

# MySQL 开发规约实战

芦火

阿里云运维专家

前言



五、MySQL 数据库

(一) 建表规约

- 1. **【强制】**表达是与否概念的字段，必须使用 is\_xxx 的方式命名，数据类型是 unsigned tinyint（1 表示是，0 表示否）。

说明：任何字段如果为非负数，必须是 unsigned。

注意：POJO 类中的

Xxx 的映射关系。数

义与取值范围。

(二) 索引规约

- 1. **【强制】**业务上具有唯一特性的字段，即使是组合字段，也必须建成唯一索引。  
说明：不要以为唯一索引影响了 insert 速度，这个速度损耗可以忽略，但提高查找速度是明显的；另外，即使在应用层做了非常完善的校验控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。
- 2. **【强制】**超过三个表禁止 join。需要 join 的字段，数据类型保持绝对一致；多表关联查询时，保证被关联的字段
- 3. **【强制】**在 varchar

(三) SQL 语句

- 1. **【强制】**不要使用 count(列名)或 count(常量)来替代 count(\*)，count(\*)是 SQL92 定义的标准统计行数的语法，跟数据库无关，跟 NULL 和非 NULL 无关。  
说明：count(\*)会统计值为 NULL 的行，而 count(列名)不会统计此列为 NULL 值的行。
- 2. **【强制】**count(distinct col) 计算该列除 NULL 之外的不重复行数，注意 count(distinct col1, col2) 如果其中一列全为 NULL，那么即使另一列有不同的值，也返回为 0。
- 3. **【强制】**当某

(四) ORM 映射

- 1. **【强制】**在表查询中，一律不要使用 \* 作为查询的字段列表，需要哪些字段必须明确写明。  
说明：1) 增加查询分析器解析成本。2) 增减字段容易与 resultMap 配置不一致。3) 无用字段增加网络消耗，尤其是 text 类型的字段。
- 2. **【强制】**POJO 类的布尔属性不能加 is，而数据库字段必须加 is\_，要求在 resultMap 中进行字段与属性之间的映射。  
说明：参见定义 POJO 类以及数据库字段定义规定，在 sql.xml 增加映射，是必须的。

# 语句规范要建立在结构规范的基础上

## 字符集

统一字符集，建议UTF8mb4

统一排序规则

常用字符集	描述	默认校对规则	最大长度	备注
latin1	cp1252 West European	latin1_swedish_ci	1	早期官方默认字符集
gbk	GBK Simplified Chinese	gbk_chinese_ci	2	非国际标准
utf8	UTF-8 Unicode	utf8_general_ci	3	alias for utf8mb3
utf8mb4	UTF-8 Unicode	utf8mb4_0900_ai_ci	4	官方8.0默认字符集

## 字段

统一字段名、类型

解决业务歧义、隐式转换问题

字段长度 varchar(255)

方便，但存在性能隐患

定义 id int primary key

UNSIGNED 容量大一倍，PK 强制

禁止Null值

Null & Null =?

## 语句规范要建立在结构规范的基础上

### 索引

80%的语句性能问题都可以靠索引解决

单列索引要充分评估

定期review索引有效性

例如：覆盖索引已cover单列

不要走极端

复合索引所有列

所有列都建单列索引

有关索引有关内容，请关注【MySQL 表和索引优化实战】课程

## 01 SQL语句编写规范

## 02 事务的使用与优化

## 02 开发中常见问题与最佳实践



# SQL语句编写规范

## 规范语法

### 不兼容语法

Select \* from sbtest.sbtest1 group by id;

Only\_full\_groupby

Select id,count(\*) from sbtest.sbtest1 group by id desc;

8.0不再支持

## 别名

所有返回列要给有意义的命名，与列名原则一致，强制 as 关键词

Select id,count(\*) id\_count from sbtest.sbtest1 group by id;

# SQL语句编写规范

## 执行顺序

- 1 .FROM, including JOINs
2. WHERE
3. GROUP BY
4. HAVING
5. WINDOW functions
6. SELECT
7. DISTINCT
8. UNION
9. ORDER BY
- 10 .LIMIT and OFFSET



## 语句性能

数据流的流向

比如: order by limit 场景

## 数据返回逻辑

数据的筛选机制

比如: left join where场景

# SQL语句编写规范

## 如何判断语句是否已最优:explain

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	range	idx_pk	idx_pk	100	NULL	1460	1.11	Using index condition; Using where; Using MRR
1	SIMPLE	b	NULL	ref	i_test	i_test	302	b.c.KeyNo	1	100.00	Using where; Using index

TYPE:

ALL: Full Table Scan, 全表扫描

index: Full Index Scan, 索引扫描

range:范围扫描

ref: 表示连接匹配条件

eq\_ref: 类似ref, 区别就在使用的索引是唯一索引

const: 常量查询, 比如pk等值

system是const类型的特例  
当查询的表只有一行的情况下, 使用system

Extra:

Using filesort

Using index

Using index condition

Using temporary

Using where

Using index & Using where

排序

使用索引可以返回请求列

通过索引初步过滤

回表再过滤其它条件

临时表

单独出现时

一般代表表上出现全表扫描过滤

使用索引返回数据

同时通过索引过滤



# SQL语句编写规范

## 禁止项

- ✓ select \*, 返回无用数据, 过多IO消耗, 以及schema变更问题
- ✓ insert语句指定具体字段名称, 不要写成insert into t1 values(...), 道理同上
- ✓ 禁止不带WHERE, 导致全表扫描以及误操作
- ✓ where条件里等号左右字段类型必须一致, 否则可能会产生隐式转换, 无法利用索引
- ✓ 索引列不要使用函数或表达式, 否则无法利用索引。  
如where length(name)= 'Admin' 或where user\_id+2=5
- ✓ Replace into, 会导致主备不一致
- ✓ 业务语句中带有DDL操作, 特别是truncate

## 建议项

- ✓ 减小三表以上Join
- ✓ 用union all 替代 union
- ✓ 使用join 替代子查询
- ✓ 不要使用 like '%abc%', 可以使用 like 'abc%'
- ✓ Order by /distinct /group by 都可以利用索引有序性
- ✓ 减少使用event/存储过程, 通过业务逻辑实现
- ✓ 减小where in() 条件数据量
- ✓ 减少过于复杂的查询. & 拼串写法

## SQL语句编写规范

用数据库的思维考虑SQL

- ✗ - 一个语句解决所有问题  
导致过于复杂的查询，执行计划不稳定
- ✗ - 开发应用的逻辑写语句  
所有的运算、判断应用逻辑都放到SQL实现
- ✗ - 存储过程使用过重  
难以调试、定位问题

- ✓ - 少即是美  
每一层结果集都要最大限度的减小
- ✓ - 数据集处理，减小单条处理
- ✓ - 减小数据访问（扫描）
- ✓ - 新feature谨慎应用到生产中

# SQL语句编写规范

## Sql改写-join

```
select count(a.id) from sbtest1 a left join sbtest2 b on a.id=b.id;
```

```
mysql> select count(a.id) from sbtest1 a left join sbtest2 b on a.id=b.id;
+-----+
| count(a.id) |
+-----+
|      20000000 |
+-----+
1 row in set (19.34 sec)
```

请注意join 键为PK，也就是左表右表应该是1对1的关系

# SQL语句编写规范

## Sql改写-join

select count(a.id) from sbtest1 a left join sbtest2 b on a.id=b.id;

等价于

select count(a.id) from sbtest1 a ;

```
mysql> explain select count(*) from sbtest1 a left join sbtest2 b on a.id=b.id;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | a | NULL | index | NULL | k_1 | 4 | NULL | 19728432 | 100.00 | Using index |
| 1 | SIMPLE | b | NULL | eq_ref | PRIMARY | PRIMARY | 4 | sbtest.a.id | 1 | 100.00 | Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set, 1 warning (0.00 sec)

mysql> explain select count(a.id) from sbtest1 a ;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | a | NULL | index | NULL | k_1 | 4 | NULL | 19728432 | 100.00 | Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> select count(a.id) from sbtest1 a ;
+-----+
| count(a.id) |
+-----+
| 20000000 |
+-----+
1 row in set (3.32 sec)
```

Sql改写一般会出现在复杂查询的Join场景中

除去显式join，还包括：

- 半连接：exists,in
- 反连接：not exists,not in

此类查询过慢时，请考虑是否可通过SQL改写优化



# SQL语句编写规范

## Sql改写-分页统计

```
select a.id from sbtest1 a left join sbtest2 b on a.id=b.id limit 200,20;
```

取总数据量:

```
select count(*) from  
(select a.id from sbtest1 a left join sbtest2 b on a.id=b.id) as a;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	a	NULL	index	NULL	k_1	4	NULL	19728432	100.00	Using index
1	SIMPLE	b	NULL	eq_ref	PRIMARY	PRIMARY	4	sbtest.a.id	1	100.00	Using index

执行计划上无本质区别，但语句冗余



# SQL语句编写规范

## Sql改写-分页统计

改写1:

```
select count(a.id) from sbtest1 a left join sbtest2 b on a.id=b.id;
```

改写2:

```
select count(a.id) from sbtest1 a;
```

此类改写目的:

- 1.精简语句，简化语句逻辑
- 2.进一步寻找优化空间

```
mysql> select count(a.id) from sbtest1 a left join sbtest2 b on a.id=b.id;
```

```
+-----+
| count(a.id) |
+-----+
|      20000000 |
+-----+
```

```
1 row in set (15.06 sec)
```

```
mysql> select count(a.id) from sbtest1 a;
```

```
+-----+
| count(a.id) |
+-----+
|      20000000 |
+-----+
```

```
1 row in set (0.25 sec)
```

**01** SQL语句编写规范

**02** 事务的使用与优化

**02** 开发中常见问题与最佳实践

# 事务的使用与优化

事务是什么？

并发控制的单位

4个属性：

- Atomicity
- Consistency
- Isolation
- Durability

隔离级别

Read Uncommitted

Read Committed (一般采用)

Repeatable Read (官方默认)

Serializable

概念不再具体解释

只强调一点：

大事务 不等于 长事务

例如：

1.

Insert table batch

2.

Begin

insert single data

sleep(3600)

Commit

# 事务的使用与优化

## 事务的问题

### 1.Undo 异常增长

ibdata空间问题, History list过长

### 2.binlog 异常增长

单个事务不拆分存放

### 3.Slave延迟

DDL类, 写入等

### 4.锁问题

死锁、阻塞

## 优化

### 大事务

- 1.大事务拆分为小事务
- 2.DDL拆分（无锁变更）

### 长事务

- 1.合并为大事务（特别是写入场景）
- 2.事务分解（不必要的请求摘除）
- 3.应用侧保证一致性

### 事务使用基本原则

在保证业务逻辑的前提下，尽可能缩短

# 事务的使用与优化

## 事务问题定位

长事务：  
Information\_schema.innodb\_trx

例如：

```
SELECT trx.trx_id, trx.trx_started,
trx.trx_mysql_thread_id FROM
INFORMATION_SCHEMA.INNODB_TRX trx WHERE
trx.trx_started < CURRENT_TIMESTAMP - INTERVAL 1
SECOND
```

## 锁问题

8.0以前：  
information\_schema.innodb\_lock\_waits、 innodb\_locks

8.0  
performance\_schema. data\_lock\_waits、 data\_locks

Field	Type	Null	Key	Default	Extra
ENGINE	varchar(32)	NO		NULL	
REQUESTING_ENGINE_LOCK_ID	varchar(128)	NO	MUL	NULL	
REQUESTING_ENGINE_TRANSACTION_ID	bigint(20) unsigned	YES	MUL	NULL	
REQUESTING_THREAD_ID	bigint(20) unsigned	YES	MUL	NULL	
REQUESTING_EVENT_ID	bigint(20) unsigned	YES		NULL	
REQUESTING_OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	NO		NULL	
BLOCKING_ENGINE_LOCK_ID	varchar(128)	NO	MUL	NULL	
BLOCKING_ENGINE_TRANSACTION_ID	bigint(20) unsigned	YES	MUL	NULL	
BLOCKING_THREAD_ID	bigint(20) unsigned	YES	MUL	NULL	
BLOCKING_EVENT_ID	bigint(20) unsigned	YES		NULL	
BLOCKING_OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	NO		NULL	



**01** SQL语句编写规范

**02** 事务的使用与优化

**02** 开发中常见问题与最佳实践

## 开发中常见问题与最佳实践

### 分页问题


#### 传统写法

```
select * from sbtest1 order by id limit M,N
```

#### 问题点:

扫描大量无效数据后，返回请求数据

#### 执行顺序

1. FROM, including JOINS 
2. WHERE
3. GROUP BY
4. HAVING
5. WINDOW functions
6. SELECT
7. DISTINCT
8. UNION
9. ORDER BY
10. LIMIT and OFFSET

# 开发中常见问题与最佳实践

## 分页问题

- **select \* from sbtest1 where id > #max\_id# order by id limit n;**
  - 适用顺序翻页的场景，每次记录上一页#max\_id#带入下一次查询中
- **select \* from sbtest1 as a inner join (select id from sbtest1 order by id limit m, n) as b on a.id = b.id order by a.id;**
  - 适用只按照id进行分页,无where条件
- **select \* from sbtest1 as a inner join (select id from sbtest1 where col=xxxx order by id limit m, n) as b on a.id = b.id order by a.id;**
  - 适用于带where条件，同时按照id顺序分页
  - 此时，需要在where条件上创建二级索引

# 开发中常见问题与最佳实践

## 大表数据清理

数据清理场景：

### ➤ 历史数据清理

- 通常按照条件（比如：时间范围）delete历史数据
- 问题
  - ① 单次delete行数过多，容易导致锁堵塞、主从复制延迟、影响线上业务
  - ② 易失败，死锁、超时等
- 建议方案
  - ① 伪代码

```
Select min(id),max(id) from t where gmt_create<$date
```

```
For l in "max(id)-min(id)/1000"
```

```
Delete from t where id>=min(id) and id<min(id)+1000 and gmt_create<$date
```

.....

- ② 定期optimize table回收碎片

### ➤ 全表数据清理

- Truncate整张表的数据
- 问题
  - 大表（如：>100G），truncate期间会造成io持续抖动
- 建议方案
  - 硬连接方式后truncate，异步trim文件

## 开发中常见问题与最佳实践

### 隐式转换问题

```
Create table testtb(id varchar(10) primary key);  
Select * from testtb where id=1;
```

隐式转换发生在比较值类型不一值的场景下，除去一些规定情况  
最终都是转换为浮点数进行比较

此类问题在编写时很难发现，上线导致严重性能问题



## 开发中常见问题与最佳实践

### 循环

#### 外部循环

应用侧实现

主要问题来自每次请求的rt  
例如:

```
for i=0;i++;i<500  
    insert (db 交互)  
next
```

$rt = \text{single rt} * \text{total count}$   
建议batch一次写入

#### 内部循环

一般常用在存储过程  
事务无法保证

```
While do  
    insert;  
    Commit;  
end while
```

频繁commit

或:

```
Begin tran  
While do  
    insert;  
end while  
Commit
```

数据一致性  
以及长事务

## 开发中常见问题与最佳实践

### 存储过程中的事务处理

```
create procedure insertTest(IN num int)
BEGIN
  DECLARE errno int;
  declare i int;
  declare continue HANDLER for sqlexception set errno=1;
  start transaction;
  set i=0;
  while i<num do
    INSERT testfor VALUES(i);
    set i=i+1;
  end while;
  if errno=1 then
    rollback;
  else
    commit;
  end if;
end;
```

# 开发中常见问题与最佳实践

## 常见问题-1

1. Where 后面的列顺序是不是要符合最左原则?

Where a=1 and b=2 等价于 Where b=2 and a=1

最左原则指的是索引顺序，不是谓词顺序，以上两个条件都匹配( a,b) 复合索引

2. Join 的顺序是不是指定了左边为驱动表

inner join场景下，在执行计划中按预估自动选中驱动表，left join ,right join 时左右写的顺序才有显式意义

3.业务上有随机返回的需求，可不可以用order by rand()

一般不建议，如果结果集非常小，勉强可用，但结果集大时由于随机数排序，会产生 sort操作，甚至溢出到磁盘，有很大性能损耗

此类需求可以考虑伪随机算法，具体不再此提供

4.Delete数据之后，为什么磁盘空间占用反而大了?

Delete数据并不能清理数据文件空间，反而会导致undo,binlog文件的增长，使用optimize收缩

# 开发中常见问题与最佳实践

## 常见问题-2

### 1. Binlog是否一定要row格式

在主从场景下，Binlog使用row格式是为了保证主从数据一致性  
单机场景下，Binlog做为增长数据备份使用，同时也包括一些语句级数据恢复的功能

### 2. 死锁、阻塞的区别

通常说的阻塞，主要是由于锁获取不到，产生的请求被阻塞了，一般需要手动解锁(kill或等待)  
死锁不等于阻塞，虽然死锁中阻塞是必现的，但是会自动回滚事物解锁，不用手动处理，但需要业务判断语句逻辑  
以上两种情况都是由于业务侧逻辑出现，并非内核原因

### 3.做DDL时是否会锁表

参考官方文档：  
<https://dev.mysql.com/doc/refman/5.6/en/innodb-online-ddl-operations.html>

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Creating or adding a secondary index	Yes	No	Yes	No
Dropping an index	Yes	No	Yes	Yes
Adding a FULLTEXT index	Yes*	No*	No	No
Changing the index type	Yes	No	Yes	Yes



