【强制】：① 存储引擎必须使用 InnoDB

解读： InnoDB 支持事物、行级锁、并发性能更好，CPU 及内存缓存页优化使得资源利用率更高。

【强制】：②每张表必须设置一个主键 ID，且这个主键 ID 使用自增主键（在满足需要的情况下尽量短），除非在分库分表环境下

解读： 由于 InnoDB 组织数据的方式决定了需要有一个主键，而且若是这个主键 ID 是单调递增的可以有效提高插入的性能，避免过多的页分裂、减少表碎片提高空间的使用率。

而在分库分表环境下，则需要统一来分配各个表中的主键值，从而避免整个逻辑表中主键重复。

【强制】：③必须使用 utf8mb4 字符集

解读： 在 MySQL 中的 UTF-8 并非"真正的 UTF-8"，而 utf8mb4"才是真正的"UTF-8"。

【强制】：④数据库表、表字段必须加入中文注释

解读： 大家都别懒。

【强制】：⑤库名、表名、字段名均小写，下划线风格，不超过 32 个字符，必须见名知意，禁止拼音英文混用

解读：约定。

【强制】：⑥单表列数目必须小于 30，若超过则应该考虑将表拆分

解读： 单表列数太多使得 MySQL 服务器处理 InnoDB 返回数据之间的映射成本太高。

【强制】：⑦禁止使用外键，如果有外键完整性约束，需要应用程序控制

解读： 外键会导致表与表之间耦合，UPDATE 与 DELETE 操作都会涉及相关联的表，十分影响 SQL 的性能，甚至会造成死锁。

【强制】：⑧必须把字段定义为 NOT NULL 并且提供默认值

解读：

- NULL 的列使索引/索引统计/值比较都更加复杂，对 MySQL 来说更难优化。
- NULL 这种类型 MySQL 内部需要进行特殊处理，增加数据库处理记录的复杂性；同等条件下，表中有较多空字段的时候，数据库的处理性能会降低很多。
- NULL 值需要更多的存储空，无论是表还是索引中每行中的 NULL 的列都需要额外的空间来标识。

【强制】：⑨禁用保留字，如 DESC、RANGE、MARCH 等

解读： 请参考 MySQL 官方保留字。

【强制】：⑩如果存储的字符串长度几乎相等，使用 CHAR 定长字符串类型

解读：能够减少空间碎片，节省存储空间。

【建议】：⑪ 在一些场景下，考虑使用 TIMESTAMP 代替 DATETIME

解读：

- 这两种类型的都能表达"yyyy-MM-dd HH:mm:ss"格式的时间，TIMESTAMP 只需要占用 4 个字节的长度，可以存储的范围为（1970-2038）年，在各个时区，所展示的时间是不一样的。
- 而 DATETIME 类型占用 8 个字节，对时区不敏感，可以存储的范围为（1001-9999）年。

【建议】：⑫当心自动生成的 Schema，建议所有的 Schema 手动编写

解读： 对于一些数据库客户端不要太过信任。

SQL 规约

【建议】：①为了充分利用缓存，不允许使用自定义函数、存储函数、用户变量

解读：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，其查询结果都不会被缓存。

比如函数 NOW() 或者 CURRENT_DATE() 会因为不同的查询时间，返回不同的查询结果。

【强制】：②在查询中指定所需的列，而不是直接使用“ *”返回所有的列

解读：

- 读取不需要的列会增加 CPU、IO、NET 消耗。
- 不能有效的利用覆盖索引。

【强制】：③不允许使用属性隐式转换

解读： 假设我们在手机号列上添加了索引，然后执行下面的 SQL 会发生什么？

explain SELECT user_name FROM parent WHERE phone=13812345678；很明显就是索引不生效，会全表扫描。

【建议】：④在 WHERE 条件的属性上使用函数或者表达式

解读： MySQL 无法自动解析这种表达式，无法使用到索引。

【强制】： ⑤禁止使用外键与级联，一切外键概念必须在应用层解决

解读： 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。

【建议】：⑥应尽量避免在 WHERE 子句中使用 or 作为连接条件

解读： 根据情况可以选择使用 UNION ALL 来代替 OR。

【强制】：⑦不允许使用 % 开头的模糊查询

解读： 根据索引的最左前缀原理，%开头的模糊查询无法使用索引，可以使用 ES 来做检索。

索引规约

【建议】：①避免在更新比较频繁、区分度不高的列上单独建立索引

解读： 区分度不高的列单独创建索引的优化效果很小，但是较为频繁的更新则会让索引的维护成本更高。

【强制】：②JOIN 的表不允许超过五个。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

解读： 太多表的 JOIN 会让 MySQL 的优化器更难权衡出一个“最佳”的执行计划（可能性为表数量的阶乘），同时要注意关联字段的类型、长度、字符编码等等是否一致。

【强制】：③在一个联合索引中，若第一列索引区分度等于 1，那么则不需要建立联合索引

解读： 索引通过第一列就能够完全定位的数据，所以联合索引的后边部分是不需要的。

【强制】：④建立联合索引时，必须将区分度更高的字段放在左边

解读： 区分度更高的列放在左边，能够在一开始就有效的过滤掉无用数据。提高索引的效率，相应我们在 Mapper 中编写 SQL 的 WHERE 条件中有多个条件时，需要先看看当前表是否有现成的联合索引直接使用，注意各个条件的顺序尽量和索引的顺序一致。

【建议】：⑤利用覆盖索引来进行查询操作，避免回表

解读： 覆盖查询即是查询只需要通过索引即可拿到所需 DATA，而不再需要再次回表查询，所以效率相对很高。

我们在使用 EXPLAIN 的结果，extra 列会出现："using index"。这里也要强调一下不要使用“SELECT * ”，否则几乎不可能使用到覆盖索引。

【建议】：⑥在较长 VARCHAR 字段，例如 VARCHAR(100) 上建立索引时，应指定索引长度，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

解读： 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，若长度为 20 的索引，区分度会高达 90% 以上，则可以考虑创建长度例为 20 的索引，而非全字段索引。

例如可以使用 SELECT COUNT(DISTINCT LEFT(lesson_code, 20))/COUNT(*) FROM lesson；来确定 lesson_code 字段字符长度为 20 时文本区分度。

【建议】：⑦如果有 ORDER BY 的场景，请注意利用索引的有序性

ORDER BY 最后的字段是联合索引的一部分，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。

解读：

- 假设有查询条件为 WHERE a=? and b=? ORDER BY c；存在索引：a_b_c，则此时可以利用索引排序。
- 反例：在查询条件中包含了范围查询，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。

【建议】：⑧在 Where 中索引的列不能某个表达式的一部分，也不能是函数的参数

解读： 即是某列上已经添加了索引，但是若此列成为表达式的一部分、或者是函数的参数，MySQL 无法将此列单独解析出来，索引也不会生效。

【建议】：⑨我们在 Where 条件中使用范围查询时，索引最多用于一个范围条件，超过一个则后边的不走索引

解读： MySQL 能够使用多个范围条件里边的最左边的第一个范围查询，但是后边的范围查询则无法使用。

【建议】：⑩在多个表进行外连接时，表之间的关联字段类型必须完全一致

解读： 当两个表进行 Join 时，字段类型若没有完全一致，则加索引也不会生效，这里的完全一致包括但不限于字段类型、字段长度、字符集、Collection 等等。