聊一聊mysql的索引下推

之前参加部门培训,dba有讲到索引下推优化这个问题,一直似懂非懂,最近复习数据库知识,又学习了一遍,借机分享给大家.

1.什么是索引下推

以下是mysql官网描述:

Index Condition Pushdown (ICP) is an optimization for the case where MySQL retrieves rows from a table using an index. Without ICP, the storage engine traverses the index to locate rows in the base table and returns them to the MySQL server which evaluates the `WHERE` condition for the rows. With ICP enabled, and if parts of the `WHERE` condition can be evaluated by using only columns from the index, the MySQL server pushes this part of the `WHERE` condition down to the storage engine. The storage engine then evaluates the pushed index condition by using the index entry and only if this is satisfied is the row read from the table. ICP can reduce the number of times the storage engine must access the base table and the number of times the MySQL server must access the storage engine.

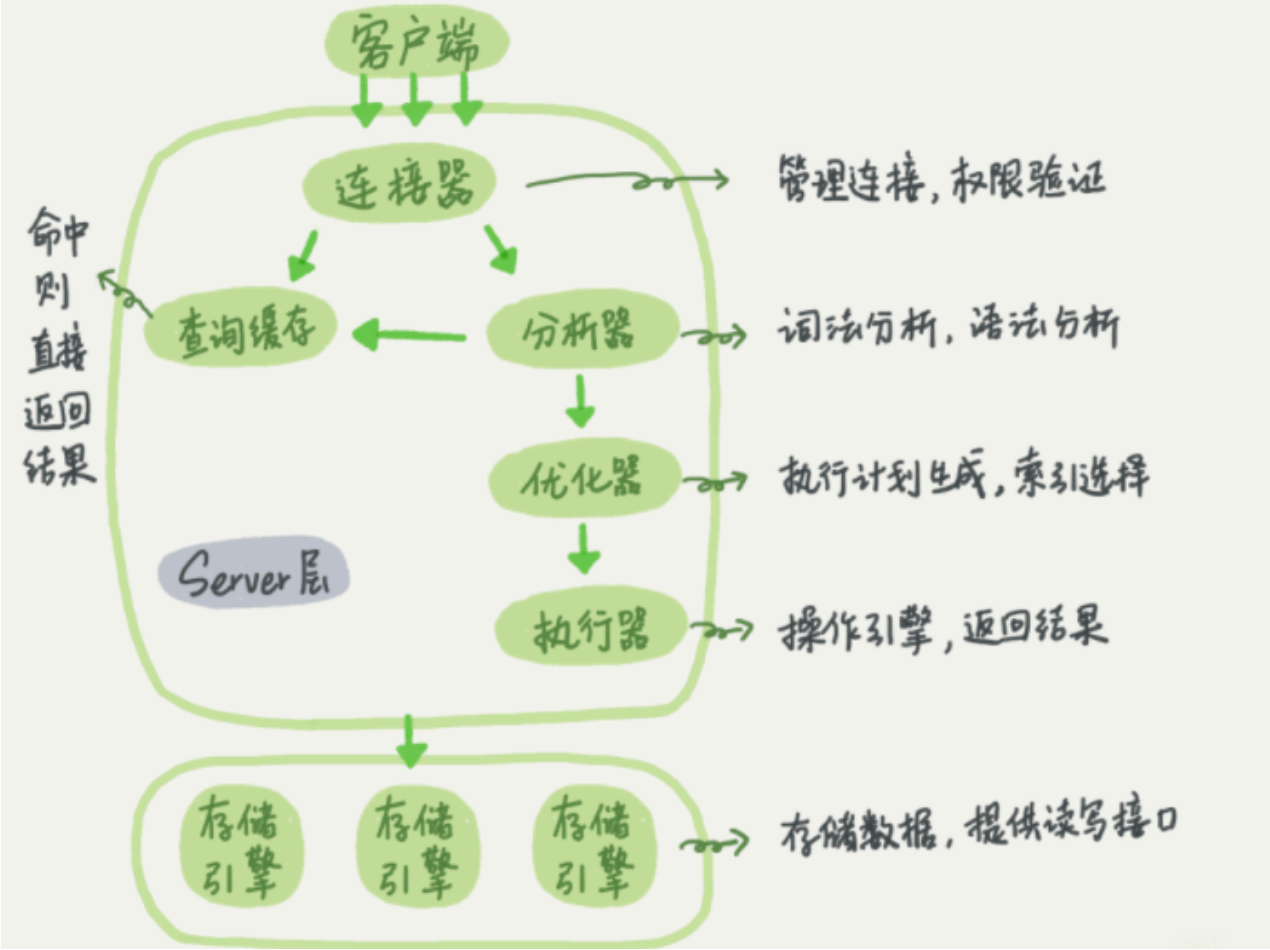
> 翻译大概意思:**索引条件下推（ICP）是针对MySQL使用索引从表中检索行的情况的一种优化。如果不使用ICP，则存储引擎将遍历索引以在基表中定位行，并将其返回给MySQL服务器，后者将评估`WHERE`行的条件。启用ICP后，如果`WHERE`可以仅使用索引中的列来评估部分条件，则MySQL服务器会将这部分条件压入`WHERE`条件下降到存储引擎。然后，存储引擎通过使用索引条目来评估推送的索引条件，并且只有在满足此条件的情况下，才从表中读取行。ICP可以减少存储引擎必须访问基表的次数以及MySQL服务器必须访问存储引擎的次数.**

其实单看这个描述让我感觉是似乎懂了,好像又没有懂.主要问题个人觉得大概是下面几点:

- **Mysql服务器**和**存储引擎**是负责做啥的?
- 什么是**基表**?
- **ICP**是具体是如何做到减少存储引擎访问基表的次数的?

2.Mysql服务器和存储引擎是作用

mysql的具体的存储模块不展开描述,直接看mysql的分层架构,更加清晰



大体来说，MySQL 可以分为 Server 层和存储引擎层两部分。

Server 层包括连接器、查询缓存、分析器、优化器、执行器等，涵盖 MySQL 的大多数核心服务功能，以及所有的内置函数（如日期、时间、数学和加密函数等），所有跨存储引擎的功能都在这一层实现，比如存储过程、触发器、视图等.

而存储引擎层负责数据的存储和提取。其架构模式是插件式的，支持 InnoDB、MyISAM、Memory 等多个存储引擎。现在最常用的存储引擎是 InnoDB，它从 MySQL 5.5.5 版本开始成为了默认存储引擎。而存储引擎再往下就是内存或者磁盘了.

这里只是粗略了解下,本文的重点也不是在讨论架构这一部分,每个模块细节不展开.

3.什么是基表

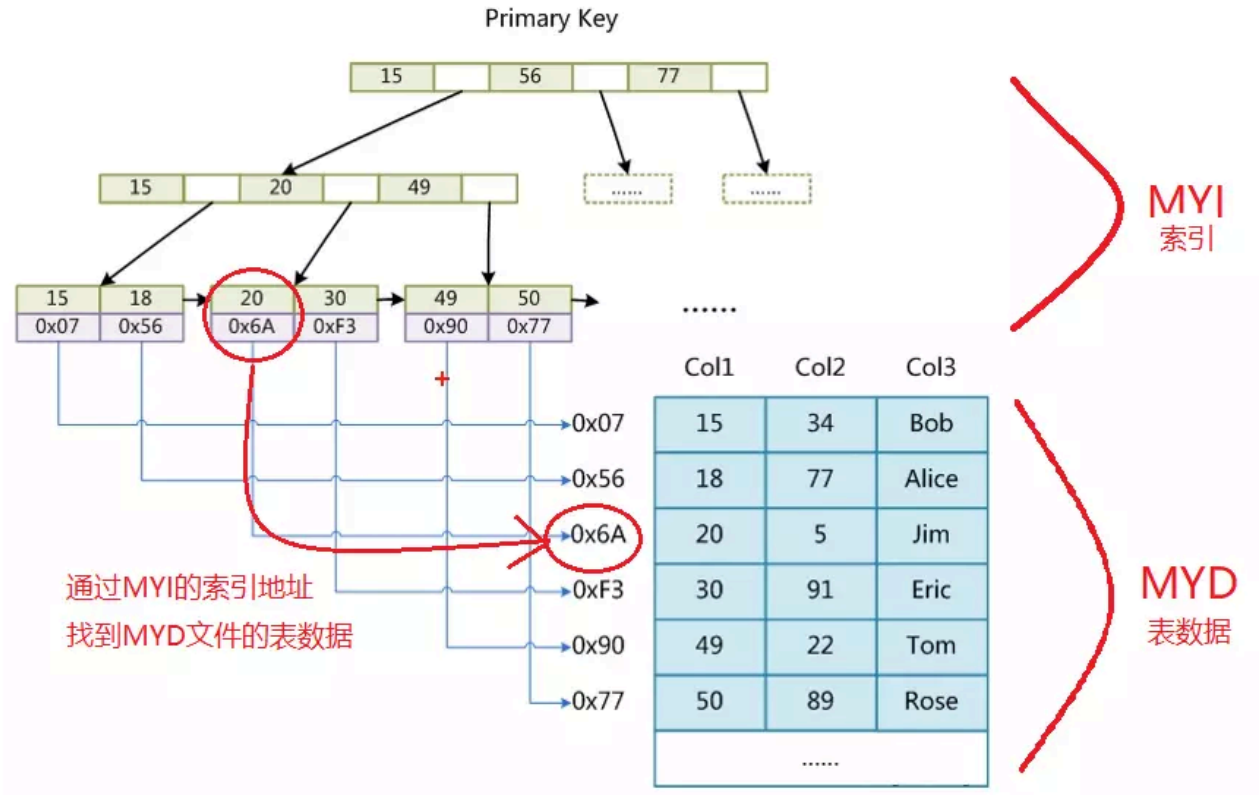
说到基表这里就必须要聊各个存储引擎的数据组织存储的方式了,直奔目的,我们直接来看最常用的InnoDB的数据组织方式.需要知道数据是怎么怎么存储的,那么我们就得看看它的数据存储文件,我们会发现InnoDB是由两个文件存储的(.frm和.ibd),在对比下MyISAM表是使用(.frm、.MYD、.MYI)三个文件存储的.

> .frm 是 MySQL 里面表结构定义的文件，不管你建表的时候选用任何一个存储引擎都会生成。

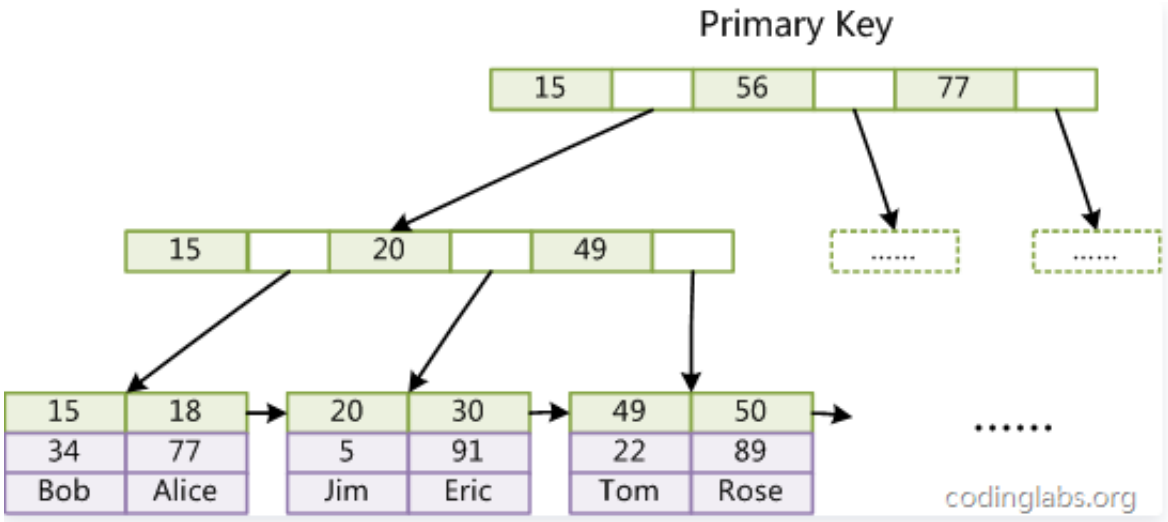
MyISAM 里面另外有两个文件：
一个是.MYD 文件，D 代表 Data，是 MyISAM 的数据文件，存放数据记录，比如我们的 user_myisam 表的所有的表数据。
一个是.MYI 文件，I 代表 Index，是 MyISAM 的索引文件，存放索引，比如我们在id 字段上面创建了一个主键索引，那么主键索引就是在这个索引文件里面。

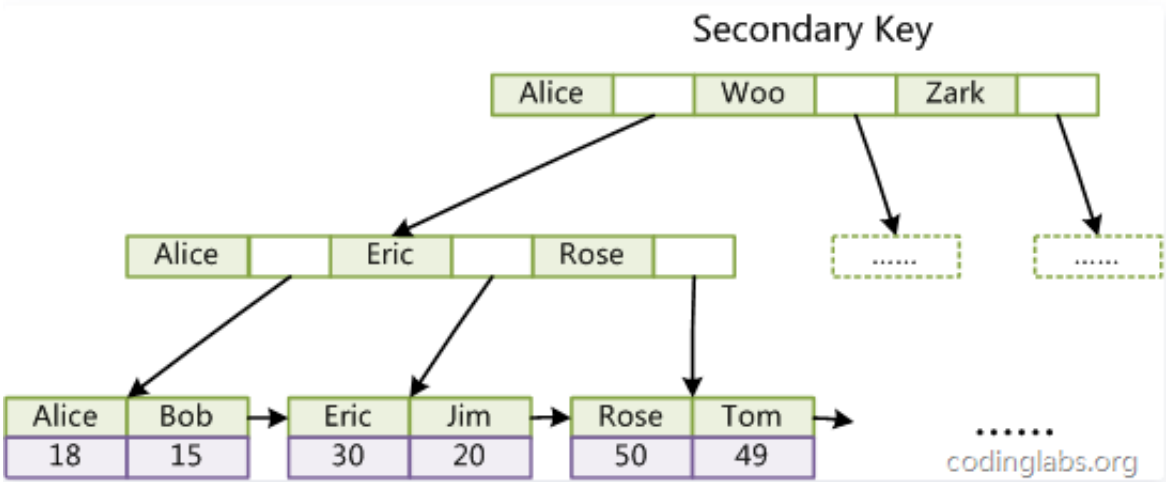
InnoDB 只有一个文件,这个文件存放了索引和数据

这两种引擎为什么会采用这两种存储方式?其实我们通过分析他们的索引存储就能得出结果了.



MyISAM索引图





InnoDB索引图

以上均为网上图片,借用一下,可以看出两个存储引擎都采用了B+树的存储方式,但是对比一下会发现MyISAM的索引和InnoDB还是有所不同的,主要是树的叶子节点上面.

> 主键索引:MyISAM的叶子节点上面存储的是数据在磁盘上面的地址,而InnoDB主键索引的上面的存储就是磁盘上面的数据
>
> 非主键索引:MyISAM的叶子节点上面存储的是数据在磁盘上面的地址,没有变化,而InnoDB非主键索引这边则产生了变化,非主键这边存储的就是主键的id,当我们通过二级索引查询数据的时候,InnoDB会在非主键索引上找到主键id,然后在到主键索引上查询到数据返回给服务器.

其实从主键索引中就可以看出之前为啥InnoDB的文件只有两个,因为它使用了索引来组织数据,所以索引和数据都存储在了.ibd文件里面.至于为啥InnoDB需要采用这种形式来组织数据,这里不深究(其实主键索引叶子节点上不只是数据,还包含主键值、事务ID、回滚指针等等),大家有兴趣继续深究.

如果InnoDB的表没有设置主键,那么它会如何组织数据呢?

- 如果我们定义了主键(PRIMARY KEY)，那么 InnoDB 会选择主键作为聚集索引。
- 如果没有显式定义主键，则 InnoDB 会选择第一个不包含有 NULL 值的唯一索引作为主键索引。
- 如果也没有这样的唯一索引，则 InnoDB 会选择内置 6 字节长的 ROWID 作为隐藏的聚集索引，它会随着行记录的写入而主键递增

这个id采用这个语句`select _rowid name from 表名称;`是可以查询到的.

> 注:这里的主键索引(聚簇索引,一级索引)和非主键索引(辅助索引,二级索引,非聚簇索引)有多种叫法,但是含义都这这个.

当innodb在查完二级索引后,这些被辅助索引命中的数据,由于这些数据只是存储了索引列数据和主键的id,当需要查询结果超出索引列的时候,则需要进行回表查询(就是拿着这些数据对应的主键id到一级索引中去查询数据),这个操作就是我们常说的**回表**,而主键索引所在的就是上面所说的**基表**.

其实,再看之前文中说`减少存储引擎访问基表的次数`这个操作不就是减少回表吗.

4.ICP是具体是如何做到减少存储引擎访问基表的次数的?

上面提到什么时候触发的回表操作,是在需要查询的数据超出索引列的时候,就是说当前如果我们查询的数据都是索引字段和主键,那么就不会触发回表操作,就是说在当前的辅助索引上就直接返回数据了.下面可以论证下是否正确:

```
CREATE TABLE `P_ATOM_INDEX` (
  `FATOM_INDEX_ID` varchar(32) NOT NULL COMMENT '原子指标id',
  `FATOM_INDEX_NAME` varchar(50) NOT NULL COMMENT '原子指标名称',
  `FATOM_INDEX_ENNAME` varchar(50) NOT NULL COMMENT '原子指标英文名称',
  `FATOM_INDEX_STATUS` int(11) NOT NULL COMMENT '状态',
  `FATOM_INDEX_SOURCE_TABLE` varchar(50) NOT NULL COMMENT '来源表',
  `FATOM_INDEX_SOURCE_FILED` varchar(50) NOT NULL COMMENT '来源字段',
  `FATOM_INDEX_TYPE` varchar(50) NOT NULL COMMENT '类型',
  `FATOM_INDEX_FORMAT` varchar(50) NOT NULL COMMENT '格式',
  `FATOM_INDEX_ARED_ID` varchar(50) NOT NULL COMMENT '数据域id',
  `FATOM_INDEX_DESC` text COMMENT '描述',
  `FATOM_INDEX_CAL_TYPE` int(11) NOT NULL COMMENT '计算类型',
  `FATOM_INDEX_CAL_SCRIPT` text NOT NULL COMMENT '计算表达式',
  `FATOM_INDEX_LAST_MODIFYBY` varchar(50) NOT NULL,
  `FATOM_INDEX_LAST_MODIFY_TIME` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `FATOM_INDEX_CREATEBY` varchar(50) DEFAULT NULL COMMENT '创建人',
  `FATOM_INDEX_CREATE_TIME` timestamp NULL DEFAULT NULL COMMENT '创建时间',
  `FLAST_MODIFYBY_ID` varchar(50) DEFAULT NULL,
  `FCREATEBY_ID` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`FATOM_INDEX_ID`) USING BTREE,
  KEY `FATOM_INDEX_NAME_INDEX` (`FATOM_INDEX_NAME`,`FATOM_INDEX_ENNAME`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC;
```

这是一个建表语句,可以看到我在`FATOM_INDEX_NAME`,`FATOM_INDEX_ENNAME`两个字段上面建了一个索引,下面我们来查询数据

```
EXPLAIN SELECT * FROM P_ATOM_INDEX WHERE FATOM_INDEX_NAME='用户访问量';
```

信息 | Result 1 | 概况 | 状态

	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
M_INDEX	(NULL)	ref	FATOM_INDEX_NAME_INDEX	FATOM_INDEX_NAME_INDEX	152	const	1	100.00	(NULL)

```
EXPLAIN SELECT FATOM_INDEX_NAME,FATOM_INDEX_ID FROM P_ATOM_INDEX WHERE FATOM_INDEX_NAME='用户访问量';
```

信息 | Result 1 | 概况 | 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	P_ATOM_INDEX	(NULL)	ref	FATOM_INDEX_NAME_INDEX	FATOM_INDEX_NAME	152	const	1	100.00	Using index

以上对比发现,第二次操作的字段不用到数据区在进行查找,就是不需要进行回表操作,这个Using index则代表着使用到了**覆盖索引**.大家都知道多一次查找就会多一次磁盘io,使用了覆盖索引就会大大减少回表操作带来的io,会大大节约资源.所以dba一般也会建议我们多建立联合索引,也是有这个一部分原因的(还有前缀索引等原因),当然这个也不是绝对的.

这里说到了在使用覆盖索引时减少了回表的操作,那么索引下推是如何减少回表的呢?

这是上面那张表的数据

FATOM_INDEX_SOURCE_FILED	FATOM_INDEX_TYPE	FATOM_INDEX_FORMAT	FATOM_INDEX_ARED_ID	FATOM_INDEX_DESC	FATOM_INDEX_CAL_TYPE	FATOM_INDEX_CAL_SCR
user_id	1		8a81828c741e734b01741f4c84da000c	用户访问量	0	count(f_user_id)
ORDER_ID	1		8a81828c7424dbeb0174250ab8e60004	订单的数量	0	count(order_id)
AMOUNT	4		8a81828c7424dbeb0174250ab8e60004	订单金额	0	sum(amount)
sd	2		ff80808174ba362a0174ba370de80000		0	dsadas

下面是开启索引下推的状况下:

下图是关掉索引下推后的状态:

从官网上可以知道:

> `EXPLAIN` output shows `Using index condition` in the `Extra` column when Index Condition Pushdown is used. It does not show `Using index` because that does not apply when full table rows must be read.

说明第一张图示使用到了索引下推的,那么这中间是怎么样的一个过程呢,关掉和没关掉区别在哪呢?

从数据图中可以看出,查询数据时候,由于like这条件是满足最左前缀原则的，最左前缀可以用于在索引中定位记录,但是其实是有满足这个条件的是有两条记录的,当没有索引下推的时候,索引并不会使用后面一个条件来进行过滤,这两条数据会一条一条的进行回表操作,然后将查出数据返回到server层,再由server将后面的条件过滤掉,最终返回一条数据给客户端.

当开启了ICP时候,在索引内部就会进行过滤操作,最终将一条满足条件的数据进行回表操作,得到数据返回给server层,如此看来确实得到了大大的优化.

> 注意:索引下推是在mysql5.6之后引入的机制,同时我们可以使用`show VARIABLES LIKE 'optimizer_switch'`;来查看是否开启了索引下推,
>
> 同时也可以使用`SET optimizer_switch = 'index_condition_pushdown=on';`来设置索引下推
>
> ICP在InnoDB中只能使用于二级索引,这个可以推测

关于ICP更多的使用场景官网上有更加详细的描述:https://dev.mysql.com/doc/refman/5.7/en/index-condition-pushdown-optimization.html

关于在平常开发中是否有必要关注这个,我认为是十分有必要的,我们这边已经有通过触发ICP来实现优化的sql案例,节约了数据库资源.具体案例:参考培训ppt