**MySQL开发规范**

## 基本原则

* SQL语句尽可能简单。
* 大语句拆小语句，减少锁时间。一条大SQL可以堵死整个库。同时增加缓存利用效率，拆分后的语句可以在缓存层分别缓存
* 尽量使用简单事务，避免使用大事务。
* 避免使用存储过程、触发器、外键、函数、视图、事件等，由客户端程序取而代之。主要原因是降低业务耦合度，在某些场景会导致主从不一致，同时不利于问题排查和统一运维

## 语句规范

### 2.1. 精简写法

* 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)

* 去掉重叠常量

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

=>

b>5 AND b=c AND a=5

* 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

* 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。

### 2.2. 数据过滤

* 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。

* 控制扫描规模，WHERE子句中的数据扫描别跨越表的30%。
* 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。
* 使用LIKE时，%不要放在首字符位置。如果%必须放在首字符位置，可考虑使用全文索引或利用搜索引擎实现。
* 值域比较多的表字段(选择率高的)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，避免表字段的隐示转换

比如：列为int，如果where 列=’1’，则会出现转换。

* 考虑使用LIMIT N，少用LIMIT M,N，特别是大表，或M比较大的时候。

### 2.3. 表关联、子查询

* 避免使用表关联，MySQL处理表关联效率较低。
* 多表联接查询时，关联字段类型尽量一致，并且都要有索引。
* 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，不一定是全表数据量小的）作为驱动表；
* 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。
* 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。

* 通常情况下，子查询的性能比较差，建议改造成JOIN写法。

### 2.4. 排序、分组、集合

* 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。

* 尽可能在索引中完成排序

利用索引排序，主要是利用了索引的有序性。

* 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

* 考虑使用UNION ALL，少使用UNION，但注意考虑去重。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';

=>

select \* from opp

where phone in ('12347856' , '42242233');

select \* from opp

where phone='010-88886666' or cellPhone='13800138000';

=>

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;

=>

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。

### 2.5. 索引

* 多用复合索引，少用多个独立索引，尤其是一些基数（Cardinality）太小（比如说，该列的唯一值总数少于255）的列就不要创建独立索引了。
* 类似分页功能的SQL，建议先用主键关联，然后返回结果集，效率会高很多。

### 2.6. DML

* 尽量使用主键进行UPDATE和DELETE。
* INSERT语句必须指明需要插入的字段名，避免由于字段变更导致的INSERT失败
* 使用INSERT ... ON DUPLICATE KEY UPDATE(INSERT IGNORE)来避免不必要的查询。
* 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。
* INSERT和SELECT明确指明字段，避免字段变更导致预期外的业务失败。
* UPDATE、DELETE语句不使用LIMIT。有主键ID的表WHERE条件应结合主键。
* 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,…)

* 使用LOAD DATA导数据

load data比insert快约20倍。

### 2.7. 其他

* 避免使用存储过程、触发器、函数、UDF、events等，容易将业务逻辑和数据库耦合在一起。
* 减少使用视图，避免复杂的语句。
* 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。
* 读取数据时，只选取所需要的列，不要每次都SELECT \*，避免产生严重的随机读问题、减少网络流量等，尤其是读到一些TEXT/BLOB列。避免由于字段变更导致的其他与该表相关业务的query失效
* OR改写为IN()

OR的效率是N级别，IN的消息时LOG(N)级别，IN的个数建议控制在200以内。

select id from t where phone=′159′ or phone=′136′;

=>

select id from t where phone in (′159′, ′136′);

* OR改写为UNION

mysql的索引合并很弱智。

select id from t where phone = ′159′ or name = ′john′;

=>

select id from t where phone=′159′

union

select id from t where name=′jonh′