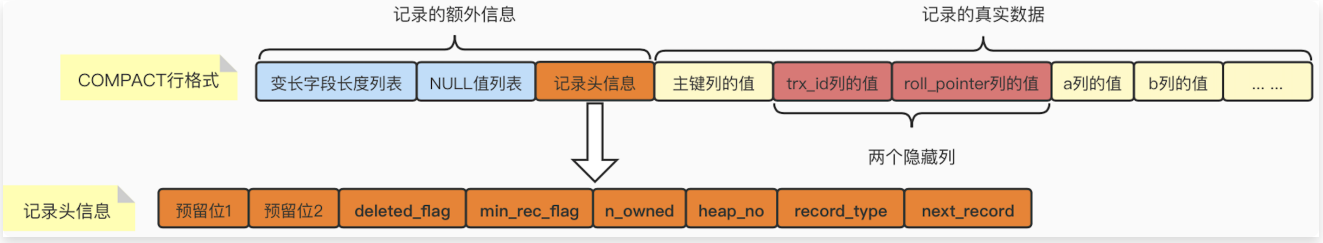
- 我们平时都是以**记录**为单位向表中插入数据的，这些记录在磁盘上的存放形式也被称为**行格式**or **记录格式**。
- 记录在页中的存储



- 在一开始生成页的时候，**没有UserRecords部分**。当插入一条记录时，就会**从Free Space中申请**一个记录大小的空间，并将这个空间**划分到User Records部分**。
- 当 `Free Space` 部分的空间全部都被 `User Records` 部分替代掉后，则这个页使用完了，如果再有新的记录，则需要去**申请新的页**了。

四、记录头信息

- COMPACT行格式示意图



- **deleted_flag**：删除标记（0:未删除 1:已删除 ）。*为什么被删除的记录还在页中？或者说，依然在磁盘上？*

> 答：这些被删除的记录之所以没有从磁盘上删除，是因为如果移除了，还需要在磁盘上**重新排列**剩余的记录，这会带来一定的**性能消耗**，所以只是打了一个 `删除的标记` 就可以避免重排。然后所有的被删除掉的记录会组成一个**垃圾链表**，记录在这个链表中占用的空间被称为**可重用空间**。之后若是有新的记录插入到表中，它们就可以**覆盖**掉被删除的这些记录占用的存储空间了

- **min_rec_flag**：B+树中**每层非叶子节点**中的**最小的目录项记录**，都会添加该标记。
- **n_owned**：一个页面被分若干组后，“带头大哥”用于保存**组中所有的记录条数**。
- **heap_no**：表示当前记录在**页面堆**中的**相对位置**。 *什么叫页面堆？heap_no作用是什么？*

> 我们向表中插入的记录都会放到User Record部分，这些记录一条条的 `连续排列` 着，InnoDB将此连续排列的结构称之为堆（heap）。为了方便管理，他们 `把一条记录在堆中的相对位置称之为heap_no` 。堆中记录的heap_no值在分配之后就**不会发生改动**了，即使删除了堆中的某条记录，这条被删除记录的heap_no值页仍然保持不变。

- *为什么用户记录的heap_no从2开始？见下图学生信息表*

> 因为创建页时，每个页会自动添加两条记录，且**都没有主键值**：
> 第一条代表页面中的**最小记录**（即：比任何用户记录都小）—— `Infimum` 记录，heap_no=0
> 第二条代表页面中的**最大记录**（即：比任何用户记录都大）—— `Supremum` 记录，heap_no=1
> 为了**区分**这两条默认记录和用户自己插入的记录，将着两条记录放到一个称为Infimum+Supremum的部分。

- **record_type**：表示当前的**记录类型**。

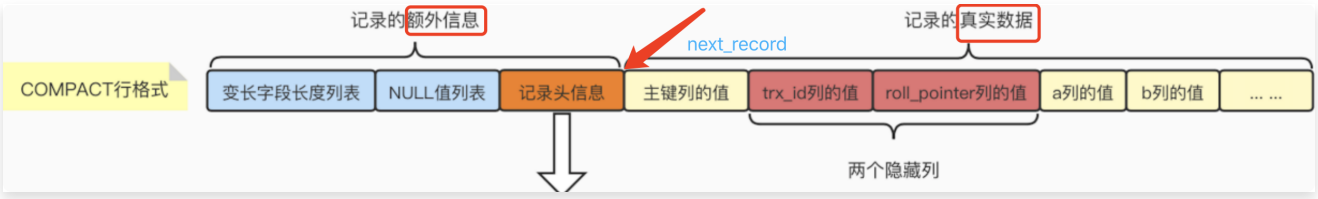
0：普通记录

1： B+树非叶子节点的目录项记录

2：表示Infimum记录

3：表示Supremum记录

- next_record：表示下一条记录的**相对位置**。就是**链表**。这个属性非常重要。它表示从当前记录的**真实数据**到下一条记录的**真实数据**的距离。如下所示：



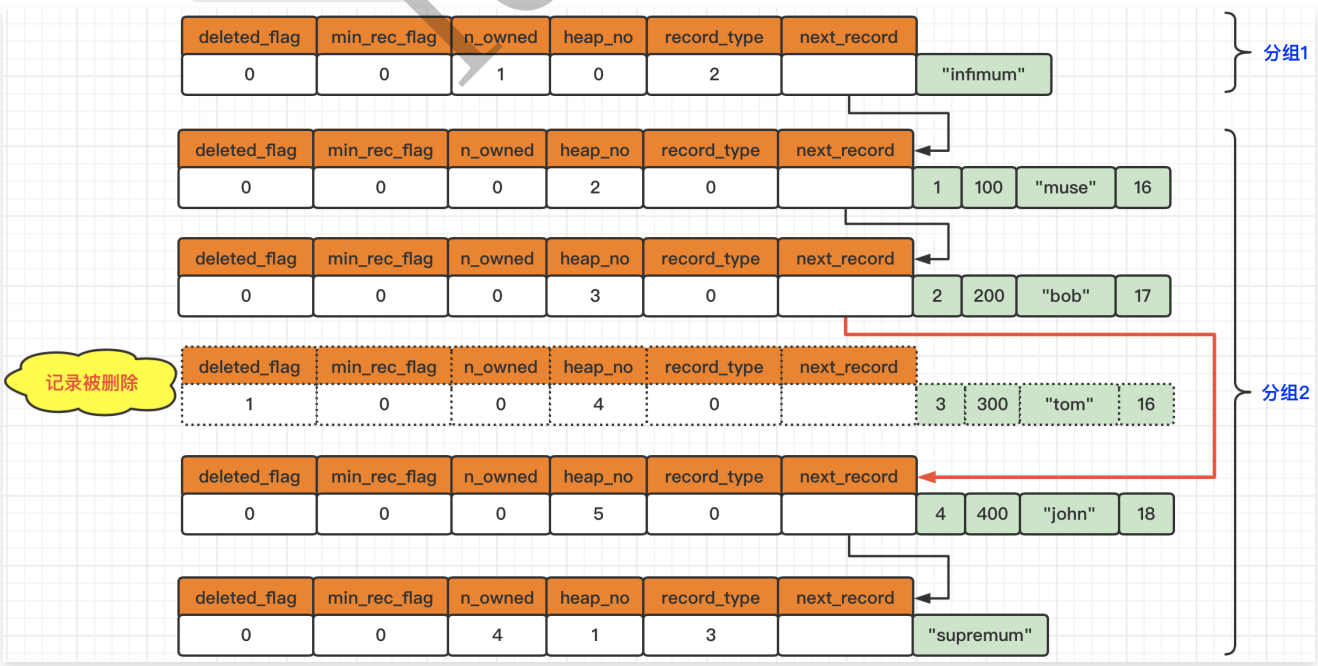
- *为什么要指向「记录头信息」和「真实数据之间」 的位置呢？而不是指向整条记录的开头位置？*

答：因为这个位置刚刚好，向左读取就是**记录头信息**；向右读取就是**真实数据**

- 该属性为**正数**——说明当前记录的下一条记录在它的**后面**。
  该属性为**负数**——说明当前记录的下一条记录在它的**前面**。

比如：一条记录的 `next_record` 值为32，意味着从当前记录的真实数据的地址处**向后找32字**节便是下一条记录的真实数据。其中：「下一条记录」指的是**按主键值** `由小到大的顺序排列` 的下一条记录。

- 通过下图，可以看出记录是**按照主键从小到大的顺序形成了一个单向链表**。
- 记录被删除对 `next_record` 的影响，如下图所示：



- `deleted_flag` 变为了1，但是并没有从磁盘中删除。

- `next_record` 变为了0，意味着没有下一条记录了。
- “bob”的 `next_record` 指向了“john”。
- supremum记录的 `n_owned` 变为了4。
- 如果再次执行插入操作 insert into tb_student values(3, 300, "tom", 16);时，InnoDB **不会因此申请新的存储空间**，而是**直接恢复**原来被删除记录的存储空间。

五、页目录Page Directory

- 记录在页中是按照**主键值从小到大**的顺序串联成为一个**单向链表**。那么如果我们要查询 `id=4` 的数据，用笨方法就是从记录的链表头开始，一直往下查找。但是如果数据量很大，那么性能就无法保证了。针对这个问题，InnoDB采取了图书目录的解决方案，即： `Page Directory` 。生成Page Directory步骤如下：
- 首先，将**非删除**的数据（包含 `Infimum` 记录和 `Supremum` 记录）划分几个组。

【分组规则如下】

- 对于Infimum记录所在的分组只能有**1条**记录。
- 对于Supremum记录所在的分组只能在**1～8条**记录之间。
- 剩下的记录所在的分组只能在**4～8条**记录之间。

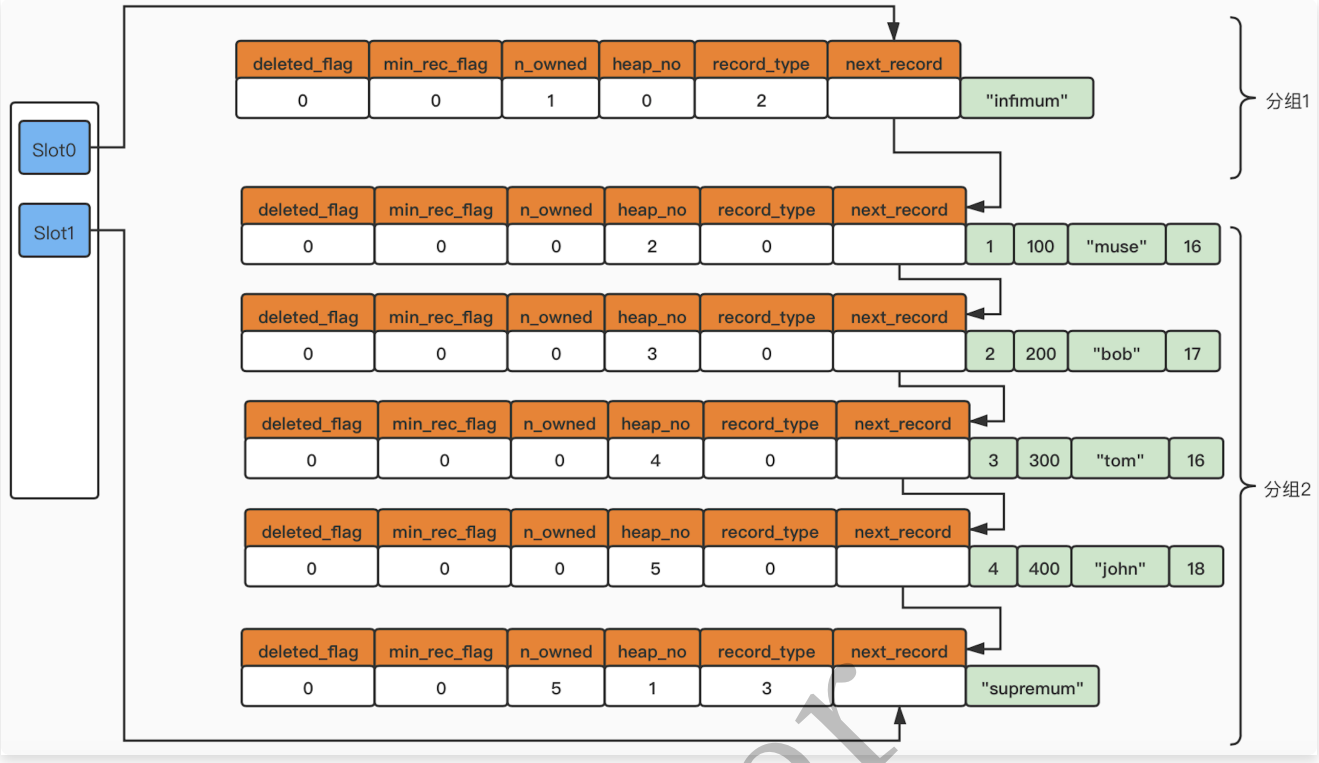
【分组步骤如下】

- 初始情况下，一个数据页中只有 `Infimum` 记录和 `Supremum` 记录这两条，所以分为两个组。
- 之后每当插入一条记录时，都会从页目录中找到对应记录的主键值比待插入记录的主键值大，并且差值最小的槽，然后把该槽对应的 `n_owned` 加1。
- 当一个组中的记录数等于 `8` 时，当再插入一条记录的时候，会将组中的**记录拆分成两个组**（一个组中 `4` 条记录，另一个组中 `5` 条记录）。并在拆分过程中，会在 `Page Directory` 中新增一个槽，并记录这个新增分组中最大的那条记录的偏移量。

- 每个**组的最后一条记录**（即：也是这个组里，最大的那条记录）——“带头大哥”，其余的记录均为“组内小弟”；“大哥”记录的头信息中的 `n_owned` 属性表示该组内共有几条记录，而“小弟”的 `n_owned` 属性都为 `0` ；
- 将“大哥”在页面中的**地址偏移量**取出来，按顺序存储到靠近 `Page Trailer` 的地方。这个地方就是 `Page Directory` 。
- Page Directory中的这些地址偏移量被称为**槽**（ `Slot` ），每个槽占用**2个字节**。

一个正常的页面为16KB，即：16384字节。而2个字节可以表示的地址偏移量范围是0～(2^16-1)，即：0～65535。所以2个字节表示一个槽足够了。

- **Page Directory就是由多个槽组成的**。数据记录和页目录的关系，如下所示，分为 2 组。



- 页目录生成完毕后，则可以通过**二分法**快速进行查找。
- 在一个数据页中查找指定主键值的记录时，过程分为两步：

  - 第一步：通过二分法确定该记录所在**分组对应的Slot**，然后找到该Slot所在分组中**主键值最小的那条记录**。
    - *每个槽对应的都是组内主键值最大的记录，那么怎么定位一个组中主键值最小的记录呢?*

      答：由于**每个槽都是挨着**的，所以，我们可以通过找到**前一个槽**中的最大主键值记录，这个记录的下一条记录（next_record），就是本槽的最小主键值记录。
  - 第二步：通过记录的**next_record属性遍历**该槽所在组中的各个记录

六、页面头部Page Header

- Page Header用于存储数据页中的**记录的状态信息**，该部分占用56个字节，专门用于存储记录的各种状态信息。详细信息如下表所示：

名称	大小	描述
PAGE_N_DIR_SLOTS	2bits	Page Directory中槽位的数量
PAGE_HEAP_TOP	2bits	未使用空间最小地址，从该地址之后就是Free Space

PAGE_N_HEAP	2bits	第1位：本记录是否为紧凑型记录 剩余15位：本页的堆中记录的数量（包含Infimum和Supremun和标记删除记录）
PAGE_FREE	2bits	每个已删除的记录通过next_record组成一个单向链表，他们的空间可以被重新利用，PAGE_FREE表示该链表头节点对应记录在页面中的偏移量
PAGE_GARBAGE	2bits	已删除记录占用的字节数
PAGE_LAST_INSERT	2bits	最后插入记录的位置
PAGE_DIRECTION	2bits	记录插入的方向【下面会有解释】
PAGE_N_DIRECTION	2bits	一个方向连续插入的记录数量【下面会有解释】
PAGE_N_RECS	2bits	该页中用户记录的数量（不包含Infimum和Supremum和被删除记录）
PAGE_MAX_TRX_ID	8bits	修改当前页的最大事务id，该值仅在二级索引页面中定义
PAGE_LEVEL	2bits	当前页在B+树中所处的层级，从0层开始
PAGE_INDEX_ID	8bits	索引ID，表示当前页属于哪个索引
PAGE_BTR_SEG_LEAF	10bits	B+树叶子节点段的头部信息，仅在B+树的根页面中定义
PAGE_BTR_SEG_TOP	10bits	B+树非叶子节点段的头部信息，仅在B+树的根页面中定义

- PAGE_DIRECTION

> 假如新插入的一条记录的主键值比上一条记录的主键值大，我们就说这条记录插入方向是右边，反之则是左边。

- PAGE_N_DIRECTION

> 假如连续几次插入新记录的方向都是一致的，InnoDB会把沿着同一个方向插入记录的条数记下来，这个条数就是PAGE_N_DIRECTION，**如果最后一条记录的插入方向发生了改变，这个状态值就会被清零后重新统计**。

七、文件头部File Header

- File Header部分是在**所有类型的页中通用**的。
- File Header的详细信息如下表所示：

名称	大小	描述
FIL_PAGE_SPACE_OR_CHKSUM	4bits	表示页的“校验和”（checksum）【下面会有解释】
FIL_PAGE_OFFSET	4bits	页号【下面会有解释】
FIL_PAGE_PREV	4bits	上一个页的页号【下面会有解释】
FIL_PAGE_NEXT	4bits	下一个页的页号【下面会有解释】
FIL_PAGE_LSN	8bits	页面被最后修改时对应的LSN值
FIL_PAGE_TYPE	2bits	该页的类型【下面会有解释】
FIL_PAGE_FILE_FLUSH_LSN	8bits	仅在系统表空间的第一个页中定义，代表文件至少被刷新到了对应的LSN值
FIL_PAGE_ARCH_LOG_NO_OR_SPACE_ID	4bits	该页属于哪个表空间

- FIL_PAGE_SPACE_OR_CHKSUM

> - MySQL4.0.14以下，该值表示本页所在的表空间ID，之后的版本，代表着 `checksum` 值。
> - *什么是checksum呢?*
>
> 就是将一个很长的字节串通过某种算法转变为较短的值来代表这个长的字节串，这个比较短的值就称为**校验和**（ `checksum` ）。这样在比较两个很长的字节串之前，**先比较他们的checksum，如果不相同，则说明这两个字节串肯定是不同的**，这样就省去了直接比较两个长字节串的时间损耗。

- FIL_PAGE_OFFSET

> 每个页都有一个**独一无二的页号**，如身份证一样。InnoDB**通过页号来唯一定位一个页**。

- FIL_PAGE_TYPE

InnoDB为了不同的目的而把页分为不同的类型，我们前面存储记录的页类型是数据页/索引页。页的类型如下所示：

名称	十六进制	描述
FIL_PAGE_TYPE_ALLOCATED	0x0000	最新分配，还未使用
FIL_PAGE_UNDO_LOG	0x0002	undo日志页
FIL_PAGE_INODE	0x0003	存储段的信息
FIL_PAGE_IBUF_FREE_LIST	0x0004	Change Buffer空闲列表
FIL_PAGE_IBUF_BITMAP	0x0005	Change Buffer的一些属性
FIL_PAGE_TYPE_SYS	0x0006	存储一些系统数据
FIL_PAGE_TYPE_TRX_SYS	0x0006	事务系统数据
FIL_PAGE_TYPE_FSP_HDR	0x0008	表空间头部信息
FIL_PAGE_TYPE_XDES	0x0009	存储区的一些属性
FIL_PAGE_TYPE_BLOB	0x000A	溢出页
FIL_PAGE_INDEX	0x45BF	索引页，也就是我们所说的数据页

- FIL_PAGE_PREV和FIL_PAGE_NEXT

通过这两个参数，就可以建立一个双向链表把许多的页串联起来，而无须这些页在物理上是真正紧挨在一起的。这里需要注意的是，**并不是所有类型的页都有这两个属性的**。

八、文件尾部File Trailer

- 我们知道，InnoDB会把数据最终持久化到磁盘上，但是磁盘的读取速度太慢了，所以，读取的时候，就会把数据放到内存中进行缓存，如果对缓存中的数据进行修改后，也会定时或者根据某些触发条件将修改后的内容刷新到磁盘上。那么，如果这个过程中出现了异常或者断电了怎么办？**为了检测一个页是否完整**，File Trailer就应运而生了，它与File Header类似，都通用于所有类型的页。它是由8个字节组成，可以分为如下两个小部分。
- 第一部分：页的checksum（占前4个字节）

当一个页在内存中被修改时，在刷新到磁盘之前首先是要计算出checksum值的。由于File Header在页面的前边，所以File Header中的checksum会被优先刷新到磁盘，当完全写完后，checksum的值再被写到File Trailer。如果页面刷新成功，那么**File Header和File Trailer的checksum值应该是一致的**。否则，就意味着刷新期间发生了错误。

- 第二部分：页被最后修改时对应的LSN的后4字节（占后4个字节）

> 正常情况下File Trailer的这部分值应该与File Header的FIL_PAGE_LSN的后4的字节相同。这部分也是**用于校验页的完整性的**。

九、总结

- InnoDB的数据页有**7个**组成部分。
- 每个**数据页**在物理结构上**可以不是相连**的，但是可以通过一个**双向链表**在逻辑上相互关联。
- 通过**File Header文件头部**中的**FIL_PAGE_PREV**和**FIL_PAGE_NEXT**构成双向链表。

> FIL_PAGE_PREV：记录 `上一个页` 的页号。
> FIL_PAGE_NEXT：记录 `下一个页` 的页号。

- 每个数据页中的**记录**，会**按照主键值从小到大**的顺序组成一个**单向链表**。
- 每个数据页都会为存储在它里面的记录生成一个**页目录**。
- 在通过主键查找某条记录的时候，可以在页目录中使用**二分法**快速定位到对应的**槽**（Slot），然后**再遍历**该槽对应**分组中**的记录，就可以快速找到指定的记录了。

吾尝终日而思矣，不如须臾之所学也；
吾尝跂而望矣，不如登高之博见也。
登高而招，臂非加长也，而见者远；
顺风而呼，声非加疾也，而闻者彰。
假舆马者，非利足也，而致千里；
假舟楫者，非能水也，而绝江河。
君子生非异也，善假于物也。
------ 摘自《劝学》

愿本文可以成为大家的“山”、“风”、“马”、“舟”，助大家在技术之路上乘风破浪，大展宏图~~

同时，也欢迎大家关注我的公众号“爪哇缪斯” ~\(^o^)/~ 「干货分享，每天更新」