ICS 点击此处添加 ICS 号
CCS 点击此处添加 CCS 号

T/

江 苏 省 软 件 行 业 协 会 团 体 标 准

T/JSIA 2026—XXXX

软件静态 SQL 代码质量规范

（征求意见稿）

在提交反馈意见时，请将您知道的相关专利连同支持性文件一并附上。

XXXX－XX－XX 发布　　XXXX－XX－XX 实施

江苏省软件行业协会　　发 布

目　次

前　言

本文件按照GB/T 1.1—2020《标准化工作导则　第1部分：标准化文件的结构和起草规则》的规定起草。

本文件由××××提出。
本文件由××××归口。
本文件起草单位：
本文件主要起草人：

引　言

近年来，我国软件产业持续高速发展，金融、政务、医疗等重点行业的信息化系统规模不断扩大，软件代码的复杂度和长期维护成本随之显著上升。行业质量统计数据显示，在生产环境中因代码层缺陷导致的系统故障在所有生产事故中占比突出，其中由数据库设计不规范、SQL 语句效率低下、索引滥用等数据库相关代码问题引发的缺陷又占相当比重。

在此背景下，仅靠运行期的性能测量与评级已不足以前置发现风险——质量管控需要向开发阶段前移，在代码提交、集成、发布等环节通过静态检测规则将典型缺陷拦截在投产之前。

本文件作为 GB/T 30975—2014 的补充，聚焦静态代码质量检测领域，围绕建表设计、索引使用规则、SQL 语句编写三个核心方向，建立覆盖软件开发全生命周期的质量管控技术框架与检测规则。本文件制定过程中参考了 ISO/IEC 25010 软件产品质量模型，并结合国内典型企业的工程实践经验，经多轮行业调研及试点项目验证后形成。

软件静态 SQL 代码质量规范

1 范围

本文件规定了关系型数据库相关对象的静态代码质量检测方法与技术要求，涵盖数据库表结构设计、索引规则、SQL语句编写与评审中的典型缺陷检测、质量评价方法与结果分级框架，并给出了在代码审查、持续集成及生产环境巡检等场景下的质量管控应用指引。

本文件适用于对数据一致性与查询性能要求较高的信息系统（如金融交易系统、政务服务平台、医疗信息管理系统、能源行业信息系统等）在其研发、测试及运维过程中开展的关系型数据库（如 MySQL、Oracle、PostgreSQL 等）表结构与SQL代码静态质量检测与质量管控活动。

本文件不适用于：

——非关系型/非结构化数据库（如 MongoDB、Elasticsearch 等）的查询优化与对象检测；

实时嵌入式系统的底层驱动代码检测与评价。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

ISO/ IEC 9075 SQL 标准

GB/T 30975-2014 信息技术 基于计算机的软件系统的性能测量与评级

GB/T 25000.51-2016 系统与软件工程 系统与软件质量要求和评价（SQuaRE）

ISO/ IEC 25010:2011 系统和软件质量模型

GB/T 30994-2014 关系数据库管理系统检测规范

3 术语和定义

点击此处，以便选择适当的引导语

下列术语和定义适用于本文件。

3.1 静态代码检测 static code inspection

在不执行目标软件的条件下，对源代码或编译中间表示进行词法、语法、控制流与数据流分析，以识别潜在缺陷、违规模式及质量风险的活动。

3.2 静态 SQL 性能检测 static SQL performance inspection

对软件源码或配置中提取的 SQL 语句进行语法解析与语义分析，通过识别其访问路径、算子类型、索引可见性及资源消耗模式，推断其在运行态可能出现的响应延迟、资源占用异常、数据一致性风险及查询效率低下等问题的活动。

3.3 索引有效性 index effectiveness

在给定查询场景下，已创建的索引能够被查询优化器选中并用于缩小数据访问范围、减少扫描代价，从而使查询执行代价较全表扫描或低效访问路径出现可观测下降的属性。

3.4 深度分页 deep pagination

在使用 LIMIT OFFSET（或等价语义的分页机制）进行结果集切片时，因偏移量（OFFSET）过大，导致数据库需要先顺序扫描并丢弃大量不满足返回区间的记录，从而引起查询延迟随偏移量增长显著上升的现象。

4 检测与检测内容

4.1 通则

本文件所述的静态SQL质量检测，指在不执行被测系统的条件下，对源代码、配置文件或数据库对象定义脚本中出现的SQL语句及相关数据库对象（表、索引等）进行语法解析、语义分析与模式匹配，识别其在规范性、稳定性、执行效率及可维护性方面可能存在的缺陷或风险模式。

检测结果用于为被测软件中数据库相关代码的质量评价、缺陷整改优先级排序及质量门禁判定提供依据，也可作为GB/T 30975—2014所述动态性能指标的代码层根因输入信息。

4.2 检测类别

本文件将存储类组件的静态SQL质量检测划分为以下五个类别：

——编码规范性：SQL 语句及数据库对象定义脚本在格式、命名、注释与结构组织方面应符合统一的编码约定；

——资源利用性：识别 SQL 访问路径与资源消耗模式中可能导致非必要 I/O、扫描放大或内存压力的风险写法；

——操作稳定性：识别可能引发数据一致性风险、运行时异常或锁膨胀倾向的操作模式；

——可维护性：评估 SQL 代码的可审查性与可演进性，控制复杂度、重复度及硬编码等维护性负债；

——执行效率：识别典型低效写法，为 GB/T 30975—2014 所述执行时间、吞吐量等动态指标的劣化提供代码层根因线索。

4.3 检测准备

检测实施前，宜完成与被测系统相关的背景信息收集，至少包括：

——业务场景对一致性、并发量及响应时延的敏感程度；

——存储组件类型及版本、关键业务表的数据规模量级与增长趋势；

——SQL 操作类型分布特征及被测代码/脚本的提取方式；

——部署架构特征（单库/读写分离/分库分表等），用于识别应重点关注的缺陷模式类别；

——对象间依赖关系线索（主外键、视图引用等），用于确定扫描入口与影响面。

完成上述信息收集后，应据此从 4.2 给出的检测维度中确定本次适用的检测类别组合，并形成检测方案

4.4 检测过程

4.4.1 通则

检测过程一般包括四项活动，按顺序分别是检测策划、检测准备、检测执行、检测汇总。

4.4.2 检测策划

检测策划阶段应围绕被测系统的业务场景特征开展分析，根据检测目标、系统所处研发生命周期阶段、关键业务数据规模量级、存储组件类型及版本，以及检测结果的预期用途，确定本次适用的检测类别组合与检测内容的覆盖范围，并明确各检测项的充分性判定准则；在此基础上宜选定检测实施方式，确定所用静态分析工具的版本、启用规则集标识及自定义规则来源，识别检测范围遗漏、规则误报、多模块并行开发导致版本漂移等典型风险并制定应对措施；检测活动中宜对检测执行、结果复核与过程监督的职责进行分离设置，确保检测结果的可追溯性与结论的客观性。

4.4.3 检测准备

4.4.3.1 人员准备

人员准备要求见表1

表 1 人员配置表

工作职责	人数	具体要求
检测项目负责人	1	整个检测项目的主要责任人，负责资源调度，人员协调、以及检测结果的完备性、准确性和有效性。
检测需求计划复核员	1	复核提报的检测需求计划，分析需求计划是否合理，选择检测方向是否准确、检测范围是否有效覆盖。
检测配置员	1	合理编排检测内容，根据不同检测模块的要求以及检测对象的场景因素来配置合适的参数；精准有效地使用工具来满足检测需求。
检测执行员	1	根据检测任务的编排内容与参数，执行对应的检测工具和模块，程序进行检测，同时采集检测结果并规范化。
结果审核员	1	对检测结果进行检测，避免参数配置或者潜在因素产生检测结果的偏差。
报告整合员	1	根据检测类型和场景生成相应的检测报告，最终交付给客户。

注：其中检测项目负责人和执行人员是必须人员，其他人员可以由其他岗位人员兼职。

4.4.3.2 资源配置

根据检测需求配置相应的硬件资源，如设备的型号、可并发处理能力、可处理的软件规模；同时配置软件资源，如检测模块、检测工具的选择，多模块之间协作对硬件的要求，以及结果采集方式的要求。

4.4.3.3 参数配置

参数配置应包含以下几个方面：

——根据存储数据规模配置检测规则阈值

——配置技术框架设定检测模块入参

——根据需求编排检测任务的先后关系和参数传递

——根据技术框架特性设定检测阈值

——配置模块评估数据的基准

——根据需求中存储规模的变化趋势配置对应参数体现该变化

——配置预期运行环境的参数

依据检测场景和需求，分析件内容选择对应检测工具，编排检测模块同时根据业务场景的检测需求设定模块参数。

4.4.4 检测实施

检测实施应包含以下几个步骤：

a) 需求检测计划复核人员对检测需求的待检存储语句进行复核，确认需求准确性和代码一致性；

b) 获取源码压缩包，以及对应运行环境信息文件，对待检文件一致性检查，检查通过后，通过文件指纹生成工具获取文件唯一特征码，并且作为检测唯一 ID;

c) 根据检测策划方案，申请硬件、工具、网络等资源并初始化

d) 通过制定的安全设备上传到检测设备中

e) 将解压后的文件放置到指定目录下

f) 使用存储语句提取工具提取代检测的存储语句

g) 初始化语句生产环境参数

h) 执行编排计划，使用模块参数对模块进行初始化

i) 执行具体检测模块

j) 标记检测结果的检测环境、目的、维度等数据，生成代审核的结果数据

k) 审核人员根据检测模块和环境参数对结果复核

l) 确定审核维度，应包括：

——检测模块与所需检测需求是否一致

——检测过程是否遗漏检测项

——检测结果是否准确

——结果格式是否规范

——正反例使用是否恰当
——问题适用规则是否缺失
——原因是否缺失
——问题是否定位
m） 整个检测结果问题列表（如无问题则问题列表为空无需整改，直接跳转结果生成步骤）
n） 实施人员根据问题列表重新调整后再次检测
o） 审核人员再次复核
p） 如无问题，报告整合员根据报告模板生成标准检测报告
q） 审核人员签字
r） 项目负责人确认签字
s） 走用印流程
t） 通知项目单位领取检测报告
u） 将被检测文件与报告存档，与唯一 ID 关联，30 天后自动删除

4.4.5 检测报告

检测报告主要包含几个部分：概述、检测场景与目的、检测内容引用、具体检测项列表、附件与案例引用。

4.4.5.1

唯一标识码：报告包含被检测项目存储部分的文件唯一标识码，这个编码将检测报告与检测文件强制关联，具有不可抵赖与不可更改性。

申请单位：申请单位是对代码质量要求方，对代码检测维度目标提出检测要求。

项目单位：提供被检测的存储代码的单位。

联系方式：申请单位与项目单位提供负责人姓名与相关联系方式，包括不限于手机号、微信、QQ、邮箱。

4.4.5.2 检测场景与目的

检测范围明确指向被测存储语句的实际运行场景，涵盖请求规模、硬件与网络环境配置、数据总量、调用密度分布及未来规模增长趋势等关键要素。

本次检测旨在达成以下目标：作为代码交付与验收的客观依据；量化评估数据库资源消耗，为容量成本测算提供数据支撑；综合评估存储语句在编码规范性、可读性与可维护性方面的静态质量；以及在大数据量及高负载压力下，验证其性能表现与运行的稳定性。

4.4.5.3 检测内容引用

检测内容需要包含语言类型、存储框架信息、存储系统类型如（mysql、oracle等）
被检测语句在代码中的具体位置，位置中包含具体文件相对了路径、内容在文件中的相对位置
被检测语句的具体内容片段，片段中需要包含被检测点
检测结果需有检测结果相关的依据和规则
相关依据可以是公开的各种标准，也可以是企业、行业或者组织自行制定的标准
依据与规则具体描述检测的内容，检测结果判定的方法与纠正方法

4.4.5.4 具体检测项目

每个检测结果项的主要五部分组成：存储语句片段位置、被检测存储语句内容、规则内容或依据、检测结果、完善检测。每个检测场景有特定结果项，根据实际情况对上述结果列进行调整。

a） 存储语句片段位置：
——文件路径：代码文件所在项目的项目路径，可以通过该路径获取被检测语句所在的文件
——语句行号：表示被检测语句在文件起始行与结束行，通过行号可以获取被检测的语句片段
——地址链接：通过该链接可以直接访问被检测语句片段，也可以通过该链接下载被检测语句
b） 被检测存储语句内容：
——具体内容可以是代码片段，表示具体的使用逻辑。
——以 xml/json 等形式保存的存储语句内容

——在代码中被引用的片段

c) 规则内容或依据：

——具体内容：规则内容表示对代码片段的具体要求，包含具体特征，违反的原则，可能引发的后果。

——内容表述：规则内容需要与代码实施具有高相关性，可以通过人工走查的方式复现效果；如无法复现，需描述清晰，名词使用准确；

d) 检测结果：

检测结果包含被检测内容片段，代码中存在的问题项，具体问题项详细描述了代码存在的问题。

——文本描述：通过文本描述表示代码在指定维度上检测的结果，被检测单位可以通过文本信息了解该维度的代码质量。

——量化指标：列举指定场景下，该维度的指标并量化，如临时表述数据规模、单位时间的吞吐率、稳定性波动幅度。

——问题示例：列举改代码在指定维度上可能出现的现象，可以给出相关实现。

——趋势分析：对指定维度上的趋势分析和展示，为后续预防提供依据。

e) 完善建议：

——强制修改：表明该问题发生的几率极大或者发生后具有极强的破坏力，对这类问题强制要求在上线前修改。

——建议修改：该维度上缺陷对代码运行具有较大影响，具有一定的破坏力，希望此类问题代码得到修改；除非特定的场景需求或者客观现实限制，需要保持现有方式，可不修改。

——推荐：代码可以满足现有需求，但是实现方式不是更高效的实现方式，会造成一定影响，引发严重事故的概率相对较小，此类问题的积累与叠加可能会引发更大的事故，如内存资源过度占用，多个此类应用的内存资源过度占用，可能在某个时刻导致资源击穿，导致全面宕机。需根据场景来决定是否完善该段代码。

——参考：代码可以满足现有需求，引发问题概率较小，有更好更新的方式实现，作为被检测单位的参考依据。

4.4.5.5 附件与案例引用

附件：检测报告的原始代码、报告中不好放置的文本内容、文件引用、问题描述等内容都会放到附件中，进行唯一编号，在报告中通过唯一编号关联具体文件。

案例引用：针对检测项给出具体的正例和反例，具体内容无法完整在报告中显示，那么保存内容到文件，并编号与报告的中的具体引用点关联。

5 结果评价

结果评价方法分为分级制、通过制、量化制。

5.1 分级制

分级制检测报告中应包含综合评价结论，对检测软件规范与实施质量进行评估，确定最终 软件质量。综合评价方法如下：

a) 经过静态代码质量检测或者整改后，系统代码中没有强制类规范，且建议类缺陷数量不大于 5（不含必改缺陷），则整体结论为软件质量为“高”；

b) 经过静态代码质量检测或者整改后，系统代码中没有高风险缺陷，且建议类缺陷数量大于 5 且不大于 10（不含必改缺陷），则整体结论为软件质量为“较高”；

c) 经过静态代码质量检测或者整改后，系统代码中没有高风险缺陷，且建议类缺陷数量大于 10 且不大于；或者强制类缺陷数量不大于 3 且建议类风险缺陷数量在 0 到 15 之间，则整体结论为代码安全性为“中”；

d) 经过静态代码检测或者整改后在基本项都满足的情况下，系统代码中强制类缺陷数量大于 3，或者推荐类缺陷数量大于 15，则整体结论为代码安全性为“低”；

e) 整体结论为代码质量“高”或者“较高”，则满足系统代码总体软件质量要求；

f） 检测的最终输出是检测报告，检测报告应给出各个检测项质量缺陷的检测结论，并报告信息系统代码的整体综合评价结论；

g） 必改缺陷包含：强制类

综合评价方法图表表示方法见表1。

表 2 分级制综合评价方法

安全性结论	评价要求
高	强制类缺陷个数=0,0≤建议类缺陷个数≤5
较高	强制类缺陷个数=0,5<建议类缺陷个数≤10
中	强制类缺陷个数=0, 10<建议类缺陷个数≤15 或强制类缺陷个数≤3,0≤建议类缺陷个数≤15
低	强制类个数>3 或推荐类缺陷个数>15

5.2 通过制

通过制是指检测单位列出具体检测项，以及检测项覆盖的维度，结果内容；客户单位根据自身需求选定需要检测的检测项，同时根据具体检测结果属性，设定通过与不同过的条件的条件。

5.3 量化制

量化制是指部分结果可以转换为具体的量化数据，指标量化后是对代码实现影响的直观评估；便于判断趋势、资源分配、预前方案制定、人员管理、规则制定等。

6 工具

代码质量检测工具是一类通过静态或者动态分析代码，自动发现程序中的缺陷、漏洞、规范问题与性能风险，从而提升软件质量、安全性和可维护性的工具。代码质量检测工具见表3：

表 3 代码质量检测工具

工具类型	功能与特征说明	举例	备注
静态检测工具	对软件编码质量、结构设计、资源消耗、稳定性、可维护性等维度进行检测的工具	复杂度分析、规范性分析、可维护性分析、可靠性分析、稳定性分析等工具	针对软件编码质量、结构设计、可靠性、稳定性、资源利用率、可维护性等维度检测的工具很少
支持检测过程活动的其他工具	支持静态解析、语法语义解析、结构访问的工具	检测策划、资源配置、人员安排、规则加载、检测数据一致性等工具	抽象语法树结构访问工具可以帮助引擎执行具体检测规则。

附 录 A
（资料性）
检测服务内容

A.1 基础规范

a) 低效表名：检查表名是否以数字开头、是否两个下划线中只有数字，表名是否与应用名 一致。
b) 命名大小写：检查表/字段名是否包含大小写，不同操作系统处理后会造成系统无法运 行的问题。
c) 命名包含关键字：表名或字段名包含数据库保留字（关键字）， 这个命名可能带来误 解或创建表失败。
d) 命名理解性：检查表名/字段名包含非常规缩写，理解性差命名降低协作能力，增加后 续的运维成本。
e) 索引命名：检查索引名称，是否已类型缩略前缀来命名，引发研发/运维过程中效率问 题。
f) 字段类型：检查是否 小数类型为 decimal， 禁止使用 float 和 double， 存储时精度损失问 题。
g) 必备字段：检查缺失 id,create_time, modified_time 字段的表，缺失必要字段，加重追踪效率问题。
h) 注释：检查建库语句，建库/字段是否加注释问题，缺失注释后续增加维护难度的问题。
i) 唯一索引：检查建表语句中是否包含唯一索引，没有唯一索引，数据不具备唯一性，引 发脏数据问题。
j) 默认值：检查 int/decimal/varchar 等类型是否设置默认值，没有设置默认值增加代码 复杂度和 null 值判断难度
k) 非空未设默认值(提醒)： 检查创建表时要求字段非空，但是没有设置默认值；插入时， 如未赋值产生插入异常。
l) 外键：检查建表语句中是否包含外键，通常不得使用外键与级联，一切外键概念必须在 应用层解决。
m) 数据权限：检查建表语句中为用户修改权限问题，所有权限应该有人为指定。

A.2 索引规则

a) Varchar 类型建索引：检查 Varchar 字段上建索引且未指定索引长度，根据实际需要设定 索引长度，过长内容设置全部内容为索引引发索引不稳定性。
b) 索引数量：检查建立过多索引的表，索引过多修改代价和性能太大的问题。
c) Join 表数量：检查 join 表是否超过 3 张表，需要 join 的字段类型是否保持一致问题。 左模糊匹配：检查查询语句包含左模糊匹配，左模糊匹配可能导致索引失效的问题。
d) 全模糊匹配：检查查询语句中包含全模糊匹配，全模糊匹配导致索引失效降低搜索性能 问题。
e) 查询条件顺序：检查查询条件顺序，依据最左前缀原则解析语句时，最左边的条件包含 索引失效关键字时，后续可用索引都会失效，引发性能问题。
f) 冗余索引：检查（a,b）(a)索引同时存在，且索引之间作用存在包含关系，增加维护索 引的代价问题。

A.3 语句规则

a) 使用 Distinct *：检查语句中 distinct *，未指定列进行去重可能引发性能瓶颈，仅部分去 重操作所有列导致冗余处理，对 NULL 的处理不一定符合预期问题。
b) 使用 Select *： 检查语句返回所有结果列，未明确指定需要的所有列，导致结果集庞 大、冗余数据太多、全列数据反复提取与释放增加内存资源使用的不稳定性。
c) Count(*) ：检查使用 count(列名) 代替 Count(*)情况，统计时忽略 NULL 的问题
d) Delete 条件： 检查 delete 后缺失条件的情况，缺失条件可能导致数据被误删问题。

e) Update 条件：检查 Update 后缺失条件情况，更新数据时缺失条件约束会误更新数据 导致数据不一致问题。
f) Limit 存在：检查查询语句返回结果是否有 limit 约束，没有 limit 约束的结果大小不可预 期，可能太大导致返回失败，使用内存大小波动较大引发内存击穿问题。
g) Limit 深度分页：检查查询语句中包含语句，但是是 start end 结构，不是 length 结构， 大数据量查询时发生深度分页问题。
h) Or 优化：检查条件中包含 or 并且前后条件语句可以使用索引的情况，or 将导致后一个 索引失效导致性能下降问题。
i) 低效函数：检查语句中函数，函数可能导致索引失效，降低处理性能，可以通过其他方 式替代的问题。
j) 隐式关联：检测表连接的隐式关联，以及关联项的数据类型一致性，类型不一致可能导 致潜在数据一致性问题。
k) 可拆分 join：检查结果列表与查询条件，可拆分单表查却使用多表 join ，中间表过大或 者冗余数据过多。
l) In 范围：检查 in 的范围过大，for 循环添加，规模不可控可能超过语句长度。
m) Exist 替换：检查 in 类型，表规模在 existing 模式下效率更高问题。
n) 子查询深度：检查子查询嵌套层数过多，结构复杂，维护难度大问题。
o) 子查询效率：检查低效且可被其他 join 或拆分单表查询的子查询问题。
p) 临时表溢出：检查临时表过大导致内存空间过分占用，严重情况下内存崩溃问题。
q) ORM 的 Update 误插入：检查 update 语句中主键是否指定，未指定主键的 update 变成 插入，增加脏数据与数据不一致的潜在风险。
r) 冗余 SQL 语句结果字段：检查 sql 语句结果集中的字段是否被应用使用，未使用字段将 空耗系统资源和效率问题。
s) 禁用存储过程：检查普通应用是否使用了存储过程，存储过程难以调试与拓展，提高程 序员编程能力要求问题。
t) 表级联：检查是否存在表级联问题，表级联容易产生数据库更新风暴问题。
u) 删除未先查：检查删除数据钱未先查询而导致数据误删除，最终导致数据丢失或者数据 不一致问题。
v) Update 缺乏 limit: 检查缺乏 limit 的更新语句，错误的 update，没有 limit 保护将导致大 规模的数据错误问题。
w) Delete 缺乏 limit: 检查缺乏 limit 的删除语句，执行错误的删除语句，没有 limit 控制作 用范围将导致误删全局数据造成较大损失的问题。
x) 索引类使用函数：检查对索引列使用函数计算，导致索引失效，降低查询效率问题。
y) 表达式使用： 检查对字段进行表达式操作的 where 子句，表达式佛事引擎放弃索引进 而全表扫描问题。
z) 最值查询效率：查找最值时未使用 limit ，全数据后拿最大一个导致拿过多冗余数据， 同时未利用引擎优化特性。
aa) SQL 注入：检查使用${}符号的变量未使用#{}预编译，此种用法可能导致 SQL 注入的风险。

附 录 B
（资料性）
编码示例

检查项	正例	反例
低效表 名	user_info（与应用名一致，不 含数字开头）	123_user （ 数 字 开 头 ） 或 table__12 （下划线中间仅数字）
命名大 小写	user_id（全小写）	UserName（大小写混合）
命名包 含关键 字	description（非保留字）	desc（MySQL 保留字）
命名理 解性	account_number（明确无缩写）	acc_no（非常规缩写）
索引命 名	idx_user_name（前缀 idx_表示 普通索引）	index1（无类型前缀）
字段类 型	amount DECIMAL(10,2) （避免 精度损失）	amount FLOAT（精度可能丢失）
必备字 段	表 包 含 id 、create_time 、modified_time	表缺失 id 或时间字段
注释	CREATE TABLE user (id INT COMMENT '用户 ID')	建表语句未添加任何注释
唯一索 引	uk_user_email（唯一约束邮箱 字段）	未对唯一性字段（如手机号）建立索 引
默认值	status INT NOT NULL DEFAULT 0	status INT （未设置默认值 ，可能为
	（数值型默认0）	NULL）
非空未 设默认 值	name VARCHAR(20) NOT NULL DEFAULT ''（非空且有默认值）	name VARCHAR(20) NOT NULL（未设默 认值，插入需显式赋值）
外键	不在建表语句中使用外键约 束,应用层通过逻辑校验数据关联 性。例如：CREATE TABLE orders (id INT PRIMARY KEY, user_id INT);（应用代码校验 user_id 合法性）	使用外键约束或级联操作 ，如 ： CREATE TABLE orders (id IN PRIMARY KEY, user_id INT, FOREIGN KEY (user_id) REFERENCE users(id) ON DELETE CASCADE);
数据权 限	显 式 指 定 最 小 必 要 权 限 ， 如 ： GRANT SELECT, UPDATE ON db.table TO 'user'@'192.168.1.%';	开放全局权限或模糊授权,如：GRAN ALL PRIVILEGES ON db.* TO 'user'@'%';
Varchar 索引长 度	INDEX idx_email (email(20))（指 定索引长度）	INDEX idx_email (email)（未指定长度 可能过长）
索引数 量	单表索引数 ≤5	单表索引数 ≥10（影响写性能）
Join 表数 量	JOIN 3 张表且字段类型一致	JOIN 5 张表或字段类型不一致（如 IN 与 VARCHAR 关联）
左模糊 匹配	WHERE name LIKE '张%'（右模 糊）	WHERE name LIKE '%张'（左模糊导致 索引失效）
全模糊 匹配	使用搜索引擎处理模糊查询	WHERE name LIKE '% 张 %'（全模糊导致 全表扫描）
查询条	WHERE a=1 AND b=2 （组合索 引(a,b)生	WHERE b=2 AND a=1（违反最

件顺序	效）	左前缀原 则）
冗余索 引	仅有索引 (a,b)	同时存在 (a,b) 和 (a)（冗余索引）
使用 Distinct*	SELECT DISTINCT user_id FROM orders（明确列）	SELECT DISTINCT * FROM orders（未指 定列，性能差）
使用 Select*	SELECT id, name FROM user（指 定所需列）	SELECT * FROM user（返回冗余列）
Count(*)	SELECT COUNT(*) FROM user （统计所有行）	SELECT COUNT(id) FROM user （ 忽 略 NULL 行）
Delete 条件	DELETE FROM user WHERE id=1 （带条件）	DELETE FROM user（无条件，误删全 表）
Update 条件	UPDATE user SET name=' 张 三 ' WHERE id=1（带条件）	UPDATE user SET name='张三 ' （无条 件，误更新全表）
Limit 存 在	SELECT * FROM user LIMIT 100 （限制结果集）	SELECT * FROM user （未限制，可能导 致内存溢出）
Limit 深 度分页	SELECT * FROM user LIMIT 1000, 10（分页优化）	SELECT * FROM user WHERE id>100 LIMIT 10（未优化深度分页）
Or 优化	WHERE a=1 UNION WHERE b=2 （拆分 OR 条件）	WHERE a=1 OR b=2（导致索引失效）
低效函 数	WHERE DATE(create_time)='2023-01-01'（改 用范围查询）	WHERE create_time BETWEE '2023-01-01' AND '2023-01-02'（避免函数操 作）
隐式关联类型不一致	JOIN 字段类型均为 INT	JOIN 字段类型为 INT 与 VARCHAR（隐 式转换导致性能问题）
可拆分 Join	单表查询后程序合并结果	多表 JOIN 生成冗余中间表
In 范围 过大	WHERE id IN (1,2,3) （元 素 数 ≤1000）	WHERE id IN (1,2,..., 1001)（超出可控范 围）
Exist 替 换	WHERE EXISTS (SELECT 1 FROM order WHERE user.id =order.user_id)	WHERE user.id IN (SELECT user_i FROM order)（表大时效率低）
子查询 深度	子查询嵌套 ≤2 层	子查询嵌套 ≥4 层（难以维护）
子查询 效率	使用 JOIN 替代子查询	SELECT * FROM user WHERE id I (SELECT user_id FROM order) （可 优化 为 JOIN）
ORM 的 Update 误插入	UPDATE user SET name=' 张 三 ' WHERE id=1（指定主键）	UPDATE user SET name='张三 '（未指定 主键，可能误插入）
冗 余 SQL 结果字 段	仅返回应用需要的字段	返 回 未 使 用 的 字 段 （ 如 SELECT *仅用 id）
禁用存储过程	应用层处理逻辑	使用存储过程实现业务规则
表级联	应用层控制外键逻辑	使用数据库级联删除（如 ON DELET CASCADE）
删除未 先查	先 执 行 SELECT 确 认 数 据 范 围 ， 再 DELETE	直 接 执 行 DELETE FROM user（未确认 数据）
Update 缺乏	UPDATE user SET status=1	UPDATE user SET status=1 WHERE id=

Limit	WHERE id=1 LIMIT 1 （限制影响行 数）	（未限制，可能误更新多行）
Delete 缺乏 Limit	DELETE FROM user WHERE status=0 LIMIT 100（分批删除）	DELETE FROM user WHERE status=0（一 次性删除大量数据）
索引列 使用函 数	WHERE create_time >= '2023-01-01'（避免函数）	WHERE YEAR(create_time)=2023（索引 失效）
表达式 操作	WHERE age > 18（直接比较）	WHERE age+10 > 30（表达式导致全表 扫描）
最值查 询效率	SELECT MAX(id) FROM user LIMIT 1（利用索引优化）	SELECT MAX(id) FROM user（未优化， 可能全表扫描）
SQL 注入	使用预编译参数（如#{}）	拼接 SQL 变量（如${}）