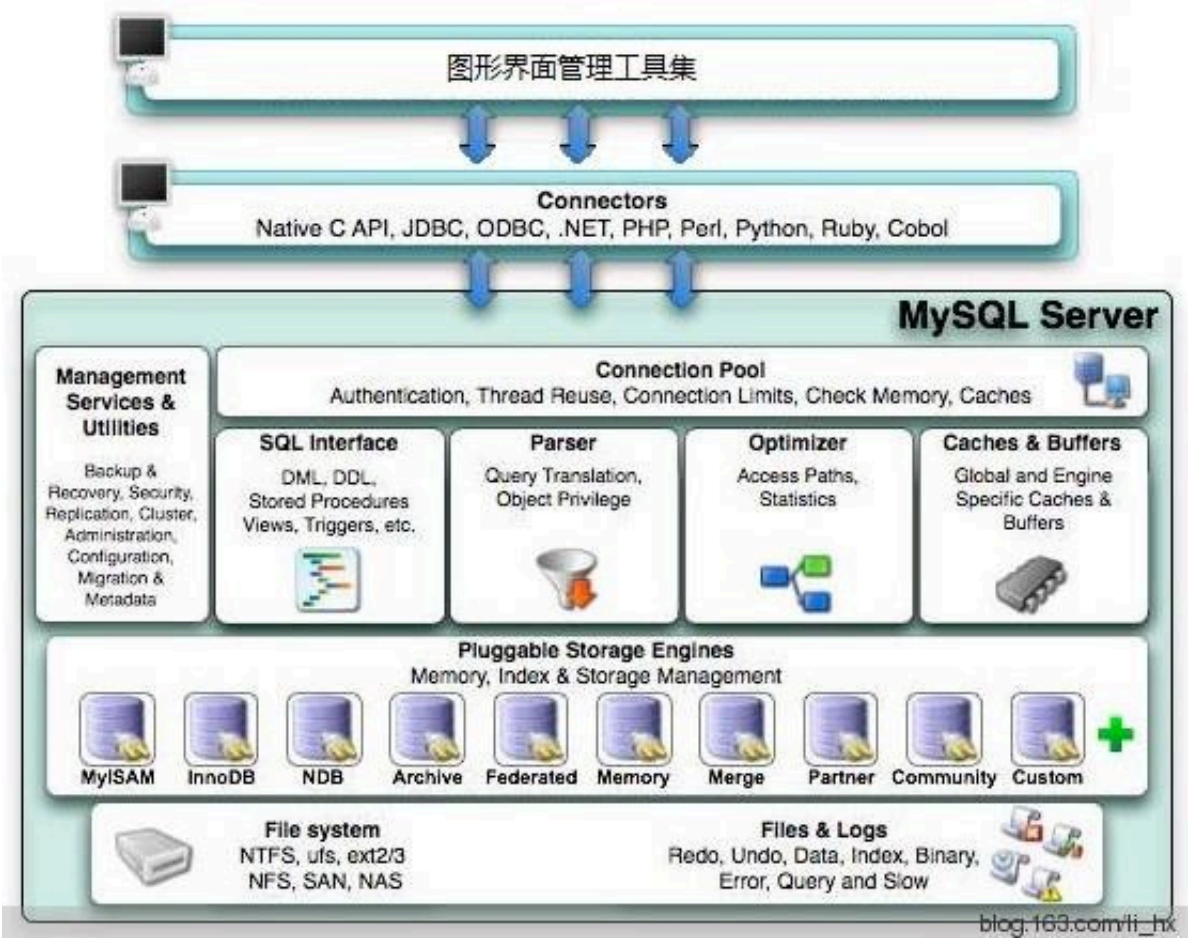
第一节-InnoDb行格式、数据页结构以及索引底层原理分析

鲁班学院-周瑜

曾参与大型电商平台、互联网金融产品等多家互联网公司的开发，曾就职于大众点评，任项目经理等职位，参与并主导千万级并发电商网站与系统架构搭建

Mysql架构图



MySQL服务器中负责对表中数据的读取和写入工作的部分是存储引擎，而服务器又支持不同类型的存储引擎，比如InnoDB、MyISAM、Memory啥的，不同的存储引擎一般是由不同的人为实现不同的特性而开发的，真实数据在不同存储引擎中存放的格式一般是不同的，甚至有的存储引擎比如Memory都不用磁盘来存储数据，也就是说关闭服务器后表中的数据就消失了。

InnoDB是一个将表中的数据存储到磁盘上的存储引擎，所以即使关机后重启我们的数据还是存在的。而真正处理数据的过程是发生在内存中的，所以需要把磁盘中的数据加载到内存中，如果是处理写入或修改请求的话，还需要把内存中的内容刷新到磁盘上。而我们知道读写磁盘的速度非常慢，和内存读写差了几个数量级，所以当我们想从表中获取某些记录时，InnoDB存储引擎需要一条一条的把记录从磁盘上读出来么？不，那样会慢死，InnoDB采取的方式是：将数据划分为若干个页，以页作为磁盘和内存之间交互的基本单位，InnoDB中页的大小一般为 16 KB。也就是在一般情况下，一次最少从磁盘中读取16KB的内容到内存中，一次最少把内存中的16KB内容刷新到磁盘中。

InnoDB数据页结构

页是InnoDB管理存储空间的基本单位，一个页的大小一般是16KB。

```
SHOW GLOBAL STATUS like 'Innodb_page_size';
```

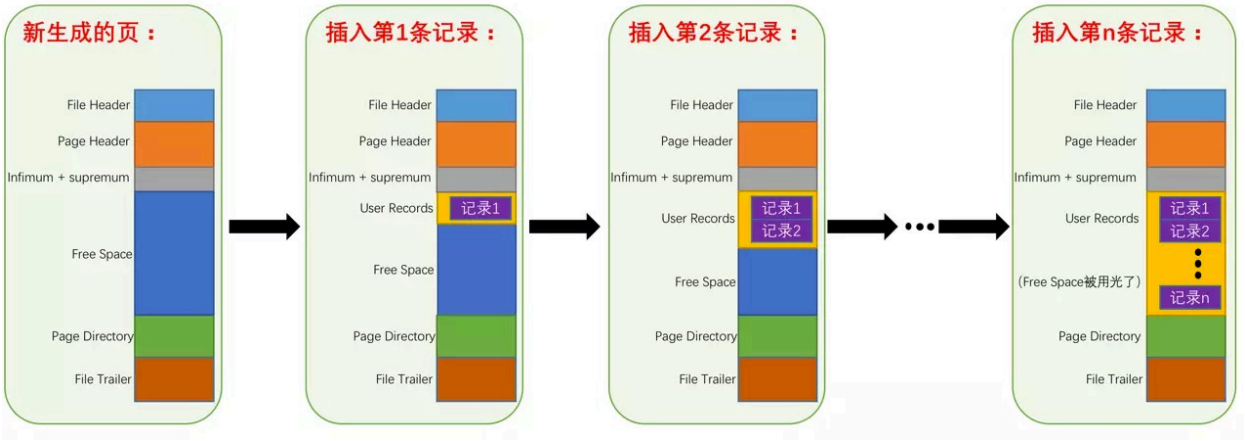
存放记录的页为数据页。

数据页代表的这块16KB大小的存储空间可以被划分为多个部分，不同部分有不同的功能，各个部分如图所示：

一个InnoDB数据页的存储空间大致被划分成了7个部分，有的部分占用的字节数是确定的，有的部分占用的字节数是不确定的。

名称	中文名	占用空间大小	简单描述
File Header	文件头部	38 字节	页的一些通用信息
Page Header	页面头部	56 字节	数据页专有的一些信息
Infimum + Supremum	最小记录和最大记录	26 字节	两个虚拟的行记录
User Records	用户记录	不确定	实际存储的行记录内容
Free Space	空闲空间	不确定	页中尚未使用的空间
Page Directory	页面目录	不确定	页中的某些记录的相对位置
File Trailer	文件尾部	8 字节	校验页是否完整

在页的7个组成部分中，我们自己存储的记录会存储到User Records部分。但是在一开始生成页的时候，其实并没有User Records这个部分，每当我们插入一条记录，都会从Free Space部分，也就是尚未使用的存储空间中申请一个记录大小的空间划分到User Records部分，当Free Space部分的空间全部被User Records部分替代掉之后，也就意味着这个页使用完了，如果还有新的记录插入的话，就需要去申请新的页了，这个过程的图示如下：



InnoDB行格式

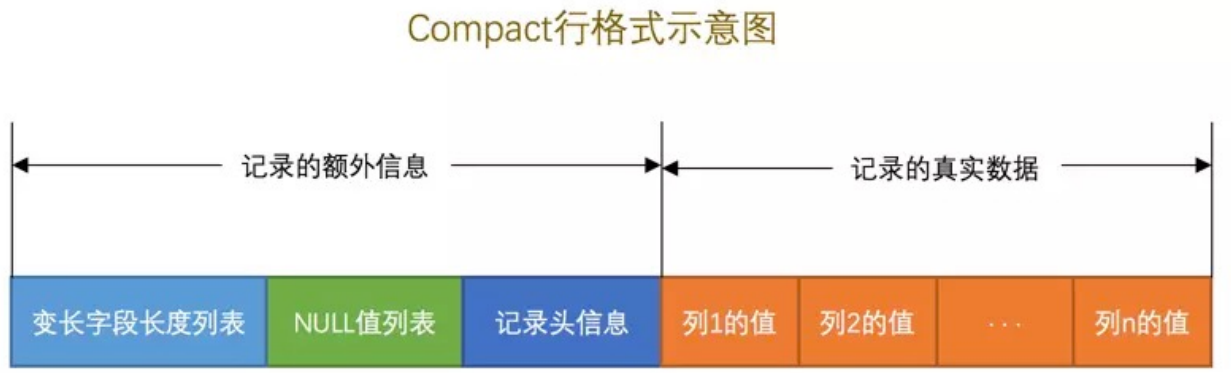
我们平时是以记录为单位来向表中插入数据的，这些记录在磁盘上的存放方式也被称为行格式或者记录格式。设计InnoDB存储引擎的大叔们到现在为止设计了4种不同类型的行格式，分别是Compact、Redundant、Dynamic和Compressed行格式，随着时间的推移，他们可能会设计出更多的行格式，但是不管怎么变，在原理上大体都是相同的。

我们可以在创建或修改表的语句中指定行格式：

```
CREATE TABLE 表名 (列的信息) ROW_FORMAT=行格式名称

ALTER TABLE 表名 ROW_FORMAT=行格式名称
```

COMPACT行格式



记录的额外信息

这部分信息是服务器为了描述这条记录而不得不额外添加的一些信息，这些额外信息分为3类，分别是变长字段长度列表、NULL值列表和记录头信息。

变长字段长度列表

我们知道MySQL支持一些变长的数据类型，比如VARCHAR(M)、VARBINARY(M)、各种TEXT类型，各种BLOB类型，我们也可以把拥有这些数据类型的列称为变长字段，变长字段中存储多少字节的数据是不固定的，所以我们在存储真实数据的时候需要顺便把这些数据占用的字节数也存起来。

在Compact行格式中，把所有变长字段的真实数据占用的字节长度都存放在记录的开头部位，从而形成一个变长字段长度列表。

> CHAR是一种固定长度的类型，VARCHAR则是一种可变长度的类型，VARCHAR(M)，M代表最大能存多少个字符。

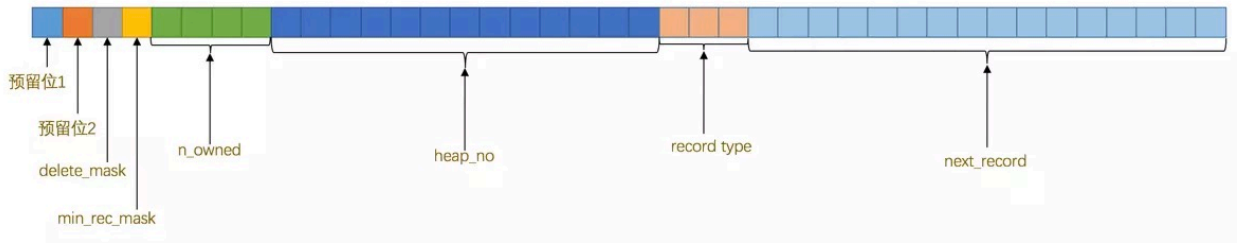
NULL值列表

我们知道表中的某些列可能存储NULL值，如果把这些NULL值都放到记录的真实数据中存储会很占地方，所以Compact行格式把这些值为NULL的列统一管理起来，存储到NULL值列表中，如果表中没有允许存储 NULL 的列，则 NULL值列表也不存在了，将每个允许存储NULL的列对应一个二进制位：

- 二进制位的值为1时，代表该列的值为NULL。
- 二进制位的值为0时，代表该列的值不为NULL。

记录头信息

除了变长字段长度列表、NULL值列表之外，还有一个用于描述记录的记录头信息，它是由固定的5个字节组成。5个字节也就是40个二进制位，不同的位代表不同的意思，如图：



记录的真实数据

记录的真实数据除了我们自己定义的列的数据以外，MySQL会为每个记录默认的添加一些列（也称为隐藏列），具体的列如下：

列名	是否必须	占用空间	描述
row_id	否	6 字节	行ID，唯一标识一条记录
transaction_id	是	6 字节	事务ID
roll_pointer	是	7 字节	回滚指针

> 实际上这几个列的真正名称其实是：DB_ROW_ID、DB_TRX_ID、DB_ROLL_PTR。

InnoDB表对主键的生成策略：优先使用用户自定义主键作为主键，如果用户没有定义主键，则选取一个Unique键作为主键，如果表中连Unique键都没有定义的话，则InnoDB会为表默认添加一个名为row_id的隐藏列作为主键。所以我们从上表中可以看出：InnoDB存储引擎会为每条记录都添加 transaction_id 和 roll_pointer 这两个列，但是 row_id 是可选的（在没有自定义主键以及Unique键的情况下才会添加该列）。这些隐藏列的值不用我们操心，InnoDB存储引擎会自己帮我们生成的。

行溢出数据

对于VARCHAR(M)类型的列最多可以占用65535个字节。其中的M代表该类型最多存储的字符数量，如果我们使用ascii字符集的话，一个字符就代表一个字节，我们看看VARCHAR(65535)是否可用：

```
mysql> CREATE TABLE varchar_size_demo(
    ->     c VARCHAR(65535)
    -> ) CHARSET=ascii ROW_FORMAT=Compact;
ERROR 1118 (42000): Row size too large. The maximum row size for the used table type,
not counting BLOBs, is 65535. This includes storage overhead, check the manual. You
have to change some columns to TEXT or BLOBs
mysql>
```

从报错信息里可以看出，MySQL对一条记录占用的最大存储空间是有限制的，除了BLOB或者TEXT类型的列之外，其他所有的列（不包括隐藏列和记录头信息）占用的字节长度加起来不能超过65535个字节。所以MySQL服务器建议我们把存储类型改为TEXT或者BLOB的类型。这个65535个字节除了列本身的数据之外，还包括一些其他的数据，比如说我们为了存储一个VARCHAR(M)类型的列，其实需要占用3部分存储空间：

1. 真实数据
2. 真实数据占用字节的长度
3. NULL值标识，如果该列有NOT NULL属性则可以没有这部分存储空间

如果该VARCHAR类型的列没有NOT NULL属性，那最多只能存储65532个字节的数据，因为真实数据的长度可能占用2个字节，NULL值标识需要占用1个字节。

```
mysql> CREATE TABLE varchar_size_demo(
    ->     c VARCHAR(65532)
    -> ) CHARSET=ascii ROW_FORMAT=Compact;
Query OK, 0 rows affected (0.02 sec)
```

```
CREATE TABLE varchar_size_demo(
    c VARCHAR(65533) not null
) CHARSET=ascii ROW_FORMAT=Compact;
Query OK, 0 rows affected (0.02 sec)
```

记录中的数据太多产生的溢出

MySQL中磁盘和内存交互的基本单位是页，也就是说MySQL是以页为基本单位来管理存储空间的，我们的记录都会被分配到某个页中存储。而一个页的大小一般是16KB，也就是16384字节，而一个VARCHAR(M)类型的列就最多可以存储65533个字节，这样就可能造成一个页存放不了一条记录。

在Compact和Reduntant行格式中，对于占用存储空间非常大的列，在记录的真实数据处只会存储该列的一部分数据，把剩余的数据分散存储在几个其他的页中，然后记录的真实数据处用20个字节存储指向这些页的地址（当然这20个字节中还包括这些分散在其他页面中的数据的占用的字节数），从而可以找到剩余数据所在的页。

Dynamic和Compressed行格式

这两种行格式类似于COMPACT行格式，只不过在处理行溢出数据时有点儿分歧，它们不会在记录的真实数据处存储一部分数据，而是把所有的数据都存储到其他页面中，只在记录的真实数据处存储其他页面的地址。

另外，Compressed行格式会采用压缩算法对页面进行压缩。

索引

索引的产生过程看视频吧...

聚簇索引

B+树索引有两个特点：

1. 使用记录主键值的大小进行记录和页的排序，这包括三个方面的含义：
    - 页内的记录是按照主键的大小顺序排成一个单向链表。
    - 各个存放用户记录的页也是根据页中用户记录的主键大小顺序排成一个双向链表。
    - 存放目录项记录的页分为不同的层次，在同一层次中的页也是根据页中目录项记录的主键大小顺序排成一个双向链表。
2. B+树的叶子节点存储的是完整的用户记录。所谓完整的用户记录，就是指这个记录中存储了所有列的值（包括隐藏列）。

我们把具有这两种特性的B+树称为聚簇索引，所有完整的用户记录都存放在这个聚簇索引的叶子节点处。这种聚簇索引并不需要我们在MySQL语句中显式的使用INDEX语句去创建。InnoDB存储引擎会自动的为我们创建聚簇索引。在InnoDB存储引擎中，聚簇索引就是数据的存储方式（所有的用户记录都存储在了叶子节点），也就是所谓的索引即数据，数据即索引。

二级索引（复制索引）

聚簇索引只能在搜索条件是主键值时才能发挥作用，因为B+树中的数据都是按照主键进行排序的。当我们想以别的列作为搜索条件时我们可以多建几棵B+树，不同的B+树中的数据采用不同的排序规则。

二级索引与聚簇索引有几处不同：

1. 所有排序都按对应的索引列来进行排序
2. B+树的叶子节点存储的并不是完整的用户记录，而只是索引列+主键这两个列的值。
3. 目录项记录中不再是主键+页号的搭配，而变成了索引列+页号的搭配。
4. 在对二级索引进行查找数据时，需要根据主键值去聚簇索引中再查找一遍完整的用户记录，这个过程叫做**回表**

为什么我们还需要一次回表操作呢？直接把完整的用户记录放到叶子节点不就好了么？你说的对，如果把完整的用户记录放到叶子节点是可以不用回表，但是太占地方了呀~相当于每建立一棵B+树都需要把所有的用户记录再都拷贝一遍，这就有点太浪费存储空间了。因为这种按照非主键列建立的B+树需要一次回表操作才可以定位到完整的用户记录，所以这种B+树也被称为二级索引（英文名secondary index），或者辅助索引。

联合索引

以多个列的大小为排序规则建立的B+树称为联合索引，本质上也是一个二级索引。

InnoDB的B+树索引的注意事项

根页面

我们前边介绍B+树索引的时候，为了大家理解上的方便，先把存储用户记录的叶子节点都画出来，然后接着画存储目录项记录的内节点，实际上B+树的形成过程是这样的：

1. 每当为某个表创建一个B+树索引（聚簇索引不是人为创建的，默认就有）的时候，都会为这个索引创建一个根节点页面。最开始表中没有数据的时候，每个B+树索引对应的根节点中既没有用户记录，也没有目录项记录。
2. 随后向表中插入用户记录时，先把用户记录存储到这个根节点中。
3. 当根节点中的可用空间用完时继续插入记录，此时会将根节点中的所有记录复制到一个新分配的页，比如页a中，然后对这个新页进行页分裂的操作，得到另一个新页，比如页b。这时新插入的记录根据键值（也就是聚簇索引中的主键值，二级索引中对应的索引列的值）的大小就会被分配到页a或者页b中，而根节点便升级为存储目录项记录的页。

这个过程需要大家特别注意的是：一个B+树索引的根节点自诞生之日起，便不会再移动。这样只要我们对某个表建立一个索引，那么它的根节点的页号便会被记录到某个地方，然后凡是InnoDB存储引擎需要用到这个索引的时候，都会从那个固定的地方取出根节点的页号，从而来访问这个索引。

内节点中目录项记录的唯一性

我们需要保证在B+树的同一层内节点的目录项记录除页号这个字段以外是唯一的。所以对于二级索引的内节点的目录项记录的内容实际上是由三个部分构成的：

- 索引列的值
- 主键值
- 页号

MyISAM中的索引方案简单介绍

InnoDB中索引即数据，也就是聚簇索引的那棵B+树的叶子节点中已经把所有完整的用户记录都包含了，而MyISAM的索引方案虽然也使用树形结构，但是却将索引和数据分开存储：

- 将表中的记录按照记录的插入顺序单独存储在一个文件中，称之为数据文件。这个文件并不划分为若干个数据页，有多少记录就往这个文件中塞多少记录就成了。我们可以通过行号而快速访问到一条记录。
- 使用MyISAM存储引擎的表会把索引信息另外存储到一个称为索引文件的另一个文件中。MyISAM会单独为表的主键创建一个索引，只不过在索引的叶子节点中存储的不是完整的用户记录，而是主键值 + 行号的组合。也就是先通过索引找到对应的行号，再通过行号去找对应的记录！这一点和InnoDB是完全不相同的，在InnoDB存储引擎中，我们只需要根据主键值对聚簇索引进行一次查找就能找到对应的记录，而在MyISAM中却需要进行一次回表操作，意味着MyISAM中建立的索引相当于全部都是二级索引！
- 如果有需要的话，我们也可以对其它的列分别建立索引或者建立联合索引，原理和InnoDB中的索引差不多，不过在叶子节点处存储的是相应的列 + 行号。这些索引也全部都是二级索引。

> 为什么不自动为每个列都建立个索引呢？别忘了，每建立一个索引都会建立一棵B+树，每插入一条记录都要维护各个记录、数据页的排序关系，这是很费性能和存储空间的。