- **①存储引擎**

MySQL常见的两种存储引擎：MyISAM与InnoDB的爱恨情仇

https://juejin.im/post/5b1685bef265da6e5c3c1c34

- **②字符集及校对规则**

字符集指的是一种从二进制编码到某类字符符号的映射。校对规则则是指某种字符集下的排序规则。Mysql中每一种字符集都会对应一系列的校对规则。

Mysql采用的是类似继承的方式指定字符集的默认值，每个数据库以及每张数据表都有自己的默认值，他们逐层继承。比如：某个库中所有表的默认字符集将是该数据库所指定的字符集（这些表在没有指定字符集的情况下，才会采用默认字符集） PS：整理自《Java工程师修炼之道》

详细内容可以参考： MySQL字符集及校对规则的理解

https://www.cnblogs.com/geaozhang/p/6724393.html#mysqlyuzifuji

- **③索引相关的内容（数据库使用中非常关键的技术，合理正确的使用索引可以大大提高数据库的查询性能）**

Mysql索引使用的数据结构主要有**BTree索引** 和 **哈希索引** 。对于哈希索引来说，底层的数据结构就是哈希表，因此在绝大多数需求为单条记录查询的时候，可以选择哈希索引，查询性能最快；其余大部分场景，建议选择BTree索引。

Mysql的BTree索引使用的是B数中的B+Tree，但对于主要的两种存储引擎的实现方式是不同的。

MyISAM: B+Tree叶节点的data域存放的是数据记录的地址。在索引检索的时候，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域的值为地址读取相应的数据记录。这被称为“非聚簇索引”。

InnoDB: 其数据文件本身就是索引文件。相比MyISAM，索引文件和数据文件是分离的，其表数据文件本身就是按B+Tree组织的一个索引结构，树的叶节点data域保存了完整的数据记录。这个索引的key是数据表的主键，因此InnoDB表数据文件本身就是主索引。这被称为“聚簇索引（或聚集索引）”。而其余的索引都作为辅助索引，辅助索引的data域存储相应记录主键的值而不是地址，这也是和MyISAM不同的地方。**在根据主索引搜索时，直接找到key所在的节点即可取出数据；在根据辅助索引查找时，则需要先取出主键的值，在走一遍主索引。 因此，在设计表的时候，不建议使用过长的字段作为主键，也不建议使用非单调的字段作为主键，这样会造成主索引频繁分裂。** PS：整理自《Java工程师修炼之道》

详细内容可以参考：

干货：mysql索引的数据结构

https://www.jianshu.com/p/1775b4ff123a

MySQL优化系列（三）--索引的使用、原理和设计优化

https://blog.csdn.net/Jack_Frost/article/details/72571540

- **④查询缓存的使用**

my.cnf加入以下配置，重启Mysql开启查询缓存

```
query_cache_type=1
query_cache_size=600000
```

Mysql执行以下命令也可以开启查询缓存

```
set global  query_cache_type=1;
set global  query_cache_size=600000;
```

如上，**开启查询缓存后在同样的查询条件以及数据情况下，会直接在缓存中返回结果。**这里的查询条件包括查询本身、当前要查询的数据库、客户端协议版本号等一些可能影响结果的信息。因此任何两个查询在任何字符上的不同都会导致缓存不命中。此外，如果查询中包含任何用户自定义函数、存储函数、用户变量、

临时表、Mysql库中的系统表，其查询结果也不会被缓存。

缓存建立之后，Mysql的查询缓存系统会跟踪查询中涉及的每张表，如果这些表（数据或结构）发生变化，那么和这张表相关的所有缓存数据都将失效。

缓存虽然能够提升数据库的查询性能，但是缓存同时也带来了额外的开销，每次查询后都要做一次缓存操作，失效后还要销毁。 因此，开启缓存查询要谨慎，尤其对于写密集的应用来说更是如此。如果开启，要注意合理控制缓存空间大小，一般来说其大小设置为几十MB比较合适。此外，**还可以通过sql_cache和sql_no_cache来控制某个查询语句是否需要缓存：**

```
1 select sql_no_cache count(*) from usr;
```

- ## ⑤事务机制

关系性数据库需要遵循ACID规则，具体内容如下：

事务的特性

1. **原子性：** 事务是最小的执行单位，不允许分割。事务的原子性确保动作要么全部完成，要么完全不起作用；
2. **一致性：** 执行事务前后，数据保持一致；
3. **隔离性：** 并发访问数据库时，一个用户的事物不被其他事物所干扰，各并发事务之间数据库是独立的；
4. **持久性:** 一个事务被提交之后。它对数据库中数据的改变是持久的，即使数据库 发生故障也不应该对其有任何影响。

为了达到上述事务特性，数据库定义了几种不同的事务隔离级别：

- **READ_UNCOMMITTED（未授权读取）：**最低的隔离级别，允许读取尚未提交的数据变更，**可能会导致脏读、幻读或不可重复读**
- **READ_COMMITTED（授权读取）：**允许读取并发事务已经提交的数据，**可以阻止脏读，但是幻读或不可重复读仍有可能发生**
- **REPEATABLE_READ（可重复读）：**对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，**可以阻止脏读和不可重复读，但幻读仍有可能发生。**
- **SERIALIZABLE（串行）：**最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，**该级别可以防止脏读、不可重复读以及幻读。**但是这将严重影响程序的性能。通常情况下也不会用到该级别。

这里需要注意的是：**Mysql 默认采用的 REPEATABLE_READ隔离级别 Oracle 默认采用的 READ_COMMITTED隔离级别.**

事务隔离机制的实现基于锁机制和并发调度。其中并发调度使用的是MVVC（多版本并发控制），通过保存修改的旧版本信息来支持并发一致性读和回滚等特性。

详细内容可以参考： 可能是最漂亮的Spring事务管理详解

https://blog.csdn.net/qq_34337272/article/details/80394121

- ## ⑥锁机制与InnoDB锁算法

MyISAM和InnoDB存储引擎使用的锁：

MyISAM采用表级锁(table-level locking)。

InnoDB支持行级锁(row-level locking)和表级锁,默认为行级锁

表级锁和行级锁对比：

表级锁： Mysql中锁定 **粒度最大** 的一种锁，对当前操作的整张表加锁，实现简单，资源消耗也比较少，加锁快，不会出现死锁。其锁定粒度最大，触发锁冲突的概率最高，并发度最低，MyISAM和 InnoDB引擎都支持表级锁。

行级锁： Mysql中锁定 **粒度最小** 的一种锁，只针对当前操作的行进行加锁。 行级锁能大大减少数据库操作的冲突。其加锁粒度最小，并发度高，但加锁的开销也最大，加锁慢，会出现死锁。

详细内容可以参考： Mysql锁机制简单了解一下

https://blog.csdn.net/qq_34337272/article/details/80611486

InnoDB存储引擎的锁的算法有三种：

Record lock：单个行记录上的锁

Gap lock：间隙锁，锁定一个范围，不包括记录本身

Next-key lock：record+gap 锁定一个范围，包含记录本身

相关知识点：

1. innodb对于行的查询使用next-key lock
2. Next-locking keying为了解决Phantom Problem幻读问题
3. 当查询的索引含有唯一属性时，将next-key lock降级为record key
4. Gap锁设计的目的是为了阻止多个事务将记录插入到同一范围内，而这会导致幻读问题的产生
5. 有两种方式显式关闭gap锁： （除了外键约束和唯一性检查外，其余情况仅使用record lock） A. 将事务隔离级别设置为RC B. 将参数innodb_locks_unsafe_for_binlog设置为1

- ## ⑦大表优化

当MySQL单表记录数过大时，数据库的CRUD性能会明显下降，一些常见的优化措施如下：

1. **限定数据的范围：** 务必禁止不带任何限制数据范围条件的查询语句。比如：我们当用户在查询订单历史的时候，我们可以控制在一个月的范围内。；
2. **读/写分离：** 经典的数据库拆分方案，主库负责写，从库负责读；
3. **缓存：** 使用MySQL的缓存，另外对重量级、更新少的数据可以考虑使用应用级别的缓存；

4. **垂直分区：**

   根据数据库里面数据表的相关性进行拆分。 例如，用户表中既有用户的登录信息又有用户的基本信息，可以将用户表拆分成两个单独的表，甚至放到单独的库做分库。

   简单来说垂直拆分是指数据表列的拆分，把一张列比较多的表拆分为多张表。 如下图所示，这样来说大家应该就更容易理解了。

   img

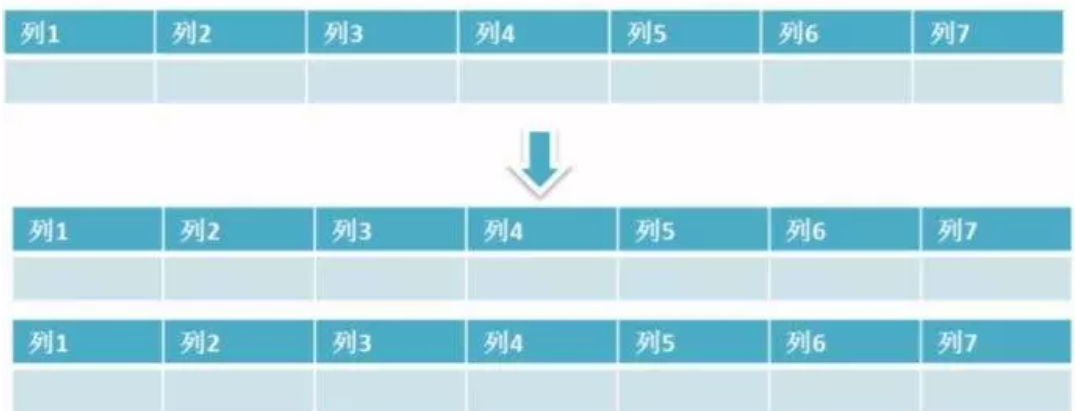
   垂直拆分的优点： 可以使得行数据变小，在查询时减少读取的Block数，减少I/O次数。此外，垂直分区可以简化表的结构，易于维护。

   垂直拆分的缺点： 主键会出现冗余，需要管理冗余列，并会引起Join操作，可以通过在应用层进行Join来解决。此外，垂直分区会让事务变得更加复杂；

5. **水平分区：**

   保持数据表结构不变，通过某种策略存储数据分片。这样每一片数据分散到不同的表或者库中，达到了分布式的目的。 水平拆分可以支撑非常大的数据量。

   水平拆分是指数据表行的拆分，表的行数超过200万行时，就会变慢，这时可以把一张的表的数据拆成多张表来存放。举个例子：我们可以将用户信息表拆分成多个用户信息表，这样就可以避免单一表数据量过大对性能造成影响。



数据库水平拆分

水品拆分可以支持非常大的数据量。需要注意的一点是:分表仅仅是解决了单一表数据过大的问题，但由于表的数据还是在同一台机器上，其实对于提升MySQL并发能力没有什么意义，所以 **水品拆分最好分库** 。

水平拆分能够 **支持非常大的数据量存储，应用端改造也少**，但 **分片事务难以解决** ，跨界点Join性能较差，逻辑复杂。《Java工程师修炼之道》的作者推荐 **尽量不要对数据进行分片，因为拆分会带来逻辑、部署、运维的各种复杂度** ，一般的数据表在优化得当的情况下支撑千万以下的数据量是没有太大问题的。如果实在要分片，尽量选择客户端分片架构，这样可以减少一次和中间件的网络I/O。

下面补充一下数据库分片的两种常见方案：

- **客户端代理： 分片逻辑在应用端，封装在jar包中，通过修改或者封装JDBC层来实现。** 当当网的 **Sharding-JDBC** 、阿里的TDDL是两种比较常用的实现。
- **中间件代理： 在应用和数据中间加了一个代理层。分片逻辑统一维护在中间件服务中。** 我们现在谈的 **Mycat**、360的Atlas、网易的DDB等等都是这种架构的实现。