

MySQL逻辑架构和存储引擎

1、MySQL逻辑架构

大体来说，MySQL可以分为 **Server 层**和**存储引擎层**。

Server 层包括连接器、查询缓存、解析器、优化器和执行器等，涵盖了 MySQL 大多数核心服务功能。

存储引擎完成数据的存储和提取，它负责和文件系统打交道。

MySQL 的存储引擎是插件式的。不同的存储引擎支持不同的特性。

选择合适的存储引擎对应用非常重要。

2、常用存储引擎

2.1、MyISAM存储引擎

特点：

1. 查询速度比较快
2. **支持表锁**
3. 支持全文索引
4. **不支持事务**
5. 有三个文件保存一张表，.frm保存表结构，.myi保存索引，.myd保存数据，像这样的数据和索引分开存放的叫**非聚集索引**。

2.2、InnoDB存储引擎

1. 5.5以及之后的版本默认的存储引擎
2. **支持事务**
3. **支持表锁和行锁的**
4. 支持MVCC
5. 支持外键约束
6. 有两个文件保存一整张表，.frm保存表结构，.ibd文件保存索引和数据，索引和数据是保存在一起的，这样的叫**聚集索引**。

2.3、Memory存储引擎

1. 数据保存在内存中，数据库重启之后，数据消失。
2. 支持表锁。
3. 用temporary关键字可以创建临时表，这样的临时表只是在当前的连接中看到。
4. 用memory创建的表，在不同的连接中都可以看到。

3、MySQL中的锁

3.1、锁的概念

使用锁可以对有限的资源进行保护，解决隔离和并发的矛盾

3.2、锁的分类

按照对数据的锁定范围划分：

1. 行级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。
2. 表级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

按照对数据的访问类型划分：

1. 读锁（共享锁）：同一份数据，多个读操作可以同时进行而互不影响。
2. 写锁（排它锁）：当前操作没有完成之前，它会阻断其他读锁和写锁。

3.3、MyISAM表锁

1. 如何加锁？

1. 加读锁：lock table 表名 read
2. 加写锁：lock table 表名 write
3. 解锁：unlock table;

2. 加锁之后的效果？

1. 当前会话加了读锁，可以读加锁的这张表，其他会话更新这张表时（写数据时），写操作会阻塞，直到解锁。

```
mysql>
mysql> select * from class;
+----+-----+-----+-----+
| ID | name  | gender | age |
+----+-----+-----+-----+
| 3  | xiaoming | male  | 24  |
| 2  | xiaoli  | male  | 23  |
+----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
mysql> lock table class read;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql>
mysql>
```

第一个会话加了读锁

```
mysql>
mysql>
mysql>
mysql> select * from class;
+----+-----+-----+-----+
| ID | name  | gender | age |
+----+-----+-----+-----+
| 3  | xiaoming | male  | 24  |
| 2  | xiaoli  | male  | 23  |
+----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
mysql> insert into class (ID,name,gender,age) values (
5,'linghuchong',male,25);
```

第二个会话里可以读数据，但是修改数据会阻塞

2. 一个会话对一张表加了锁之后，只能访问这张表，不能访问其他表。一心不能二用：只能对一张表加锁，只能访问加了锁的表，没有加锁的表不能访问。

```
mysql>
mysql> lock table class read;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql>
mysql>
mysql> select * from member;
ERROR 1100 (HY000): Table 'member' was not locked with LOCK TABLES

mysql>
mysql>
mysql>
```

3. 一个会话对一张表加了读锁，只能在这张表中读数据，不能写数据。

```
mysql> insert into class (ID,name,gender,age) values (7,'duanyu',male,23);
ERROR 1054 (42S22): Unknown column 'male' in 'field list'
mysql>
```

3. 读锁写锁总结：读锁会阻塞写，但是不会堵塞读。而写锁则会把读和写都堵塞。

```
mysql> show open tables;
+-----+-----+
| performance_schema | variables_by_thread |
| mysql              | gtid_executed       |
| mysql              | time_zone_transition_type |
| performance_schema | hosts               |
| test               | int_type            |
+-----+-----+
129 rows in set (0.00 sec)
```

左边加了读锁，右边会话可以再加读锁，但是不能再加写锁，加写锁会阻塞

```
mysql> lock table member read;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> lock table member read;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> lock table member write;
```

4. 查看锁的争用情况：show open tables;

in_use表示对这张表加锁的进程数

```
mysql> show open tables;
+-----+-----+-----+-----+
| Database | Table | In_use | Name_locked |
+-----+-----+-----+-----+
| performance_schema | events_waits_summary_by_thread_by_event_name | 0 | 0 |
| performance_schema | events_waits_summary_global_by_event_name | 0 | 0 |
+-----+-----+-----+-----+
```

3.4、InnoDB行锁

1. 行锁的特点，只对一行加锁，如果访问同一行，会冲突。

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)
```

左右两个会话在事务中修改了同一行数据，第二个写操作会阻塞

```
mysql> update order_table set price=7000 where order_id=2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1 Changed: 0 Warnings: 0
```

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> update order_table set price=8000 where order_id=2;
```

2. 如果访问不同行数据，不会冲突，写操作不阻塞。

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> update order_table set price=7000 where order_id=2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1 Changed: 0 Warnings: 0
```

左右两边的会话在各自的事务中修改不同行的数据，写操作不冲突，SQL的执行不会阻塞

```
mysql> update order_table set price=8000 where order_id=4;
Query OK, 0 rows affected (0.01 sec)
Rows matched: 1 Changed: 0 Warnings: 0
```

3. 加读锁: `SELECT * from lock in share mode;`
4. 加写锁: `SELECT for update;`
5. 间隙锁: 当我们查找的数据是一个范围, 可能会有符合查询条件, 但是这条数据并不存在, 加入另外一个进程插入了符合查询条件的数据, 在某些场景下, 会很大的影响效率。

4、业务设计

4.1、逻辑设计

范式设计:

1. 第一范式: 表中的每一个字段原子性不可再分。
2. 第二范式: 所有非主键字段完全依赖主键, 不能产生部分依赖。
如果想设计出满足第二范式的表, 可以用这样的思路: 多对多: 三张表, 关系表中用外键建立另外两张表的主键间的。
3. 第三范式: 第三范式需要确保数据表中的所有非主键字段直接依赖主键, 不能产生传递依赖。
4. 范式设计总结: 建立**冗余较小**、结构合理的数据库, 需要满足一定的规范。
 1. 优点:
可以尽量减少数据的冗余, 符合范式设计的表更小, 更新数据时速度也会更快。
 2. 缺点:
 1. 范式化的表, 在查询的时候经常需要很多join关联, 增加让查询的代价。
 2. 不容易做索引优化。

反范式设计

1. 概念：允许存在少量冗余，换句话说反范式化就是使用空间来换取时间
2. 优点：可以减少表的关联，提高效率，能够更好地做索引优化
3. 缺点：存在数据冗余，使得数据的维护和修改成本更高。
4. 总结：三大范式只是一般设计数据库的基本理念，不能一味的去追求范式建立数据库，在实际应用中经常需要混用，可能使用部分范式化的技巧。

4.2、物理设计

1. 定义数据库、表及字段的命名规范
2. 选择合适的存储引擎：根据不同存储引擎的特性，比如要求事务和并发，选择InnoDB。
3. 选择合适的数据类型，优先选择整数型

5、索引

聚集索引：

1. 数据跟索引保存在一个文件中，InnoDB存储引擎的表都是采用这种结构。
2. 聚集索引中叶子节点保存key值和一整行完整的数据。
3. InnoDB的表中，有且只有一个聚集索引。

辅助索引：

1. 叶子节点上面保存的是key值和对应的主键ID。

回表：当通过辅助索引来寻找数据的时候，InnoDB会遍历辅助索引，并通过辅助索引的叶子节点，获取主键。然后再通过聚集索引来找到一个完整的行记录。这个过程我们称之为回表。

```
select name from citizen where name='xiaoming';
```

这条语句执行时，加入name字段建立了索引，先到name字段的B树上面查找。

假如执行的是

```
select * from citizen where name='xiaoming';
```

这条语句执行时，由于保存name的B树上面只保存了name和主键，没有其他字段，要查找其他字段，拿到name='xiaoming'的主键值，用这个主键值到聚集索引上再查找一次，这个过程就是回表。

联合索引:联合索引是指对表的多个列进行索引。

最左前缀原则

我们利用一张公民表来分析这个问题。建立 (age,name)来分析这个问题 age, age和name联合的

```
select * from citizen where name = xxx and age =xxx;
```

#符合

```
select * from citizen where name = xxx; #不符合
```

```
select * from citizen where age = xxx; #符合
```

```
select * from citizen where name like 'ab%'; #不符合
```

原因，原因是用的查找条件是name字段

```
select * from citizen where name like '%ab'; #不符合
```

如果是建立了 (name, age)的索引

```
select * from citizen where name like 'ab%'; #符合，
```

原因是用的查找条件是name字段，同时字符串也是从左开始匹配的

```
select * from citizen where name like '%ab'; #不符合
```

原因，原因是字符串是从中间的ab匹配的

索引覆盖：InnoDB 存储引擎支持覆盖索引 (covering index), 即从辅助索引中就可以得到要查询的信息，而不需要回表。