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：

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。

（三）SQL 语句

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。

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。

2.【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。

3.【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。

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。

1） NULL<>NULL 的返回结果是 NULL，而不是 false。

2） NULL=NULL 的返回结果是 NULL，而不是 true。

3） 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（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。

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】如果有国际化需要，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 utf8mb4 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。

（四）ORM 映射

1. 【强制】在表查询中，一律不要使用 * 作为查询的字段列表，需要哪些字段必须明确写明。

说明：1）增加查询分析器解析成本。2）增减字段容易与 resultMap 配置不一致。3）无用字段增加网络消耗，尤其是 text 类型的字段。

2. 【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。

说明：参见定义 POJO 类以及数据库字段定义规定，在<resultMap>中增加映射，是必须的。在 MyBatis Generator 生成的代码中，需要进行对应的修改。

3. 【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 POJO 类与之对应。

说明：配置映射关系，使字段与 DO 类解耦，方便维护。

4. 【强制】sql.xml 配置参数使用：#{}，#param# 不要使用${} 此种方式容易出现 SQL 注入。

5. 【强制】iBATIS 自带的 queryForList(String statementName,int start,int size)不推荐使用。

说明：其实现方式是在数据库取到 statementName 对应的 SQL 语句的所有记录，再通过 subList 取 start,size 的子集合。

正例：
```
Map<String, Object> map = new HashMap<>();
map.put("start", start);
map.put("size", size);
```

6. 【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。

7. 【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。

8. 【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。

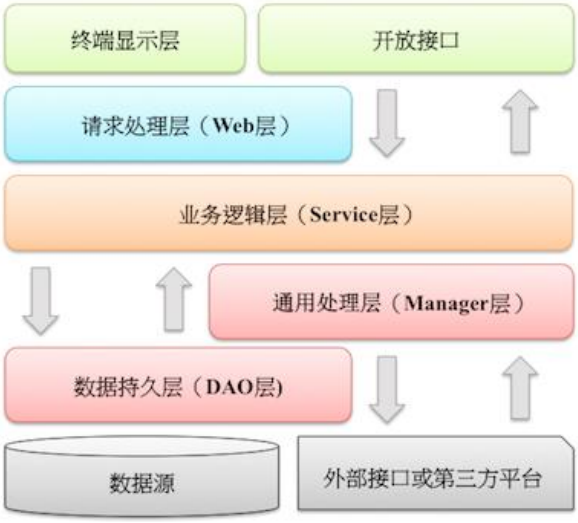
9. 【参考】@Transactional 事务不要滥用。事务会影响数据库的 QPS，另外使用事务的地方需要考虑各方面的回滚方案，包括缓存回滚、搜索引擎回滚、消息补偿、统计修正等。

10. 【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。

六、工程结构

(一)应用分层

1．【推荐】图中默认上层依赖于下层，箭头关系表示可直接依赖，如：开放接口层可以依赖于 Web 层，也可以直接依赖于 Service 层，依此类推：



- **开放接口层**：可直接封装 Service 方法暴露成 RPC 接口；通过 Web 封装成 http 接口；进行网关安全控制、流量控制等。
- **终端显示层**：各个端的模板渲染并执行显示的层。当前主要是 velocity 渲染，JS 渲染，JSP 渲染，移动端展示等。
- **Web 层**：主要是对访问控制进行转发，各类基本参数校验，或者不复用的业务简单处理等。
- **Service 层**：相对具体的业务逻辑服务层。
- **Manager 层**：通用业务处理层，它有如下特征：
  1） 对第三方平台封装的层，预处理返回结果及转化异常信息；
  2） 对 Service 层通用能力的下沉，如缓存方案、中间件通用处理；
  3） 与 DAO 层交互，对多个 DAO 的组合复用。
- **DAO 层**：数据访问层，与底层 MySQL、Oracle、Hbase 等进行数据交互。
- **外部接口或第三方平台**：包括其它部门 RPC 开放接口，基础平台，其它公司的 HTTP 接口。

2．【参考】 （分层异常处理规约）在 DAO 层，产生的异常类型有很多，无法用细粒度的异常进行 catch，使用 catch(Exception e)方式，并 throw new DAOException(e)，不需要打印日志，因为日志在 Manager/Service 层一定需要捕获并打印到日志文件中去，如果同台服务器再打日志，浪费性能和存储。在 Service 层出现异常时，必须记录出错日志到磁盘，尽可能带上参数信息，相当于保护案发现场。如果 Manager 层与 Service 同机部署，日志方式与 DAO 层处理一致，如果是单独部署，则采用与 Service 一致的处理方式。Web 层绝不应该继续往上抛异常，因为已经处于顶层，如果意识到这个异常将导致页面无法正常渲染，那么就应该直接

跳转到友好错误页面，加上用户容易理解的错误提示信息。开放接口层要将异常处理成错误码和错误信息方式返回。

3．【参考】分层领域模型规约：

- **DO**（Data Object）：此对象与数据库表结构一一对应，通过 DAO 层向上传输数据源对象。
- **DTO**（Data Transfer Object）：数据传输对象，Service 或 Manager 向外传输的对象。
- **BO**（Business Object）：业务对象，由 Service 层输出的封装业务逻辑的对象。
- **AO**（Application Object）：应用对象，在 Web 层与 Service 层之间抽象的复用对象模型，极为贴近展示层，复用度不高。
- **VO**（View Object）：显示层对象，通常是 Web 向模板渲染引擎层传输的对象。
- **Query**：数据查询对象，各层接收上层的查询请求。注意超过 2 个参数的查询封装，禁止使用 Map 类来传输。

（二）二方库依赖

1．【强制】定义 GAV 遵从以下规则：

1） **G**roupID 格式：com.{公司/BU }.业务线 [.子业务线]，最多 4 级。

说明：{公司/BU} 例如：alibaba/taobao/tmall/aliexpress 等 BU 一级；子业务线可选。

正例：com.taobao.jstorm 或 com.alibaba.dubbo.register

2） **A**rtifactID 格式：产品线名-模块名。语义不重复不遗漏，先到中央仓库去查证一下。

正例：dubbo-client / fastjson-api / jstorm-tool

3） **V**ersion：详细规定参考下方。

2．【强制】二方库版本号命名方式：主版本号.次版本号.修订号

1） **主版本号**：产品方向改变，或者大规模 API 不兼容，或者架构不兼容升级。

2） **次版本号**：保持相对兼容性，增加主要功能特性，影响范围极小的 API 不兼容修改。

3） **修订号**：保持完全兼容性，修复 BUG、新增次要功能特性等。

说明：注意起始版本号必须为：**1.0.0**，而不是 **0.0.1** 正式发布的类库必须先去中央仓库进行查证，使版本号有延续性，正式版本号不允许覆盖升级。如当前版本：1.3.3，那么下一个合理的版本号：1.3.4 或 1.4.0 或 2.0.0

3．【强制】线上应用不要依赖 SNAPSHOT 版本（安全包除外）。

说明：不依赖 SNAPSHOT 版本是保证应用发布的幂等性。另外，也可以加快编译时的打包构建。

4．【强制】二方库的新增或升级，保持除功能点之外的其它 jar 包仲裁结果不变。如果有改变，必须明确评估和验证，建议进行 dependency:resolve 前后信息比对，如果仲裁结果完全不一致，那么通过 dependency:tree 命令，找出差异点，进行<excludes>排除 jar 包。