

3、加分项——MyIsam 和 InnoDB 引擎的调研

3.1、引擎概论

3.1.1 定义：什么是数据引擎

数据库引擎是用于存储、处理和保护数据的核心服务。利用数据库引擎可控制访问权限并快速处理事务，从而满足企业内大多数需要处理大量数据的应用程序的要求。使用数据库引擎创建用于联机事务处理或联机分析处理数据的关系数据库。这包括创建用于存储数据的表和用于查看、管理和保护数据库安全的数据库对象（如索引、视图和存储过程）。

3.1.2 存储引擎作用

- （1）设计并创建数据库以保存系统所需的关系或 XML 文档。
- （2）实现系统以访问和更改数据库中存储的数据。包括实现网站或使用数据的应用程序，还包括生成使用 SQL Server 工具和实用工具以使用数据的过程。
- （3）为单位或客户部署实现的系统。
- （4）提供日常管理支持以优化数据库的性能。

3.1.3、为什么要合理选择数据库存储引擎：

MySQL 中的数据用各种不同的技术存储在文件（或者内存）中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力。通过因地制宜选择不同的技术，针对不同的需求选择不同的引擎，就能够获得额外的速度或者功能，从而改善应用的整体功能

3.2、MySQL 各大存储引擎：

存储引擎主要有：1. MyIsam , 2. InnoDB, 3. Memory, 4. Blackhole, 5. CSV, 6. Performance_Schema, 7. Archive, 8. Federated , 9 Mrg_Myisam

但是我们在项目过程中只遇到了 MyIsam 和 InnoDB，所以主要分析这两者。

3.2.1 InnoDB：

- （1）定义：（默认的存储引擎）

InnoDB 是一个事务型的存储引擎，有行级锁定和外键约束。

InnoDB 引擎提供了对数据库 ACID 事务的支持，并且实现了 SQL 标准的四种隔离级别。该引擎还提供了行级锁和外键约束，它的设计目标是处理大容量数据库系统，它本身其实就是基于 MySQL 后台的完整数据库系统，MySQL 运行时 InnoDB 会在内存中建立缓冲池，用于缓冲数据和索引。但是该引擎不支持 FULLTEXT 类型的索引，而且它没有保存表的行数，当 SELECT COUNT(*) FROM TABLE 时需要扫描全表。当需要使用数据库事务时，该引擎当然是首选。由于锁的粒度更小，写操作不会锁定全表，所以在并发较高时，使用 InnoDB 引擎会提升效率。但是使用行级锁也不是绝对的，如果在执行一个 SQL 语句时 MySQL 不能确定要扫

描的范围，InnoDB 表同样会锁全表。

(2) MySQL 官方对 InnoDB 的讲解：

①InnoDB 给 MySQL 提供了具有提交、回滚和崩溃恢复能力的事务安全（ACID 兼容）存储引擎。

②InnoDB 锁定在行级并且也在 SELECT 语句提供一个 Oracle 风格一致的非锁定读，这些特色增加了多用户部署和性能。没有在 InnoDB 中扩大锁定的需要，因为在 InnoDB 中行级锁定适合非常小的空间。

③InnoDB 也支持 FOREIGN KEY 强制。在 SQL 查询中，你可以自由地将 InnoDB 类型的表与其它 MySQL 的表的类型混合起来，甚至在同一个查询中也可以混合。

④InnoDB 是为处理巨大数据量时的最大性能设计，它的 CPU 效率可能是任何其它基于磁盘的关系数据库引擎所不能匹敌的。

⑤ InnoDB 被用来在众多需要高性能的大型数据库站点上产生。

(3) 适用场景

所以综合上述文档的信息，我们认为 InnoDB 的适用场景有：

①经常更新的表，适合处理多重并发的更新请求。比如本项目中的近期的预约这样的热数据（需要更改预约状态）。

②支持事务。

③可以从严重错误中恢复（通过 bin-log 日志等）。

④外键约束。只有他支持外键。

⑤支持自动增加列属性 auto_increment。

3.2.2 MyISAM:

(1) 定义：

MyISAM 是 MySQL 默认的引擎（可能版本不同导致差异），但是它没有提供对数据库事务的支持，也不支持行级锁和外键，因此当 INSERT(插入)或 UPDATE(更新)数据时即写操作需要锁定整个表，效率便会低一些。

引擎在创建表的时候，会创建三个文件，一个是.frm 文件用于存储表的定义，一个是.MYD 文件用于存储表的数据，另一个是.MYI 文件，存储的是索引。操作系统对大文件的操作是比较慢的，这样将表分为三个文件，那么.MYD 这个文件单独来存放数据自然可以优化数据库的查询等操作。有索引管理和字段管理。MyISAM 还使用一种表格锁定的机制，来优化多个并发的读写操作，其代价是你需要经常运行 OPTIMIZE TABLE 命令，来恢复被更新机制所浪费的空间。

(2) 特性：

ISAM 执行读取操作的速度很快，而且不占用大量的内存和存储资源。

在设计之初就预想数据组织成有固定长度的记录，按顺序存储的。—ISAM 是一种静态索引结构。

(3) 缺点：

①它不支持事务处理

②也不能够容错。如果你的硬盘崩溃了，那么数据文件就无法恢复了。如果你正在把 ISAM 用在关键任务应用程序里，那就必须经常备份你所有的实时数据，通过其复制特性，MYSQL 能够支持这样的备份应用程序。

由于这些特性的不同，MYISAM 和 InnoDB 在适用场景上也不一样。

(4) 适用场景：

①不支持事务的设计（但是并不代表着有事务操作的项目不能用 MyIsam 存储引擎，可以在 service 层进行根据自己的业务需求进行相应的控制）。

②不支持外键的表设计。

③查询速度很快，如果数据库 insert 和 update 的操作比较多的话比较适用。

④对表进行加锁的场景。

⑤MyISAM 极度强调快速读取操作。

⑥MyIASM 中存储了表的行数，于是 SELECT COUNT(*) FROM TABLE 时只需要直接读取已经保存好的值而不需要进行全表扫描。如果表的读操作远远多于写操作且不需要数据库事务的支持，那么 MyIASM 也是很好的选择。（比如在预约拿号的时候）

3.2.3 InnoDB 和 MyIsam 使用及其原理对比：

(1) 使用的效果与区别展示：


MySQL 数据库实战例子（存储引擎、视图、锁机制、分表）

①在一个普通数据库中创建两张分别以 MyIsam 和 InnoDB 作为存储引擎的表。

```
1 create table testMyIsam(  
2   id int unsigned primary key auto_increment,  
3   name varchar(20) not null  
4   )engine=myisam;  
  
1 create table testInnoDB(  
2   id int unsigned primary key auto_increment,  
3   name varchar(20) not null  
4   )engine=innodb;
```

创建空表后的区别如下：

	testinnodb	索引长度
	表	0 bytes (0)
		数据长度
		16.00 KB (16,384)
行		最大数据长度
0		0 bytes (0)
引擎		数据可用空间
InnoDB		0 bytes (0)
自动递增		排序规则
1		utf8_general_ci

	testmyisam	索引长度	1.00 KB (1,024)
	表	数据长度	0 bytes (0)
行	0	最大数据长度	256.00 TB (281,474,976,710,655)
引擎	MyISAM	数据可用空间	0 bytes (0)
自动递增	1	排序规则	utf8_general_ci

②对比插入效率（百万级插入）：

为了更好地对比，我们可以使用函数的方式或者存储过程的方式。

创建储存过程

```

1 delimiter $$
2 drop procedure if exists ptestmyisam;
3 create procedure ptestmyisam()
4 begin
5 declare pid int ;
6 set pid = 1000000;
7 while pid>0
8 do
9 insert into testmyisam(name) values(concat("fuzhu", pid));
10 set pid = pid-1;
11 end while;
12 end $$

```

```

1 delimiter $$
2 drop procedure if exists ptestInndb;
3 create procedure ptestInndb()
4 begin
5 declare pid int ;
6 set pid = 1000000;
7 while pid>0
8 do
9 insert into testinnodb(name) values(concat("fuzhu", pid));
10 set pid = pid-1;
11 end while;
12 end $$

```

运行时间同为 1 分钟，innodb 只插入了 800 条。

COUNT(*)

800

而 Myisam 是接近两倍的速度。

信息 结果 1 剖析 状态

COUNT(*)

1538

而如果我们把事务关闭，那就会很快。

```

3  set autocommit = 0;
4
5  call ptestInndb();
6
7  set autocommit = 1;
8

```

修改数据之后插入十万条只需要 5.8s

```

COUNT(*)
└─ 100000

set autocommit = 0
> OK
> 时间: 0s

call ptestInndb()
> OK
> 时间: 5.861s

set autocommit = 1
> OK
> 时间: 0.346s

```

百万条也只用 84s

```

COUNT(*)
└─ 1000000

set autocommit = 0
> OK
> 时间: 0s

call ptestInndb()
> OK
> 时间: 84.73s

set autocommit = 1
> OK
> 时间: 4.545s

SELECT COUNT(*) FROM testinnodb
> OK
> 时间: 0.127s

```

而 MyISAM 不支持事务，故不用实验。

但是可以用插入的方法快速地写入 mysiam

```

#DELETE FROM `vote_records` WHERE id>0;
INSERT INTO testmysiam SELECT * FROM `testinnodb`
> Affected rows: 1000000
> 时间: 5.18s

```

虽然看上去速度上 MyISAM 快，但是增删改是涉及事务安全的，所以用 InnoDB 相对好很多。

③对比更新：

```
update testinnodb set name = 'fuzhu3' where id>0 and id<2074414
> Affected rows: 9999
> 时间: 1.651s

update testmysam set name = 'fuzhu3' where id>0 and id<2074414
> Affected rows: 9999
> 时间: 0.188s
```

还是 Myisam 会快一些

④查询对比：

1) 查询总数目

```
SELECT count(*) FROM testinnodb
> OK
> 时间: 0.213s

SELECT count(*) FROM testmysam
> OK
> 时间: 0s
```

Myisam 几乎不花时间

2) 查询无索引的列：

```
select * from testMyIsam where name > "fuzhu100"
> OK
> 时间: 1.984s
```

```
select * from testInnoDB where name > "fuzhu100"
> OK
> 时间: 2.186s
```

3) 查询有索引的列：

```
select * from testMyIsam where id > 10
> OK
> 时间: 2.713s
```

```
select * from testinnodb where id > 10
> OK
> 时间: 2.91s
```

时间都差不多

4) 存储大小：

	testinnodb	索引长度	0 bytes (0)
表		数据长度	368.00 KB (376,832)
行	0	最大数据长度	0 bytes (0)
引擎	InnoDB	数据可用空间	40.00 MB (41,943,040)
自动递增	2,064,415		

	testmysam	索引长度	203.00 KB (207,872)
表		数据长度	390.65 KB (400,024)
行	0	最大数据长度	256.00 TB (281,474,976,710,655)
引擎	MyISAM	数据可用空间	390.65 KB (400,024)
自动递增	2,064,415		

（2）效果对比总述：

①事务。**MyISAM** 类型不支持事务处理等高级处理，而 **InnoDB** 类型支持，提供事务支持已经外部键等高级数据库功能。

InnoDB 表的行锁也不是绝对的，假如在执行一个 SQL 语句时 **MySQL** 不能确定要扫描的范围，**InnoDB** 表同样会锁全表，例如 `updatetable set num=1 where name like "a%"`

就是说不确定的范围时，**InnoDB** 还是会锁表的。

②性能主题。**MyISAM** 类型的表强调的是性能，其执行速度比 **InnoDB** 类型更快。

③行数保存。**InnoDB** 中不保存表的具体行数，也就是说，执行 `select count() fromtable` 时，**InnoDB** 要扫描一遍整个表来计算有多少行，但是 **MyISAM** 只要简单的读出保存好的行数即可。注意的是，当 `count()` 语句包含 `where` 条件时，两种表的操作是一样的。

④索引存储。对于 **AUTO_INCREMENT** 类型的字段，**InnoDB** 中必须包含只有该字段的索引，但是在 **MyISAM** 表中，可以和其他字段一起建立联合索引。

MyISAM 支持全文索引（**FULLTEXT**）、压缩索引，**InnoDB** 不支持

⑤服务器数据备份。**InnoDB** 必须导出 SQL 来备份，`LOAD TABLE FROM MASTER` 操作对 **InnoDB** 是不起作用的，解决方法是首先把 **InnoDB** 表改成 **MyISAM** 表，导入数据后再改成 **InnoDB** 表，但是对于使用的额外的 **InnoDB** 特性(例如外键)的表不适用。

而且 **MyISAM** 应对错误编码导致的数据恢复速度快。**MyISAM** 的数据是以文件的形式存储，所以在跨平台的数据转移中会很方便。在备份和恢复时可单独针对某个表进行操作。

（3）使用建议：

以下两点必须使用 **InnoDB**：

①可靠性高或者要求事务处理，则使用 **InnoDB**。这个是必须的。

②表更新和查询都相当的频繁，并且表锁定的机会比较大的情况指定 InnoDB 数据引擎的创建。

对比之下，MyISAM 的使用场景：

- ①做很多 count 的计算的。如一些日志，调查的业务表以及该项目中的预约取号。
- ②插入修改不频繁，查询非常频繁的。

MySQL 能够允许你在表这一层应用数据库引擎，所以你可以只对需要事务处理的表格来进行性能优化，而把不需要事务处理的表格交给更加轻便的 MyISAM 引擎。对于 MySQL 而言，灵活性才是关键。

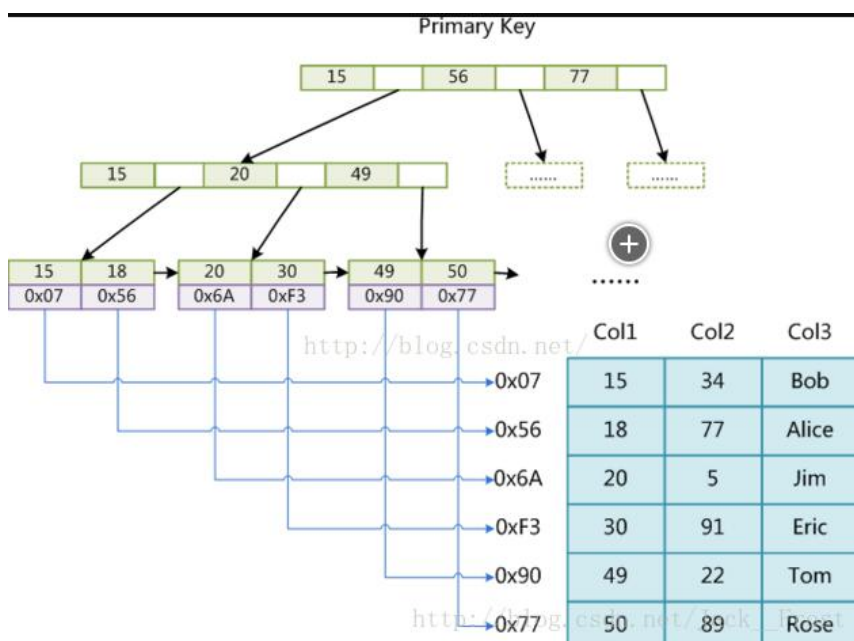
3.2.4 InnoDB 和 MyISAM 引擎原理：

在此之前，先去理解什么是聚簇和非聚簇索引。

(1) MyISAM 引擎的索引结构：

MyISAM 索引结构: MyISAM 索引用的 B+ tree 来储存数据, MyISAM 索引的指针指向的是键值的地址，地址存储的是数据。

B+Tree 的数据域存储的内容为实际数据的地址，也就是说它的索引和实际的数据是分开的，只不过是索引指向了实际的数据，这种索引就是所谓的非聚集索引。



因此，过程为：MyISAM 中索引检索的算法为首先按照 B+Tree 搜索算法搜索索引，如果指定的 Key 存在，则取出其 data 域的值，然后以 data 域的值作为地址，根据 data 域的值去读取相应数据记录。

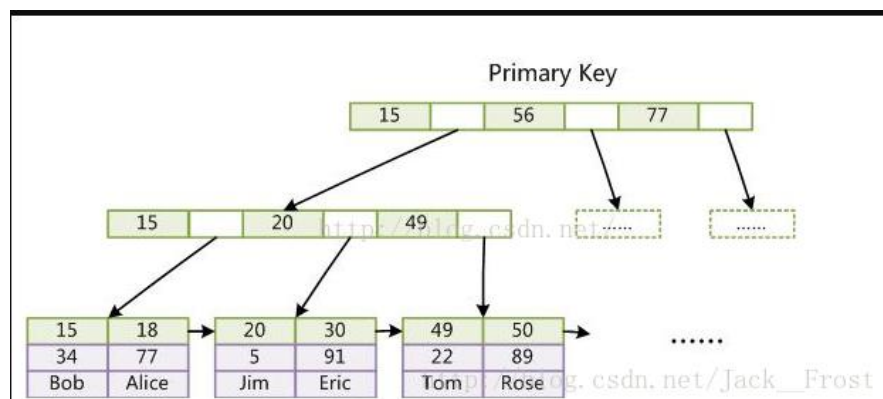
(2) InnoDB 引擎的索引结构：

也是 B+Tree 索引结构。InnoDB 的索引文件本身就是数据文件，即 B+Tree 的数据域存储的就是实际的数据，这种索引就是聚集索引。这个索引的 key 就是数据表的主键，因此 InnoDB 表数据文件本身就是主索引。

InnoDB 的辅助索引数据域存储的也是相应记录主键的值而不是地址，所以当以辅助索

引查找时，会先根据辅助索引找到主键，再根据主键索引找到实际的数据。所以 InnoDB 不建议使用过长的主键，否则会使辅助索引变得过大。

建议使用自增的字段作为主键，这样 B+Tree 的每一个结点都会被顺序的填满，而不会频繁的分裂调整，会有有效的提升插入数据的效率。



上图，可以看到叶节点包含了完整的数据记录。这种索引叫做聚集索引。因为 InnoDB 的数据文件本身要按主键聚集，所以 InnoDB 要求表必须有主键（MyISAM 可以没有），如果没有显式指定，则 MySQL 系统会自动选择一个可以唯一标识数据记录的列作为主键，如果不存在这种列，则 MySQL 自动为 InnoDB 表生成一个隐含字段作为主键，这个字段长度为 6 个字节，类型为长整形。

而且，与 MyISAM 索引的不同是 InnoDB 的辅助索引 data 域存储相应记录主键的值而不是地址。换句话说，InnoDB 的所有辅助索引都引用主键作为 data 域。

因此，过程为：将主键组织到一棵 B+ 树中，而行数据就储存在叶子节点上，若使用 "where id = 13" 这样的条件查找主键，则按照 B+ 树的检索算法即可查找到对应的叶节点，之后获得行数据。若对 Name 列进行条件搜索，则需要两个步骤：第一步在辅助索引 B+ 树中检索 Name，到达其叶子节点获取对应的主键。第二步使用主键在主索引 B+ 树种再执行一次 B+ 树检索操作，最终到达叶子节点即可获取整行数据。

参考链接：

<https://www.cnblogs.com/sunsky303/p/8274586.html>

<https://www.cnblogs.com/tufujie/p/9413852.html>