当前使用的 mysql 版本号

5.6 版本

mysql 存储引擎

innodb: 特点: 事务型存储引擎, 支持事务, 支持外键, 支持行级锁定和表锁, 功能多但是查询速度比 myisam 慢.
场景: 需要事务支持, 并且需要较高的并发读写频率, 数据更新较为频繁的场景, 都可以选择 innodb

myisam: 特点: 不支持事务和外键, 锁的粒度范围到表锁, 查询速度快.
场景: 对事务没有要求或者以读取为主的应用基本都可以选择 myiasm

memory: 特点:采用的存储介质是系统内存, 所以能得到最快的响应时间. 虽然在内存中存储表数据确实会提供很高的性能, 但如果出现宕机或者其他故障的话, 很容易出现数据丢失现象, 很少使用.
场景: 目标数据较小, 而且被非常频繁地访问; 数据是临时的而且是必须立即可用; 或者存在内存中的数据一旦丢失不会对应用服务产生影响.

merge: 特点: Merge 存储引擎是一种 MyIsam 表的集合，这些 MyIsam 表结构必须完全一致，说白了，merge 表只是一堆 MyIsam 表的集合器，merge 表中并没有数据，对 merge 表的增删改查操作实际上是对 MyIsam 表的操作；删除 merge 表时，只是删除 merge 表的定义，对内部数据没有影响.

关系型和非关系型数据库区别

关系型: 表和表之间有联系的一大堆表组成的数据库(mysql, oracle, sql server, sqlite)

优点:

1. 容易理解
2. 易于维护：都是使用表结构，格式一致；
3.使用方便：SQL 语言通用，可用于复杂查询；
4.支持复杂操作：支持 SQL，可用于一个表以及多个表之间复杂查询。

缺点:

1、读写性能比较差，尤其是海量数据的高效率读写；
2、固定的表结构，灵活度稍欠；
3、高并发下硬盘 I/O 是一个很大的瓶颈。

非关系型: 一种结构化数据存储方法的集合 , 数据以键值对的形式进行存储(MongoDB, redis, memcache)

优点：

1、格式灵活：存储数据的格式支持 key-value, 文档, 图片等
2、速度快：nosql 可以使用硬盘或者随机存储器作为存储介质, 其设计理念之初便适合海量数据的高并发读写
3、高扩展性: 很容易进行集群扩展，适应海量数据增减
4、成本低: 非关系型数据库部署简单, 基本都是开源软件

缺点：

1、不提供 sql 支持，学习和使用成本较高；
2、数据结构相对复杂，复杂查询方面稍欠。

mysql 索引机制

mysql 索引底层存储机制是使用 B+树实现；B+树数据结构是 B 树实现的增强版本。尽管 B+树支持 B 树索引的所有特性，它们之间最显著的不同点在于 B+树中底层数据是根据索引列进行排序的。B+树还通过叶子节点之间的附加引用(每个叶子节点增加一个指向相邻叶子节点的指针)来优化扫描性能。

B+树的特性:

非叶子节点相当于是叶子节点的索引，叶子节点相当于数据层。
所有关键字(存储值)都只能出现在叶子节点的链表中，叶子节点相当于存储数据的数据层。
不可能在非叶子节点上命中。

mongodb 索引机制

mysql 索引底层存储机制是使用 B 树实现, B 树的特性:

关键字集合分布在整颗树中(非叶子节点和叶子节点都有存储)；
任何一个关键字出现且只出现在一个结点中；
搜索有可能在非叶子结点结束；
其搜索性能等价于在关键字全集内做一次二分查找；

b+树相比于 b 树的查询优势

b+树的中间节点不保存数据，所以磁盘页能容纳更多节点元素，更“矮胖”；
b+树查询必须查找到叶子节点，b 树只要匹配到即可不用管元素位置，因此 b+树查找更稳定（并不慢）；
对于范围查找来说，b+树只需遍历叶子节点链表即可，b 树却需要重复地中序遍历

mysql 索引种类

索引: 用于提高数据查询效率

根据唯一性区分

普通索引：仅加速查询

唯一索引：加速查询 + 列值唯一（可以有 null）

主键索引：加速查询 + 列值唯一（不可以有 null）+ 表中只有一个

组合索引：多列值组成一个索引，专门用于组合搜索，其效率大于索引合并

全文索引: 主要用来查找文本中的关键字，而不是直接与索引中的值相比较。fulltext 索引跟其它索引大不相同，它更像是一个搜索引擎，而不是简单的 where 语句的参数匹配

根据存储的方式区分

聚集索引和非聚集索引

聚集索引: 聚集索引数据的存放顺序与索引顺序是一致的。即只要索引是相邻的，那么索引所对应的数据也一定是相邻的存放在磁盘上。

聚集索引好处: 设想一下做范围查询时，比如查询 1000-10000 的主键 ID，又或者以 a,b,c 开头的用户名，如果是聚集索引的话，实际情况只需要一次磁盘 IO。

覆盖索引

在索引文件中就能把要查找的数据查到

索引合并

使用多个单列索引去查找数据, 不需要所有索引放一起查.

最左匹配原则

当使用联合索引或联合唯一索引时, 要查询的索引列一定要包含最左边的那个索引才能命中索引.

内连接

只返回两个表中连接字段相等的行

左连接(左外连接)

返回包括左表中的所有记录和右表中联结字段相等的记录

右连接(右外连接)

返回包括右表中的所有记录和左表中联结字段相等的记录

全外连接

左表和右表都不做限制，所有的记录都显示，两表不足的地方用 null 填充。

mysql 读写分离

把写专门交给写服务器处理, 那么需要把读的任务分配多台服务器来完成，就叫做读写分离。即写服务器为主服务器, 读服务器为从服务器, 如何实现主从服务器的数据一致性? ---> 主从复制技术.

数据库优化

sql 语句优化, 添加索引, 加缓存, 硬件扩容, 表的水平拆分和垂直拆分

垂直拆分: 数据中有大文本数据, 可以将大文本这一列拆分出去

水平拆分: 根据数据量或业务来拆, 数据量大时, 把主表主键索引 id 末尾数(0,1,2,3…9)的不同来计算一个哈希值, 每一个哈希值对应添加一个从表, 然后将这几个从表分布在不同的服务器上, 可以加快查询.

无法命中索引

1. where 查询的字段没有建索引, 走全局
2. where 数据类型不一致(passwd 字段是一个 char 类型, 查询却是数字), 走全局
3. 索引列参与到计算中走全局
4. 模糊查询 like 条件尽量以精确匹配开头"ale%", 如果"%xx.."走全局
5. or 条件中有未建立索引的走全局(主键除外)
6. order_by 对索引排序, 如果 select 字段没有建立索引则走全局(主键除外)
7. 联合索引或联合唯一索引没有遵循最左匹配原则

sql 语句优化

1. 建索引, 选择一些区分度高的列建索引(这样索引树的高度就会很低, 查询 I/O 时间少, 效率高)
2. 尽量避免使用 select * from...和 count * 之类的
3. 查询条件中避免一些 >, <, != , between ... and ... 查询范围大的条件

4. 尽量使用 and(mysql 从左到右优先找区分度高的字段查), 不使用 or(只能从左到右依次查询, 无法自动进行优选)
5. 表的字段中优先使用 char()类型, 查询快
6. 使用 join 连表查询代替子查询
7. 经常使用多个条件查询时, 可以用组合索引代替多个单列索引
总之要尽可能地命中索引

mysql 使用场景

mysql: 大多集中用于互联网方向, 对于数据安全性要求低, 灵活度要求高, 没有充足预算的话就选 mysql.
oracle: 主要用在传统行业的数据化业务中，业务模式比较固定. 比如：银行、金融这样的对可用性、健壮性、安全性、实时性要求极高的业务；零售、物流这样对海量数据存储分析要求很高的业务。

存储过程

一堆 sql 语句的封装, 通过调用存储过程名字可以执行其内部的 sql 语句, 说白了就是对 sql 语句的批处理.

视图

在查询某个数据时可能需要连上好几张表, 我们可以把这些 sql 语句封装起来作为视图, 以后再想查询该数据时就无需重写复杂的 sql 了，直接去视图中查找即可.
视图是一个虚拟表只存储表结构没有数据.
```
create view teacher_view as select tid from teacher where tname='李平老师';
```

触发器

对某一张表进行增删改前后的一些操作可以通过触发器实现.(new 表示即将要插入的数据行, old 表示即将要删除或修改的数据行)
```
create trigger tri_insert_cmd after insert on cmd for each row
```

mysql 的三范式

1NF——每一列的原子性，字段不可再分（关系型数据库都自动符合）
2NF——唯一性，在 1NF 基础上，设置主键（一条完全一样的数据不可能同时出现 2 次）
3NF——冗余性约束，在 2NF 基础上，设置外键
逆范式：逆范式化指的是通过增加冗余或重复的数据来提高数据库的读性能。

事务特性

原子性: 原子性是指事务是一个不可分割的工作单位, 事务中的操作要么都发生,要么都不发生.
隔离性: 隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。
即要达到这么一种效果：对于任意两个并发的事务 T1 和 T2，在事务 T1 看来，T2 要么在 T1 开始之前就已经结束，要么在 T1 结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。
一致性: 事务前后数据的完整性必须保持一致
永久性: 持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响

事务隔离级别

事务隔离主要分为 4 个级别
RU(未提交读): 该级别事务内容易出现脏读的情况，即事务 A 读到了事务 B 没有提交的数据；为了解决脏读问题，可以提高事务隔离级别到 RC(提交读)
RC(已提交读): 此时事务 A 不会读到其他事务未提交的数据，但又产生了一个新的现象：事务 A 执行的过程中，有可能另外一个事务 B 提交了数据，此时事务读取的数据和之前不一致，即出现了不可重复读的问题；所以 mysql 的 InnoDB 本身默认采用了第三个事务隔离级别 RR(可重复读)
RR(可重复读): 该级别使用 MVCC(多版本并发控制)解决重复读的问题，一般的 RR 级别会出现幻读的问题，及同一个事务多次执行同一个 select，读取到的数据行发生了改变，这是因为数据行发生了行数减少或者新增；但 mysql 中因为引入了 Gap Lock(间隙锁)的概念，所以 mysql 的 RR 级别同时也解决了幻读的问题
SE(可序列化): 最高的事务隔离级别是 SE(可序列化)，该方式下事务相当于串行执行，对性能和效率的影响很大，生产环境中很少会使用该隔离级别
脏读: 事务 B 读取事务 A 还没有提交的数据
幻读:事务 A 对数据进行修改, 同时事务 B 对数据进行插入, 事务 A 查询时发现还有一条数据未修改, 跟幻象一样.
不可重复读: 事务 A 执行的过程中，有可能事务 B 修改并提交了数据使两次读到的数据不一样.

mysql 和 redis 的事务

redis 事务: 半事务支持: 如果没有语法错误的话事务依旧生效, sql 语句依旧执行, 语法错误则事务不生效
mysql 事务: 全事务支持: 只要报错了 sql 语句不执行, 事务就不生效

MVCC

是一种多版本并发控制机制, 是一种并发控制的方法，一般在数据库管理系统中，实现对数据库的并发访问.大多数 mysql 事务型存储引擎，如 InnoDB 等都不使用一种简单的行锁机制。事实上，他们都和 MVCC 多版本控制一起来使用。行锁机制可以控制并发操作，但是其系统开销较大，而 MVCC 可以在大多数情况下代替行级锁，使用 MVCC 能降低其系统开销。

Gap Lock

MySQL InnoDB 支持三种行锁定方式：
行锁（Record Lock）: 锁直接加在索引记录上面(不是行数据)，锁住的是 key。
间隙锁（Gap Lock）: 锁定索引记录间隙，确保索引记录的间隙不变。间隙锁是针对事务隔离级别为可重复读或以上级别的。
Next-Key Lock : 行锁和间隙锁组合起来就叫 Next-Key Lock。
默认情况下，InnoDB 工作在可重复读隔离级别下，并且会以 Next-Key Lock 的方式对数据行进行加锁，这样可以有效防止幻读的发生。Next-Key Lock 是行锁和间隙锁的组合，当 InnoDB 扫描索引记录的时候，会首先对索引记录加上行锁（Record Lock），再对索引记录两边的间隙加上间隙锁（Gap Lock）。加上间隙锁之后，其他事务就不能在这个间隙修改或者插入记录。
Gap Lock 在 InnoDB 的唯一作用就是防止其他事务的插入操作，以此防止幻读的发生。

乐观锁

乐观锁，即先进行业务操作，不到最后一步不进行加锁，在最后一步更新数据的时候再进行加锁. 乐观锁的实现方式一般为每一条数据加一个版本号, 修改数据的时候首先把这条数据的版本号查出来，update 时判断这个版本号是否和数据库里的一致，如果一致则表明这条数据没有被其他用户修改，若不一致则表明这条数据在操作期间被其他客户修改过，此时需要在代码中抛异常或者回滚等
乐观锁适用于写比较少的情况下（多读场景），多读场景下冲突很少, 这样可以省去了锁的开销，加大了系统的整个吞吐量。

悲观锁

悲观锁对数据加锁持有一种悲观的态度。因此，在整个数据处理过程中，将数据处于锁定状态.
悲观锁用数据库提供的锁的机制.
悲观锁适用于多写的情况，如果使用乐观锁则会降低性能，所以一般多写的场景下用悲观锁就比较合适.

redis 使用场景

1. 用作缓存: 使用 Redis 把常被请求的内容缓存起来，能够大大的降低页面请求的延迟，用 Redis 来缓存页面，这就是页面静态化的一种方式。
2. 用于 session: 相比较于类似 memcache 的 session 存储，Redis 具有缓存数据持久化的能力，当缓存因出现问题而重启后，之前的缓存数据还在那儿，避免了因为 session 突然消失带来的用户体验问题。
3. 用于计数器: 商品的浏览量、视频网站视频的播放数等。为了保证数据实时效，每次浏览都得给+1，并发量高时如果每次都请求数据库操作无疑是种挑战和压力。Redis 提供的 incr 命令来实现计数器功能
4. 用于排行榜 Top 100: 使用有序集合来存储数据(sorted-set): zadd key score1 member1,
   查数据: arevrange key 0 99
5. 最新评论: Redis 使用列表结构， ltrim 可用来限制列表的数量(假如为 2000 个)，这样列表永远为 2000 个 ID，当要查询的最新评论数小于 2000 直接去访问 redis, 超过 2000 再去磁盘上的数据库找.
6. 有时效性的数据: 比如 redis 通过字符串去存储一些有时间限制的验证码, 超时自动清除.

redis 数据类型

字符串类型, 哈希类型, 列表类型, 集合类型, 有序集合类型

redis 持久化方式

以 rdb 文件进行存储: 是将数据先存储在内存，在指定的时间间隔内(每过 900 秒)将变化的数据一次性写入 rdb 数据文件.**特点**: 速度快, 适合作备份
以 aof 文件进行存储: 记录服务器执行的所有变更操作命令（例如 set del 等）, 并将之追加到文件末尾。
特点: 能够最大程度地保证数据不会丢失, 如果设置追加 file 的时间是 1s，如果 redis 发生故障，最多会丢失 1s 的数据. 但是日志记录会非常大.

MongoDB 和 Redis 区别

Redis: 支持多种数据结构, 可以对数据设置过期时间.
MongoDB: 是面向文件存储的数据库, mongodb 支持丰富的数据表达，索引，最类似关系型数据库，支持的查询语言非常丰富, mongoDB 适合大数据量的存储，依赖操作系统的内存管理，吃内存也比较厉害.

MongoDB 使用场景

Redis：数据量较小的高性能操作和运算上
MongoDB: 主要解决海量数据的访问效率问题(爬虫数据的后期存储)

MongoDB 版本号

3.2 版本, 在 4.0 版本开始支持事务操作

MongoDB 存储引擎

MMAP: out, 已经过时了的
Wiredtiger:

3.0 以前锁的粒度是库级别的, 3.0 引入了 wiredtiger 存储引擎, 锁的粒度精确到文档级别(相当于行锁);
磁盘数据是经过压缩的, 更有利于海量数据的存储;
删除数据时会被立即删除;
性能较之前版本有 4-7 倍的增长

Rocksdb:

RocksDB 针对写入做了优化，将随机写入转换成了顺序写入，能保证持续高效的数据写入。

写场景较多时刚开始 Wiredtiger 的写入性能远超 RocksDB，而随着数据量越来越大，Wiredtiger 的写入能力开始下降，而 RocksDB 的写入一直比较稳定。