MySQL开发规范

目录

[1. 简介 4](#_Toc444860740)

[1.1 规范是死的，人是活的。 4](#_Toc444860741)

[1.2 目的 4](#_Toc444860742)

[1.3 范围 4](#_Toc444860743)

[2. 数据库设计 4](#_Toc444860744)

[2.1 引擎及版本选择 4](#_Toc444860745)

[2.2 schema设计 4](#_Toc444860746)

[2.2.1 尽量不在数据库做运算 4](#_Toc444860747)

[2.2.2 复杂运算移到程序端CPU 4](#_Toc444860748)

[2.2.3 尽可能简单应用MySQL 4](#_Toc444860749)

[2.2.4 库和表预估 4](#_Toc444860750)

[2.2.5 单表控制 4](#_Toc444860751)

[2.2.6 拒绝3B 5](#_Toc444860752)

[2.3 全文检索设计 5](#_Toc444860753)

[2.3.1 最差的设计 5](#_Toc444860754)

[2.3.2 MySQL相关引擎支持 5](#_Toc444860755)

[2.3.3 使用外部开源全文检索引擎 5](#_Toc444860756)

[2.4 分页设计 5](#_Toc444860757)

[2.4.1 传统分页 5](#_Toc444860758)

[2.4.2 LIMIT原理 5](#_Toc444860759)

[2.4.3 LIMIT方式推荐分页 5](#_Toc444860760)

[2.4.4 LIMIT的高效分页 6](#_Toc444860761)

[2.4.5 LIMIT分页举例 6](#_Toc444860762)

[2.4.6 LIMIT分页与缓存结合 6](#_Toc444860763)

[2.5 表字段设计 7](#_Toc444860764)

[2.5.1 主键 7](#_Toc444860765)

[2.5.2 类型溢出 8](#_Toc444860766)

[2.5.3 类型转换 9](#_Toc444860767)

[2.5.4 字段定义 10](#_Toc444860768)

[2.5.5 字段选择 12](#_Toc444860769)

[2.5.6 技巧 12](#_Toc444860770)

[3. sql语句 13](#_Toc444860771)

[3.1 MySQL语句编写规范 13](#_Toc444860772)

[3.1.1 去掉不必要的括号 13](#_Toc444860773)

[3.1.2 去掉重叠常量 13](#_Toc444860774)

[3.1.3 去除常量条件(由于常量重叠需要) 13](#_Toc444860775)

[3.1.4 去掉无意义的连接用条件 14](#_Toc444860776)

[3.1.5 开发过程中不使用拼字符串的方式来完成where子句 14](#_Toc444860777)

[3.1.6 多使用等值操作，少使用非等值操作 14](#_Toc444860778)

[3.1.7 常数表优先，字典表或小表其次，大表最后 14](#_Toc444860779)

[3.1.8 减少或避免临时表 14](#_Toc444860780)

[3.1.9 where子句中的数据扫描别跨越表的30% 14](#_Toc444860781)

[3.1.10 where子句中同一个表的不同的字段组合建议小于等于5组 14](#_Toc444860782)

[3.1.11 不使用is null或is not null 14](#_Toc444860783)

[3.1.12 使用like时，%不要放在首字符位置。 14](#_Toc444860784)

[3.1.13 值域比较多的表字段放在前面 15](#_Toc444860785)

[3.1.14 表字段组合中出现比较多的表字段放在前面 15](#_Toc444860786)

[3.1.15 表字段不能有表达式或是函数 15](#_Toc444860787)

[3.1.16 注意表字段的类型，避免表字段的隐式转换 15](#_Toc444860788)

[3.1.17 考虑使用union all，少使用union，注意考虑去重 15](#_Toc444860789)

[3.1.18 不同字段的值or或in大于等于3次，考虑用union all替换；同一字段的值or用in替换 15](#_Toc444860790)

[3.1.19 用Where子句替换HAVING子句 15](#_Toc444860791)

[3.1.20 对同一表的order by和group by操作分别小于3组 15](#_Toc444860792)

[3.1.21 尽量使用主键进行update和delete 16](#_Toc444860793)

[3.1.22 小心text/blobs等大字段 16](#_Toc444860794)

[3.1.23 使用INSERT ... ON DUPLICATE KEY update (INSERT IGNORE)来避免不必要的查询 16](#_Toc444860795)

[3.1.24 考虑使用limit N，少用limit M,N，特别是大表，或M比较大的时候 16](#_Toc444860796)

[3.1.25 减少或避免排序，如：group by语句中如果不需要排序，可以增加order by null 16](#_Toc444860797)

[3.1.26 增删改语句中不使用不确定值函数和随机函数，如：RAND()和NOW()等。 16](#_Toc444860798)

[3.1.27 INSERT语句使用batch提交（INSERT INTO table VALUES(),(),()„„），values的个数不超过500。 16](#_Toc444860799)

[3.1.28 避免使用存储过程、触发器、函数、UDF、events等，容易将业务逻辑和DB耦合在一起，并且MySQL的存储过程、触发器、函数、UDF、events中存在一定的bug。 16](#_Toc444860800)

[3.1.29 避免使用JOIN。 16](#_Toc444860801)

[3.1.30 使用合理的SQL语句减少与数据库的交互次数。 16](#_Toc444860802)

[3.1.31 减少使用视图，避免复杂的语句。 16](#_Toc444860803)

[3.1.32 SQL语句中IN包含的值不超过500。 16](#_Toc444860804)

[3.1.33 UPDATE、DELETE语句不使用LIMIT。有主键id的表WHERE条件应结合主键。 16](#_Toc444860805)

[3.1.34 使用prepared statement，可以提供性能并且避免SQL注入。 16](#_Toc444860806)

[3.1.35 InnoDB表避免使用COUNT(\*)操作，计数统计实时要求较强可以使用memcache或者redis，非实时统计可以使用单独统计表，定时更新。 16](#_Toc444860807)

[3.1.36 禁止在Update语句，将“,”写成“and”，非常危险。 16](#_Toc444860808)

[4. 索引 17](#_Toc444860809)

[4.1 非唯一索引按照“idx\_字段名称\_字段名称[\_字段名]”进行命名。 17](#_Toc444860810)

[4.2 唯一索引按照“uk\_字段名称\_字段名称[\_字段名]”进行命名。 17](#_Toc444860811)

[4.3 索引名称使用小写。 17](#_Toc444860812)

[4.4 索引中的字段数不超过5个。 17](#_Toc444860813)

[4.5 唯一键由3个以下字段组成，并且字段都是整形时，使用唯一键作为主键。 17](#_Toc444860814)

[4.6 没有唯一键或者唯一键不符合5中的条件时，使用自增（或者通过发号器获取）id作为主键。 17](#_Toc444860815)

[4.7 唯一键不和主键重复。 17](#_Toc444860816)

[4.8 索引字段的顺序需要考虑字段值去重之后的个数，个数多的放在前面。 17](#_Toc444860817)

[4.9 ORDER BY，GROUP BY，DISTINCT的字段需要添加在索引的后面。 17](#_Toc444860818)

[4.10 单张表的索引数量控制在5个以内，若单张表多个字段在查询需求上都要单独用到索引，需要经过DBA评估。查询性能问题无法解决的，应从产品设计上进行重构。 17](#_Toc444860819)

[4.11 PDATE、DELETE语句需要根据WHERE条件添加索引。 17](#_Toc444860820)

[4.12 合理创建联合索引（避免冗余），(a,b,c) 相当于 (a) 、(a,b) 、(a,b,c)。 17](#_Toc444860821)

[4.13 合理利用覆盖索引。 17](#_Toc444860822)

# 简介

持续借鉴、收集并整理一些开发规范和技巧，期望能更充分利用MySQL的特性，得到更好的性能。

## 规范是死的，人是活的。

现在定义的规范，是为以后推翻准备的。

## 目的

提供给开发人员参考，方便写成更有效率的开发。

方便数据库统一管理和优化

## 范围

文档涉及的范围：需要基于MySQL做应用开发的人员。

# 数据库设计

目标三个：功能实现，可伸缩性，可用性。

关键点：平衡业务技术各个方面，做好取舍。

80%的性能优化来自架构设计的优化。

## 引擎及版本选择

根据目前我们业务的特点，引擎建议使用InnoDB

建议使用Percona Server 5.6.20，后续MySQL5.7比较稳定后再进行考量和评估。

## schema设计

### 尽量不在数据库做运算

### 复杂运算移到程序端CPU

### 尽可能简单应用MySQL

如：md5() 或Order by Rand()或计算字段等操作不在数据库表上进行。

### 库和表预估

常见的有100库100表，1000库10表等。

建议单库不超过300-400个表。

总空间容量不超过100G。

### 单表控制

* 考虑因素：

IO高效；全表遍历；表修复快；提高并发；alter table快。

* 字段数量：

建议上限20~50个。

一年内的单表数据量预估

建议纯INT不超1000W，含CHAR不超500W。

* 举例

单表1G体积 500W行评估：

顺序读1G文件需N秒

单行不超过200Byte

单表不超50个纯INT字段

单表不超20个CHAR(10)字段

### 拒绝3B

大SQL (BIG SQL)

大事务 (BIG Transaction)

大批量 (BIG Batch)

### 表和字段注释

这里强调，所有的表和字段都要有明确的注释！不解释！

## 全文检索设计

### 最差的设计

直接使用sql语句where条件中使用like %fulltext%

直接全表扫描或全索引扫描，性能最差，无任何扩展，基本不可接受。

### MySQL相关引擎支持

MyISAM全文索引，使用match()函数搜索。InnoDB从MySQL5.6.4开始支持全文索引，对中文支持不好，使用MATCH()…AGAINST。

并发不高，数据量不大，业务逻辑简单，可以考虑。

### 使用外部开源全文检索引擎

目前常用的有sphinx和lucene等。

适合并发高，数据量大，业务逻辑复杂的场景。

主要关注预热、增量更新及分片功能的实现。

## 分页设计

### 传统分页

Select \* from table limit 10000,10;

### LIMIT原理

Limit 10000,10

偏移量越大则越慢

### LIMIT方式推荐分页

Select \* from table WHERE id>=23423 limit 11; #10+1 (每页10条)

select \* from table WHERE id>=23434 limit 11;

### LIMIT的高效分页

可能需按场景分析并重组索引。

* **分页方式二**

Select \* from table WHERE id >= ( select id from table limit 10000,1 ) limit 10;

* **分页方式三**

SELECT \* FROM table INNER JOIN (SELECT id FROM table LIMIT 10000,10) USING (id) ;

* **分页方式四**

程序取ID：select id from table limit 10000,10;

Select \* from table WHERE id in (123,456…) ;

### LIMIT分页举例

MySQL> select sql\_no\_cache \* from post limit 10,10;

10 row in set (0.01 sec)

MySQL> select sql\_no\_cache \* from post limit 20000,10;

10 row in set (0.13 sec)

MySQL> select sql\_no\_cache \* from post limit 80000,10;

10 rows in set (0.58 sec)

MySQL> select sql\_no\_cache id from post limit 80000,10;

10 rows in set (0.02 sec)

MySQL> select sql\_no\_cache \* from post WHERE id>=323423 limit 10;

10 rows in set (0.01 sec)

MySQL> select \* from post WHERE id >= ( select sql\_no\_cache id from post limit 80000,1 ) limit 10 ;

10 rows in set (0.02 sec)

### LIMIT分页与缓存结合

类似sina微博的首页，总共保留最新的500条微博，分10页。

## 表字段设计

### 主键

如果使用的是InnoDB并且不需要特殊的聚簇。定义一个代理键（surrogate key）是个好的主意。意思就是这个主键并不是来自于你的应用程序的数据（与业务逻辑无关，而应用程序的数据如果有唯一的候选列可以做成唯一键），最简单的方法就是使用AUTO\_INCREMENT列。这能保证数据插入保持着连续的顺序并且对于使用主键连接会获得更好的性能。最好避免使用随机的聚簇键。 对每张表，最重要的就是一定要有主键。

* **主键设计之UUID**

从性能的角度来说，使用UUID是个最不好的方法：它使聚簇索引的插入是随机的。这是最不好的场景了。并且对于数据的聚集也没有什么帮助。

* **UUID\_SHORT()**

UUID\_SHORT()所占用的存储空间比UUID要小。（UUID\_SHORT()可能要使用bigint占用8个字节，而UUID可能要使用字符串用char(32)）。另外uuid\_short()是顺序的，这个也解决了随机导致的问题，但是uuid\_short()也有一些限制和bug。

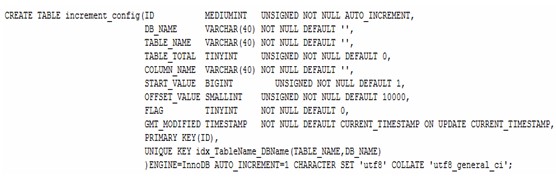
* **主键设计之自增列**

没什么好说的，自增列简单实用。

仅注意一下锁(gap lock)问题即可。

* **主键设计之程序控制**

建立类似如下表：



避免自增列引起的一些锁问题，统一管理，并发性更高。

* **主键设计之使用中间件**

更高的并发性，可以考虑从数据库中剥离，使用自己开发或第三方中间件，如：

<https://github.com/twitter/snowflake>

### 类型溢出

* **举例**

以MySQL5版本，int类型为例：

#建表

root@localhost(test2)14:46>create table test2 (a int(10) UNSIGNED);

Query OK, 0 rows affected (0.12 sec)

#插入数据

root@localhost(test2)14:56>insert test2 values (10);

Query OK, 1 row affected (0.00 sec)

#模拟更新溢出

root@localhost(test2)14:56>update test2 set a=a-11;

Query OK, 1 row affected, 1 warning (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 1

#查看warnings

root@localhost(test2)14:57>show warnings;

+---------+------+--------------------------------------------+

| Level | Code | Message |

+---------+------+--------------------------------------------+

| Warning | 1264 | Out of range value for column 'a' at row 1 |

+---------+------+--------------------------------------------+

1 row in set (0.00 sec)

#确定实际得到的值已经溢出

root@localhost(test2)14:57>select \* from test2;

+------------+

| a |

+------------+

| 4294967295 |

+------------+

1 row in set (0.00 sec)

#清理数据

root@localhost(test2)14:59>delete from test2;

Query OK, 1 row affected (0.00 sec)

#模拟插入溢出

root@localhost(test2)14:59>insert test2 values (-1);

Query OK, 1 row affected, 1 warning (0.00 sec)

#查看warnings

root@localhost(test2)14:59>show warnings;

+---------+------+--------------------------------------------+

| Level | Code | Message |

+---------+------+--------------------------------------------+

| Warning | 1264 | Out of range value for column 'a' at row 1 |

+---------+------+--------------------------------------------+

1 row in set (0.00 sec)

#确定实际得到的值已经溢出

root@localhost(test2)14:59>select \* from test2;

+------+

| a |

+------+

| 0 |

+------+

1 row in set (0.00 sec)

* **原因**

int占用4个字节，而int又分为无符号型和有符号性。对于无符号型的范围是0 到 4294967295；有符号型的范围是-2147483648 到 2147483647。

举一反三，其他类型都可能有类似问题，均需要考量。

* **控制方法**

可以通过sql\_mode参数控制，但一般建议程序控制，比如：对表单项的值进行校验。

### 类型转换

* **举例**

溢出和隐含转换导致的运行缓慢。

discuz论坛, uid是pk, mediumint类型, MyISAM表,总行数255万#现象：更新比较慢，快7秒

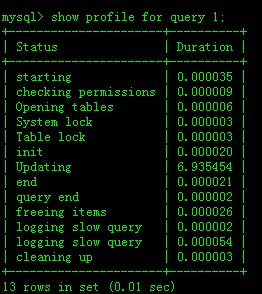
MySQL> UPDATE cdb\_members SET email='qin@zol.com.cn' WHERE uid='486851368';Query OK, 0 rows affected (6.94 sec)Rows matched: 0 Changed: 0 Warnings: 0

#通过观察语句发现有类型转换，因为uid为mediumint，而传过来的值是字符串'2982891440'

#查看该执行计划MySQL> explain select \* from cdb\_members WHERE uid='2982891440';+----+-------------+-------+------+---------------+------+---------+------+------+-----------------------------------------------------+| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |+----+-------------+-------+------+---------------+------+---------+------+------+-----------------------------------------------------+| 1 | SIMPLE | NULL | NULL | NULL | NULL | NULL | NULL | NULL | Impossible WHERE noticed after reading const tables |+----+-------------+-------+------+---------------+------+---------+------+------+-----------------------------------------------------+1 row in set (0.00 sec)#查看传过来的值是mediumint类型的2982891440的执行计划

MySQL> explain select \* from cdb\_members WHERE uid=2982891440;+----+-------------+-------+------+---------------+------+---------+------+------+-----------------------------------------------------+| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |+----+-------------+-------+------+---------------+------+---------+------+------+-----------------------------------------------------+| 1 | SIMPLE | NULL | NULL | NULL | NULL | NULL | NULL | NULL | Impossible WHERE noticed after reading const tables |+----+-------------+-------+------+---------------+------+---------+------+------+-----------------------------------------------------+1 row in set (0.00 sec)

#profiling的效果



这里的uid是mediumint， 这个条件的uid大小明显超过了范围。（参考 [字段定义](#_字段定义)）

去掉uid=后面值的''，速度就正常了。但是对于uid没有溢出的,加不加引号速度都一样。

* **原因**

在non-strict mode下，MySQL会自动帮你把字符串转换成整形，但是如果数值超出了范围，转换就会失败，所以MySQL就按照字符串来处理，因此不能使用索引。而从explain的结果上，并没有表现出这样的差别。

* **控制方法**

可以通过sql\_mode参数控制，一般建议程序控制，比如：对表单项的值进行校验。

### 字段定义

|  |  |  |
| --- | --- | --- |
| 列类型 | 表达的范围 | 存储需求 |
| TINYINT[(M)] [UNSIGNED] [ZEROFILL] | -128到127 或 0到255 | 1个字节 |
| SMALLINT[(M)] [UNSIGNED] [ZEROFILL] | -32768到32767 或 0到65535 | 2个字节 |
| INT[(M)] [UNSIGNED] [ZEROFILL] | -2147483648到2147483647 或 0到4294967295 | 4个字节 |
| BIGINT[(M)] [UNSIGNED] [ZEROFILL] | -9223372036854775808到9223372036854775807  或 0到18446744073709551615 | 8个字节 |
| DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL] | 整数最大位数（M）为65，小数位数最大（D）为30 | 变长 |
| DATE | YYYY-MM-DD | 3个字节 |
| DATETIME | YYYY-MM-DD HH:MM:SS(1001年到9999年的范围) | 8个字节 |
| TIMESTAMP | YYYY-MM-DD HH:MM:SS（1970年到2037年的范围） | 4个字节 |
| CHAR(M) | 0<M<=255（建议CHAR(1)外，超过此长度的用VARCHAR） | M个字符（所占空间跟字符集等有关系） |
| VARCHAR(M) | 0<M<65532/N | M个字符（N大小由字符集，以及是否为中文还是字母数字等有关系） |
| TEXT | 64K个字符 | 所占空间跟字符集等有关系 |

### 字段选择

a) 字段定义为NOT NULL。

b) 表字符集选择UTF8，对恶魔字符集emoji用utf8mb4，如果MySQL版本不支持时，应用收到这种字符集的时候先转码成定义的串再传入数据库，例如传来\x0F，应用先替换成[i\_smile]这样的特定字符串再存入数据库。

c) 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）使用TIMESTAMP类型或INT。使用时间字段作为查询条件，尤其是以时间段查询时，应使用INT保存。

d) 整形定义中不添加长度，比如使用INT，而不是INT[4]。

e) 使用VARBINARY存储变长字符串。

f)自增序列类型的字段只能使用INT或者BIGINT，且明确标识出为无符号型(UNSIGNED)，除非确实会出现负数，仅当该字段数字取值会超过42亿，才使用BIGINT类型；

g) int(10)和int(1)没有什么区别，10和1仅是宽度而已，在zerofill等扩展属性的时候有用或者特殊的命令行交互工具

h)少量枚举或状态定义类需求不建议使用ENUM、SET、VARCHAR类型，使用TINYINT来代替。需在COMMENT信息中标明被枚举的含义。例如：

`is\_disable` int(11) DEFAULT '0' COMMENT '0:启用 1:禁用 2:异常';

i)存储精确浮点数使用DECIMAL替代FLOAT和DOUBLE。

j)使用UNSIGNED存储非负数值。

k)使用INT UNSIGNED存储IPV4。

l) 使用短数据类型，比如取值范围为0-80时，使用TINYINT UNSIGNED。

m) 尽可能不使用TEXT、BLOB类型，如果确实需要将过大字段拆分到其他表中。

n) VARCHAR(N)，N表示的是字符数不是字节数，比如VARCHAR(255)，可以最大可存储255个汉字，需要根据实际的宽度来选择N，VARCHAR(N)，N>5000时，使用BLOB类型。如果N<256时会使用一个字节来存储长度，如果N>=256则使用两个字节来存储长度。

o) 不在数据库中使用VARBINARY、BLOB存储图片、文件等。

p) 禁止在数据库中存储明文密码

q) 数据库中存放IP时，按功能确定字段类型。仅作展示功能的使用CHAR，作为查询功能的应使用INT类型存放。参见“[将字符转化为数字](#_将字符转化为数字)”

r) 严格禁止在库名、表名中使用大写字母。

### 技巧

* **将字符转化为数字**

更高效，查询更快，占用空间更小

举例：用无符号INT存储IP，而非CHAR(15)

INT UNSIGNED

INET\_ATON()

INET\_NTOA()

root@localhost(test) 16:21>SELECT INET\_ATON('192.168.0.1');

+--------------------------+

| INET\_ATON('192.168.0.1') |

+--------------------------+

| 3232235521 |

+--------------------------+

1 row in set (0.00 sec)

root@localhost(test) 16:21>SELECT INET\_NTOA('3232235521');

+-------------------------+

| INET\_NTOA('3232235521') |

+-------------------------+

| 192.168.0.1 |

+-------------------------+

1 row in set (0.01 sec)

* **将日期转化为数字**

更高效，查询更快，占用空间更小

举例：用无符号INT存储日期和时间

INT UNSIGNED

FROM\_UNIXTIME()

UNIX\_TIMESTAMP()

* **避免使用NULL字段**

原因：

很难进行查询优化；

NULL列加索引，需要额外空间；

含NULL复合索引无效。

举例：

不使用：`a` char(32) DEFAULT NULL

不使用：`b` int(10) NOT NULL

使用：`c` int(10) NOT NULL DEFAULT 0

# sql语句

## MySQL语句编写规范

### 去掉不必要的括号

如： ((a AND b) AND c OR (((a AND b) AND (c AND d))))

修改成 (a AND b AND c) OR (a AND b AND c AND d)

### 去掉重叠常量

如： (a<b AND b=c) AND a=5

修改成 b>5 AND b=c AND a=5

### 去除常量条件(由于常量重叠需要)

如： (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)

修改成 B=5 OR B=6

### 去掉无意义的连接用条件

如：1=1，2>1，1<2等 直接从where子句中去掉。

### 开发过程中不使用拼字符串的方式来完成where子句

### 多使用等值操作，少使用非等值操作

WHERE条件中的非等值条件（IN、BETWEEN、<、<=、>、>=）会导致后面的条件使用不了索引，因为不能同时用到两个范围条件。

### 常数表优先，字典表或小表其次，大表最后

常数表指：空表或只有1行的表。与在一个PRIMARY KEY或UNIQUE索引的WHERE子句一起使用的表。如：

SELECT \* FROM t WHERE primary\_key=1;

SELECT \* FROM t1,t2 WHERE t1.primary\_key=1 AND t2.primary\_key=t1.id;

字典表指：小数量的行。如：自定义的自增字段表，而不使用MySQL的AUTO\_INCREMENT。

### 减少或避免临时表

如果有一个ORDER BY子句和不同的GROUP BY子句，或如果ORDER BY或GROUP BY包含联接队列中的第一个表之外的其它表的列，则创建一个临时表。

### where子句中的数据扫描别跨越表的30%

比如：where primary\_key <> 1或者primary\_key not in(…)，这样跨表的数据肯定超过30%了。

where status=1，其中1值非常少，主要是0值，比如一个表的记录删除用了一个状态位，而删除的记录又比较少。

### where子句中同一个表的不同的字段组合建议小于等于5组

否则考虑业务逻辑或分表

### 不使用is null或is not null

字段设计时建议not null，若麻烦可折中考虑给一默认值

原因参见“[避免使用NULL字段](#_避免使用NULL字段)”。

### 使用like时，%不要放在首字符位置。

如果%必须放在首字符位置，参见“[全文检索设计](#_全文检索设计)”

### 值域比较多的表字段放在前面

比如：id，date字段放在前面，而status这样的字段放在后面，具体的可以通过执行计划来把握。

### 表字段组合中出现比较多的表字段放在前面

方便综合评估索引，缓解因为索引过多导致的增删改的一些性能问题。

### 表字段不能有表达式或是函数

如：where abs(列)>3或where 列\*10>100

### 注意表字段的类型，避免表字段的隐式转换

参见“[表字段设计](#_表字段设计)”

比如：列为int，如果where 列=’1’，则会出现转换。

### 考虑使用union all，少使用union，注意考虑去重

union all不去重，而少了排序操作，速度相对比union要快，如果没有去重的需求，优先使用union all。

### 不同字段的值or或in大于等于3次，考虑用union all替换；同一字段的值or用in替换

Select \* from opp WHERE phone=‘12347856' or phone=‘42242233';

考虑用

Select \* from opp WHERE phone in ('12347856' , '42242233');

Select \* from opp WHERE phone='010-88886666' or cellPhone='13800138000';

考虑用

Select \* from opp WHERE phone='010-88886666'

union all

Select \* from opp WHERE cellPhone='13800138000';

### 用Where子句替换HAVING子句

select id,count(\*) from table group by id having age>=30 order by null;

考虑用

select id,count(\*) from table where age>=30 group by id order by null;

### 对同一表的order by和group by操作分别小于3组

否则考虑业务逻辑或分表

### 尽量使用主键进行update和delete

### 小心text/blobs等大字段

如果确实不需要这样的大字段，则不用放入sql语句中

Text/blob大字段的用法参见“[字段选择](#_字段选择)”

### 使用INSERT ... ON DUPLICATE KEY update (INSERT IGNORE)来避免不必要的查询

### 考虑使用limit N，少用limit M,N，特别是大表，或M比较大的时候

参考“[分页设计](#_分页设计)”

### 减少或避免排序，如：group by语句中如果不需要排序，可以增加order by null

### 增删改语句中不使用不确定值函数和随机函数，如：RAND()和NOW()等。

### INSERT语句使用batch提交（INSERT INTO table VALUES(),(),()„„），values的个数不超过500。

### 避免使用存储过程、触发器、函数、UDF、events等，容易将业务逻辑和DB耦合在一起，并且MySQL的存储过程、触发器、函数、UDF、events中存在一定的bug。

### 避免使用JOIN。

### 使用合理的SQL语句减少与数据库的交互次数。

INSERT ... ON DUPLICATE KEY UPDATE

REPLACE INTO、INSERT IGNORE 、INSERT INTO VALUES(),(),()

UPDATE … WHERE ID IN(10,20,50,…)

### 减少使用视图，避免复杂的语句。

### SQL语句中IN包含的值不超过500。

### UPDATE、DELETE语句不使用LIMIT。有主键id的表WHERE条件应结合主键。

### 使用prepared statement，可以提供性能并且避免SQL注入。

### InnoDB表避免使用COUNT(\*)操作，计数统计实时要求较强可以使用memcache或者redis，非实时统计可以使用单独统计表，定时更新。

### 禁止在Update语句，将“,”写成“and”，非常危险。

正确示例：update Table set uid=uid+1000**,**gid=gid+1000 where id <=2 ;

错误示例：update Table set uid=uid+1000 **and** gid=gid+1000 where id <=2 ;

此时“uid=uid+1000 and gid=gid+1000”将作为值赋给uid，并且**无Warning！！！**

# 索引

## 非唯一索引按照“idx\_字段名称\_字段名称[\_字段名]”进行命名。

## 唯一索引按照“uk\_字段名称\_字段名称[\_字段名]”进行命名。

## 索引名称使用小写。

## 索引中的字段数不超过5个。

## 唯一键由3个以下字段组成，并且字段都是整形时，使用唯一键作为主键。

## 没有唯一键或者唯一键不符合5中的条件时，使用自增（或者通过发号器获取）id作为主键。

## 唯一键不和主键重复。

## 索引字段的顺序需要考虑字段值去重之后的个数，个数多的放在前面。

## ORDER BY，GROUP BY，DISTINCT的字段需要添加在索引的后面。

## 单张表的索引数量控制在5个以内，若单张表多个字段在查询需求上都要单独用到索引，需要经过DBA评估。查询性能问题无法解决的，应从产品设计上进行重构。

## PDATE、DELETE语句需要根据WHERE条件添加索引。

## 合理创建联合索引（避免冗余），(a,b,c) 相当于 (a) 、(a,b) 、(a,b,c)。

## 合理利用覆盖索引。