## 第一章：mysql架构与历史

### 1.锁粒度：

表级锁：性能开销小

行级锁：性能开销大，最大程度支持并发

### 2.事务：ACID

原子性：事务是最小工作单元，不可分割

一致性：事务总是从一个一致性状态转换到另一个一致性的状态

隔离性：事务之间互不影响

持久性：一旦事务提交，就会永久保存在数据库

### 3.隔离级别：

未提交读：一个事务读到了另一个事务未提交的数据，脏读，不可重复读，幻读

提交读：一个事务读取到了另一个事务提交的数据，不可重复读，幻读

可重复读：事务多次读取得到一致的数据，幻读。Mysql默认隔离级别

可串行化：强制事务串行执行，最高隔离级别，读取的每一行数据都加锁，耗费性能。

### 4.死锁：

两个或以上事务争夺资源造成相互等待。只有部分或者完全回滚一个事务才能解决死锁。InnoDB存储引擎处理死锁的方式是将持有最少行级锁的事务进行回滚，使影响最小。

### 5.Mysql中的事务：

Mysql默认自动提交模式。Show variables like ‘autocommit’;可以查询当前模式，1或者ON表示弃用，0或者OFF表示禁止。

Mysql的服务器层不管理事务，事务是由下层存储引擎实现的。

InnoDB采用两阶段锁定协议。在事务执行过程中，随时都可以执行锁定，锁只有在执行commit或者rollback的时候才会释放，并且所有的锁是在同一时刻被释放。

### 6.多版本并发控制

Mysql的大多数事务性存储引擎的实现都不是简单的行级锁。为了提升并发性能，一般都实现了多版本并发控制（MVCC）。MVCC的实现，是通过保存数据在某个时间点的快照来实现，以便每个事务看到的数据都是一致的。

InnoDB的MVCC，是通过在每行记录后面保存两个隐藏的列实现。一个保存了行的创建时间，一个保存了行的过期时间（或删除时间）。具体存储的是系统版本号，不是时间值。

对于select：

InnoDB会根据以下两个条件检查每行记录

a. InnoDB只查找版本小于等于当前事务版本的数据行，这样可以保证事务读取的 行在事务开始之前已经存在。

b. 行的删除版本号要么未定义，要么大于当前事务版本号，这样可以保证事务读 取到的行在事务开始之前未被删除。

只有符合上述两个条件的记录才能作为查询结果返回。

对于insert：

InnoDB为新插入的每一行保存当前系统版本号作为行版本号。

对于delete：

InnoDB为删除的每一行保存当前系统版本号作为行删除标识。

对于update：

InnoDB为插入一行新纪录保存当前系统版本号为行版本号，同时保存当前系统版 本号到原来的行作为行删除标识。

保存这两个额外的系统版本号，是大多数读操作都可以不用加锁，提高了性能。不足的是每行记录都需要额外的存储空间，需要更多的行检查工作以及额外的维护工作。MVCC只在可重复读和提交读两个隔离级别下工作。未提交读（每次读取最新数据）和串行化（对读取的行都加锁）不兼容。

### 7. Mysql的存储引擎

创建表会同时创建一个表同名的.frm文件，用于保存表的定义。可以使用show table status命令（mysql 5.0以后可以查询 information\_schema中对应的表）显示表的相关信息。

#### 7.1 InnoDB存储引擎

InnoDB的数据存储在表空间（tablespace）中，表空间由一系列的数据文件组成。InnoDB可以将每个表的数据和索引放在单独的文件中。

事务型存储引擎。InnoDB采用MVCC来支持高并发，并且实现了四个标准的隔离级别。默认为可重复读，并且通过间隙锁（next-key locking）策略防止幻读的出现。间隙锁使得InnoDB不仅仅锁定查询涉及的行，还会对索引中的间隙进行锁定。

InnoDB表是基于聚簇索引建立的，使主键查询有很高性能。不过二级索引中必须包含主键列，所以如果主键列很大，其他索引也会很大，因此表的主键要尽可能的小。

#### 7.2 MyISAM存储引擎

MyISAM支持全文索引、压缩、空间函数GIS等，但不支持事务和行级锁，崩溃后无法安全恢复。

MyISAM会将表存储在两个文件中：数据文件和索引文件，分别以.MYD和.MYI为扩展名。MyISAM表可以包含动态或静态（长度固定）行。他可以存储的行记录数，受限于可用的磁盘空间，或者操作系统中单个文件的最大尺寸。

MyISAM特性：

1.加锁与并发

只支持表级锁，读取数据会对表加共享锁，写入时对表加排它锁。读取数据的同时，可以往表中插入新的记录（并发插入）

2.修复

对于MyISAM表，mysql可以手工或者自动执行检查和修复操作

3.索引特性

对于BLOB和TEXT长字段，也可以基于前500个字符创建索引。MyISAM也支持全文索引，是基于分词创建的索引

4.延迟更新索引键

创建MyISAM表时，如果指定DELAY\_KEY\_WRITE选项，在每次修改执行完成时，会先将索引数据写入内存的键缓冲区，在清理键缓冲区或者关闭表的时候，才会将修改真正写入磁盘，极大的提升了写入性能

## 第四章：schema与数据类型优化

### 4.1 选择优化的数据类型

选择数据类型的简单原则：

1.更小的数据类型：占用更少的磁盘、内存和CPU缓存

2.简单数据类型：简单数据类型通常需要更少的cpu周期

3.尽量避免使用null：最好指定not null，允许为null对于mysql很难优化

#### 4.1.1 整数类型：

Tinyint,smallint,mediumint,int,bigint。分别使用8,16,24,32,64位存储空间，存储的值范围为-2(N-1)到2(N-1)-1，N为存储的位数。可以选择unsigned属性，表示不允许为负数，可以使正数的上限提高一倍，比如tinyint为-128~127，标记unsigned以后，范围为0~255。

对于执行整数类型的宽度，int(11)，只是限制了客户端显示的字符长度，不会限制存储的范围，所以int(1)和int(20)对于存储和计算来说是相同的。

#### 4.1.2 实数类型

实数是带有小数部分的数字。Float和double为近似计算，decimal为精确小数。CPU不支持decimal类型的计算，所以mysql服务器实现了decimal的计算，需要额外的空间和计算开销。

#### 4.1.3 字符串类型

Varchar

Varchar类型用于存储可变长字符串。比定长类型节省空间，但是需要额外的1或者2个额外字节记录字符串长度：如果列小于等于255，只使用1个字节表示，否则使用2个字节。因为是变长的，所以如果行增长到内存页内不能存放，MyISAM会将行拆成不同的片段存储，InnoDB需要分裂页使行可以放入页内。

适用场景：

1.字符串列的最大长度比平均长度大很多

2.列的更新很少，所以不用担心碎片问题

3.使用了像utf-8这样复杂的字符集，每个字符都使用不同的字节数进行存储

Varchar(5)和Varchar(200)存储‘hello’的空间开销是一样的，但是使用varchar(5)是有优势的，因为更长的列会消耗更多的内存，因为mysql通常会分配固定大小的内存块来保存内部值。使用内存临时表进行排序或操作会特别糟糕。

Char

Char类型是定长的：mysql总是根据定义的字符串长度分配足够的空间。Char类型会删除末尾的所有空格。

适用场景：

1. Char适合存储很短的字符串，或者所有值都接近同一个长度。比如MD5

2.数据经常变更，因为定长不容易产生碎片

BLOB与TEXT

Blob存储很大的数据，采用二进制的方式进行存储，没有排序规则和字符集。包括tinyblob, smallblob, blob, mediumblob, longblob。Blob与smallblob相同

Text存储很大的数据，采用字符方式进行存储，有排序规则和字符集。包括tinytext, smalltext, text, mediumtext, longtext。Text与smalltext相同。

Mysql把每个blob和text都当做独立的对象处理。当blob和text值太大时，InnoDB会使用外部存储区域进行存储，行内使用1~4个字节存储对应指针

#### 4.1.4 时间类型

Datetime和timestamp

Datetime范围更大，从1001年到9999年，精度为秒；timestamp从1970年1月1日算起，范围小，但是效率高。尽量使用timestamp。

### 4.2 mysql 数据库设计注意问题

1. 不要有太多的列

如果使用非常宽的表，但是使用数据的时候只会用到一小部分列，那么cpu占用可能会非常高。

1. 单个查询不要有太多的关联

最好在12个表以内做关联

### 4.3 范式

第一范式：每列保持原子性，不可分解

第二范式：每列都和主键相关，而不是和主键的一部分相关

第三范式：每列都和主键直接相关，而不是间接相关

### 4.5 alter table操作

Mysql执行大部分修改表结构的操作，比如增加索引，都是按照新的结构创建一个新的空表，然后查询旧表中的数据插入新表，最后删除旧表。

## 第五章：创建高性能的索引