数据库设计规范

Mysql 数据库设计规范

一、设计规范

1.【推荐】字段允许适当冗余，以提高查询性能，但必须考虑数据一致。冗余字段应遵循：

- 不是频繁修改的字段。
- 不是 `varchar` 超长字段，更不能是 `text` 字段。

> 正例：商品类目名称使用频率高，字段长度短，名称基本一成不变，可在相关联的表中冗余存 储类目名称，避免关联查询。

2.【推荐】单表行数超过 500 万行或者单表容量超过 2GB，才推荐进行分库分表。 说明：如果预计2年后的数据量根本达不到这个级别，请不要在创建表时就分库分表。

3.【推荐】id必须是主键，每个表必须有主键，且保持增长趋势的， 小型系统可以依赖于 MySQL 的自增主键，大型系统或者需要分库分表时才使用内置的 ID 生成器

4.【强制】id类型没有特殊要求，必须使用 `bigint unsigned`，禁止使用 `int`，即使现在的数据量很小。id如果是数字类型的话，必须是8个字节。参见最后例子

- 方便对接外部系统，还有可能产生很多废数据
- 避免废弃数据对系统id的影响

- 未来分库分表，自动生成id，一般也是8个字节

5.【推荐】字段尽量设置为 NOT NULL， 为字段提供默认值。 如字符型的默认值为一个空字符值串”；数值型默认值为数值 0；逻辑型的默认值为数值 0；

6.【推荐】每个字段和表必须提供清晰的注释

7.【推荐】时间统一格式： `YYYY-MM-DD HH：MM：SS`

8.【强制】更新数据表记录时，必须同时更新记录对应的 `update_time` 字段值为当前时间，

二、命名规范

1.【强制】表达是与否概念的字段，必须使用 `xxx_flag` 的方式命名，数据类型是 `tinyint`（1表示是，0表示否)。
正例：表达逻辑删除的字段名 `delete_flag`，1 表示删除，0 表示未删除。

2.【强制】表名、字段名必须使用小写字母或数字，禁止出现数字开头，禁止两个下划线中间只 出现数字。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名称需要慎重考虑。 说明：MySQL 在 Windows 下不区分大小写，但在 Linux 下默认是区分大小写。因此，数据库 名、表名、字段名，都不允许出现任何大写字母，避免节外生枝。 正例： `health_user`， `rdc_config`， `level3_name` 反例： `HealthUser`， `rdcConfig`， `level_3_name`

3.【强制】表名不使用复数名词。 说明：表名应该仅仅表示表里面的实体内容，不应该表示实体数量，对应于 DO 类名也是单数 形式，符合表达习惯。

4.【强制】禁用保留字，如 `desc`、 `range`、 `match`、 `delayed` 等，请参考 MySQL 官方保留字。

5.【强制】主键索引名为 `pk_字段名`；唯一索引名为 `uk_字段名`；普通索引名则为 `idx_字段名`。
说明： `pk_` 即 `primary key`； `uk_` 即 `unique key`； `idx_` 即 `index` 的简称。

6.【强制】小数类型为 `decimal`，禁止使用 `float` 和 `double`。
说明： `float` 和 `double` 在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不 正确的结果。如果存储的数据范围超过 `decimal` 的范围，建议将数据拆成整数和小数分开存储。

7.【强制】如果存储的字符串长度几乎相等，使用 `char` 定长字符串类型。

8.【强制】 `varchar` 是可变长字符串，不预先分配存储空间，长度不要超过 5000，如果存储长 度大于此值，定义字段类型为 `text`，独立出来一张表，用主键来对应，避免影响其它字段索 引效率。

9.【强制】按照公司业务系统规范，表必备字段： `id, delete_flag, tenantsid, create_id, create_name, create_time, update_id, update_name, update_time`。 说明：其中id必为主键，类型为 `unsigned bigint`、单表时自增、步长为1。 `create_time`， `update_time` 的类型均为 `date_time` 类型，前者现在时表示主动创建，后者在有数据更新时被动更新。

10.【强制】所有命名必须使用全名，有默认约定的除外，如果超过 30 个字符，使用缩写，请尽量名字易懂简短，如 description --> desc；information --> info；address --> addr 等

11.【推荐】表的命名最好是加上 `业务名称_表` 的作用。 正例：health_user / trade_config

12.【推荐】库名与应用名称尽量一致。如 health

13.【推荐】如果修改字段含义或对字段表示的状态追加时，需要及时更新字段注释

14.【推荐】所有时间字段，都以 `_time` 结尾，后面加上动词的过去式，例如： `create_time`

三、类型规范

1.表示状态字段(0-255)的使用 `TINYINT UNSINGED`，禁止使用枚举类型，注释必须清晰地说明每个枚举的含义，以及是否多选等

2.表示boolean类型的都使用 `TINYINT(1)`，因为mysql本身是没有boolean类型的，在自动生成代码的时候，DO对象的字段就是boolean类型，例如 `delete_flag`；其余所有时候都使用 `TINYINT(4)`

> TINYINT(4)，这个括号里面的数值并不是表示使用多大空间存储，而是最大显示宽度，并且只有字段指定zerofill时有用，没有zerofill，(m)就是无用的，例如id BIGINT ZEROFILL NOT NULL，所以建表时就使用默认就好了，不需要加括号了，除非有特殊需求，例如TINYINT(1)代表boolean类型。

> TINYINT(1)，TINYINT(4)都是存储一个字节，并不会因为括号里的数字改变。例如TINYINT(4)存储22则会显示0022，因为最大宽度为4，达不到的情况下用0来补充。

3.【参考】合适的字符存储长度，不但节约数据库表空间、节约索引存储，更重要的是提升检索速度。

类型	字节	表示范围
tinyint	1	无符号值： 0～255；有符号值： -128~127
smallint	2	无符号值： 0～65536；有符号值： -32768~32767
mediumint	3	无符号值： 0～16777215；有符号值： -8388608~8388607
int	4	无符号值： 0~4294967295；有符号值： -2147483648~2147483647
bigint	8	无符号值： 0~((2^32×2^2)-1)；有符号值： -(2^32×2^2)/2 ~ (2^32×2^2)/2-1

4.非负的数字类型字段，都添加上 UNSINGED， 如可以使用 INT UNSINGED 字段存 IPV4

5.时间字段使用时间日期类型，不要使用字符串类型存储，日期使用DATE类型，年使用YEAR类型，日期时间使用DATETIME

6.字符串VARCHAR(N)， 其中 N表示字符个数，请尽量减少 N 的大小，参考：code VARCHAR(32)；name VARCHAR(32)；memo VARCHAR(512)；

7.Blob 和 Text 类型所存储的数据量大，删除和修改操作容易在数 据表里产生大量的碎片，避免使用 Blob 或 Text 类型

四、索引规范

1.【强制】业务上具有唯一特性的字段，即使是多个字段的组合，也必须建成唯一索引。

> 不要以为唯一索引影响了 insert 速度，这个速度损耗可以忽略，但提高查找速度是明 显的；另外，即使在应用层做了非常完善的校验控制，只要没有唯一索引，根据墨菲定律，必 然有脏数据产生。

2.【强制】超过三个表禁止 join。需要 join 的字段，数据类型必须绝对一致；多表关联查询时， 保证被关联的字段需要有索引。

> 即使双表 join 也要注意表索引、SQL 性能。

3.【强制】在 varchar 字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，根据 实际文本区分度决定索引长度即可。 说明：索引的长度与区分度是一对矛盾体，一般对字符串类型数据，长度为 20 的索引，区分度会高达 90%以上，可以使用 count(distinct left(列名， 索引长度))/count(*)的区分度来确定。

4.【强制】页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决。

> 索引文件具有 B-Tree 的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。

5.【推荐】如果有 order by 的场景，请注意利用索引的有序性。order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file_sort 的情况，影响查询性能。

> 正例：where a=? and b=? order by c； 索引：a_b_c 反例：索引中有范围查找，那么索引有序性无法利用，如：WHERE a>10 ORDER BY b； 索引 a_b 无法排序。

6.【推荐】利用覆盖索引来进行查询操作，避免回表。
说明：如果一本书需要知道第11章是什么标题，会翻开第11章对应的那一页吗?目录浏览一下就好，这个目录就是起到覆盖索引的作用。

> 正例：能够建立索引的种类：主键索引、唯一索引、普通索引，而覆盖索引是一种查询的效果，用explain的结果，extra列会出现：using index。

7.【推荐】利用延迟关联或者子查询优化超多分页场景。
说明：MySQL并不是跳过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行，返回 N 行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写。

> 正例：先快速定位需要获取的 id 段，然后再关联：

SELECT a.* FROM 表 1 a， (select id from 表 1 where 条件 LIMIT 100000，20) b where a.id=b.id

8.【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：

- 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。
- ref 指的是使用普通的索引(normal index)。
- 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。

9.【推荐】建组合索引的时候，区分度最高的在最左边。
正例：如果 where a=? and b=? ， a 列的几乎接近于唯一值，那么只需要单建 idx_a 索引即 可。

> 说明：存在非等号和等号混合判断条件时，在建索引时，请把等号条件的列前置。如：where a>? and b=? 那么即使 a 的区分度更高，也必须把 b 放在索引的最前列。

10【推荐】防止因字段类型不同造成的隐式转换，导致索引失效。

11.【参考】创建索引时避免有如下极端误解

- 宁滥勿缺。认为一个查询就需要建一个索引。
- 宁缺勿滥。认为索引会消耗空间、严重拖慢更新和新增速度。
- 抵制惟一索引。认为业务的惟一性一律需要在应用层通过“先查后插”方式解决。

12. 总结

- 索引占磁盘空间，不要重复的索引，尽量短
  - 只给常用的查询条件加索引
  - 过滤性高的列建索引，取值范围固定的列不建索引
  - 唯一的记录添加唯一索引
  - 频繁更新的列不要建索引
  - 不要对索引列运算
  - 同样过滤效果下，保持索引长度最小
  - 合理利用组合索引，注意索引字段先后顺序
  - 多列组合索引，过滤性高的字段最前
  - `order by` 字段建立索引，避免 `filesort`
  - 组合索引，不同的排序顺序不能使用索引
  - <>!=无法使用索引

五、SQL规范

1.【强制】不要使用 count(列名)或 count(常量)来替代 `count(*)`， `count(*)`是 SQL92 定义的 标准统计行数的语法，跟数据库无关，跟 NULL 和非 NULL 无关。

> count(*)会统计值为 NULL 的行，而 count(列名)不会统计此列为 NULL 值的行。

2.【强制】count(distinct col) 计算该列除 NULL 之外的不重复行数，

> count(distinct col1， col2) 如果其中一列全为NULL，那么即使另一列有不同的值，也返回为0。

3.【强制】当某一列col的值全是 NULL 时，count(col)的返回结果为 0，但 sum(col)的返回结果为 NULL，因此使用 sum()时需注意 NPE 问题。

> 正例：可以使用如下方式来避免sum的NPE问题：SELECT IF(ISNULL(SUM(g))，0，SUM(g)) FROM table；

4.【强制】使用 ISNULL()来判断是否为 NULL 值。 说明：NULL 与任何值的直接比较都为 NULL。

- NULL<>NULL的返回结果是NULL，而不是false。
- NULL=NULL的返回结果是NULL，而不是true。
- NULL<>1的返回结果是NULL，而不是true。

5.【强制】 在代码中写分页查询逻辑时，若 count 为 0 应直接返回，避免执行后面的分页语句。

6.【强制】不得使用外键与级联，一切外键概念必须在应用层解决。 说明：以学生和成绩的关系为例，学生表中的 student_id 是主键，那么成绩表中的 student_id 则为外键。如果更新学生表中的 student_id，同时触发成绩表中的 student_id 更新，即为 级联更新。外键与级联更新适用于单机低并发，不适合分布式、高并发集群；级联更新是强阻 塞，存在数据库更新风暴的风险；外键影响数据库的插入速度。

7.【强制】禁止使用存储过程，存储过程难以调试和扩展，更没有移植性。

8.【强制】数据订正时，删除和修改记录时，要先 select，避免出现误删除，确认无误才能执行更新语句。

9.【推荐】in操作能避免则避免，若实在避免不了，需要仔细评估 in 后边的集合元素数量，控
制在 1000 个之内。

10.【参考】如果有全球化需要，所有的字符存储与表示，均以 utf-8 编码，注意字符统计函数 的区别。

> 说明：

SELECT LENGTH(“轻松工作”)； 返回为12
SELECT CHARACTER_LENGTH(“轻松工作”)； 返回为4 如果需要存储表情，那么选择 utfmb4 来进行存储，注意它与 utf-8 编码的区别。

11.【参考】TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少，但 TRUNCATE 无事务且不触发 trigger，有可能造成事故，故不建议在开发代码中使用此语句。 说明：TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同。

12.【推荐】不要写一个大而全的数据更新接口。传入为 POJO 类，不管是不是自己的目标更新字 段，都进行 update table set c1=value1，c2=value2，c3=value3； 这是不对的。执行 SQL 时，不要更新无改动的字段，一是易出错；二是效率低；三是增加 binlog 存储。

13.总结

```
• 能够快速缩小结果集的 WHERE 条件写在前面，如果有恒量条 件，也尽量放在前面 ，例如 where 1=1
• 避免使用 GROUP BY、DISTINCT 等语句的使用，避免联表查 询和子查询
• 能够使用索引的字段尽量进行有效的合理排列
• 针对索引字段使用 >， >=， =， <， <=， IF NULL 和 BETWEEN 将会 使用索引，如果对某个索引字段进行 LIKE 查
询，使用 LIKE ‘%abc%’不能使用索引，使用 LIKE ‘abc%’将能够使用索引
• 如果在 SQL 里使用了 MySQL部分自带函数，索引将失效
• 避免直接使用 select *，只取需要的字段，增加使用覆盖索引使用的可能
• 对于大数据量的查询，尽量避免在 SQL 语句中使用 order by 字
句
• 连表查询的情况下，要确保关联条件的数据类型一致，避免嵌
套子查询
• 对于连续的数值，使用 between 代替 in
• where 语句中尽量不要使用 CASE 条件
• 当只要一行数据时使用 LIMIT 1
```

创建表示例:

```
CREATE TABLE `kcf_application_auth` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT COMMENT '序号',
  `app_id` varchar(64) NOT NULL COMMENT '应用id',
  `app_name` varchar(255) NOT NULL COMMENT '应用名称',
  `enable_flag` tinyint unsigned NOT NULL DEFAULT 1 COMMENT '状态,0:停用,1:启用',
  `effective_date` datetime DEFAULT NULL COMMENT '生效日',
  `invalidation_date` datetime DEFAULT NULL COMMENT '失效日',
  `delete_flag` tinyint unsigned NOT NULL DEFAULT 0 COMMENT '是否删除，0-未删除，1-删除，默认为0',
  `tenantsid` bigint(32) NOT NULL COMMENT '租户 sid',
  `create_id` varchar(64) NOT NULL COMMENT '创建人 id',
  `create_name` varchar(64) NOT NULL COMMENT '创建人 name',
  `create_time` datetime DEFAULT CURRENT_TIMESTAMP COMMENT ' 创建时间',
  `update_id` varchar(64) DEFAULT NULL COMMENT '操作人 id',
  `update_name` varchar(64) DEFAULT NULL COMMENT '操作人 name',
  `update_time` datetime DEFAULT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`)
) COMMENT='KCF 授权表';
```

MongoDB 数据库设计规范

一、库设计规范

1. 【强制】数据库命名规范：db_xxxx
2. 【强制】库名全部小写，禁止使用任何_以外的特殊字符，禁止使用数字打头的库名，如：123_abc；

说明：库以文件夹的形式存在，使用特殊字符或其它不规范的命名方式会导致命名混乱

3. 【强制】数据库名称最多为 64 个字符。
4. 【强制】在创建新的库前应尽量评估该库的体积、QPS等，提前与DBA讨论是应该新建一个库还是专门为该库创建一个新的集群。

二、集合设计规范

1.【强制】集合名全部小写，禁止使用任何_以外的特殊字符，禁止使用数字打头的集合名，如：123_abc，禁止system打头; system是系统集合前缀；

2.【强制】集合名称最多为64字符；

3.【建议】一个库中写入较大的集合会影响其它集合的读写性能，如果业务比较繁忙的集合在一个DB中，建议最多80个集合，同时也要考虑磁盘I/O的性能；

4.【建议】如果评估单集合数据量较大，可以将一个大表拆分为多个小表，然后将每一个小表存放在独立的库中或者sharding分表；

5.【建议】MongoDB的集合拥有”自动清理过期数据”的功能，只需在该集合中文档的时间字段增加一个TTL索引即可实现该功能，但需要注意的是该字段的类型则必须是mongoDate()，一定要结合实际业务设计是否需要；

6.【建议】设计轮询集合—集合是否设计为Capped限制集，一定要结合实际业务设计是否需要。

创建集合规则

不同的业务场景是可以使用不同的配置;

```
db.createCollection("logs",
{ "storageEngine": { "wiredTiger":
{ "configString": "internal_page_max=16KB,
leaf_page_max=16KB,leaf_value_max=8KB,os_cache_max=1GB"} }
})
```

a. 如果是读多写少的表在创建时我们可以尽量将 page size 设置的比较小，比如 16KB，如果表数据量不大(“internal_page_max=16KB，leaf_page_max=16KB，leaf_value_max=8KB，os_cache_max=1GB”)

b. 如果这个读多写少的表数据量比较大，可以为其设置一个压缩算法，例如：”block_compressor=zlib，internal_page_max=16KB，leaf_page_max=16KB，leaf_value_max=8KB”

c. 注意：该zlib压缩算法不要使用，对cpu消耗特别大，如果使用snapp消耗20% cpu，而且使用zlib能消耗90%cpu，甚至100%

d. 如果是写多读少的表，可以将 leaf_page_max 设置到 1MB，并开启压缩算法，也可以为其制定操作系统层面 page cache 大小的 os_cache_max 值，让它不会占用太多的 page cache 内存，防止影响读操作

读多写少的表 internal_page_max=16KB 默认为4KB leaf_page_max=16KB 默认为32KB leaf_value_max=8KB 默认为64MB os_cache_max=1GB 默认为0 读多写少的表 而且数据量比较大 block_compressor=zlib 默认为snappy internal_page_max=16KB 默认为4KB leaf_page_max=16KB 默认为32KB leaf_value_max=8KB 默认为64M

三、文档设计规范

1.【强制】集合中的 key 禁止使用任何 “_”（下划线）以外的特殊字符。

2.【强制】尽量将同样类型的文档存放在一个集合中，将不同类型的文档分散在不同的集合中；相同类型的文档能够大幅度提高索引利用率，如果文档混杂存放则可能会出现查询经常需要全表扫描的情况；

3.【建议】禁止使用 `_id`，如：向 `_id` 中写入自定义内容；
说明：MongoDB的表与InnoDB相似,都是索引组织表,数据内容跟在主键后,而 *id是MongoDB中的默认主键,一旦* id的值为非自增,当数据量达到一定程度之后,每一次写入都可能导致主键的二叉树大幅度调整,这将是一个代价极大的写入, 所以写入就会随着数据量的增大而下降,所以一定不要在_id中写入自定义的内容。

4.【建议】尽量不要让数组字段成为查询条件;

5.【建议】如果字段较大，应尽量压缩存放;

不要存放太长的字符串，如果这个字段为查询条件，那么确保该字段的值不超过1KB；MongoDB的索引仅支持1K以内的字段，如果你存入的数据长度超过1K，那么它将无法被索引

6.【建议】尽量存放统一了大小写后的数据；

7.【建议】如果评估单集合数据量较大，可以将一个大表拆分为多个小表，然后将每一个小表存放在独立的库中或者sharding分表。

四、索引设计规范

1.【强制】MongoDB 的组合索引使用策略与 MySQL 一致，遵循”最左原则”；

2.【强制】索引名称长度不要超过 128 字符;

3.【强制】应尽量综合评估查询场景,通过评估尽可能的将单列索引并入组合索引以降低所以数量，结合1，2点;

4.【建议】优先使用覆盖索引;

5.【建议】创建组合索引的时候，应评估索引中包含的字段，尽量将数据基数大(唯一值多的数据)的字段放在组合索引的前面；

6.【建议】MongoDB 支持 TTL 索引，该索引能够按你的需要自动删除XXX秒之前的数据并会尽量选择在业务低峰期执行删除操作；看业务是否需要这一类型索引；

7.【建议】在数据量较大的时候，MongoDB 索引的创建是一个缓慢的过程，所以应当在上线前或数据量变得很大前尽量评估，按需创建会用到的索引；

8.【建议】如果你存放的数据是地理位置信息，比如：经纬度数据。那么可以在该字段上添加 MongoDB 支持的地理索引：2d 及 2dsphere，但他们是不同的，混用会导致结果不准确。

五、API使用规范

1.【强制】在查询条件的字段或者排序条件的字段上必须创建索引；

2.【强制】查询结果只包含需要的字段，而不查询所有字段；

3.【强制】在文档级别更新是原子性的，这意味着一条更新 10 个文档的语句可能在更新 3 个文档后由于某些原因失败。应用程序必须根据自己的策略来处理这些失败；

4.【建议】单个文档的BSON size不能超过16M；

5.【建议】禁用不带条件的update、remove或者find语句；

6.【建议】限定返回记录条数，每次查询结果不超过 2000 条。如果需要查询 2000 条以上的数据，在代码中使用多线程并发查询；

7.【建议】在写入数据的时候，如果你需要实现类似 MySQL 中 `INSERT INTO ON DUPLICATE KEY UPDATE` 的功能，那么可以选择 upsert() 函数；

8.【建议】写入大量数据的时候可以选择使用 batchInsert，但目前 MongoDB 每一次能够接受的最大消息长度为48MB，如果超出48MB，将会被自动拆分为多个48MB的消息;

9.【建议】索引中的-1和1是不一样的，一个是逆序，一个是正序，应当根据自己的业务场景建立适合的索引排序，需要注意的是{a:1,b:-1} 和 {a:-1,b:1}是一样的；

10.【建议】在开发业务的时候尽量检查自己的程序性能,可以使用 explain() 函数检查你的查询执行详情，另外 hint() 函数相当于 MySQL 中的 force index()；

11.【建议】如果你结合体积大小/文档数固定，那么建议创建 capped（封顶）集合，这种集合的写入性能非常高并无需专门清理老旧数据，需要注意的是 capped 表不支持remove() 和 update()操作;

12.【建议】查询中的某些操作符可能会导致性能低下，如ne，not，exists，nin，or，尽量在业务中不要使用；

> exist：因为松散的文档结构导致查询必须遍历每一个文档
> ne：如果当取反的值为大多数，则会扫描整个索引
> not：可能会导致查询优化器不知道应当使用哪个索引，所以会经常退化为全表扫描
> nin：全表扫描
> or：有多少个条件就会查询多少次，最后合并结果集，所以尽可能的使用in

13.【建议】不要一次取出太多的数据进行排序，MongoDB 目前支持对32MB以内的结果集进行排序，如果需要排序，那么请尽量限制结果集中的数据量；

14.【建议】MongoDB 的聚合框架非常好用，能够通过简单的语法实现复杂的统计查询，并且性能也不错；

15.【建议】如果需要清理掉一个集合中的所有数据，那么 remove() 的性能是非常低下的，该场景下应当使用 drop()；remove() 是逐行操作，所以在删除大量数据的时候性能很差；

16.【建议】在使用数组字段做为查询条件的时候，将与覆盖索引无缘;这是因为数组是保存在索引中的，即便将数组字段从需要返回的字段中剔除，这样的索引仍然无法覆盖查询；

17.【建议】在查询中如果有范围条件,那么尽量和定值条件放在一起进行过滤,并在创建索引的时候将定值查询字段放在范围查询字段前。

六、连接规范

1.【强制】正确连接副本集，副本集提供了数据的保护、高可用和灾难恢复的机制。如果主节点宕 机，其中一个从节点会自动提升为从节点。

2.【建议】合理控制连接池的大小，限制连接数资源，可通过Connection String URL中的 maxPoolSize 参数来配置连接池大小。

3.【建议】复制集读选项 默认情况下，复制集的所有读请求都发到Primary，Driver可通过设置的Read Preference 来将 读请求路由到其他的节点。

Elasticsearch开发规范

一、集群部署优化建议

1. 选择合理的硬件配置：尽可能使用SSD

Elasticsearch 最大的瓶颈往往是磁盘读写性能，尤其是随机读取性能。使用SSD（PCI-E接口SSD卡/SATA接口SSD盘）通常比机械硬盘（SATA盘/SAS盘）查询速度快5~10倍，写入性能提升不明显。

对于文档检索类查询性能要求较高的场景，建议考虑 SSD 作为存储，同时按照 1:10 的比例配置内存和硬盘。

对于日志分析类查询并发要求较低的场景，可以考虑采用机械硬盘作为存储，同时按照 1:50 的比例配置内存和硬盘。

单节点存储数据建议在2TB以内，不要超过5TB，避免查询速度慢、系统不稳定。

2. 给JVM配置机器一半的内存，但是不建议超过32G

修改 `**conf/jvm.options**` 配置，-Xms 和 -Xmx 设置为相同的值，推荐设置为机器内存的一半左右，剩余一半留给操作系统缓存使用。

JVM 内存建议不要低于 2G，否则有可能因为内存不足导致 ES 无法正常启动或内存溢出，**JVM 建议不要超过 32G，否则 JVM 会禁用内存对象指针压缩技术，造成内存浪费。**

机器内存大于 64G 内存时，推荐配置 -Xms30g -Xmx30g。

JVM 堆内存较大时，内存垃圾回收暂停时间比较长，建议配置 ZGC 或 G1 垃圾回收算法。

3. 规模较大的集群配置专有主节点，避免脑裂问题

Elasticsearch 主节点负责集群元信息管理、index 的增删操作、节点的加入剔除，定期将最新的集群状态广播至各个节点。

在集群规模较大时，建议配置专有主节点只负责集群管理，不存储数据，不承担数据读写压力。

具体配置如下：

```
# 专有主节点配置(conf/elasticsearch.yml):
node.master: true
node.data: false
node.ingest: false
# 数据节点配置(conf/elasticsearch.yml):
node.master: false
node.data: true
node.ingest: true
复制代码
```

Elasticsearch 默认每个节点既是候选主节点，又是数据节点。最小主节点数量参数 **minimum_master_nodes** 推荐配置为候选主节点数量一半以上，该配置告诉 Elasticsearch 当没有足够的 master 候选节点的时候，不进行 master 节点选举，等 master 节点足够了才进行选举。

例如对于 3 节点集群，最小主节点数量从默认值 1 改为 2。

```
# 最小主节点数量配置(conf/elasticsearch.yml):
discovery.zen.minimum_master_nodes: 2
复制代码
```

4. Linux操作系统调优

关闭交换分区，防止内存置换降低性能。

具体配置如下：

```
# 将 /etc/fstab 文件中包含swap的行注释掉
sed -i '/swap/s/^/#/' /etc/fstab
swapoff -a
# 单用户可以打开的最大文件数量，可以设置为官方推荐的65536或更大些
echo "* - nofile 65536" >> /etc/security/limits.conf
# 单用户线程数调大
echo "* - nproc 131072" >> /etc/security/limits.conf
# 单进程可以使用的最大map内存区域数量
echo "vm.max_map_count = 655360" >> /etc/sysctl.conf
# 参数修改立即生效
sysctl -p
复制代码
```

二、索引性能调优建议

1. 严格设置存储和索引开关，不要将数据全部存储或者全部索引

Elasticsearch支持持久化和索引，但是Elasticsearch主要场景是全文检索，成本昂贵。建议根据业务只将需要索引的Field进行索引，不仅节省索引成本，而且在操作时速度更快、效率更高！

2. 禁止使用多type索引，避免索引稀疏

Elasticsearch支持对同一个索引，划分不同的type进行读写，数据以type级别进行隔离。但是Elasticsearch多type索引共用_id，其version往往容易产生冲突，而且6.x版本开始逐渐转向单type索引的设计，7.x之后将不再支持多type索引。

3. 分片数根据业务数据量和集群规模进行设置

Elasticsearch默认分片数是5， 不同的业务索引分片数应该根据实际情况进行设置，如果索引数据量特别小且未来的增量有限，可以适当减少分片数，因为分片作为Elasticsearch集群的元数据在分片数很多的情况下，集群的同步管理和故障恢复会存在一定的风险。

如果索引数据量特别大，需要根据数据量和集群规模进行设置，一个分片建议在10亿级别，最大数据量建议是40亿。同时分片数也受集群规模影响，集群节点数太少不宜设置过多的分片数，会导致同一个索引的多个分片存在于一个Elasticsearch节点中，出现故障恢复较慢。

您可以在集群节点上保存的分片数量与您可用的堆内存大小成正比，但这在Elasticsearch中没有的固定限制。 **一个很好的经验法则是：确保每个节点的分片数量保持在低于每1GB堆内存对应集群的分片在20-25之间。** 因此，具有30GB堆内存的节点最多可以有600-750个分片，但是进一步低于此限制，您可以保持更好。 这通常会帮助群体保持处于健康状态。

4. 副本数不大于3

Elasticsearch默认副本数是1，即存在1个primary和1个replica共两份数据，主备形式同步，为了提升数据的安全性，避免数据丢失，可以设置副本数为2，即存在1个primary和两个replica共三份数据。但是没有必要设置过大的副本数，Elasticsearch的复制会对集群性能产生影响，特别是refresh_interval设置比较小的集群。

5. 禁止自动创建索引

Elasticsearch的索引默认可以自动创建，必须禁用掉。

自动创建索引无法避免误操作带来的元数据治理问题，设置`action.auto_create_index`为`false`。

6. 创建完索引，禁用掉自动类型mapping

Elasticsearch的索引默认可自动类型mapping，必须禁用掉。自动创建mapping会带来类型与实际类型不匹配的问题，设置`index.mapper.dynamic`为`false`。

7. 字符串mapping设置，优先使用keyword

Elasticsearch的索引默认支持keyword和text，text用于分词场景，keyword不支持分词，不使用分词时，需要明确指定类型为keyword。

8. mapping设置，慎用嵌套类型

Elasticsearch的索引默认支持mapping嵌套，正常情况下慎用嵌套类型，如果使用嵌套类型，建议深度不要超过2，嵌套类型文档内部更新无法原子更新。

9. join设计，慎用nested和parent-child

Elasticsearch的索引默认不支持join，建议应用程序自身处理。如果需要在父子两个文档都需要进行过滤，或者需要返回父子两个文档的field，建议使用nested，其他场景建议使用parent-child。

通常情况下，nested比parent-child查询快5-10倍，但parent-child更新文档比较方便。

10. 索引字段数不要太多，字段类型占用内存越少越好

Elasticsearch的索引最多默认支持1000个字段，Elasticsearch的字段数越多，对于集群缓存的消耗越严重，通常建议不要超过50个字段。

建议设计过程中，非索引和聚合字段不要存放到Elasticsearch中，可以使用其他存储引擎，比如mysql、hbase等。

11. 禁用掉`_all`和`_source`

Elasticsearch的索引默认开启 *all和* source，除了正常的存储和索引之外，还存放着原始写入记录、其他的元数据信息，这些信息可以直接使用正常的存储和索引filed进行替代，禁用掉可以提升集群性能，也可以在进行查询和搜索时进行exclude或者include设置。

12. 设置合理延迟分片平衡时间

Elasticsearch的集群对网络的稳定性有很大的依赖，默认延迟分片平衡时间是1m，即副本在1分钟内恢复到集群，则不会进行重新分片。如果运维过程中，断网时间较长，建议修改该参数，避免来回创建分片、移动分片，为运维和节点异常恢复提供缓冲时间。