数据库设计规范

MySQL 版

隆承志　收集整理

目录

1 数据库开发设计规范

1.1 术语说明

- 强制：必须遵守，如果需要违背，需要规范负责人确认
- 推荐：最佳实践，建议遵守。如果有更好的方案，与规范负责人确认，更新文档。
- 参考：可选遵守。如果有更好的方案，与规范负责人确认，更新文档。

1.2 数据库开发规范

开发规范本身也包含几部分：基本命名和约束规范，建表规范、字段设计规范，索引规范，SQL 规范等

1.3 规范存在意义

- 确保在开发成员或开发团队之间的工作可以顺利交接，不必花很大的力气便能理解已编写的数据库对象
- 保证线上数据库 schema 规范
- 提高程序的稳定性，减少出问题概率
- 方便自动化管理
- 规范需要长期坚持，对开发和 DBA 是一个双赢的事情
- 提升性能

1.4 约束规范

【强制】表字符集选择 UTF8 ，如果需要存储 emoj 表情，需要使用 UTF8mb4(MySQL 5.5.3 以后支持)

【强制】存储引擎使用 InnoDB

【推荐】变长字符串尽量使用 VARCHAR VARBINARY

【推荐】字段数据类型长度选择遵守够用最小原则。

【强制】禁止在线上做数据库压力测试

【建议】禁止从测试、开发环境直连数据库

【推荐】同种字段保证数据类型和长度的一致性

【强制】禁止在表定义时显式指定 字段/表 的字符集以及字符集校验规则。原因是数据库字符集默认为 utf8，字符集默认校验规则为 utf8_general_ci，在 DB 服务器部署标准里已经做了定义。

【强制】表结构定义禁止指定索引类型以及索引排序规则

【强制】禁止在数据库中存储明文密码

1.5 命名规范

【强制】库名、表名、字段名、索引名使用名词作为数据库名称，并且只用英文，不用中文拼音

【推荐】库名使用英文字母，全部小写，控制在 3-7 个字母以内

【强制】库名、表名、字段名禁止使用保留字。如 desc、range、match、delayed 等，请参考 MySQL 官方保留字。

【强制】库名、表名、字段名、索引名使用小写字母，以下划线分割 ，需要见名知意

【强制】库名、表名、字段名、索引名不要设计过长，禁止超过 32 个字符，尽可能用最少的字符表达出表的用途

【推荐】临时库、临时表名必须以 tmp 为前缀，并以日期为后缀

【推荐】备份库、表必须以 bak 为前缀，并以日期为后缀

【强制】禁止在数据库里定义数据约束/字段主外键关系，由程序维护

1.6 建表规范

在设计数据库表间的关系时也要遵循相同原则，职责分离降低耦合，但同时要考虑到性能情况，做到适当冗余而不导致修改逻辑复杂。

【推荐】库名与应用名称尽量一致。

【推荐】表的命名最好是加上“业务名称_表的作用”。

【强制】表名不使用复数名词。

【强制】表达是与否概念的字段，必须使用 is_xxx 的方式命名，数据类型是 unsigned tinyint （ 1 表示是，0 表示否）。

【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：

1）不是频繁修改的字段。

2）不是 varchar 超长字段，更不能是 text 字段。

【推荐】表中的第一个 id 字段一定是主键。关于主键：

- 表必须有主键：数据库直接复制需要必须有主键
- 值域范围够用且存储长度越短
- 数值的有序性增加
- 不使用更新频繁的列
- 尽量不选择字符串列
- 不使用 UUID MD5 HASH
- 默认使用非空的唯一键
- 建议选择自增或发号器
- 禁止使用 varchar 类型作为主键语句设计

【强制】表必须包含 create_time 和 modify_time 字段，即表必须包含记录创建时间和修改时间的字段。

【推荐】时间字段，除特殊情况一律采用 int 来记录 unix_timestamp

- 存储年使用 YEAR 类型。
- 存储日期使用 DATE 类型。

- 存储时间（精确到秒）建议使用 TIMESTAMP 类型，因为 TIMESTAMP 使用 4 字节，DATETIME 使用 8 个字节。

datetime 与 timestamp 有什么不同?

相同点：TIMESTAMP 列的显示格式与 DATETIME 列相同。显示宽度固定在 19 字符，并且格式为 YYYY-MM-DD HH:MM:SS。

不同点：

- TIMESTAMP4 个字节储存，时间范围：1970-01-01 08:00:01 ~ 2038-01-19 11:14:07 值以 UTC 格式保存，涉及时区转化 ，存储时对当前的时区进行转换，检索时再转换回当前的时区。
- datetime8 个字节储存，时间范围：1000-01-01 00:00:00 ~ 9999-12-31 23:59:59
- 实际格式储存，与时区无关

如何使用 TIMESTAMP 的自动赋值属性?

将当前时间作为 ts 的默认值：ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP。

当行更新时，更新 ts 的值：ts TIMESTAMP DEFAULT ON UPDATE CURRENT_TIMESTAMP。

可以将 1 和 2 结合起来：ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP。

【推荐】字段均应用 COMMENT 列属性来描述此表、字段所代表的真正含义，如枚举值则建议将该字段中使用的内容都定义出来。此外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

【推荐】所有字段均定义为 NOT NULL ，最好指定默认值

【推荐】表必须包含 is_del，用来标示数据是否被删除，原则上数据库数据不允许物理删除

【推荐】能用 int 的就不用 char 或者 varchar

【推荐】对表加新字段，不允许指定字段位置

【推荐】能用 tinyint 的就不用 int

【推荐】使用 UNSIGNED 存储非负数值。

【推荐】不建议使用 ENUM、SET 类型，使用 TINYINT 来代替

【推荐】使用短数据类型，比如取值范围为 0-80 时，使用 TINYINT UNSIGNED

【推荐】存储精确浮点数必须使用 DECIMAL 替代 FLOAT 和 DOUBLE

【推荐】尽可能不使用 TEXT、BLOB 类型

【推荐】禁止在数据库里存图片/邮件正文/文章/等大数据。建议使用其他方式存储（TFS/SFS），MySQL 只保存指针信息。必须使用 TEXT,BLOB 数据类型时，把大字段拆分到单独的表

【推荐】单条记录大小禁止超过 8k（列长度(中文)*3(UTF8)+列长度(英文)*1）

【推荐】使用 UNSIGNED INT 存储 IPv4 地址而不是 CHAR(15)，通过 MySQL 函数 inet_ntoa 和 inet_aton 来进行转化。ipv6 地址目前没有转化函数，需要使用 DECIMAL 或者两个 bigINT 来存储。

【推荐】使用 VARCHAR 存储变长字符串，当然要注意 varchar(M)里的 M 指的是字符数不是字节

【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。

什么时候用 CHAR，什么时候用 VARCHAR?

CHAR 和 VARCHAR 类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。CHAR 和 VARCHAR 类型声明的长度表示你想要保存的最大字符数。例如，CHAR(30)可以占用 30 个字符。

CHAR 列的长度固定为创建表时声明的长度。长度可以为从 0 到 255 的任何值。当保存 CHAR 值时，在它们的右边填充空格以达到指定的长度。当检索到 CHAR 值时，尾部的空格被删除掉。在存储或检索过程中不进行大小写转换。

VARCHAR 列中的值为可变长字符串。长度可以指定为 0 到 65,535 之间的值。(VARCHAR的最大有效长度由最大行大小和使用的字符集确定。整体最大长度是65,532字节）。

同 CHAR 对比，VARCHAR 值保存时只保存需要的字符数，另加一个字节来记录长度(如果列声明的长度超过 255，则使用两个字节)。VARCHAR 值保存时不进行填充。当值保存和检索时尾部的空格仍保留，符合标准 SQL。char 适合存储用户密码的 MD5 哈希值，它的长度总是一样的。对于经常改变的值，char 也好于 varchar,因为固定长度的行不容易产生碎片，对于很短的列，char 的效率也高于 varchar。char(1)字符串对于单字节字符集只会占用一个字节，但是 varchar(1)则会占用 2 个字节，因为 1 个字节用来存储长度信息。

【推荐】整形定义中不添加长度，比如使用 INT，而不是 INT[4]

INT[M]****，M 值代表什么含义？

- 注意数值类型括号后面的数字只是表示宽度而跟存储范围没有关系。很多人他们认为 INT(4)和 INT(10)其取值范围分别是 (-9999 到 9999)和(-9999999999 到 9999999999)，这种理解是错误的。其实对整型中的 M 值与 ZEROFILL 属性结合使用时可以实现列值等宽。不管 INT[M]中 M 值是多少，其取值范围还是 (-2147483648 到 2147483647 有符号时)，(0 到 4294967295 无符号时)。
- 显示宽度并不限制可以在列内保存的值的范围,也不限制超过列的指定宽度的值的显示。当结合可选扩展属性 ZEROFILL 使用时默认补充的空格用零代替。例如：对于声明为 INT(5) ZEROFILL 的列，值 4 检索为 00004。请注意如果在整数列保存超过显示宽度的一个值,当 MySQL 为复杂联接生成临时表时会遇到问题，因为在这些情况下 MySQL 相信数据适合原列宽度，如果为一个数值列指定 ZEROFILL，MySQL 自动为该列添加 UNSIGNED 属性。

1.7 索引规范

【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

【强制】禁止在区分度很差的字段上建索引

【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。说明：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

【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

说明：即使双表 join 也要注意表索引、SQL 性能。

【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。

【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。

说明：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。

【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。

【推荐】核心 SQL 优先考虑覆盖索引，利用覆盖索引来进行查询操作，避免回表。

【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

【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。

【推荐】单个索引字段数不超过 5，单表索引数量不超过 5，索引设计遵循 B+ Tree 索引最左前缀匹配原则

为什么一张表中不能存在过多的索引？

InnoDB 的 secondary index 使用 b+tree 来存储，因此在 UPDATE、DELETE、INSERT 的时候需要对 b+tree 进行调整，过多的索引会减慢更新的速度。

【建议】索引根据左前缀原则

【推荐】建组合索引的时候，区分度最高的在最左边。

【推荐】议优先考虑前缀索引，必要时可添加伪列并建立索引

【推荐】建立的索引能覆盖 80%主要的查询，不求全，解决问题的主要矛盾

【建议】DML 和 order by 和 group by 字段要建立合适的索引

【推荐】防止因字段类型不同造成的隐式转换，导致索引失效。隐式转换例子，字段定义为 varchar，但传入的值是个 int，就会导致全表扫描，要求程序端要做好类型检查

【参考】创建索引时避免有如下极端误解： 1）宁滥勿缺。认为一个查询就需要建一个索引。 2）宁缺勿滥。认为索引会消耗空间、严重拖慢更新和新增速度。 3）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。

【强制】禁止使用触发器

【推荐】区分度最大的字段放在前面

【推荐】选择筛选性更优的字段放在最前面，比如单号、userid 等，type，status 等筛选性一般不建议放在最前面

【推荐】重要的 SQL 必须被索引:UPDATE、DELETE 语句的 WHERE 条件列;ORDER BY、GROUP BY、DISTINCT 的字段; 多表 JOIN 的字段

【推荐】不要索引常用的小型表

【推荐】不在低基数列上建立索引，例如“性别”

【推荐】不在索引列进行数学运算和函数运算

【推荐】尽量不使用外键 ，外键用来保护参照完整性，可在业务端实现；对父亲和子表的操作会相互影响，降低可用性 ； INNODB 本身对 online DDL 的限制

- 外键用来保护参照完整性，
- 可在业务端实现对父表和子表的操作会相互影响，
- 降低可用性 INNODB 本身对 online DDL 的限制

【建议】不使用%前导的查询，如 like “%ab”

【建议】不使用负向查询，如 not in/like ″无法使用索引，导致全表扫描

【建议】合理创建联合索引(避免冗余),(a,b,c) 相当于 (a) 、(a,b) 、(a,b,c)

MYSQL 中索引的限制

- MYISAM 存储引擎索引长度的总和不能超过 1000 字节
- BLOB 和 TEXT 类型的列只能创建前缀索引
- MYSQL 目前不支持函数索引
- 使用不等于 (!= 或者 <>) 的时候，MYSQL 无法使用索引。
- 过滤字段使用函数运算 (如 abs (column)) 后，MYSQL 无法使用索引。
- join 语句中 join 条件字段类型不一致的时候 MYSQL 无法使用索引
- 使用 LIKE 操作的时候如果条件以通配符开始 (如 ‘%abc…’)时，MYSQL 无法使用索引。
- 使用非等值查询的时候，MYSQL 无法使用 Hash 索引。

1.8 SQL类规范

【强制】程序里不允许出现 DELETE * FROM T_TEST;TRUNCATE TABLE T_TEST;DROP TABLE T_TEST;DROP DATABASE 等高危操作。说明: 对数据的大批量修改/删除要有 DBA 参与确定处理方案。对库结构的任何修改必须通过 DBA 审核。

【强制】不允许出现 INSERT ••• SELECT •••.。注：INSERT ••• SELECT •••会对 SELECT 的表记录加 X 锁，影响并发。

【建议】GROUP BY 去除排序

【建议】不带条件的 COUNT(*)统计必须缓存。说明：INNODB 表的 COUNT(*)需要全表扫描，表越大效率越低，要控制 COUNT(*)次数

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

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

【强制】当某一列的值全是 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。

【强制】不允许在 IN 列表里做查询。用 JOIN 替换。

【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。

【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。 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。外键对于主备复制是禁止的，有了外键，主备复制只能是单线程。

【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。

【强制】数据订正（特别是删除、修改记录操作）时，要先 select，避免出现误删除，确认无误才能执行更新语句。

【推荐】使用 in 代替 or，in 操作也要能避免则避免，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。

【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。

【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。

【推荐】表之间的关联用 ON 而不是 WHERE

【推荐】子查询都改成连表查询（JOIN）

【建议】 一次返回给客户端的结果集不超过 100 行记录。超过部分分页处理

【推荐】不允许出现 REPLACE INTO …。注：代码判断是否记录是否存在，再确定做 UPDATE 还是 INSERT

【推荐】多表连接查询，最大连接表数不允许超过 3 个。注：打散大查询。

【推荐】使用预编译语句，只传参数，比传递 SQL 语句更高效，降低 SQL 注用概率

【推荐】充分利用前缀索引

【推荐】尽量不使用存储过程、触发器、函数等，让数据库做最擅长的事

【推荐】避免使用大表的 JOIN，MySQL 优化器对 join 优化策略过于简单

- MySQL 最擅长的是单表的主键/二级索引查询
- JOIN 消耗较多内存，产生临时表

【推荐】避免在数据库中进行数学运算和其他大量计算任务

- MySQL 不擅长数学运算和逻辑判断
- 无法使用索引

【推荐】SQL 合并，主要是指的 DML 时候多个 value 合并，减少和数据库交互

【推荐】合理的分页，尤其大分页

【推荐】UPDATE、DELETE 语句不使用 LIMIT ，容易造成主从不一致

【推荐】INSERT 语句使用 batch 提交(INSERT INTO table VALUES(),(),()……)，values 的个数不超过 500

【推荐】禁止使用 order by rand()

【推荐】sql 语句避免使用临时表

【推荐】使用 union all 而不是 union。从效率上说，union all 要比 union 快很多

【推荐】禁止单条 SQL 语句同时更新多个表

【强制】读取数据时，只选取所需要的列，不要每次都 SELECT *，避免产生严重的随机读问题，尤其是读到一些 TEXT/BLOB 列

【推荐】通常情况下，子查询的性能比较差，建议改造成 JOIN 写法

【推荐】多表联接查询时，关联字段类型尽量一致，并且都要有索引

【推荐】多表连接查询时，把结果集小的表（注意，这里是指过滤后的结果集，不一定是全表数据量小的）作为驱动表

【推荐】多表联接并且有排序时，排序字段必须是驱动表里的，否则排序列无法用到索引

【推荐】多用复合索引，少用多个独立索引，尤其是一些基数（Cardinality）太小（比如说，该列的唯一值总数少于 255）的列就不要创建独立索引了

【推荐】类似分页功能的 SQL，建议先用主键关联，然后返回结果集，效率会高很多

【推荐】不使用负向查询，如 not in/like

- 无法使用索引，导致全表扫描
- 全表扫描导致 buffer pool 利用率降低

1.9 ORM映射

【强制】POJO 类的布尔属性不能加 is，而数据库字段必须加 is_，要求在 resultMap 中进行字段与属性之间的映射。

【强制】不要用 resultClass 当返回参数，即使所有类属性名与数据库字段一一对应，也需要定义；反过来，每一个表也必然有一个与之对应。

【强制】不允许直接拿 HashMap 与 Hashtable 作为查询结果集的输出。

说明：resultClass=”Hashtable”，会置入字段名和属性值，但是值的类型不可控。

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

【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。

【参考】@Transactional 事务不要滥用。事务会影响数据库的 QPS，另外使用事务的地方需要考虑各方面的回滚方案，包括缓存回滚、搜索引擎回滚、消息补偿、统计修正等。

【参考】<isEqual>中的 compareValue 是与属性值对比的常量，一般是数字，表示相等时带上此条件；<isNotEmpty>表示不为空且不为 null 时执行；<isNotNull>表示不为 null 值时执行。

【强制】程序应有捕获 SQL 异常的处理机制

1.10 MySQL集群方案

- 基于主从复制；
- 基于中间件/proxy
- 基于 NDB 引擎
- 基于 Galera 协议

优先推荐 MHA：可以采用一主多从，或者双主多从的模式，这种模式下，可以采用 MHA 或 MMM 来管理整个集群，最新的 MHA 也已支持 MySQL 5.6 的 GTID 模式了

MHA 的优势很明显：

- 开源，用 Perl 开发，代码结构清晰，二次开发容易；
- 方案成熟，故障切换时，MHA 会做到较严格的判断，尽量减少数据丢失，保证数据一致性；
- 提供一个通用框架，可根据自己的情况做自定义开发，尤其是判断和切换操作步骤；
- 支持 binlog server，可提高 binlog 传送效率，进一步减少数据丢失风险。

不过 MHA 也有些限制：

- 需要在各个节点间打通 ssh 信任，这对某些公司安全制度来说是个挑战，因为如果某个节点被黑客攻破的话，其他节点也会跟着遭殃；
- 自带提供的脚本还需要进一步补充完善，当然了，一般的使用还是够用的。

1.11 分库分表

- 解决单机写入压力过大和容量问题
- 有垂直拆分和水平拆分两种方式

- 拆分要适度，切勿过度拆分
- 有中间层控制拆分逻辑最好，否则拆分过细管理成本会很高

1.12 分表规范

单表一到两年内数据量超过 500w 或数据容量超过 10G 考虑分表（如果预计三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），需提前考虑历史数据迁移或应用自行删除历史数据，采用等量均衡分表或根据业务规则分表均可。要分表的数据表必须与 DBA 商量分表策略。

- 拆分大字段和访问频率低的字段，分离冷热数据
- 使用 HASH 进行散表，表名后缀使用十进制数，下标从 0 开始
- 按日期时间分表需符合 YYYY[MM][DD][HH]格式。例如千库十表、十库百表等采用合适的分库分表策略

禁止使用分区表，分区表对分区键有严格要，分区表在表变大后执行 DDL、SHARDING、单表恢复等都变得更加困难。

1.13 数据备份

- 全量备份 VS 增量备份
- 热备 VS 冷备
- 物理备份 VS 逻辑备份
- 延时备份
- 全量 binlog 备份

建议方式：

- 热备＋物理备份
- 核心业务：延时备份＋逻辑备份
- 全量 binlog 备份

主要做的几点：

- 备份策略集中式调度管理
- xtrabackup 热备
- 备份结果统计分析
- 备份数据一致性校验
- 采用分布式文件系统存储备份

备份系统采用分布式文件系统原因：

- 解决存储分配的问题
- 解决存储 NFS 备份效率低下问题
- 存储集中式管理
- 数据可靠性更好

1.14 职责分离原则

通常指数据的产生和使用，每个系统相互独立，通常取决于以下几点

- 数据的产生：通常谁产生谁负责，维护数据的正个生命周期，产生，修改，销毁等周期。
- 使用数据者：谁使用谁维护
- 考虑高内聚，低耦合：在存放数据的时候如果考虑到数据使用原则导致了相关度非常高的数据存放在多个地方，需要多个系统来维护这个数据就有可能导致系统间的耦合性增强，应当尽量避免

1.15 在线处理和分析分离

为了保证生产环境数据处理性能，需要将一些分析相关的数据及结果单独库存储，避免在数据分析的时候导致业务数据吞吐量下降，引起系统问题。

专门用于存放离线报表数据，并提供线上数据查询方法，建议将统计结果，汇总的数据都从在线处理数据库中移走。

原则上要将在线用户请求和后台统计请求分开：

a. 将后台统计与生产库分开(一般使用 slave)，缺点是数据量大了玩不转。

b. 建立离线报表，专门存放统计结果，计算与展示异步处理，缺点是实时业务响应差。

c. 实时拉取 mysql row binlog，做数据的异构处理(tungsten, canal)，将增量结果处理后(storm)，保存在数据库中，基本实时。

1.16 事物与日志分离

通常用户生成的内容和用户行为的日志要分开：

- 游戏 DB 里存放玩家的基础信息，装备，属性，好友列表等等，这些放到数据库里面。但是玩家的行为日志，比如消耗金币，今天下过哪些副本，买过什么顶级装备，这些属于行为日志，应该单独存放并分析处理。
- 对于 web，有好多用户置顶，刷新，竞价，展示等行为，要求实时并且量很大，一定要和贴子分开。
- 行为日志，需要做分析处理，并且由于时效性不宜存储在 mysql 中，后期维护就是地雷。

1.17 历史可追溯

保障数据可追溯，应当遵循一些简单的约定，事后方便数据的查询和统计：

对于状态数据，应当设计相应状态的字段来保存该数据的最后状态，同时记录下来该数据的初始创建人，时间以及该数据的最后修改人和修改时间；所以在交易数据（如订单合同），广告数据，账户表等都应该默认有状态（status），创建人（creator/creator_name），创建时间（created_at），最后修改人（modifier/modifier_name），最后修改时间（modified_at）等字段用来表明数据的当前状态，创建信息及修改信息。

针对需要跟踪每次修改的数据，需要在数据发生变化的时候记录一张日志表，用于记录该数据发生变化的全生命周期。针对只需要关注关键字段变化的情况，则日志表中只需要记录关键字段变化即可，但操作人，操作类型，时间应当准确记录，日志表数据一旦生成不允许进行修改。如用户账户的充值流水，消费流水都是一些业务紧相关的日志。而审核日志，操作记录等日志则属于与业务关联较小的日志。

针对所有历史需要保留的数据则需要每次变化都生成一个新的版本，比如类目信息等，对原始数据永远只做 insert 操作，不做 delete 及 update 操作。但这种情况仅限于极端数据历史要求极高的情况下使用。

2 行为规范

- 批量导入、导出数据必须提前通知 DBA 协助观察
- 禁止在线上从库执行后台管理和统计类查询
- 禁止有 super 权限的应用程序账号存在
- 产品出现非数据库导致的故障时及时通知 DBA 协助排查
- 推广活动或上线新功能必须提前通知 DBA 进行流量评估
- 数据库数据丢失，及时联系 DBA 进行恢复
- 对单表的多次 alter 操作必须合并为一次操作
- 不在 MySQL 数据库中存放业务逻辑
- 重大项目的数据库方案选型和设计必须提前通知 DBA 参与
- 对特别重要的库表，提前与 DBA 沟通确定维护和备份优先级
- 不在业务高峰期批量更新、查询数据库其他规范
- 提交线上建表改表需求，必须详细注明所有相关 SQL 语句