

# MySQL中，21个写SQL的好习惯（修正版）

---

 [mp.weixin.qq.com/s/Ew\\_jC0NQC5m\\_EcJ0eih7lg](https://mp.weixin.qq.com/s/Ew_jC0NQC5m_EcJ0eih7lg)

捡田螺的小男孩 程序员大咖 2020-11-13 10:24

---

**Python实战社群**

**Java实战社群**

长按识别下方二维码，按需求添加

扫码关注添加客服

**进Python社群▲**

扫码关注添加客服

**进Java社群▲**



**Python实战技术交流群**



该二维码7天内有效，重新进入将更新



Java实战技术交流群



该二维码7天内有效，重新进入将更新

作者 | 捡田螺的小男孩

---

来源 | 捡田螺的小男孩

---

## 1. 写完SQL先explain查看执行计划（SQL性能优化）

日常开发写SQL的时候，尽量养成这个好习惯呀：写完SQL后，用explain分析一下，尤其注意走不走索引。

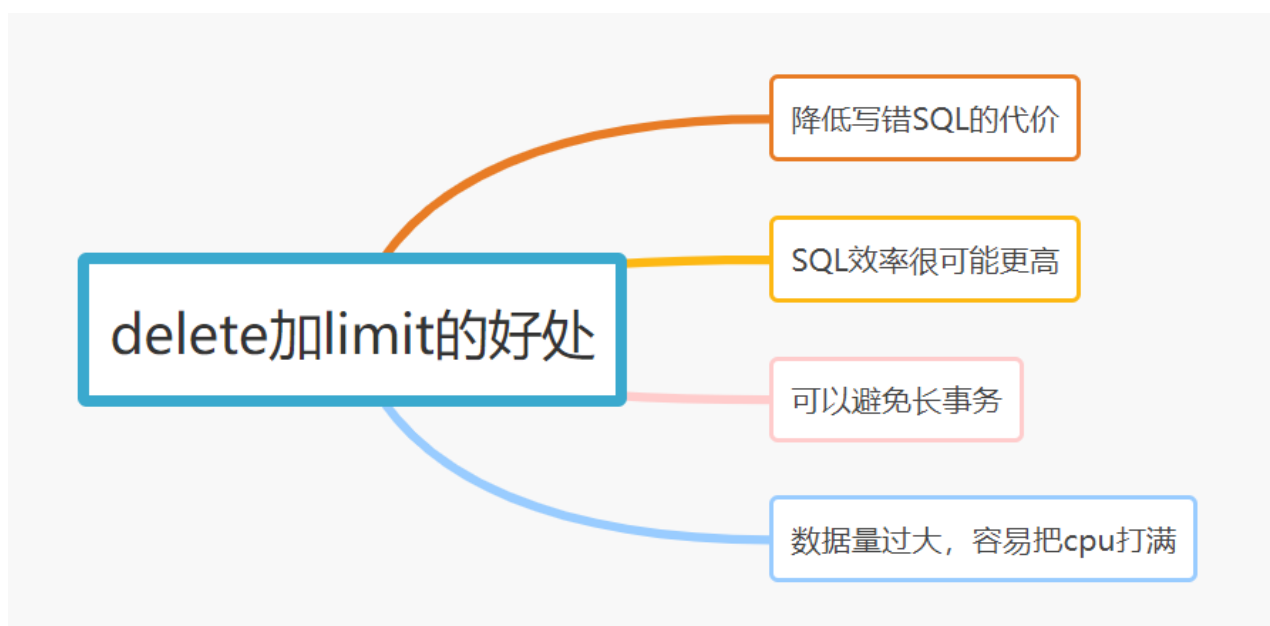
```
explain select userid,name,age from user
where userid =10086 or age =18;
```

## 2. 操作delete或者update语句，加个limit(SQL后悔药)

在执行删除或者更新语句，尽量加上limit，以下面的这条 SQL 为例吧：

```
delete from euser where age > 30 limit 200;
```

因为加了limit 主要有这些好处：



- 「降低写错SQL的代价」，你在命令行执行这个SQL的时候，如果不加limit，执行的时候一个「不小心手抖」，可能数据全删掉了，如果「删错」了呢？加了limit 200，就不一样了。删错也只是丢失200条数据，可以通过binlog日志快速恢复的。
- 「SQL效率很可能更高」，你在SQL行中，加了limit 1，如果第一条就命中目标return，没有limit的话，还会继续执行扫描表。
- 「避免了长事务」，delete执行时,如果age加了索引，MySQL会将所有相关的行加写锁和间隙锁，所有执行相关行会被锁住，如果删除数量大，会直接影响相关业务无法使用。
- 「数据量大的话，容易把CPU打满」,如果你删除数据量很大时，不加 limit限制一下记录数，容易把cpu打满，导致越删越慢的。

## 3. 设计表的时候，所有表和字段都添加相应的注释（SQL规范优雅）

这个好习惯一定要养成啦，设计数据库表的时候，所有表和字段都添加相应的注释，后面更容易维护。

「正例：」

```
CREATE TABLE `account` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键Id',
  `name` varchar(255) DEFAULT NULL COMMENT '账户名',
  `balance` int(11) DEFAULT NULL COMMENT '余额',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`id`),
  KEY `idx_name` (`name`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1570068 DEFAULT CHARSET=utf8 ROW_FORMAT=REDUNDANT CO
户表';
```

「反例：」

```
CREATE TABLE `account` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `balance` int(11) DEFAULT NULL,
  `create_time` datetime NOT NULL ,
  `update_time` datetime NOT NULL ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `idx_name` (`name`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1570068 DEFAULT CHARSET=utf8;
```

#### 4. SQL书写格式，关键字大小保持一致，使用缩进。（SQL规范优雅）

---

「正例：」

```
SELECT stu.name, sum(stu.score)
FROM Student stu
WHERE stu.classNo = '1班'
GROUP BY stu.name
```

「反例：」

```
SELECT stu.name, sum(stu.score) from Student stu WHERE stu.classNo = '1
班' group by stu.name.
```

显然，统一关键字大小写一致，使用缩进对齐，会使你的SQL看起来更优雅~

#### 5. INSERT语句标明对应的字段名称（SQL规范优雅）

---

「反例：」

```
insert into Student values ('666','捡田螺的小男孩','100');
```

「正例：」

```
insert into Student(student_id,name,score) values ('666','捡田螺的小男孩','100');
```

#### 6. 变更SQL操作先在测试环境执行，写明详细的操作步骤以及回滚方案，并在上生产前review。（SQL后悔药）

---

- 变更SQL操作先在测试环境测试，避免有语法错误就放到生产上了。
- 变更Sql操作需要写明详细操作步骤，尤其有依赖关系的时候，如：先修改表结构再补充对应的数据。
- 变更Sql操作有回滚方案，并在上生产前，review对应变更SQL。

## 7.设计数据库表的时候，加上三个字段：主键，create\_time,update\_time。（SQL规范优雅）

---

「反例：」

```
CREATE TABLE `account` (
  `name` varchar(255) DEFAULT NULL COMMENT '账户名',
  `balance` int(11) DEFAULT NULL COMMENT '余额',
) ENGINE=InnoDB AUTO_INCREMENT=1570068 DEFAULT CHARSET=utf8 ROW_FORMAT=REDUNDANT CO
户表';
```

「正例：」

```
CREATE TABLE `account` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键Id',
  `name` varchar(255) DEFAULT NULL COMMENT '账户名',
  `balance` int(11) DEFAULT NULL COMMENT '余额',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`id`),
  KEY `idx_name` (`name`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1570068 DEFAULT CHARSET=utf8 ROW_FORMAT=REDUNDANT CO
户表';
```

「理由：」

- 主键一般都要加上的，没有主键的表是没有灵魂的
- 创建时间和更新时间的话，还是建议加上吧，详细审计、跟踪记录，都是有用的。

阿里开发手册也提到这个点，如图

### 9.【强制】表必备三字段：id，gmt\_create，gmt\_modified。

**说明：**其中id必为主键，类型为bigint unsigned、单表时自增、步长为1。gmt\_create，gmt\_modified的类型均为datetime类型，前者现在时表示主动创建，后者过去分词表示被动更新。

## 8.写完SQL语句，检查where,order by,group by后面的列，多表关联的列是否已加索引，优先考虑组合索引。（SQL性能优化）

---

「反例：」

```
select * from user
where address ='深圳' order by age;
```

localhost test2 运行 停止 解释

```
1 explain select * from user where address = '深圳' order by age;
```

信息	结果 1	剖析	状态									
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra	
1	SIMPLE	user	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	1	100	Using	

「正例：」

添加索引

```
alter table user add index idx_address_age (address,age)
```

localhost test2 运行 停止 解释

```
1 explain select * from user where address = '深圳' order by age;
```

信息	结果 1	剖析	状态									
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra	
1	SIMPLE	user	(Null)	ref	idx_address_age	idx_ad	768	const	1	100	Using	

## 9.修改或删除重要数据前，要先备份，先备份，先备份（SQL后悔药）

如果要修改或删除数据，在执行SQL前一定要先备份要修改的数据，万一误操作，还能吃口「后悔药」~

## 10. where后面的字段，留意其数据类型的隐式转换（SQL性能优化）

「反例：」

//userid 是varchar字符串类型

```
select * from user where userid =123;
```

localhost test2 运行 停止 解释

```
1 explain select * from user where userid=123
```

信息	结果 1	剖析	状态									
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra	
1	SIMPLE	user	(Null)	ALL	idx_userid	(Null)	(Null)	(Null)	1	100	Using where	

「正例：」

```
select * from user where userid ='123';
```

1 explain select * from user where userid='123';											
信息	结果 1	剖析	状态								
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	user	(Null)	ref	idx_userid	idx_userid	98	const	1	100	(Null)

「理由：」

因为不加单引号时，是字符串跟数字的比较，它们类型不匹配，MySQL会做隐式的类型转换，把它们转换为浮点数再做比较，最后导致索引失效

## 11. 尽量把所有列定义为NOT NULL (SQL规范优雅)

- 「NOT NULL列更节省空间」，NULL列需要一个额外字节作为判断是否为NULL的标志位。
- 「NULL列需要注意空指针问题」，NULL列在计算和比较的时候，需要注意空指针问题。

## 12.修改或者删除SQL，先写WHERE查一下，确认后再补充 delete 或 update (SQL后悔药)

尤其在操作生产的数据时，遇到修改或者删除的SQL，先加个where查询一下，确认OK之后，再执行update或者delete操作

## 13.减少不必要的字段返回，如使用select <具体字段> 代替 select \* (SQL性能优化)

「反例：」

```
select * from employee;
```

「正例：」

```
select id, name from employee;
```

理由：

- 节省资源、减少网络开销。
- 可能用到覆盖索引，减少回表，提高查询效率。

## 14.所有表必须使用Innodb存储引擎 (SQL规范优雅)

Innodb 「支持事务，支持行级锁，更好的恢复性」，高并发下性能更好，所以呢，没有特殊要求（即Innodb无法满足的功能如：列存储，存储空间数据等）的情况下，所有表必须使用Innodb存储引擎

## 15.数据库和表的字符集尽量统一使用UTF8 (SQL规范优雅)

尽量统一使用UTF8编码



- 可以避免乱码问题
- 可以避免，不同字符集比较转换，导致的索引失效问题

「如果需要存储表情，那么选择utf8mb4来进行存储，注意它与utf-8编码的区别。」

## 16. 尽量使用varchar代替 char。（SQL性能优化）

---

「反例：」

```
`deptName` char(100) DEFAULT NULL COMMENT '部门名称'
```

「正例：」

```
`deptName` varchar(100) DEFAULT NULL COMMENT '部门名称'
```

理由：

因为首先变长字段存储空间小，可以节省存储空间。

## 17. 如果修改字段含义或对字段表示的状态追加时，需要及时更新字段注释。（SQL规范优雅）

---

这个点，是阿里开发手册中，Mysql的规约。你的字段，尤其是表示枚举状态时，如果含义被修改了，或者状态追加时，为了后面更好维护，需要即时更新字段的注释。

## 18. SQL命令行修改数据，养成begin + commit 事务的习惯(SQL后悔药)

---

「正例：」

```
begin;
update account set balance =1000000
where name ='捡田螺的小男孩';
commit;
```

「反例：」

```
update account set balance =1000000
where name ='捡田螺的小男孩';
```

## 19. 索引命名要规范，主键索引名为 pk\_ 字段名；唯一索引名为 uk\_ 字段名；普通索引名则为 idx\_ 字段名。（SQL规范优雅）

---

说明：pk\_即primary key；uk\_即unique key；idx\_即index 的简称。

## 20. WHERE从句中不对列进行函数转换和表达式计算

---

假设loginTime加了索引

「反例：」

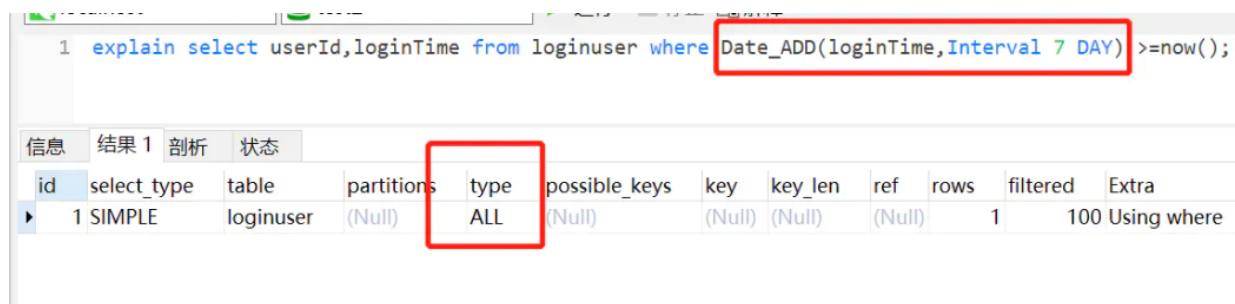
```
select userId,loginTime
from loginuser
where Date_ADD(loginTime,Interval 7 DAY) >=now();
```

「正例：」

```
explain select userId,loginTime
from loginuser
where loginTime >= Date_ADD(NOW(),INTERVAL - 7 DAY);
```

「理由：」

索引列上使用mysql的内置函数，索引失效



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	loginuser	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	1	100	Using where

## 21.如果修改/更新数据过多，考虑批量进行。

反例：

```
delete from account limit 100000;
```

正例：

```
for each(200次)
{
delete from account limit 500;
}
```

理由：

- 大批量操作会造成主从延迟。
- 大批量操作会产生大事务，阻塞。
- 大批量操作，数据量过大，会把cpu打满。

## 参考与感谢

- [delete后加 limit是个好习惯么]  
([https://blog.csdn.net/qq\\_39390545/article/details/107519747](https://blog.csdn.net/qq_39390545/article/details/107519747))
- 《阿里开发手册》



程序员专栏  
扫码关注填加客服  
长按识别下方二维码进群



程序员大咖实战技术交流群



该二维码7天内有效，重新进入将更新

---

近期精彩内容推荐：


- ➔ [程序员买房前后对比，看完后已哭瞎...](#)
- ➔ [内部泄露版！互联网大厂的薪资和职级一览](#)
- ➔ [Google 出品的 Java 编码规范](#)
- ➔ [Python编程 高阶函数使用技巧](#)



长按关注回复：100  
进程序员大咖实战技术交流群

长按下方二维码  
关注公众号



在看点这里  好文分享给更多人↓↓