**MySQL开发规范**

编写部门：

编写人：刘晶晶

编写时间：2015.07.01

**修订说明**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **修改日期** | **修改内容** | **修改人** |
| 1.0 | 2015.7.01 | 创建 | 刘晶晶 |
| 1.1 | 2015.7.05 | 修改 | 胡成龙 |
|  |  |  |  |
|  |  |  |  |

目录

[1 内容 4](#_Toc424544340)

[1.1 目的 4](#_Toc424544341)

[1.2 范围 4](#_Toc424544342)

[1.3 关键字 4](#_Toc424544343)

[1.4 术语和定义 6](#_Toc424544344)

[1.5 大小写规范 7](#_Toc424544345)

[1.6 命名规范 7](#_Toc424544346)

[1.7 基础规范 8](#_Toc424544347)

[1.8 库表设计规范 9](#_Toc424544348)

[1.9 字段设计规范 10](#_Toc424544349)

[1.10 索引设计规范 11](#_Toc424544350)

[1.11 SQL设计规范 13](#_Toc424544351)

[1.12 行为规范 14](#_Toc424544352)

[1.13 缩进风格 15](#_Toc424544353)

[1.14 空格及换行 16](#_Toc424544354)

[1.15 注释规范 17](#_Toc424544355)

[2 附录 19](#_Toc424544356)

[2.1 数据类型 19](#_Toc424544357)

[2.1.1 数值类型 19](#_Toc424544358)

[2.1.2 日期和时间类型 20](#_Toc424544359)

[2.1.3 字符串类型 21](#_Toc424544360)

[2.1.4 数据类型属性 22](#_Toc424544361)

# 内容

## 目的

本规范的目的是确保公司物理数据库的设计和编码标准化、规范化。通过建立编码规范，促使每个开发人员养成良好的编码风格和习惯，增强程序可读性，降低程序BUG率，实现程序一致性，增强程序可维护性，提升程序的性能。

## 范围

本规范仅适用于关系型数据库系统MySQL以及其对应的开发脚本语言。

## 关键字

|  |  |  |
| --- | --- | --- |
| ADD | ALL | ALTER |
| ANALYZE | AND | AS |
| ASC | ASENSITIVE | BEFORE |
| BETWEEN | BIGINT | BINARY |
| BLOB | BOTH | BY |
| CALL | CASCADE | CASE |
| CHANGE | CHAR | CHARACTER |
| CHECK | COLLATE | COLUMN |
| CONDITION | CONNECTION | CONSTRAINT |
| CONTINUE | CONVERT | CREATE |
| CROSS | CURRENT\_DATE | CURRENT\_TIME |
| CURRENT\_TIMESTAMP | CURRENT\_USER | CURSOR |
| DATABASE | DATABASES | DAY\_HOUR |
| DAY\_MICROSECOND | DAY\_MINUTE | DAY\_SECOND |
| DEC | DECIMAL | DECLARE |
| DEFAULT | DELAYED | DELETE |
| DESC | DESCRIBE | DETERMINISTIC |
| DISTINCT | DISTINCTROW | DIV |
| DOUBLE | DROP | DUAL |
| EACH | ELSE | ELSEIF |
| ENCLOSED | ESCAPED | EXISTS |
| EXIT | EXPLAIN | FALSE |
| FETCH | FLOAT | FLOAT4 |
| FLOAT8 | FOR | FORCE |
| FOREIGN | FROM | FULLTEXT |
| GOTO | GRANT | GROUP |
| HAVING | HIGH\_PRIORITY | HOUR\_MICROSECOND |
| HOUR\_MINUTE | HOUR\_SECOND | IF |
| IGNORE | IN | INDEX |
| INFILE | INNER | INOUT |
| INSENSITIVE | INSERT | INT |
| INT1 | INT2 | INT3 |
| INT4 | INT8 | INTEGER |
| INTERVAL | INTO | IS |
| ITERATE | JOIN | KEY |
| KEYS | KILL | LABEL |
| LEADING | LEAVE | LEFT |
| LIKE | LIMIT | LINEAR |
| LINES | LOAD | LOCALTIME |
| LOCALTIMESTAMP | LOCK | LONG |
| LONGBLOB | LONGTEXT | LOOP |
| LOW\_PRIORITY | MATCH | MEDIUMBLOB |
| MEDIUMINT | MEDIUMTEXT | MIDDLEINT |
| MINUTE\_MICROSECOND | MINUTE\_SECOND | MOD |
| MODIFIES | NATURAL | NOT |
| NO\_WRITE\_TO\_BINLOG | NULL | NUMERIC |
| ON | OPTIMIZE | OPTION |
| OPTIONALLY | OR | ORDER |
| OUT | OUTER | OUTFILE |
| PRECISION | PRIMARY | PROCEDURE |
| PURGE | RAID0 | RANGE |
| READ | READS | REAL |
| REFERENCES | REGEXP | RELEASE |
| RENAME | REPEAT | REPLACE |
| REQUIRE | RESTRICT | RETURN |
| REVOKE | RIGHT | RLIKE |
| SCHEMA | SCHEMAS | SECOND\_MICROSECOND |
| SELECT | SENSITIVE | SEPARATOR |
| SET | SHOW | SMALLINT |
| SPATIAL | SPECIFIC | SQL |
| SQLEXCEPTION | SQLSTATE | SQLWARNING |
| SQL\_BIG\_RESULT | SQL\_CALC\_FOUND\_ROWS | SQL\_SMALL\_RESULT |
| SSL | STARTING | STRAIGHT\_JOIN |
| TABLE | TERMINATED | THEN |
| TINYBLOB | TINYINT | TINYTEXT |
| TO | TRAILING | TRIGGER |
| TRUE | UNDO | UNION |
| UNIQUE | UNLOCK | UNSIGNED |
| UPDATE | USAGE | USE |
| USING | UTC\_DATE | UTC\_TIME |
| UTC\_TIMESTAMP | VALUES | VARBINARY |
| VARCHAR | VARCHARACTER | VARYING |
| WHEN | WHERE | WHILE |
| WITH | WRITE | X509 |
| XOR | YEAR\_MONTH | ZEROFILL |

注：

关键字不区分大小写，在建库、表、字段、索引等不能使用关键字命名。

## 术语和定义

本规范采用以下的术语描述：

★ 规则：编程时强制必须遵守的原则。

★ 建议：编程时必须加以考虑的原则。

★ 说明：对此规则或建议进行必要的解释。

★ 示例：对此规则或建议从正、反两个方面给出。

## ****书写规范****

在Linux下：

1. 数据库名、表名、表的别名默认是区分大小写的（lower\_case\_table\_names=0）。
2. 字段名、字段别名、索引名、存储程序名、关键字、变量名在任何平台均不区分大小写。
3. 字段值的大小写由MySQL的校对规则来控制。提到校对规则，就不得不说字符集。字符集是一套符号和编码，校对规则是在字符集内用于比较字符的一套规则。一般而言，校对规则以其相关的字符集名开始，通常包括一个语言名，并且以\_ci（大小写不敏感）、\_cs（大小写敏感）或\_bin（二元）结束 。例如：字符集utf8的默认校对为utf8\_general\_ci。

在Windows下：

库名、表名、表别名不区分大小写。

## ****命名规范****

1. 库名、表名、字段名应具有描述性，杜绝一切拼音或拼音英文混杂的命名方式。
2. 库名必须使用小写字母，数字和下划线，不允许使用其他字符。库名、表名必须使用单词开头，不可以使用数字和下划线开头。
3. 库名禁止超过32个字符。
4. 库名、表名、字段名禁止使用MySQL关键字。

注：

表名使用小写是为了方便数据库在Linux和Windows下的转储，Linux大小写敏感，Windows不敏感；表名使用大写是为了方便数据库在MySQL和Oracle下的迁移。

* 命名示例

对象命名规则

| **对象类型** | **规则** | **示例** | **备注** |
| --- | --- | --- | --- |
| 表（Table） | 表义名 | FINANACE\_ACCOUNT ACCOUNT\_INFO  ACCOUNT\_MAPPING | 表义名使用英文 |
| 临时表 | TEMP\_表义名 | TEMP\_PARTY | 表义名使用英文 |
| 关联表 | A表\_B表\_MAPPING | USER\_PARTY\_MAPPING |  |
| 视图（View） | V\_表义名 | V\_PARTY | 表义名使用英文 |
| 非唯一索引（Index） | IND\_表名SN | IND\_PARTY01 |  |
| 唯一索引 | UK\_表名SN | UK\_PARTY01 |  |
| 主键（Primary Key） | PK\_表名 | PK\_PARTY | 不超过30字符，否则用缩写表示 |
| 外键（Foreign Key） | FK\_所在表名\_对应的主表名SN | FK\_PARTY\_USER01 | 不超过30字符，否则用缩写表示 |
| 函数（Function） | FUN\_函数名 | FUN\_ USER\_MODIFY | 不超过30字符，否则用缩写表示 |
| 存储过程（Stored Procedure） | PRO\_存储过程名 | PRO\_USER\_MODIFY | 不超过30字符，否则用缩写表示 |
| 触发器（Trigger） | TRI\_触发器名 | TRI\_USER\_MODIFY | 不超过30字符，否则用缩写表示 |
| 表分区 | R/L/H/K\_表名\_区域/时间 | R\_FINANACE\_ENTRY01 R\_FINANACE\_HISTORY201504 |  |

函数、过程变量命名规则

|  |  |  |  |
| --- | --- | --- | --- |
| **变量类型** | **前缀** | **示例** | **备注** |
| 会话变量 | l变量类型\_变量名称 | ld\_ update |  |
| 全局变量 | g变量类型\_变量名称 | gd\_ update |  |
| 游标变量 | cur\_参数名称 | cur\_party |  |
| 输入参数 | i变量类型\_参数名称 | id\_ update |  |
| 输出参数 | o变量类型\_参数名称 | od\_ update |  |
| 输入输出参数 | io变量类型\_参数名称 | iod\_update |  |

字段命名规则

|  |  |  |  |
| --- | --- | --- | --- |
| **字段类型** | **后缀** | **示例** | **类型** |
| 自增列 | ID | PRODUCT\_ID | INT UNSIGNED |
| 编码类 | CODE | PRODUCT\_CODE | VARCHAR |
| 号码类 | NO | PRODUCT\_NO | BIGINT |
| 名称类 | NAME | PRODUCT\_NAME | VARCHAR |
| 类型类 | TYPE | PRODUCT\_TYPE | TINYINT UNSIGNED |
| 标志类 | FLAG | PRODUCT\_FLAG | TINYINT UNSIGNED |
| 级次类 | LEV | PRODUCT\_LEV | TINYINT UNSIGNED |
| 金额类 | MNY | PRODUCT\_MNY | DECIMAL |
| 自定义项类 | DEF | PRODUCT\_DEF | VARCHAR |
| 税类 | TAX | PRODUCT\_TAX | DECIMAL |
| 数量类 | NUM | PRODUCT\_NUM | INT |
| 比率类 | RATE | PRODUCT\_RATE | DECIMAL |
| 日期类 | DATE | PRODUCT\_DATE | DATE |
| 时间类 | TIME | PRODUCT\_TIME | TIME |
| 状态 |  | STATUS\_ID | TINYINT UNSIGNED |
| 备注 |  | REMARK | VARCHAR |
| 创建时间 | TIME | CREATE\_TIME | TIMESTAMP |
| 更新时间 | TIME | UPDATE\_TIME | TIMESTAMP |

## ****基础规范****

1. 建议使用InnoDB存储引擎。适合场景如下：

* MyISAM适合：(1)做很多count的计算；(2)插入不频繁，查询非常频繁；(3)没有事务。
* InnoDB适合：(1)可靠性要求比较高，或者要求事务；(2)表更新和查询都相当的频繁，并且表锁定的机会比较大的情况。(3)性能较好的服务器，比如单独的数据库服务器，像阿里云的关系型数据库RDS就推荐使用InnoDB引擎。

1. 数据库字符集使用UTF8字符集。
2. 所有表都需要添加注释，除主键外的其他字段都需要增加注释。
3. 禁止在数据库中存储图片、文件等大数据。
4. 每张表数据量建议控制在5000W以内。超过这个量级建议使用分区。
5. 禁止在线上做数据库压力测试。
6. 禁止在生产数据库上进行测试。

## ****库表设计规范****

1. 表名应具有描述性，杜绝一切拼音或拼音英文混杂的命名方式。
2. 表名使用大写字母，数字和下划线，不允许使用其他字符。表名使用单词开头，不可以使用数字和下划线开头。
3. 表名长度不能超过32个字符。
4. 表名禁止使用MySQL关键字。
5. 将大字段、访问频率低的字段拆分到单独的表中存储，分离冷热数据。
6. 按日期时间分表需符合YYYY[MM][DD][HH]格式，例如2013071601。年份必须用4位数字表示。例如按日散表R\_USER20110209、 按月散表R\_USER201102。
7. 采用合适的分库分表策略。例如千库十表、十库百表等。
8. 数据库中不允许存储大文件，或者照片，可以将大对象放到磁盘上，数据库中存储它的路径。
9. 分区限制

* 每个表的最大分区数8192。
* 不支持空间类型。
* 不能对临时表进行分区。
* 不支持外键。
* 不支持全文索引。
* 无全局索引：每个分区都有各自的索引。
* 仅可能在以下情况下进行子分区：
* 通过RANGE和LIST进行分区时。
* 通过LINEAR HASH或LINEAR KEY进行时。

1. 分区表达式限制

* 用于RANGE、LIST和HASH分区的表达式的结果必须为整数。
* RANGE COLUMNS和LIST COLUMNS允许更大范围的数据类型。
* 不能在分区表达式中使用TEXT和BLOB。
* 禁止使用UDF、存储函数、变量、某些运算符和某些内置函数。
* 运算符：|、&、^、<<、>>、~
* 不应在创建表后更改SQL模式。
* 分区表达式中不支持子查询。
* 分区表达式中使用的所有列都必须是表的所有唯一索引的一部分。

## 字段设计规范

1. 字段名应具有描述性，杜绝一切拼音或拼音英文混杂的命名方式。
2. 字段名使用大写字母、数字和下划线，不允许使用其他字符。字段名鼓励使用与所在表的内容相关单词开头，允许但不鼓励使用数字和其他字符开头。
3. 字段个数建议上限20~50个。
4. 字段名长度不能超过32个字符。
5. 数据类型和长度在不同数据表中必须保证一致性，不允许出现同一字段在一个表中为整型但在另外一个表中为字符型的情况。
6. 字段名禁止使用MySQL关键字。
7. 当几个表间的字段有关连时，要注意表与表之间关连字段命名的统一，如XS\_ORDERS表中的UID与XS\_CARTS表中的UID，都保存有XS\_USERS表中的UID。
8. 每个表都建议有一个代表ID自增量的字段，目前我们是使用‘表名第一个单词’\_ID。若与其他表产生重复，则再添加表名的最后一个单词。如：TRANS\_HISTORY\_ID。
9. 建议使用UNSIGNED存储非负数值。
10. 用DECIMAL代替FLOAT和DOUBLE存储精确小数，例如支付相关数据。
11. INT类型固定占4字节存储，例如INT(4)仅代表显示字符宽度为4位，不代表存储长度。建议INT后不加显示宽度，避免混淆。目前标明是为了避免数据库迁移时，不清楚字段长度。例如INT(10)是为了标明INT的最大长度为10位。
12. 当ENUM存储数值字符时，强烈建议使用TINYINT来代替ENUM类型。因为数值字符很容易和ENUM的数值索引混淆。
13. 尽可能不使用TEXT、BLOB类型。VARCHAR性能比TEXT高很多。实在避免不了BLOB请拆表。
14. 使用VARCHAR BINARY存储大小写敏感的变长字符串。
15. 区分使用DATETIME和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，不含日期)建议使用TIME类型。
16. 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
17. 使用尽可能小的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。
18. 禁止在数据库中存储明文密码，把密码加密后存储。
19. 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(15)，作为查询功能的应使用INT UNSIGNED类型存放。通过MySQL函数INET\_NTOA和INET\_ATON来进行转化。例如：

SELECT INET\_ATON('209.207.224.40');

3520061480

SELECT INET\_NTOA(3520061480);

209.207.224.40

1. 字段长度满足需求前提下，尽可能选择长度小的。
2. 字段属性尽量都加上NOT NULL约束，NULL字段很难查询优化，NULL字段的索引需要额外空间，NULL字段的复合索引无效。

## 索引设计规范

1. 索引名称使用大写字母、数字和下划线，不允许使用其他字符。
2. 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。对父表和子表的操作会相互影响，降低可用性。对外键不可以使用复合索引。
3. 不要索引text/blob类型的字段。
4. 索引长度不能超过32个字符。
5. 单张表中索引数量不超过5个。复合索引中字段数不超过5个。索引越少越好，索引占用空间，一个表拥有的索引越多，插入和删除的速度越慢。
6. 索引命名：

* 非唯一索引按照“IND\_表名称+序号”进行命名。例如IND\_ ACCOUNT\_INFO01。
* 唯一索引按照“UK\_表名称+序号”进行命名。例如UK\_ ACCOUNT\_INFO01。
* 主键按照“PK\_表名称”进行命名。例如PK\_ACCOUNT\_INFO。
* 外键按照“FK\_表名称+序号”进行命名。例如FK\_ACCOUNT\_INFO01。

1. 表必须有主键，推荐使用INT UNSIGNED自增列作为主键。
2. 唯一键由3个以下字段组成，并且字段都是整形时，可使用唯一键作为主键。其他情况下，建议使用自增列作主键。
3. 使用EXPLAIN判断SQL语句是否合理使用索引，尽量避免extra列出现：Using File Sort，Using Temporary。
4. UPDATE、DELETE语句需要根据WHERE条件添加索引。ORDER BY、GROUP BY、DISTINCT的字段要添加索引。
5. 对于复合索引，MySQL从左到右的使用索引中的字段，一个查询可以只使用索引中的一部份，但只能是最左侧部分。例如索引是key index (a,b,c)。可以支持(a)、(a,b)、(a,b,c)三种组合进行查找，但不支持(b,c)进行查找。当最左侧字段是常量引用时，索引就十分有效。下面用几个例子对比查询条件的不同对性能影响：

create table test(

a int,

b int,

c int,

KEY a(a,b,c)

);

优: select \* from test where a=10 and b>50

差: select \* from test where a>50

优: select \* from test where order by a

差: select \* from test where order by b

差: select \* from test where order by c

优: select \* from test where a=10 order by a

优: select \* from test where a=10 order by b

差: select \* from test where a=10 order by c

优: select \* from test where a>10 order by a

差: select \* from test where a>10 order by b

差: select \* from test where a>10 order by c

优: select \* from test where a=10 and b=10 order by a

优: select \* from test where a=10 and b=10 order by b

优: select \* from test where a=10 and b=10 order by c

优: select \* from test where a=10 and b=10 order by a

优: select \* from test where a=10 and b>10 order by b

差: select \* from test where a=10 and b>10 order by c

1. 禁止冗余索引。如果创建了索引(A,B)，再创建索引(A)就是冗余索引，因为这只是前一个索引的前缀索引。因此索引(A,B)也可以当作索引(A)来使用（这种冗余只是对B-Tree索引来说的）。但是如果再创建索引(B,A)，则不是冗余索引，索引(B)也不是。另外不同类型的索引之间不是冗余索引，如：哈希、全文索引，不会是B-Tree索引的冗余索引。
2. 禁止重复索引。重复索引是指在相同的列上按照相同的顺序创建的相同类型的索引。

例如：

CREATE TABLE test （

id INT NOT NULL PRIMIARY KEY,

a INT NOT NULL,

UNIQUE(id),

INDEX(id)

)ENGINE=InnoDB;

MySQL的唯一限制和主键限制都是通过索引实现的，上面的写法实际上在相同的列上创建了三个重复的索引。

1. 联表查询时，JOIN条件列的数据类型必须相同，并且要建立索引。
2. 不在低基数列上建立索引，例如“性别”。不在索引列进行数学运算和函数运算。
3. 对过长的VARCHAR字段使用前缀索引，前缀索引长度不超过8个字符。例如

mysql> alter table test add key (city(7));

1. 索引字段的默认值不能为NULL。NULL非常影响索引的查询效率。
2. 能使用唯一索引就要使用唯一索引，提高查询效率。
3. 合理使用覆盖索引减少IO，避免排序。如果一个索引包含(或者说覆盖)所有需要查询的字段的值，我们称之为“覆盖索引”。例如:可以考虑在list\_1，list\_2上加一个多列索引，就可以使用这个索引做覆盖索引。

mysql> select list\_1,list\_2 from table\_name;

1. 索引限制问题：

* 没有查询条件，或者查询条件没有建立索引
* 查询的数量是表的30％以上。早期是这样的，现在优化器还会考虑其他因素比如表大小、表行数和I/O块大小
* 查询条件使用函数在索引列上，或者对索引列进行运算，运算包括(+，-，\*，/，! 等)例如：

select \* from test where id-1=9;

* 隐式转换导致索引失效。这一点应当引起重视，也是开发中经常会犯的错误。由于表的字段col1定义为VARCHAR(20)，但在查询时把该字段作为NUMERIC类型以where条件传给MySQL，这样会导致索引失效。例如：

select \* from test where col1=10;

* 使用不等于操作符(<>, !=)
* 禁止使用%前导查询，例如：like“%abc”，无法利用到索引。
* 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。

## SQL设计规范

1. 用IN代替OR。(OR的效率不如IN)SQL语句中IN包含的值不应过多，应少于500个。
2. 禁止隐式转换。数值类型禁止加引号，字符串类型必须加引号。
3. 减少与数据库交互次数，尽量采用批量SQL语句。例如：

insert into t(name) values(‘Jane’),(‘Join’),(‘Lucy’),…

1. 拆分复杂SQL为多个小SQL，避免大事务。（充分利用QUERY CACHE和充分利用多核CPU）
2. 获取大量数据时，建议分批次获取数据，每次获取数据少于2000条，结果集应小于1M。
3. 用UNION ALL代替UNION。UNION会去重，如果数据量大，会导致磁盘排序。
4. 统计行数用COUNT(\*)而不是COUNT(1)。MySQL对COUNT(\*)做了内部优化。
5. SELECT只获取必要的字段，禁止使用SELECT \*。
6. SQL中避免出现NOW()、RAND()、SYSDATE()、CURRENT\_USER()等不确定结果的函数。
7. INSERT语句必须指定字段列表，禁止使用INSERT INTO TABLE\_NAME VALUES()。
8. 禁止单条SQL语句同时更新多个表。
9. 程序应有捕获SQL异常的处理机制，必要时通过rollback显式回滚。
10. 通常情况下，子查询的性能比较差，建议改成JOIN写法。
11. 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，不是全表数据量小的）作为驱动表。
12. 去掉不必要的括号：

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

-> (a AND b AND c) OR (a AND b AND c AND d)

1. 去掉重叠常量：

(a<b AND b=c) AND a=5

-> b>5 AND b=c AND a=5

1. 去除常量条件(由于常量重叠需要) ：

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

-> B=5 OR B=6

1. 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。
2. 能不用NOT IN就不用，坑太多。会把空和NULL查出来。
3. 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。
4. 常数表优先于其他任何表 。常数表指：

* 空表或只有1行的表。
* 表的WHERE从句中使用PRIMARY KEY或UNIQUE索引，并且这些索引与常数表达式进行比较。如：

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

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

1. 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。
2. Select中将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。
3. 如果不使用GROUP BY或聚合函数时（COUNT()、MIN()等），将HAVING与WHERE合并。
4. 尽量使用主键进行UPDATE和DELETE 。
5. 对于SQL语句，要求所有研发人员，SQL关键字全部使用大写，每个词之间只允许有一个空格。
6. **LIMIT分页注意效率。LIMIT越大，效率越低。可以改写LIMIT，比如：**

select id from t limit 10000,10;

-> select id from t where id > 10000 limit 10;

1. 对同一个表的多次ALTER操作必须合并为一次操作。MySQL对表的修改绝大部分操作都需要锁表并重建表，而锁表则会对线上业务造成影响。为减少这种影响，必须把对表的多次ALTER操作合并为一次操作。例如要给表t增加一个字段b,同时给已有的字段a建立索引, 通常的做法分为两步：

alter table t add column b VARCHAR(10);

alter table t add index ind\_a(a);

正确的做法是：

alter table t add column b VARCHAR(10),add index ind\_a(a);

1. 减少使用视图，避免复杂的语句。
2. 使用PREPARED STATEMENT，可以提供性能并且避免SQL注入。
3. 禁止在UPDATE语句，将“,”写成“and”，非常危险。

正确示例：

update t set uid=uid+1000, gid=gid+1000 where id <=2;

错误示例：

update t set uid=uid+1000 and gid=gid+1000 where id <=2;

此时“uid+1000 and gid=gid+1000”将作为值赋给uid，并且无Warnings！！

## ****行为规范****

1. 提交线上建表改表需求，必须详细注明涉及到的所有SQL语句（包括INSERT、DELETE、UPDATE），建议提前发邮件给DBA，便于DBA进行审核和优化。
2. 所有的建表需要确定建立哪些索引后才可以建表上线。
3. 禁止有SUPER权限的应用程序账号存在。
4. 禁止有DDL、DCL权限的应用程序账号存在。
5. 重要项目的数据库方案选型和设计必须提前通知DBA参与。
6. 批量导入、导出数据必须通过DBA审核，并在执行过程中观察服务。
7. 批量更新数据，如UPDATE、DELETE操作，必须DBA进行审核，并在执行过程中观察服务。
8. 产品出现因数据库导致的故障时，如被攻击，必须及时通知DBA，便于维护服务稳定。
9. 业务部门程序出现BUG等影响数据库服务的问题，必须及时通知DBA，便于维护服务稳定。
10. 业务部门推广活动或上线新功能，必须提前通知DBA进行服务和访问量评估，并留出必要时间以便DBA完成扩容。
11. 出现业务部门人为误操作导致数据丢失，需要恢复数据的，必须第一时间通知DBA，并提供准确时间地点、误操作语句等重要线索。

## 缩进风格

1. 脚本代码块采用缩进风格书写，保证代码清晰易读，风格一致。缩进格数统一为2个空格，但需要关键字右对齐或者参数/字段向上对齐的时候例外。

示例：

IF lm\_count1 > 0 THEN

lm\_count1;

↑

两个空格

END IF;

1. 缩进必须使用空格键，不允许使用TAB键。
2. 同一条语句占用多于一行时，每行的第一个关键字应当右对齐，但INSERT...SELECT语句例外。

示例：

SELECT FIELD1, FIELD2, ...

FROM T\_CSP\_TABLENAME

WHERE FIELD1 > 1

AND (FIELD2 < SYSDATE OR FIELD2 > SYSDATE + 3)

AND FIELD3 LIKE 'HUAWEI%'

GROUP BY FIELD1, FIELD2

ORDER BY FIELD1, FIELD2;

↑

关键字右对齐

1. 在INSERT...SELECT语句中，如果需要换行时，应使INSERT INTO部分的字段与SELECT部分的字段一一对应，以增强可读性。

示例：

BEGIN

INSERT INTO T\_DEST\_TABLE\_NAME (

DESTFIELDNAME1 -- 注释

,DESTFIELDNAME2 -- 注释

,DESTFIELDNAME3 -- 注释

,DESTFIELDNAME4 -- 注释

,DESTFIELDNAME5 -- 注释

,DESTFIELDNAME6 -- 注释

,DESTFIELDNAME7 -- 注释

)

SELECT SRCFIELDNAME1 -- 注释

,SRCFIELDNAME2 -- 注释

,SRCFIELDNAME3 -- 注释

,SRCFIELDNAME4 -- 注释

,SRCFIELDNAME5 -- 注释

,SRCFIELDNAME6 -- 注释

,SRCFIELDNAME7 -- 注释

FROM T\_SRC\_TABLE\_NAME;

END;

1. INSERT INTO语句中，如果需要对每个字段增加注释，应将每个字段单独列为一行，并在行尾增加注释。

示例：

INSERT INTO T\_DEST\_TABLE\_NAME

(

DESTFIELDNAME1 -- 注释1

,DESTFIELDNAME2 -- 注释2

,DESTFIELDNAME3 -- 注释3

)

VALUES

(

FieldValue1

,FieldValue2

,FieldValue3

);

1. begin、end 独立成行。

示例：

BEGIN

NULL;

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

## 空格及换行

1. 不允许把多个短语句写在一行中，即一行只写一条语句。

示例：

SET @li\_Variable1 = 1;

SET @lv\_Variable2 = 'abc';

说明：两个赋值语句不能写在一行中，必须分两行写。

1. 相对独立的程序块之间、变量说明之后必须加空行。

示例：

-- 初始化局部变量

SET @li\_Variable1 = 1;

-- 判断开始时间

IF vd\_begindate IS NULL THEN

vd\_begindate = SYSDATE - 15;

END IF;

说明：两个程序块在逻辑上相对独立，应用空行加以分隔，同时增加注释。

1. 调用函数或过程时，如果参数列表超过120字符，应根据逻辑内容进行换行，或者每个参数占用一行。

示例：

P\_PROCEDURE(i\_Param1, -- 注释

i\_Param2, -- 注释

i\_Param3, -- 注释

o\_Param1, -- 注释

o\_Param2, -- 注释

...) -- 注释

1. 双目运算符、操作符前后应以空格分隔，间隔符之后应以空格分隔。

示例：

vd\_datevar := TO\_DATE('2001-01-01 01:30:00', 'YYYY-MM-DD HH24:MI:SS');

↑ ↑

赋值符前后加空格 逗号后面加空格

vi\_integervar1 := vi\_integervar2 + vi\_integervar3;

↑ ↑

赋值符前后加空格 加号前后加空格

IF vi\_number > 0 THEN

↑

比较符号前后加空格

## 注释规范

1. MySQL支持的注释符：单行注释符“#”，“-- ”（注意--后面必须有空格）；多行注释符“/\*…\*/”。特别的“/\*! …\*/”、“/\*!12345 …\*/”，MySQL不把它当成注释，但是其他的SQL服务器会当做注释。
2. 在数据库脚本文件头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、内容、功能、与其它文件的关系、修改日志等，头文件的注释中还应有函数或过程功能简要说明。

示例：

--------------------------------------------------------------

-- File Name : 该脚本文件的文件名

-- Author : 该脚本文件的作者姓名与工号

-- Version : 该脚本文件的版本号

-- Date : 该脚本文件的创建日期

-- Copyright : 该脚本文件的版权信息

-- Description : 该脚本文件的描述信息

-- Usage : 简要描述该脚本的使用/加载方法

-- History : 该脚本文件的维护历史信息

==============================================================Version Date Author Modification

版本号 修改日期（YYYY-MM-DD） 修改人姓名与工号 修改说明

--------------------------------------------------------------

1. 规则：函数/存储过程头部应进行注释，列出：名称、编码人、创建日期、版本信息、版权信息、功能说明、修改记录等，输入参数/输出参数/返回值的说明直接添加到参数后面。

示例：

/\*

Function\_Name : 该函数的名称

Author : 姓名 工号

Date : 创建日期YYYY-MM-DD

Version : 版本

Copyright : 该函数的版权信息

Description : 该函数的功能说明

History : 历史

==============================================================

Version Date Author Modification

版本号 修改日期（YYYY-MM-DD） 修改人姓名与工号 修改说明

\*/

1. 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。
2. 注释的内容要清楚、明了，含义准确，防止注释二义性。

说明：错误的注释不但无益反而有害。

1. 避免在注释中使用缩写，特别是非常用缩写。

说明：在使用缩写时或之前，应对缩写进行必要的说明。

1. 注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

示例：

BEGIN

……

-- i\_dateparam必须早于当前时间

IF i\_dateparam >= SYSDATE THEN

RETURN 1;

END IF;

……

END;

1. 注释与所描述内容进行同样的缩进。
2. 不允许在一行代码或表达式的中间插入注释。

说明：不应在代码或表达中间插入注释，否则容易使代码可理解性变差。

1. 通过对函数或过程、变量、结构等合适的命名以及合理地组织代码的结构，使代码成为自注释的。
2. 在代码的功能、意图层次上进行注释，提供有用、额外的信息。

示例：

如下注释意义不大

-- i\_retcode = 0

IF i\_retcode = 0 THEN

而如下的注释则给出了额外有用的信息。

-- 流程传入的标志为发送成功

IF i\_retcode = 0 THEN

# 附录

## 数据类型

### 数值类型

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 占用空间(字节) | 最小值(signed/Unsigned) | 最大值(signed/unsigned) |
| TINYINT | 1 | -128  0 | 127  255 |
| SMALLINT | 2 | -32768  0 | 32767  65535 |
| MEDIUMINT | 3 | -8388608  0 | 8388607  16777215 |
| INT | 4 | -2147483648  0 | 2147483647  4294967295 |
| BIGINT | 8 | -9223372036854775808  0 | 9223372036854775807  18446744073709551615 |

在 MySQL 中支持的 5 个主要整数类型是 TINYINT，SMALLINT，MEDIUMINT，INT 和 BIGINT。这些类型在很大程度上是相同的，只有它们存储的值的大小是不相同的。

MySQL 支持的三个小数类型是 FLOAT、DOUBLE 和 DECIMAL 类型。FLOAT 数值类型用于表示单精度浮点数值，而 DOUBLE 数值类型用于表示双精度浮点数值。与整数一样，这些类型也带有附加参数：一个精度指示器和一个小数指示器。比如语句 FLOAT(7,3) 规定显示的值不会超过 7 位数字，小数点后面带有 3 位数字。

对于小数点后面的位数超过允许范围的值，MySQL 会自动将它四舍五入为最接近它的值，再插入它。

DECIMAL[(M[,D])]用于保存必须为确切精度的值。同义词有NUMERIC[(M[,D])]、DEC[(M[,D])]、FIXED[(M[,D])]。对于如工资数据类型，当声明该类型的列，可以(并且通常)指定精度和标度，例如：SALARY DECIMAL(5,2)。

M是精度，D是标度。精度表示保存值的有效位数，标度表示小数点后面可以保存的位数。M的最大值是65，D的最大值是30。如果省略了标度，默认是0。如果省略了精度，默认是10。

### 日期和时间类型

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 占用空间(字节) | 取值范围（最小值/最大值） | 显示格式 |
| DATETIME | 8 | 1000-01-01 00:00:00  9999-12-31  23:59:59 | YYYY-MM-DD HH:MM:SS |
| DATE | 3 | 1000-01-01  9999-12-31 | YYYY-MM-DD |
| TIMESTAMP | 4 | 1970-01-01 00:00:01UTC  2038-01-19  03:14:07UTC | YYYY-MM-DD  HH:MM:SS |
| YEAR(4) | 1 | 1901  2155 | YYYY |
| TIME | 3 | -838:59:59  838:59:59 | HH:MM:SS  Or  HHH:MM:SS |

* DATE、TIME 和 YEAR 类型

MySQL 用 DATE 和 YEAR类型存储简单的日期值，使用 TIME 类型存储时间值。这些类型可以描述为字符串或不带分隔符的整数序列。如果描述为字符串，DATE 类型的值应该使用连字号作为分隔符分开，而 TIME 类型的值应该使用冒号作为分隔符分开。

需要注意的是，没有冒号分隔符的 TIME 类型值，将会被 MySQL 理解为持续的时间，而不是时间戳。TIME类型的时间可以大于23，是因为TIME类型不仅可以用来保存一天中的时间，也可以用来保存时间间隔，同时解释了为什么TIME类型也可以存在负值。

MySQL 还对日期的年份中的两个数字的值，或是 SQL 语句中为 YEAR类型输入的两个数字进行最大限度的通译。因为所有 YEAR类型的值必须用 4 个数字存储。MySQL 试图将 2 个数字的年份转换为 4 个数字的值。把在 00-69 范围内的值转换到 2000-2069 范围内。把 70-99 范围内的值转换到 1970-1979 之内。如果 MySQL 自动转换后的值并不符合我们的需要，请输入 4 个数字表示的年份。这里规定年份必须用4个数字表示。

* DATEYIME 和 TIMESTAMP 类型

除了日期和时间数据类型，MySQL 还支持 DATEYIME 和 TIMESTAMP 这两种混合类型。它们可以把日期和时间作为单个的值进行存储。这两种类型通常用于自动存储包含当前日期和时间的时间戳，并可在需要执行大量数据库事务和需要建立一个调试和审查用途的审计跟踪的应用程序中发挥良好作用。

### 字符串类型

* CHAR和VARCHAR

CHAR[(M)]用来保存固定长度的字符串，M的取值范围：0~255(latin)。默认M为1。CHAR 类型用于定长字符串，并且必须在圆括号内用一个大小修饰符来定义。这个大小修饰符的范围从 0-255。比指定长度大的值将被截短，而比指定长度小的值将会用空格作填补。

CHAR 类型可以使用 BINARY 修饰符。当用于比较运算时，这个修饰符使 CHAR 以二进制方式参于运算，而不是以传统的区分大小写的方式。

CHAR 类型的一个变体是 VARCHAR 类型。它是一种可变长度的字符串类型，并且也必须带有一个范围在 0-255 之间的指示器。CHAR 和 VARCHGAR 不同之处在于 MySQL 数据库处理这个指示器的方式：CHAR 把这个大小视为值的大小，长度不足的情况下就用空格补足。而 VARCHAR 类型把它视为最大值并且只使用存储字符串实际需要的长度（增加一个额外字节来存储字符串本身的长度）来存储值。所以短于指示器长度的 VARCHAR 类型不会被空格填补，但长于指示器的值仍然会被截短。

VARCHAR(N)用来保存变长字符类型，N的取值范围：0~65535(latin)，其中N都代表字符长度，而非字节长度。VARCHAR 类型在使用 BINARY 修饰符时与 CHAR 类型完全相同。

* BINARY和VARBINARY

BINARY和VARBINARY存储的是二进制的字符串，而非字符集型字符串。也就是说，BINARY和VARBINARY没有字符集的概念，对于排序和比较都是按照二进制值进行对比。

BINARY(N)和VARBINARY(N)中的N指的是字节长度，而非字符长度。

* ENUM和SET类型

ENUM和SET类型都是复合类型，一张表中ENUM和SET加起来不能超过255个唯一的元素。

* ENUM类型

ENUM 类型因为只允许在集合中取得一个值，有点类似于单选项。在处理相互排拆的数据时容易让人理解，比如人类的性别。ENUM 类型字段可以从集合中取得一个值或使用 NULL值，除此之外的输入将会使 MySQL 在这个字段中插入一个空字符串。ENUM 类型在系统内部存储为数字，并且从1开始用数字做索引。一个 ENUM 类型最多可以包含 65536 个元素。

MySQL认为 ENUM 类型集合中出现的值是合法输入，除此之外其它任何输入都将失败。这说明通过搜索包含空字符串或对应数字索引为 0 的行就可以很容易地找到错误记录的位置。

* SET 类型

SET 类型与 ENUM 类型相似但不相同。SET 类型可以从预定义的集合中取得任意数量的值。并且与 ENUM 类型相同的是任何试图在 SET 类型字段中插入非预定义的值都会使 MySQL 插入一个空字符串。如果插入一个即有合法的元素又有非法的元素的记录，MySQL 将会保留合法的元素，除去非法的元素。

一个 SET 类型最多可以包含 64 项元素。在 SET 元素中值被存储为一个分离的“位”序列，这些“位”表示与它相对应的元素。“位”是创建有序元素集合的一种简单而有效的方式。并且它还去除了重复的元素，所以 SET 类型中不可能包含两个相同的元素。

希望从 SET 类型字段中找出非法的记录只需查找包含空字符串或二进制值为 0 的行。

### 数据类型属性

1. AUTO\_INCREMENT

为新插入的行赋一个唯一的整数标识符。为列赋此属性将为每个新插入的行赋值为上一次插入的ID+1。MYSQL要求将AUTO\_INCREMENT属性用于作为主键的列。此外，每个表只允许有一个AUTO\_INCREMENT列。例如：

ID SMALLINT NOT NULL AUTO\_INCREMENT PRIMARY KEY

1. CHARACTER SET

只用于字符串类型值，指定要使用的字符集。

1. COLLATE

只用于字符串类型值，指定要使用的字符集校对。

1. BINARY

只用于字符串类型值。当为列指定了该属性时，将以区分大小写的方式排序。例如：

HOSTNAME CHAR(25) BINARY NOT NULL

1. DEFAULT

确保在没有任何值可用的情况下，赋予某个常量值，这个值必须是常量，因为MYSQL不允许插入函数或表达式值。此外，此属性无法用于BLOB或TEXT列。如果已经为此列指定了NULL属性，没有指定默认值时默认值将为NULL，否则默认值将依赖于字段的数据类型。例如：

STATUS\_ID TINYINT UNSIGNED NOT NULL DEFAULT 0

1. NOT NULL

如果将一个列定义为NOT NULL，将不允许向该列插入NULL值。建议在重要情况下始终使用NOT NULL属性，因为它提供了一个基本验证，确保已经向查询传递了所有必要的值。

1. NULL

为列指定NULL属性时，该列可以保持为空，而不论行中其它列是否已经被填充。记住，NULL精确的说法是“无”，而不是空字符串或0。

1. UNSIGNED

可用于任何数值类型，规定字段只保存正值。因为不需要保存数字的正、负符号，可以在存储时节约一个“位”的空间。从而增大这个字段可以存储的值的范围。

1. ZEROFILL

可用于任何数值类型，规定 0（不是空格）可以用来填补输出的值的显示宽度。向INT (10) UNSIGNED插入4时，将表示它为0000000004。如果为数值类型指定了ZEROFILL，MySQL会自动添加UNSIGNED属性。当存储的值超过显示宽度时，MySQL执行一些复杂的连接操作生成临时表时会产生问题，所以不建议使用该属性。