MySQL 开发规范

作者: 刘世发
创建时间: 2015-01-13
稳定程度: V 0.1 版

修改历史 History

Rev	date	Originator	Comment
0.1	2015-01-13	刘世发	建立

审核 Review

Department	Name	Acceptance Date	问题

1. 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。

1.1 目的

提供给开发人员参考，方便更有效率的开发。

1.2 范围

文档涉及的范围：需要基于 MySQL 做应用开发的人员。

2. 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

2.1 引擎及版本选择

- 存储引擎建议使用 Innodb
- 根据目前我们业务的情况，建议使用 MySQL5.6 社区版和 InnoDB plugin，后续 MySQL5.7 比较稳定后再行评估和升级

2.2 架构浅谈

开发大牛都擅长，这里不多提，仅标注一下。

2.2.1 非功能性需求

2.2.2 读写分离

2.2.3 分库分表

2.2.4 热点数据

2.3 schema 设计

2.3.1 禁止在数据库端做运算，

2.3.2 复杂运算移到程序端 CPU

2.3.3 尽可能简单应用 MySQL

如：md5() 或多个字段 order by，group by 或计算字段等操作不在数据库表上进行。

2.3.4 库库控制

- 单库不超过 800-1000 个表
- 单库总空间容量不超过 1T。

2.3.5 单表控制

- 单表不超过 3000W 行
- 单表总空间容量不超过 10G。

考虑因素

IO 高效；全表遍历；提高并发；alter table 快。

字段数量

建议上限 30~50 个。

一年内的单表数据量预估

建议纯 INT 不超 2000W，含 VARCHAR 不超 3000W。

2.3.6 拒绝 3B

- 大 SQL (BIG SQL)
- 大事务 (BIG Transaction)
- 大批量 (BIG Batch)

2.4 反范式设计

2.4.1 概念

- 无外键，尽量减少多表 join 查询
- 便于分布式设计，允许适度冗余，为了容量扩展允许适度开销
- 基于业务自由优化，基于 i/o 或查询设计，无须遵循范式结构设计

2.4.2 典型场景

- 原有展现程序涉及多个表的查询，希望精简查询程序
- 数据表拆分往往基于主键，而原有数据表往往存在非基于主键的关键查询，无法在分表结构中完成
- 存在较多数据统计需求（count,sum,max 等），效率低下

2.4.3 解决思路

基于统计的冗余设计

如：

- count(*)操作，需要不精准结果，可以直接 show table status like 'tablename' 获得，需要精准结果，可以在缓存层增加 key-value 对，实时更新该 key-value。同时异

步更新到数据库中冗余字段，或冗余表中。

历史数据表

- 历史数据表对应于热点数据表
- 将需求较少又不能丢弃的数据，仅在少数情况下被访问的数据存入历史数据表,如网银日志记录表等。

2.5 全文检索设计

2.5.1 最差的设计

- 直接在 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。

2.5.2 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

2.5.3 使用外部开源全文检索引擎

- 目前常用的有 sphinx 和 solr 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

2.6 分页设计

2.6.1 传统分页

```
Select * from table limit 10000,10;
```

2.6.2 LIMIT 原理

```
Limit 10000,10   偏移量越大则越慢
```

2.6.3 LIMIT 方式推荐分页

- select * from table WHERE AutoID >=20000 limit 1000; #1000+1 (每页 1000 条)
- select * from table WHERE AutoID >=21000 limit 1000;

分页方式二

```
Select * from table WHERE AutoID >= ( select AutoID from table limit 20000,1 ) limit 1000;
```

分页方式三

SELECT * FROM table INNER JOIN (SELECT AutoID FROM table LIMIT 20000,1000) USING (AutoID) ;

分页方式四

程序取 ID：select AutoID from table limit 20000,1000;
Select * from table WHERE AutoID in (123,456…) ;

以上几种分页方式同样适用于大批量拉取数据，如风控，自营贷等业务；

2.6.4 LIMIT 分页举例

- MySQL> select * from tablename limit 20000,1000;
- MySQL> select * from tablename WHERE AutoID >= 20000 limit 1000;
- MySQL> select * from tablename WHERE AutoID >= (select AutoID from tablename limit 20000,1) limit 1000 ;

2.6.5 LIMIT 分页请求量大的业务可以考虑增加缓存

2.7 表设计规范

2.7.1 基础规范

- 涉及系统目录、文件、表、字段名
- 全部使用 InnoDB 引擎，特殊业务需要使用特殊引擎的，通知 DBA 评估
- 字符集选择：latin1 => utf8 => gbk
- 使用数据库的目的是持久化存储以及保证事务一致性，不是运算器
- 读写分离，主库只写和少量实时读取请求
- 采用队列方式合并多次写请求，持续写入，避免瞬间压力
- 超长 text/blob 进行垂直拆分，并先行压缩
- 冷热数据进行水平拆分，LRU 原则
- 尽量不对 InnoDB 引擎表做在线实时 count(*)统计,特别是快速更新的大数据表
- 单表行记录数控制在 2000 万以内，行平均长度控制在 16KB 以内，单表 10GB 以内
- 单库下表数量不超过 1000 个
- 最少授权，只授予最基础权限需求
- 压力分散，在线表和归档表（日志，备份表类）分开存储
- 线上数据库和测试数据库尽可能保持一致
- 禁止明文存储机密数据，需至少一次加密（部分数据可逆运算）

2.7.2 命名规范

- 数据库表名应该有意义，并且易于理解，最好使用可以表达功能的英文单词或缩写，如果用英文单词表示，建议使用完整的英文单词

- 表名前必须加上前缀，表的前缀用一个业务系统或模块的英文名称首字母，前缀全部大写，表名中包含的单词首字母大写
- 强烈建议只用前缀字符+下划线+业务对象+业务内容组合，如：T_User_MailAccount，T_Mail_Notice 等，如果有分表，后面加上分表标示符号，如：T_User_Bill_88，T_Bill_ShoppingSheet_88 等,同一个库的所有表前缀必须一致
- 表命名长度不超过 32 个字符
- 每张表必须有 comment
- 不使用 select、show、update 等保留字或系统变量作为表名称
- 在表命名时应该用英文单词的单数形式，如用户表：应该为 T_UserInfo 而不是 T_UsersInfo
- 在表命名时禁止使用中文
- 临时用加上 tmp/temp 前缀/后缀
- 统计表加上 stat/statistic 前缀/后缀
- 历史归档表加上最后一次归档的完整日期，例如：T_Mail_Bill_20150115
- 数据备份表加上 _bak 后缀，如：T_Mail_Bill_Bak,按照日期备份的，在后缀后面加上备份日期，如：T_Mail_Bill_Bak_20150115

2.7.3 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。仅注意一下锁(gap lock)问题即可。

2.7.4 类型溢出

举例

以 MySQL5.6 版本，int 类型为例：

```
#建表
root@localhost(test)14:46>create table test (a int(10) UNSIGNED);
Query OK, 0 rows affected (0.12 sec)
#插入数据
root@localhost(test)14:56>insert test values (10);
Query OK, 1 row affected (0.00 sec)
#模拟更新溢出
root@localhost(test)14:56>update test 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(test)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
```

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。

2.7.5 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='888888@qq.com' WHERE uid=888888;
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 888888
#查看该执行计划
MySQL> explain select * from cdb_members WHERE uid=888888;
+----+-------------+-------+------+---------------+------+---------+------+------+--------------------------
----------------------+
| 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 888888 的执行计划
MySQL> explain select * from cdb_members WHERE uid=888888;
+----+-------------+-------+------+---------------+------+---------+------+------+--------------------------
----------------------+
```

| 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

```
mysql> show profile for query 1;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000035 |
| checking permissions | 0.000009 |
| Opening tables       | 0.000006 |
| System lock          | 0.000003 |
| Table lock           | 0.000003 |
| init                 | 0.000020 |
| Updating             | 6.935454 |
| end                  | 0.000021 |
| query end            | 0.000002 |
| freeing items        | 0.000026 |
| logging slow query   | 0.000002 |
| logging slow query   | 0.000054 |
| cleaning up          | 0.000003 |
+----------------------+----------+
13 rows in set (0.01 sec)
```

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）
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。

2.7.6 字段定义

常用字段类型：

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

2.7.7 字段规范

- 显式约束：所有字段必须定义为 NOT NULL,并且要有默认值
- 字段名称必须用表达字段功能的英文单词表示，建议使用完整的英文单词
- 字段中禁止使用下划线，连接符等特殊符号，每个单词首字母必须大写，如：BillDate，ReturnAmount，LastModifyTime，AmountMoney 等
- 注意字符集问题，server => databases => tables => column 命名长度不超过 32 个字符
- 用尽量少的存储空间来存数一个字段的数据，比如能用 int 的就不用 char 或者 varchar 能用 varchar(20)的就不用 varchar(255)
- 用 timestamp（4 字节 int unsigned，且效率非常高）记录时间，而非使用 date/datetime/char/varchar，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
- 整形定义中不添加长度，比如使用 INT，而不是 INT[4]
- IPV4 地址采用 4 字节 int unsigned，内置 INET_ATON/INET_NTOA 快速转换，采用 char 至少 15 字节
- 性别、状态、是否、小范围枚举使用 tinyint（0 ~ 255，或 -128 ~ 127）
- 不在数据库中使用 text，blob 存储图片、文件等,使用 mongodb 或文件系统代替
- 存储字符型数据时，尽可能先压缩或者序列化
- 不要同时指定字符集（character set）和校验集（collect set），避免出现和默认对应关系不一致
- 显式指定自增 int/bigint unsigned not null 作为主键
- 杜绝使用 UUID/HASH/MD5 类型作为主键
- 表与表之间的相关联字段要用统一名称
- 字段无须预留，越短越好

2.7.8 字段使用技巧

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)

  IP 存储：INET_ATON(expr)，将 IP 转换为整数。

  INET_NTOA(expr)，将整数转换为 IP

  mysql> SELECT INET_ATON('183.129.178.143');

  -> 3078730383

  mysql> SELECT INET_NTOA(3078730383);

  -> 183.129.178.143

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

3. SQL 语句规范

3.1 sql 语句编写规范

3.1.1 简化每一条 SQL，短事务、无阻塞、快速执行

3.1.2 用括号显式确定 AND、OR 的先后顺序，避免混淆

3.1.3 注意引号问题会导致类型转换（where id = ‘888888’）

3.1.4 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。

3.1.5 所有查询想尽一切办法使用索引：主键=>唯一索引=>索引

3.1.6 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、<、<=、>、>=）有可能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。

3.1.7 连表查询时，常数表优先，小表其次，大表最后

3.1.8 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。

3.1.9 where 子句中的数据扫描别跨越表的 30%

比如：where status <> 1 或者 status 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。

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 3 组，否则考虑业务逻辑或分表

3.1.11 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”。

3.1.12 杜绝使用 like '%xxx%'，尽量不用/少用 like 'xxx%'。

如果%必须放在首字符位置，参见“全文检索设计”

3.1.13 值域比较多的表字段放在前面

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

3.1.14 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。

3.1.15 表字段不能有表达式或是函数

如：where abs(列)>3 或 where 列*10>100

3.1.16 注意表字段的类型，避免表字段的隐示转换

参见“表字段设计”
比如：列为 int，如果 where 列='1'，则会出现转换。

3.1.17 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。

3.1.18 不同字段的值 or 或 in 大于等于 2 次，考虑用 union all 替换；同一字段的值 or 用 in 替换

```
select * from T_User_Bill_08 where userId=5964808 and status=0 or status=1;
考虑用
select * from T_User_Bill_08 where userId=5964808 and status in (0,1);
select * from T_User_Bill_40 where userId=6959140 and bankId=14 and status<>2 and (billDate='2014-04-16 00:00:00' or paymentDueDate='2014-05-06 00:00:00');
```

考虑用

```
select * from T_User_Bill_40 where userId=6959140 and bankId=14 and status<>2 and billDate='2014-04-16 00:00:00';
union all
select * from T_User_Bill_40 where userId=6959140 and bankId=14 and status<>2 and paymentDueDate='2014-05-06 00:00:00';
```

3.1.19 对同一表的 order by 和 group by 操作分别小于 2 组，否则考虑业务逻辑或分表

3.1.20 尽量使用主键进行 update 和 delete

3.1.21 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”

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

3.1.23 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”

3.1.24 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

3.1.25 增删改查语句中不使用不确定值函数和随机函数，可能造成额外开销或者导致无法使用索引如：RAND()和 NOW()等。

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

3.1.27 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。

3.1.28 不用/少用 FOR UPDATE、LOCK IN SHARE MODE，防止锁范围扩大化

3.1.29 使用合理的 SQL 语句减少与数据库的交互次数。

3.1.30 SQL 语句中 IN 包含的值不超过 100。

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

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

3.1.33 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。

3.1.34 禁止在 Update 语句中，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!!!**

4. 索引规范

- 非唯一索引按照“idx_字段名称_字段名称[字段名称, 字段名称]”进行命名。
- 唯一索引按照“uq_字段名称[字段名]”进行命名。
- 索引名称大小写和字段大小写保持一致。
- 索引中的字段数最好不超过 3 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，可参考索引当中的 Cardinality 值
- 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。
- 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。
- UPDATE、DELETE 语句需要根据 WHERE 条件添加索引。

- 对长度大于 50 的 VARCHAR 字段建立索引时，按需求恰当的使用前缀索引，或使用其他方法。
- 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。

5. 行为规范

- 批量导入、导出数据须提前通知 DBA，请求协助观察
- 推广活动或上线新功能须提前通知 DBA，请求压力评估
- 不使用 SUPER 权限连接数据库，分配给个人的权限不得转借给他人
- 单表多次 ALTER 操作必须合并为一次操作
- 数据库 DDL 及重要 SQL 及早提交 DBA 评审
- 重要业务库须告知 DBA 重要等级、数据备份及时性要求
- 不在业务高峰期批量更新、查询数据库
- 提交到线上的 DDL 需求，所有 SQL 语句须有备注说明

6. 测试环境规范

- 启用 log_queries_not_using_indexes，在大并发的测试过程中可以发现完全或者部分没有使用索引的 sql 语句
- 设置 long_query_time 为最小值，在大并发下可发现数据量少而且没有使用索引的语句
- 授权和生产环境一致
- 关闭 Query Cache，查询缓存在大并发写情况下，严重影响写入性能
- 设置较小 InnoDB Buffer Pool、key buffer size，在测试环境当中，走内存不一定是好事
- 数据量不能太少，否则有些性能问题无法提前规避