第三版：MySQL 40 道

MySQL 中 int(10)和 char(10)以及 varchar(10)的区别

1、 int(10)的 10 表示显示的数据的长度，不是存储数据的大小；chart(10)和 varchar(10)的 10 表示存储数据的大小，即表示存储多少个字符。

2、 char(10)表示存储定长的 10 个字符，不足 10 个就用空格补齐，占用更多的存储空间

3、 varchar(10)表示存储 10 个变长的字符，存储多少个就是多少个，空格也按一个字符存储，这一点是和 char(10)的空格不同的，char(10)的空格表示占位不算一个字符

FLOAT 和 DOUBLE 的区别是什么?

1、 FLOAT 类型数据可以存储至多 8 位十进制数，并在内存中占 4 字节。

2、 DOUBLE 类型数据可以存储至多 18 位十进制数，并在内存中占 8 字节。

drop、delete 与 truncate 的区别

三者都表示删除，但是三者有一些差别：

比较	Delete	Truncate	Drop
类型	属于 DML	属于 DDL	属于 DDL
回滚	可回滚	不可回滚	不可回滚

删除内容	表结构还在，删除表的全部或者一部分数据行	表结构还在，删除表中的所有数据	从数据库中删除表，所有的数据行，索引和权限也会被删除
删除速度	删除速度慢，需要逐行删除	删除速度快	删除速度最快

因此，在不再需要一张表的时候，用 drop；在想删除部分数据行时候，用 delete；在保留表而删除所有数据的时候用 truncate。

UNION 与 UNION ALL 的区别?

1、 如果使用 UNION ALL，不会合并重复的记录行

2、 效率 UNION 高于 UNION ALL

说出一些数据库优化方面的经验?

1、 有外键约束的话会影响增删改的性能，如果应用程序可以保证数据库的完整性那就去除外键

2、 Sql 语句全部大写，特别是列名大写，因为数据库的机制是这样的，sql 语句发送到数据库服务器，数据库首先就会把 sql 编译成大写在执行，如果一开始就编译成大写就不需要了把 sql 编译成大写这个步骤了

3、 如果应用程序可以保证数据库的完整性，可以不需要按照三大范式来设计数据库

4、 其实可以不必要创建很多索引，索引可以加快查询速度，但是索引会消耗磁盘空间

5、如果是 jdbc 的话，使用 PreparedStatement 不使用 Statement，来创建 SQl，PreparedStatement 的性能比 Statement 的速度要快，使用 PreparedStatement 对象 SQL 语句会预编译在此对象中，PreparedStatement 对象可以多次高效的执行

怎么优化 SQL 查询语句吗

1、 对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引

2、 用索引可以提高查询

3、 SELECT 子句中避免使用*号，尽量全部大写 SQL

4、 应尽量避免在 where 子句中对字段进行 is null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，使用 IS NOT NULL

5、 where 子句中使用 or 来连接条件，也会导致引擎放弃使用索引而进行全表扫描

6、 in 和 not in 也要慎用，否则会导致全表扫描

你怎么知道 SQL 语句性能是高还是低

1、 查看 SQL 的执行时间

2、 使用 explain 关键字可以模拟优化器执行 SQL 查询语句，从而知道 MYSQL 是如何处理你的 SQL 语句的。分析你的查询语句或是表结构的性能瓶颈。

SQL 的执行顺序

1、 FROM：将数据从硬盘加载到数据缓冲区，方便对接下来的数据进行操作。

2、 WHERE：从基表或视图中选择满足条件的元组。（不能使用聚合函数）

3、 JOIN（如 right left 右连接-------从右边表中读取某个元组，并且找到该元组在左边表中对应的元组或元组集）

4、 ON：join on 实现多表连接查询，推荐该种方式进行多表查询，不使用子查询。

5、 GROUP BY：分组，一般和聚合函数一起使用。

6、 HAVING：在元组的基础上进行筛选，选出符合条件的元组。（一般与 GROUP BY 进行连用）

7、 SELECT：查询到得所有元组需要罗列的哪些列。

8、 DISTINCT：去重的功能。

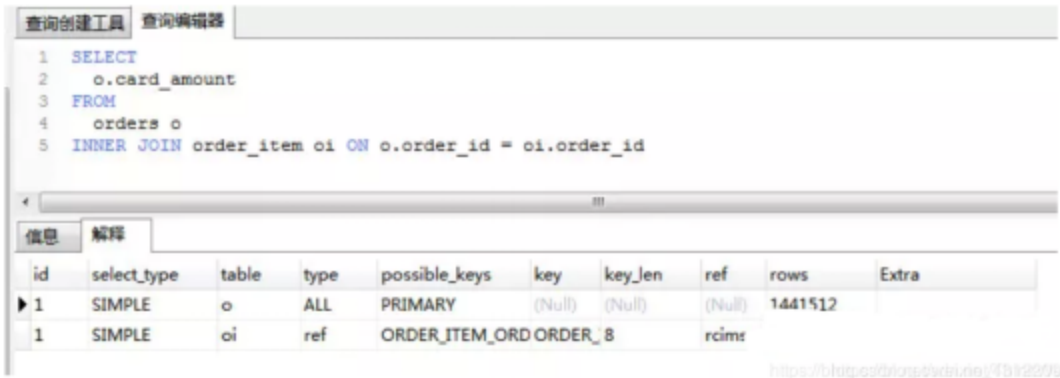
9、 UNION：将多个查询结果合并（默认去掉重复的记录）。

10、 ORDER BY：进行相应的排序。

11、 LIMIT 1：显示输出一条数据记录（元组）

如何定位及优化 SQL 语句的性能问题？创建的索引有没有被使用到?或者说怎么才可以知道这条语句运行很慢的原因?

对于低性能的 SQL 语句的定位，最重要也是最有效的方法就是使用执行计划，MySQL 提供了 explain 命令来查看语句的执行计划。 我们知道，不管是哪种数据库，或者是哪种数据库引擎，在对一条 SQL 语句进行执行的过程中都会做很多相关的优化，**对于查询语句，最重要的优化方式就是使用索引。 而执行计划，就是显示数据库引擎对于 SQL 语句的执行的详细情况，其中包含了是否使用索引，使用什么索引，使用的索引的相关信息等。**



执行计划包含的信息 **id** 有一组数字组成。表示一个查询中各个子查询的执行顺序;

1、 id 相同执行顺序由上至下。

2、 id 不同，id 值越大优先级越高，越先被执行。

3、 id 为 null 时表示一个结果集，不需要使用它查询，常出现在包含 union 等查询语句中。

select_type 每个子查询的查询类型，一些常见的查询类型。

id	select_type	description
1	SIMPLE	不包含任何子查询或 union 等查询
2	PRIMARY	包含子查询最外层查询就显示为 PRIMARY
3	SUBQUERY	在 select 或 where 字句中包含的查询
4	DERIVED	from 字句中包含的查询
5	UNION	出现在 union 后的查询语句中
6	UNION RESULT	从 UNION 中获取结果集，例如上文的第三个例子

table 查询的数据表，当从衍生表中查数据时会显示 x 表示对应的执行计划 id

partitions 表分区、表创建的时候可以指定通过那个列进行表分区。 举个例子：

```
create table tmp (
id int unsigned not null AUTO_INCREMENT,
name varchar(255),
PRIMARY KEY (id)
) engine = innodb
partition by key (id) partitions 5;
```

type(非常重要，可以看到有没有走索引) 访问类型

1、 ALL 扫描全表数据

2、 index 遍历索引

3、 range 索引范围查找

4、 index_subquery 在子查询中使用 ref

5、 unique_subquery 在子查询中使用 eq_ref

6、 ref_or_null 对 Null 进行索引的优化的 ref

7、 fulltext 使用全文索引

8、 ref 使用非唯一索引查找数据

9、 eq_ref 在 join 查询中使用 PRIMARY KEYorUNIQUE NOT NULL 索引关联。

10、 **possible_keys** 可能使用的索引，注意不一定会使用。查询涉及到的字段上若存在索引，则该索引将被列出来。当该列为 NULL 时就要考虑当前的 SQL 是否需要优化了。

11、 **key** 显示 MySQL 在查询中实际使用的索引，若没有使用索引，显示为 NULL。

12、 **TIPS**:查询中若使用了覆盖索引(覆盖索引：索引的数据覆盖了需要查询的所有数据)，则该索引仅出现在 key 列表中

13、 **key_length** 索引长度

14、 **ref** 表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值

15、 **rows** 返回估算的结果集数目，并不是一个准确的值。

16、 **extra** 的信息非常丰富，常见的有：

17、 Using index 使用覆盖索引

18、 Using where 使用了用 where 子句来过滤结果集

19、 Using filesort 使用文件排序，使用非索引列进行排序时出现，非常消耗性能，尽量优化。

20、 Using temporary 使用了临时表 sql 优化的目标可以参考阿里开发手册

推荐

SQL 性能优化的目标：至少要达到 range 级别，要求是 ref 级别，如果可以是 consts 最好

说明：

1、 consts 单表中最多只有一个匹配行（主键或者唯一索引），在优化阶段即可读取到数据。

2、 ref 指的是使用普通的索引（normal index）。

3、 range 对索引进行范围检索。

反例：

explain 表的结果，type=index，索引物理文件全扫描，速度非常慢，这个 index 级别比较 range 还低，与全表扫描是小巫见大巫。

SQL 的生命周期?

1、 应用服务器与数据库服务器建立一个连接

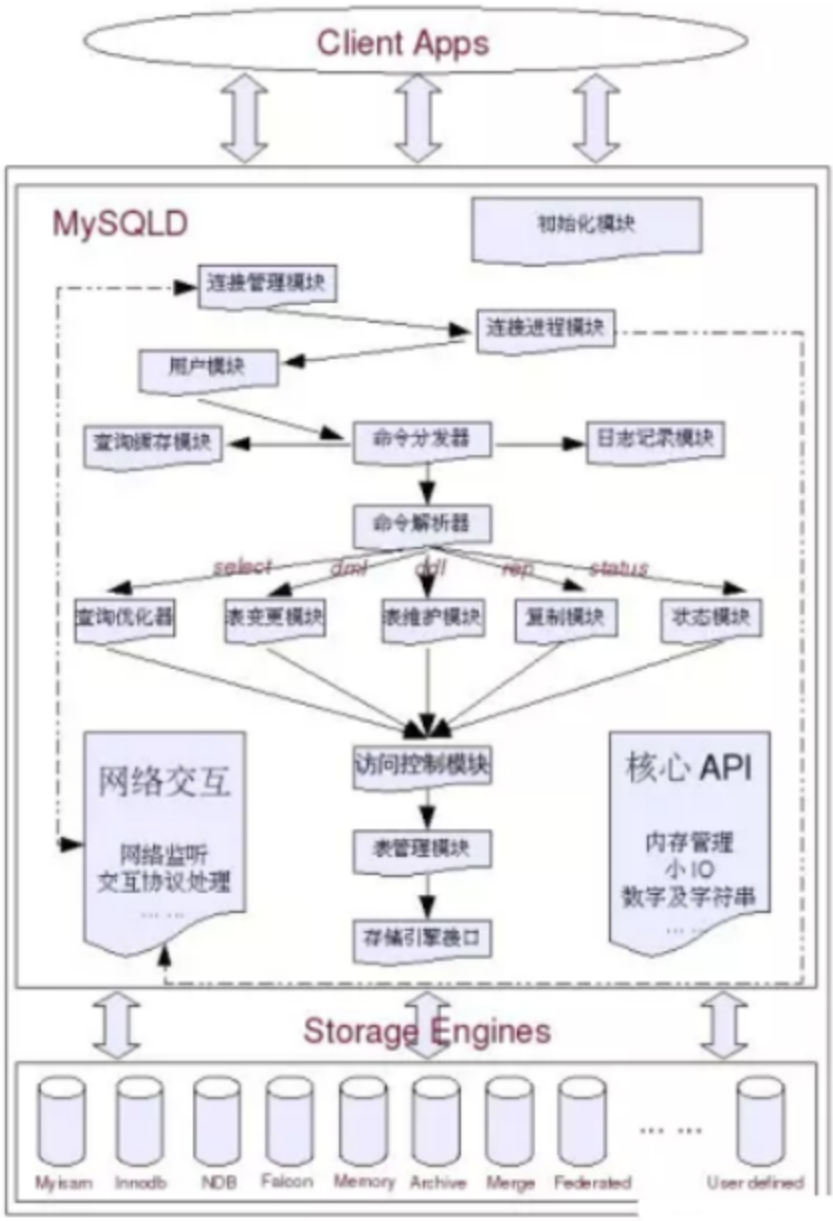
2、 数据库进程拿到请求 sql

3、 解析并生成执行计划，执行

4、 读取数据到内存并进行逻辑处理

5、 通过步骤一的连接，发送结果到客户端

6、 关掉连接，释放资源



大表数据查询，怎么优化

1、 优化 shema、sql 语句+索引；

2、 第二加缓存，Memcached，Redis；

3、 主从复制，读写分离；

4、 垂直拆分，根据你模块的耦合度，将一个大的系统分为多个小的系统，也就是分布式系统；

5、 水平切分，针对数据量大的表，这一步最麻烦，最能考验技术水平，要选择一个合理的 sharding key，为了有好的查询效率，表结构也要改动，做一定的冗余，应用也要改，sql 中尽量带 sharding key，将数据定位到限定的表上去查，而不是扫描全部的表；

超大分页怎么处理?

超大的分页一般从两个方向上来解决.

1、 数据库层面,这也是我们主要集中关注的(虽然收效没那么大),类似于 select * from table where age > 20 limit 1000000,10 这种查询其实也是有可以优化的余地的、这条语句需要 load1000000 数据然后基本上全部丢弃,只取 10 条当然比较慢、当时我们可以修改为 select * from table where id in (select id from table where age > 20 limit 1000000,10).这样虽然也 load 了一百万的数据,但是由于索引覆盖,要查询的所有字段都在索引中,所以速度会很快、同时如果 ID 连续的好,我们还可以 select * from table where id > 1000000 limit

10,效率也是不错的,优化的可能性有许多种,但是核心思想都一样,就是减少 load 的数据.

2、 从需求的角度减少这种请求...主要是不做类似的需求(直接跳转到几百万页之后的具体某一页.只允许逐页查看或者按照给定的路线走,这样可预测,可缓存)以及防止 ID 泄漏且连续被人恶意攻击.

解决超大分页,其实主要是靠缓存,可预测性的提前查到内容,缓存至 Redis 等 k-V 数据库中,直接返回即可

MySQL 分页

LIMIT 子句可以被用于强制 SELECT 语句返回指定的记录数。LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。初始记录行的偏移量是 0(而不是 1)

```
SELECT * FROM table LIMIT 5,10; // 检索记录行 6-15
```

为了检索从某一个偏移量到记录集的结束所有的记录行，可以指定第二个参数为 -1：

```
SELECT * FROM table LIMIT 95,-1; // 检索记录行 96-last.
```

如果只给定一个参数，它表示返回最大的记录行数目：

```
SELECT * FROM table LIMIT 5; //检索前 5 个记录行
```

换句话说，LIMIT n 等价于 LIMIT 0,n。

慢查询日志

用于记录执行时间超过某个临界值的 SQL 日志，用于快速定位慢查询，为我们的优化做参考

1、 开启慢查询日志

2、 配置项：slow_query_log

3、 可以使用 show variables like ‘slov_query_log’ 查看是否开启，如果状态值为 OFF，可以使用 set GLOBAL slow_query_log = on 来开启，它会在 datadir 下产生一个 xxx-slow.log 的文件。

4、 设置临界时间

5、 配置项：long_query_time

6、 查看：show VARIABLES like 'long_query_time'，单位秒

7、 设置：set long_query_time=0.5

8、 实操时应该从长时间设置到短的时间，即将最慢的 SQL 优化掉

9、查看日志，一旦 SQL 超过了我们设置的临界时间就会被记录到 xxx-slow.log 中

关心过业务系统里面的 sql 耗时吗？统计过慢查询吗？对慢查询都怎么优化过？

在业务系统中，除了使用主键进行的查询，其他的我都会在测试库上测试其耗时，慢查询的统计主要由运维在做，会定期将业务中的慢查询反馈给我们。

慢查询的优化首先要搞明白慢的原因是什么？ 是查询条件没有命中索引？是 load 了不需要的数据列？还是数据量太大？

所以优化也是针对这三个方向来的，

1、 首先分析语句，看看是否 load 了额外的数据，可能是查询了多余的行并且抛弃掉了，可能是加载了许多结果中并不需要的列，对语句进行分析以及重写。

2、 分析语句的执行计划，然后获得其使用索引的情况，之后修改语句或者修改索引，使得语句可以尽可能的命中索引。

3、 如果对语句的优化已经无法进行，可以考虑表中的数据量是否太大，如果是的话可以进行横向或者纵向的分表。

为什么要尽量设定一个主键?

主键是数据库确保数据行在整张表唯一性的保障，即使业务上本张表没有主键，也建议添加一个自增长的 ID 列作为主键。设定了主键之后，在后续的删改查的时候可能更加快速以及确保操作数据范围安全。

主键使用自增 ID 还是 UUID?

1、 推荐使用自增 ID，不要使用 UUID。

2、 因为在 InnoDB 存储引擎中，主键索引是作为聚簇索引存在的，也就是说，主键索引的 B+树叶子节点上存储了主键索引以及全部的数据(按照顺序)，如果主键索引是自增 ID，那么只需要不断向后排列即可，如果是 UUID，由于到来的 ID 与原来的大小不确定，会造成非常多的数据插入，数据移动，然后导致产生很多的内存碎片，进而造成插入性能的下降。

总之，在数据量大一些的情况下，用自增主键性能会好一些。

关于主键是聚簇索引，如果没有主键，InnoDB 会选择一个唯一键来作为聚簇索引，如果没有唯一键，会生成一个隐式的主键。

字段为什么要求定义为 not null?

null 值会占用更多的字节，且会在程序中造成很多与预期不符的情况。

如果要存储用户的密码散列，应该使用什么字段进行存储?

密码散列，盐，用户身份证号等固定长度的字符串应该使用 char 而不是 varchar 来存储，这样可以节省空间且提高检索效率。

如何优化查询过程中的数据访问

1、 访问数据太多导致查询性能下降

2、 确定应用程序是否在检索大量超过需要的数据，可能是太多行或列

3、 确认 MySQL 服务器是否在分析大量不必要的数据行

4、 避免犯如下 SQL 语句错误

5、 避免查询不需要的数据。解决办法：使用 limit 解决

6、 多表关联返回全部列。解决办法：指定列名

7、 总是返回全部列。解决办法：避免使用 SELECT *

8、 重复查询相同的数据。解决办法：可以缓存数据，下次直接读取缓存

9、 使用 explain 进行分析，如果发现查询需要扫描大量的数据，但只返回少数的行，可以通过如下技巧去优化：

10、 使用索引覆盖扫描，把所有的列都放到索引中，这样存储引擎不需要回表获取对应行就可以返回结果。

11、 改变数据库和表的结构，修改数据表范式

12、 重写 SQL 语句，让优化器可以以更优的方式执行查询。

如何优化长难的查询语句

1、 分析是一个复杂查询还是多个简单查询速度快

2、 MySQL 内部每秒能扫描内存中上百万行数据，相比之下，响应数据给客户端就要慢得多

3、 使用尽可能小的查询是好的，但是有时将一个大的查询分解为多个小的查询是很有必要的。

4、 将一个大的查询分为多个小的相同的查询

5、 一次性删除 1000 万的数据要比一次删除 1 万，暂停一会的方案更加损耗服务器开销。

6、 分解关联查询，让缓存的效率更高。

7、 执行单个查询可以减少锁的竞争。

8、 在应用层做关联更容易对数据库进行拆分。

9、 查询效率会有大幅提升。

10、 较少冗余记录的查询。

优化特定类型的查询语句

1、 count(*)会忽略所有的列，直接统计所有列数，不要使用 count(列名)

2、 MyISAM 中，没有任何 where 条件的 count(*)非常快。

3、 当有 where 条件时，MyISAM 的 count 统计不一定比其它引擎快。

4、 可以使用 explain 查询近似值，用近似值替代 count(*)

5、 增加汇总表

6、 使用缓存

优化关联查询

1、 确定 ON 或者 USING 子句中是否有索引。

2、 确保 GROUP BY 和 ORDER BY 只有一个表中的列，这样 MySQL 才有可能使用索引。

优化子查询

1、 用关联查询替代

2、 优化 GROUP BY 和 DISTINCT

3、 这两种查询据可以使用索引来优化，是最有效的优化方法

4、 关联查询中，使用标识列分组的效率更高

5、 如果不需要 ORDER BY，进行 GROUP BY 时加 ORDER BY NULL，MySQL 不会再进行文件排序。

6、 WITH ROLLUP 超级聚合，可以挪到应用程序处理

优化 LIMIT 分页

1、 LIMIT 偏移量大的时候，查询效率较低

2、 可以记录上次查询的最大 ID，下次查询时直接根据该 ID 来查询

优化 UNION 查询

UNION ALL 的效率高于 UNION

优化 WHERE 子句

多数数据库都是从左往右的顺序处理条件的，把能够过滤更多数据的条件放到前面，把过滤少的条件放在后面

SQL 语句优化的一些方法

1、 对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

2、 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num is null
-- 可以在 num 上设置默认值 0，确保表中 num 列没有 null 值，然后这样查询：
select id from t where num=0
```

3、 应尽量避免在 where 子句中使用!=或<>操作符，否则引擎将放弃使用索引而进行全表扫描。

4、 应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num=10 or num=20
```

```
-- 可以这样查询:
select id from t where num=10 union all select id from t where num=20
```

5、in 和 not in 也要慎用，否则会导致全表扫描，如：

```
select id from t where num in(1,2,3)
-- 对于连续的数值，能用 between 就不要用 in 了:
select id from t where num between 1 and 3
```

6、 下面的查询也将导致全表扫描：select id from t where name like '%李%'若要提高效率，可以考虑全文检索。

7、 如果在 where 子句中使用参数，也会导致全表扫描。因为 SQL 只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然 而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：

```
select id from t where num=@num
-- 可以改为强制查询使用索引:
select id from t with(index(索引名)) where num=@num
```

8、 应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：

```
select id from t where num/2=100
-- 应改为:
select id from t where num=100*2
```

9、 应尽量避免在 where 子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：

```
select id from t where substring(name,1,3)='abc'
-- name以abc开头的id应改为:
select id from t where name like 'abc%'
```

10、 不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

为什么要优化

1、 系统的吞吐量瓶颈往往出现在数据库的访问速度上

2、 随着应用程序的运行，数据库的中的数据会越来越多，处理时间会相应变慢

3、 数据是存放在磁盘上的，读写速度无法和内存相比

优化原则：减少系统瓶颈，减少资源占用，增加系统的反应速度

数据库结构优化

1、 一个好的数据库设计方案对于数据库的性能往往会起到事半功倍的效果。

2、 需要考虑数据冗余、查询和更新的速度、字段的数据类型是否合理等多方面的内容。

将字段很多的表分解成多个表

1、 对于字段较多的表，如果有些字段的使用频率很低，可以将这些字段分离出来形成新表。

2、 因为当一个表的数据量很大时，会由于使用频率低的字段的存在而变慢。

增加中间表

1、 对于需要经常联合查询的表，可以建立中间表以提高查询效率。

2、 通过建立中间表，将需要通过联合查询的数据插入到中间表中，然后将原来的联合查询改为对中间表的查询。

增加冗余字段

设计数据表时应尽量遵循范式理论的规约，尽可能的减少冗余字段，让数据库设计看起来精致、优雅。但是，合理的加入冗余字段可以提高查询速度。

表的规范化程度越高，表和表之间的关系越多，需要连接查询的情况也就越多，性能也就越差。

注意：

冗余字段的值在一个表中修改了，就要想办法在其他表中更新，否则就会导致数据不一致的问题。

MySQL 数据库 cpu 飙升到 500%的话他怎么处理?

1、 当 cpu 飙升到 500%时，先用操作系统命令 top 命令观察是不是 MySQLd 占用导致的，如果不是，找出占用高的进程，并进行相关处理。

2、 如果是 MySQLd 造成的， show processlist，看看里面跑的 session 情况，是不是有消耗资源的 sql 在运行。找出消耗高的 sql，看看执行计划是否准确， index 是否缺失，或者实在是数据量太大造成。

3、 一般来说，肯定要 kill 掉这些线程(同时观察 cpu 使用率是否下降)，等进行相应的调整(比如说加索引、改 sql、改内存参数)之后，再重新跑这些 SQL。

4、 也有可能是每个 sql 消耗资源并不多，但是突然之间，有大量的 session 连进来导致 cpu 飙升，这种情况就需要跟应用一起来分析为何连接数会激增，再做出相应的调整，比如说限制连接数等

大表怎么优化？分库分表了是怎么做的？分表分库了有什么问题？有用到中间件么？他们的原理知道么？

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；

还有就是通过分库分表的方式进行优化，主要有垂直分区、垂直分表和水平分区、水平分表

垂直分区

1、 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。

2、 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。

垂直拆分的优点：

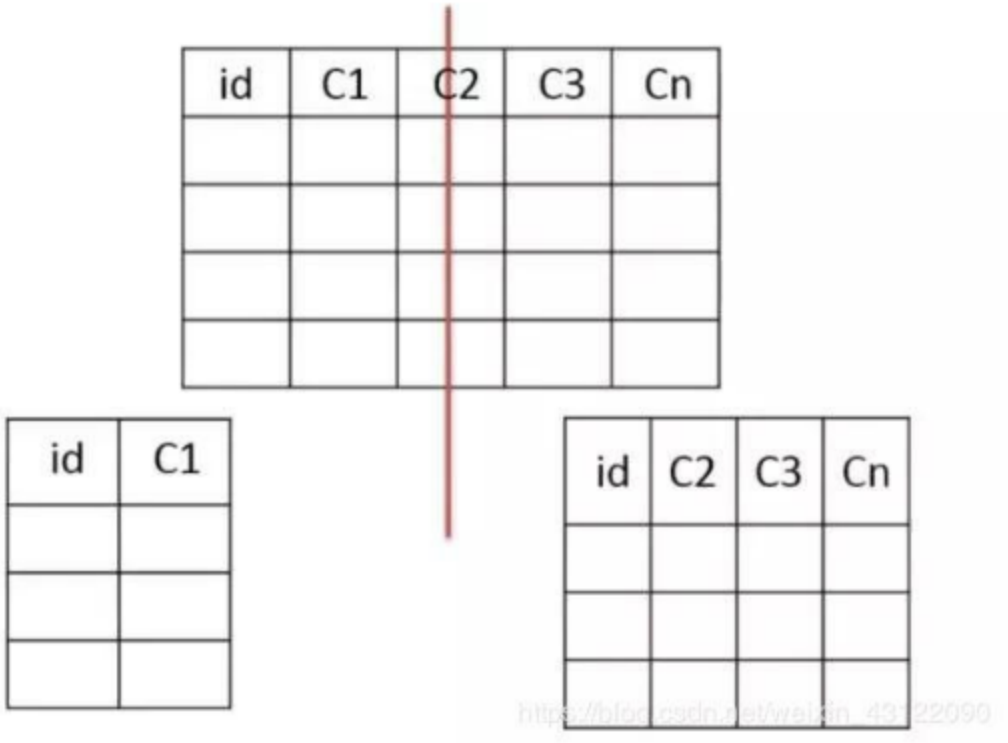
可以使得行数据变小，在查询时减少读取的 Block 数，减少 I/O 次数。此外，垂直分区可以简化表的结构，易于维护。

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；

垂直分表

把主键和一些列放在一个表，然后把主键和另外的列放在另一个表中



适用场景

1、 如果一个表中某些列常用，另外一些列不常用

2、 可以使数据行变小，一个数据页能存储更多数据，查询时减少 I/O 次数

缺点

1、 有些分表的策略基于应用层的逻辑算法，一旦逻辑算法改变，整个分表逻辑都会改变，扩展性较差

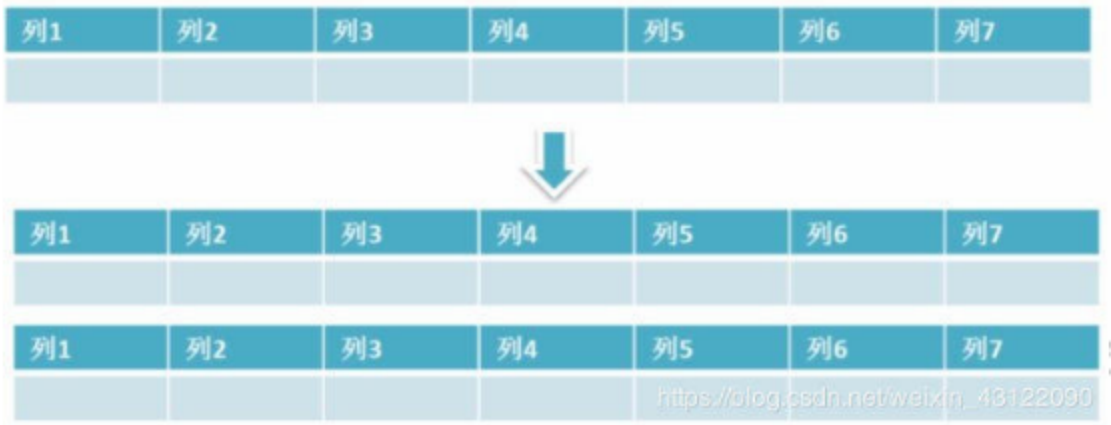
2、 对于应用层来说，逻辑算法增加开发成本

3、 管理冗余列，查询所有数据需要 join 操作

水平分区

1、 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。

2、 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。



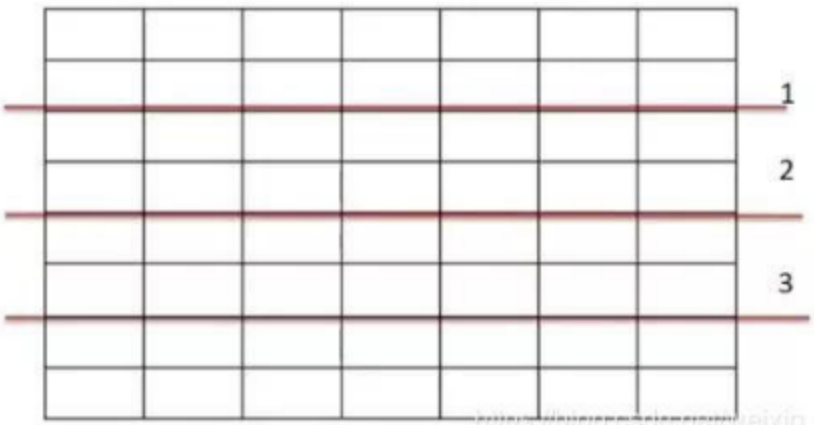
1、 水品拆分可以支持非常大的数据量。需要注意的一点是:分表仅仅是解决了单一表数据过大的问题,但由于表的数据还是在同一台机器上,其实对于提升 MySQL 并发能力没有什么意义，所以 **水平拆分最好分库** 。

2、 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。

水平分表：

表很大，分割后可以降低在查询时需要读的数据和索引的页数，同时也降低了索引的层数，提高查询次数



适用场景

1、 表中的数据本身就有独立性，例如表中分表记录各个地区的数据或者不同时期的数据，特别是有些数据常用，有些不常用。

2、 需要把数据存放在多个介质上。

水平切分的缺点

1、 给应用增加复杂度，通常查询时需要多个表名，查询所有数据都需 UNION 操作

2、 在许多数据库应用中，这种复杂度会超过它带来的优点，查询时会增加读一个索引层的磁盘次数

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。

分库分表后面临的问题

事务支持

分库分表后，就成了分布式事务了。如果依赖数据库本身的分布式事务管理功能去执行事务，将付出高昂的性能代价； 如果由应用程序去协助控制，形成程序逻辑上的事务，又会造成编程方面的负担。

跨库 join

只要是进行切分，跨节点 Join 的问题是不可避免的。但是良好的设计和切分却可以减少此类情况的发生。解决这一问题的普遍做法是分两次查询实现。在第一次查询的结果集中找出关联数据的 id,根据这些 id 发起第二次请求得到关联数据。
分库分表方案产品

跨节点的 count,order by,group by 以及聚合函数问题

这些是一类问题，因为它们都需要基于全部数据集合进行计算。多数的代理都不会自动处理合并工作。解决方案：与解决跨节点 join 问题的类似，分别在各个节点上得到结果后在应用程序端进行合并。和 join 不同的是每个结点的查询可以并行执行，因此很多时候它的速度要比单一大表快很多。但如果结果集很大，对应用程序内存的消耗是一个问题。

数据迁移，容量规划，扩容等问题

来自淘宝综合业务平台团队，它利用对 2 的倍数取余具有向前兼容的特性（如对 4 取余得 1 的数对 2 取余也是 1）来分配数据，避免了行级别的数据迁移，但是依然需要进行表级别的迁移，同时对扩容规模和分表数量都有限制。总得来说，这些方案都不是十分的理想，多多少少都存在一些缺点，这也从一个侧面反映出了 Sharding 扩容的难度。

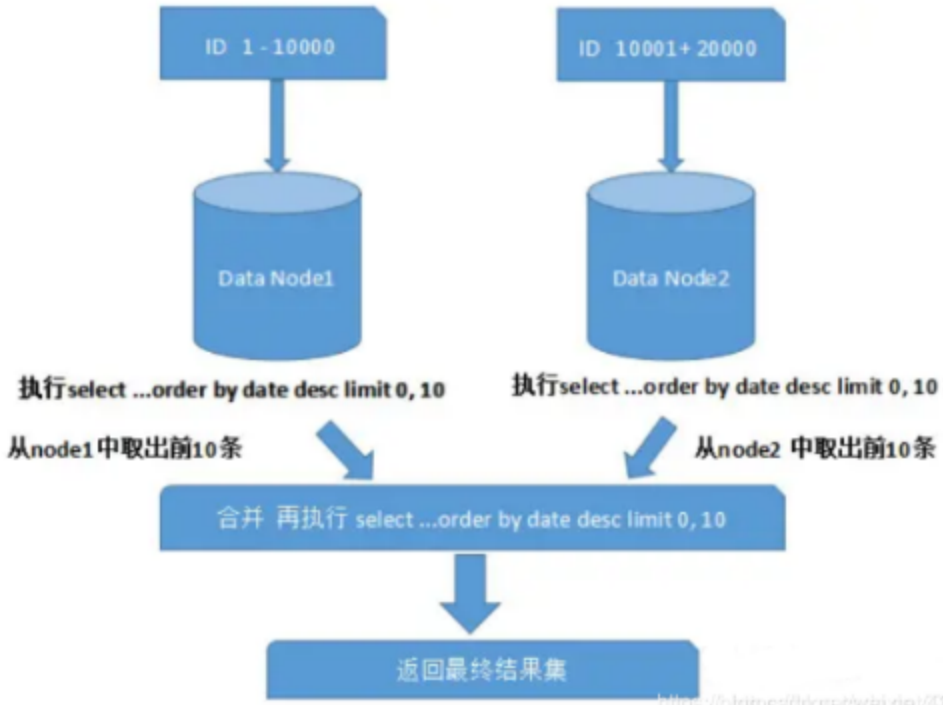
ID 问题

一旦数据库被切分到多个物理结点上，我们将不能再依赖数据库自身的主键生成机制。一方面，某个分区数据库自生成的 ID 无法保证在全局上是唯一的；另一方面，应用程序在插入数据之前需要先获得 ID,以便进行 SQL 路由、一些常见的主键生成策略

UUID 使用 UUID 作主键是最简单的方案，但是缺点也是非常明显的。由于 UUID 非常的长，除占用大量存储空间外，最主要的问题是在索引上，在建立索引和基于索引进行查询时都存在性能问题。 **Twitter 的分布式自增 ID 算法 Snowflake** 在分布式系统中，需要生成全局 UID 的场合还是比较多的，twitter 的 snowflake 解决了这种需求，实现也还是很简单的，除去配置信息，核心代码就是毫秒级时间 41 位 机器 ID 10 位 毫秒内序列 12 位。

跨分片的排序分页问题

一般来讲，分页时需要按照指定字段进行排序。当排序字段就是分片字段的时候，我们通过分片规则可以比较容易定位到指定的分片，而当排序字段非分片字段的时候，情况就会变得比较复杂了。为了最终结果的准确性，我们需要在不同的分片节点中将数据进行排序并返回，并将不同分片返回的结果集进行汇总和再次排序，最后再返回给用户。如下图所示：



MySQL 的复制原理以及流程

主从复制：

将主数据库中的 DDL 和 DML 操作通过二进制日志（BINLOG）传输到从数据库上，然后将这些日志重新执行（重做）；从而使得从数据库的数据与主数据库保持一致。

主从复制的作用

1、 主数据库出现问题，可以切换到从数据库。

2、 可以进行数据库层面的读写分离。

3、 可以在从数据库上进行日常备份。

MySQL 主从复制解决的问题

1、 数据分布：随意开始或停止复制，并在不同地理位置分布数据备份

2、 负载均衡：降低单个服务器的压力

3、 高可用和故障切换：帮助应用程序避免单点失败

4、 升级测试：可以用更高版本的 MySQL 作为从库

MySQL 主从复制工作原理

1、 在主库上把数据更高记录到二进制日志

2、 从库将主库的日志复制到自己的中继日志

3、 从库读取中继日志的事件，将其重放到从库数据中

基本原理流程，3 个线程以及之间的关联

主：

binlog 线程——记录下所有改变了数据库数据的语句，放进 master 上的 binlog 中；

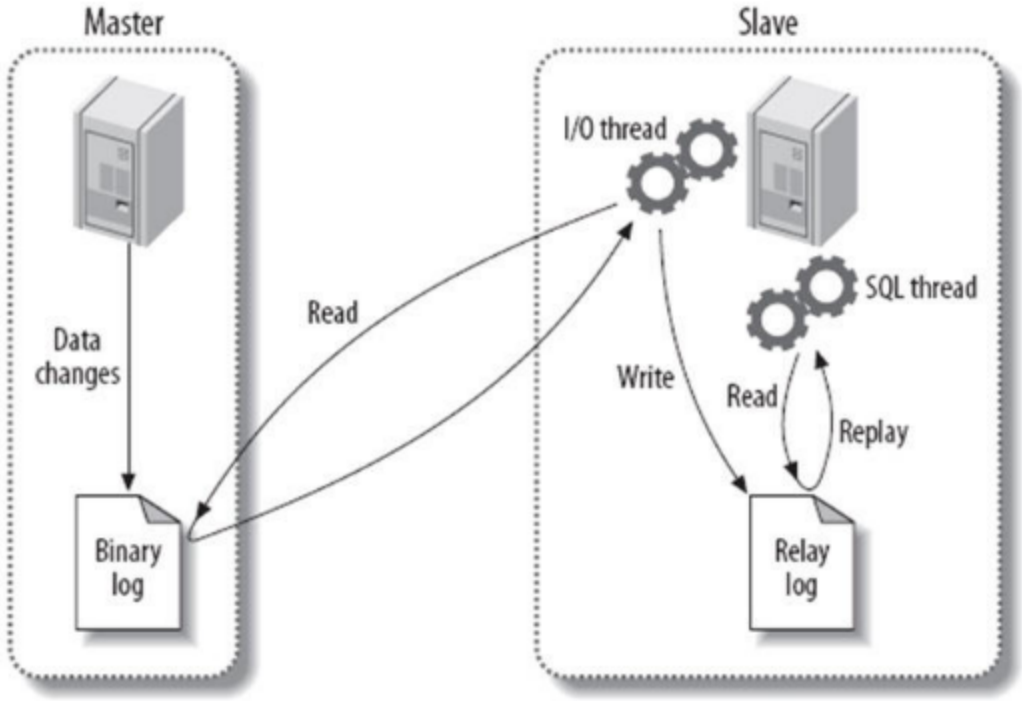
从：

io 线程——在使用 start slave 之后，负责从 master 上拉取 binlog 内容，放进自己的 relay log 中；

从：

sql 执行线程——执行 relay log 中的语句；

复制过程



Binary log：主数据库的二进制日志

Relay log：从服务器的中继日志

1、 master 在每个事务更新数据完成之前，将该操作记录串行地写入到 binlog 文件中。

2、 salve 开启一个 I/O Thread，该线程在 master 打开一个普通连接，主要工作是 binlog dump process。如果读取的进度已经跟上了 master，就进入睡眠状态

并等待 master 产生新的事件。I/O 线程最终的目的是将这些事件写入到中继日志中。

3、 SQL Thread 会读取中继日志，并顺序执行该日志中的 SQL 事件，从而与主数据库中的数据保持一致。

读写分离有哪些解决方案?

读写分离是依赖于主从复制，而主从复制又是为读写分离服务的。因为主从复制要求 slave 不能写只能读（如果对 slave 执行写操作，那么 show slave status 将会呈现 Slave_SQL_Running=NO，此时你需要按照前面提到的手动同步一下 slave）。

方案一

使用 MySQL-proxy 代理

优点：

直接实现读写分离和负载均衡，不用修改代码，master 和 slave 用一样的帐号，MySQL 官方不建议实际生产中使用

缺点：

降低性能， 不支持事务

方案二

1、 使用 AbstractRoutingDataSource+aop+annotation 在 dao 层决定数据源。

2、 如果采用了 mybatis， 可以将读写分离放在 ORM 层，比如 mybatis 可以通过 mybatis plugin 拦截 sql 语句，所有的 insert/update/delete 都访问 master 库，所有的 select 都访问 salve 库，这样对于 dao 层都是透明。 plugin 实现时可以通过注解或者分析语句是读写方法来选定主从库。不过这样依然有一个问题，也就是不支持事务，所以我们还需要重写一下 DataSourceTransactionManager，将 read-only 的事务扔进读库， 其余的有读有写的扔进写库。

方案三

1、 使用 AbstractRoutingDataSource+aop+annotation 在 service 层决定数据源，可以支持事务.

2、 缺点：类内部方法通过 this.xx()方式相互调用时，aop 不会进行拦截，需进行特殊处理。

备份计划，MySQLdump 以及 xtranbackup 的实现原理

备份计划

1、 视库的大小来定，一般来说 100G 内的库，可以考虑使用 MySQLdump 来做，因为 MySQLdump 更加轻巧灵活，备份时间选在业务低峰期，可以每天进行都进行全量备份(MySQLdump 备份出来的文件比较小，压缩之后更小)。

2、 100G 以上的库，可以考虑用 xtranbackup 来做，备份速度明显要比 MySQLdump 要快。一般是选择一周一个全备，其余每天进行增量备份，备份时间为业务低峰期。

备份恢复时间

1、 物理备份恢复快，逻辑备份恢复慢

2、 这里跟机器，尤其是硬盘的速率有关系，以下列举几个仅供参考

3、 20G 的 2 分钟（MySQLdump）

4、 80G 的 30 分钟(MySQLdump)

5、 111G 的 30 分钟（MySQLdump）

6、 288G 的 3 小时（xtra）

7、 3T 的 4 小时（xtra）

8、 逻辑导入时间一般是备份时间的 5 倍以上

备份恢复失败如何处理

首先在恢复之前就应该做足准备工作，避免恢复的时候出错。比如说备份之后的有效性检查、权限检查、空间检查等。如果万一报错，再根据报错的提示来进行相应的调整。

MySQLdump 和 xtrabackup 实现原理

MySQLdump

MySQLdump 属于逻辑备份。加入–single-transaction 选项可以进行一致性备份。后台进程会先设置 session 的事务隔离级别为 RR(SET SESSION TRANSACTION ISOLATION LEVELREPEATABLE READ)，之后显式开启一个事务(START TRANSACTION /*!40100 WITH CONSISTENTSNAPSHOT */)，这样就保证了该事务里读到的数据都是事务事务时候的快照。之后再把表的数据读取出来。如果加上–master-data=1 的话，在刚开始的时候还会加一个数据库的读锁

(FLUSH TABLES WITH READ LOCK),等开启事务后，再记录下数据库此时 binlog 的位置(showmaster status)，马上解锁，再读取表的数据。等所有的数据都已经导完，就可以结束事务

Xtrabackup:

xtrabackup 属于物理备份，直接拷贝表空间文件，同时不断扫描产生的 redo 日志并保存下来。最后完成 innodb 的备份后，会做一个 flush engine logs 的操作(老版本在有 bug，在 5.6 上不做此操作会丢数据)，确保所有的 redo log 都已经落盘(涉及到事务的两阶段提交

- 概念，因为 xtrabackup 并不拷贝 binlog，所以必须保证所有的 redo log 都落盘，否则可能会丢最后一组提交事务的数据)。这个时间点就是 innodb 完成备份的时间点，数据文件虽然不是一致性的，但是有这段时间的 redo 就可以让数据文件达到一致性(恢复的时候做的事情)。然后还需要 flush tables with read lock，把 myisam 等其他引擎的表给备份出来，备份完后解锁。这样就做到了完美的热备。

数据表损坏的修复方式有哪些?

使用 myisamchk 来修复，具体步骤：

1、 修复前将 MySQL 服务停止。

2、 打开命令行方式，然后进入到 MySQL 的/bin 目录。

3、 执行 myisamchk –recover 数据库所在路径/*.MYI

使用 repair table 或者 OPTIMIZE table 命令来修复，REPAIR TABLE table_name 修复表 OPTIMIZE TABLE table_name 优化表 REPAIR TABLE 用于修复被破坏的表。 OPTIMIZE TABLE 用于回收闲置的数据库空间，当表上的数据行被删除时，所占据的磁盘空间并没有立即被回收，使用了 OPTIMIZE TABLE 命令后这些空间将被回收，并且对磁盘上的数据行进行重排（注意：是磁盘上，而非数据库）`