1. **前言**

本规范是在项目开发中整理的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。主要提供给需要基于MySQL做应用开发的人员作为参考，方便完成更有效率的开发。

**1.1 数据库设计**

数据库设计的目标三个：功能实现，可伸缩性，可用性。设计时需要平衡业务技术各个方面，做好取舍。数据库的架构设计时最重要的，80%的性能优势来自架构设计的优势。

**2.1 版本选择**

官方版本建议使用MySQL5.7.14，分支版本建议percona 5.6 ，5.7。

**2.2 架构设计**

Mysql数据库架构设计主要考虑读写分离、分库分表、热点数据、雪崩效应与过载保护、读写优化等方面。

读写分离用的是主从库的设计，至少要两台服务器，两边数据是同步的，主库负责写数据，从库负责读数据。

分库：尽可能将访问频繁的不同业务数据分在不同的数据库来存放，这样能提高并发访问效率。

分表：尽可能将大数据量的业务表采用某种分类标识来分成不同的表。可以考虑将历史数据和现实数据分开存放。

热点数据：对于经常要重复使用的数据，必须要放在内存中缓存起来，不能每次都从磁盘读取。可以采用全局内存变量、Memory Cache等。

雪崩效应是指当并发量大时，对一些表的访问会导致大量的锁出现，这样后来的数据库访问就要建立更多的数据库连接，使数据库性能下降甚至宕机。解决方法主要是优化数据库、从业务上优化设计、及时释放锁和系统资源、使用连接池等。

读写优化：读优化和热点数据类似，主要通过在内存中缓存数据实现尽量少的读取硬盘，尽量多的读取内存。写优化主要通过主键、索引实现。

**2.3 schema设计**

**2.3.1控制库和表的个数**

MySQL是单进程多线程架构的数据库，这点与SQL Server比较类似，但与Oracle多进程的架构有所不同（Oracle的Windows版本也是单进程多线程的架构）。这也就是说，MySQL数据库实例在系统上的表现就是一个进程。

从性能上考虑，单个MySQL库不能太大，总空间容量一般不超过100G，单库不超过500个表，因为MySQL表的表结构文件、数据文件、索引文件在操作系统上存放在schema的同一个目录下，当一个schema的表个数超过100个，即同一个目录下面的文件超过300时，操作系统管理文件的成本会大幅增加，影响服务器性能。

**2.3.2控制单表数据量**

表设计主要考虑因素有：IO高效、全表遍历、表修复快、提高并发；alter table快。

单表数据量建议控制在纯INT不超1000W，含CHAR不超500W，因为Mysql在处理大表（char的表>500W行，或int表>1000W）时，性能就开始明显降低，所以要采用不同的方式控制单表容量：

A、根据数据冷热，对数据分级存储，历史归档。

B、采用分库/分表/分区表，横向拆分控制单表容量。

C、对于OLTP[系统](http://www.2cto.com/os/)，控制单事务的资源消耗，遇到大事务可以拆解，采用化整为零模式，避免特例影响大众。

D、单库不要超过500个表。

E、单表字段数不要太多，最多不要大于50个。

**2.3.3数据冗余设计**

数据库冗余设计的目的：

A、无外键时，减少多表join查询。

B、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。

C、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。

冗余设计的应用场景：

A、原有展现程序涉及多个表的查询，希望简化查询。

B、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。

C、存在较多数据统计需求（count, sum等），效率低下。

冗余的设计思路举例：

A、基于展现的冗余设计，如：

消息表message，存在字段 from\_userid,to\_userid,msg,send\_time四个字段，而展示程序需要显示发送者姓名和性别。

通常在message表中增加冗余字段from\_username和from\_user\_sex即可。

B、基于查询的冗余设计，如：

用户分表，将用户库分成若干数据表。基于用户名的查询和基于userid的查询都是高并发请求。用户分表可以基于userid分成多个表，同时基于用户名做对应冗余表。

C、基于统计的冗余设计，如：

count(\*)操作，如不需要精准结果，可以直接show table status like …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。

**2.3.4控制事务大小**

限制大SQL (BIG SQL)、大事务 (BIG Transaction)、大批量 (BIG Batch)。遇到大sql时，可以考虑根据业务分拆成几个小sql，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。

**2.3.5存储引擎选择**

默认使用InnoDB引擎。InnoDB适用于几乎99%的MySQL应用场景，而且在MySQL 5.7的系统表都改成InnoDB了，还有什么理由再死守MyISAM呢。

**2.3.6字符集**

**使用优先级 utf8mb4  > utf8 > latin1**

**2.3.7表主键**

**显示指定自增int/bigint unsigned not null 作为主键，尽量不要使用uuid/HASH/MD5类型作为主键。**

**2.4 编码规范**

**2.4.1命名规范**

**注意：数据库名长度尽量控制在14个字符以内！**

Mysql对象名称最长是64个字符，为了阅读方便，我们要求对象名控制在32个字符以内。且数据库名、表名、字段名、索引名等强烈建议只用小写字符、数字、下划线组合，不使用 **desc,select ,show ,update** 等mysql关键字，临时表加上tmp 后缀，统计表加上statistic后缀，日志表加上log后缀等。

|  |  |
| --- | --- |
| **对象** | **规范** |
| 表 | t\_应用名\_模块名\_功能名 |
| 表的字段 | 英文单词或缩写，避开关键字 |
| 视图 | v\_表名 |
| 存储过程 | p\_表名 |
| 函数 | f\_功能说明 |
| 包 | pkg\_功能说明 |
| 触发器 | tri\_表名 |
| 主键 | primary |
| 索引 | idx\_字段1\_字段2 |
| 唯一索引 | uniq\_字段1\_字段2 |

表的命名应尽量反映存储的数据内容。

2.4.2表字段的设计

字段的命名以单词或者单词缩写为主，避开数据库关键字如all、type等。

Mysql字段类型 ：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 列类型 | 表达的范围 | 存储需求 |
| 1 | TINYINT[(M)] [UNSIGNED] [ZEROFILL] | -128 到 127 或 0 到 255 | 1 个字节 |
| 2 | SMALLINT[(M)] [UNSIGNED] [ZEROFILL] | -32768 到 32767 或 0 到 65535 | 2 个字节 |
| 3 | INT[(M)] [UNSIGNED] [ZEROFILL] | -2147483648 到 2147483647 或 0 到 4294967295 | 4 个字节 |
| 4 | BIGINT[(M)] [UNSIGNED] [ZEROFILL] | -9223372036854775808 到 9223372036854775807  或 0 到 18446744073709551615 | 8 个字节 |
| 5 | 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 |
| 6 | DATE | YYYY-MM-DD | 3 个字节 |
| 7 | DATETIME | YYYY-MM-DD HH:MM:SS(1001年到 9999 年的范围 ) | 8 个字节 |
| 8 | TIMESTAMP | YYYY-MM-DD HH:MM:SS （1970年到2037年的范围） | 4 个字节 |
| 9 | 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） |
| 10 | 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） |
| 11 | TEXT | 64K 个字符 | 所占空间跟字符集等有关系 |

说明：

1. 所有动态长度字符串全部使用 VARCHAR 类型，类似于状态，有限类别的字段， 也使用可以比较明显表示出实际意义的字符串，而不应该使用INT之类的数字来代替；

2. 固定长度的字符串使用 CHAR 类型，所有单个字符的全部使用 CHAR 类型，而不应该使用VARCHAR 类型；

3. 仅仅当字符数量可能超过 20000 个的时候，可以使用 TEXT 类型来存放字符类数据。所有使用 TEXT 类型的字段必须和原表进行分拆，与原表主键单独组成另外一个表进行存放；

4. 需要精确到时间（年月日时分秒）的字段可以使用DATETIME 或TIMESTAMP，但请注意各自能表达的范围，以及是否需要用到TIMESTAMP的特性；精确到微秒建议时间类型转换为整形BIGINT存储（建议优先使用BIGINT类型存储日期）；

5. 所有只需要精确到天的字段全部使用 DATE 类型，而不应该使用 TIMESTAMP或者DATETIME 类型；

6. 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；能有tinyint的就不要用smallint ，能用smallint的就不要用int，能用int的就不要用bigint

7. 索引字段使用not null：MySQL NULL类型和Oracle的NULL有差异，会进入索引中，如果是一个组合索引，那么这个NULL类型的字段会极大影响整个索引的效率。此外，NULL 在索引中的处理也是特殊的，也会占用额外的存放空间。

8. 有小数点的字段或者精度要求高的字段用decimal，禁用float.

**2.4.3索引的设计**

索引按照“idx\_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。

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。

对超过50个长度的字符串列，最好创建前缀索引而非整列索引（例如：ALTER TABLE t1 ADD INDEX(user(50))），可以有效提高索引利用率，不过它的缺点是对这个列排序时用不到前缀索引。前缀索引的长度可以基于对该字段的统计得出，一般略大于平均长度一点就可以了。

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

**2.4.4表注释、字段注释**

Mysql中字段加注释比较麻烦，需要用alter table语句，所以尽量在新建表的时候就加上表注释和字段注释。类型字段的注释（注释中必须包括字段初始值的含义）。

**3.Sql规范**

**DDL规范：**

•字段指定  not  null  default  xxx

•所有字段，表均需要有注释  comment

•所有新表，引擎只能用  innodb

•所有新表，均有自增  ID  做为主键

•按照一定比例，限制新表text字段数量

•按照一定比例，规定新表索引数量

•按照业务常识，post\_id,  puid,  userid  均有索引

**DML规范：**

•不允许使用  load  data

•删除修改操作，where  clause  必须包含主键，唯一索引列，或是业务区分度较好列

•禁止多表  join

•禁止未决  SQL,  例如  insert  into  select

•所有  insert  语句必须指定字段

•所有  update  /  delete  建议使用绝对值，遵循可重入原则