
sql 教程 Documentation

Release 0.1

root@workgroup.cn

October 29, 2013

CONTENTS

1	设计原则	1
1.1	任何语句使用前 explain 看执行计划是否用到索引	1
1.2	不要从明细表查统计结果，定期统计插入到汇总表	1
1.3	禁止使用 SELECT *，必须指定字段名称，包括 insert table 后边加字段列表	2
1.4	明细统计时，只统计编码，不要关联名称等冗余字段	3
1.5	明细统计时，只统计编码，不要关联名称等冗余字段	3
1.6	每个查询结果集使用的内存量不要超过 256M，可以通过时间范围控制，如 RK BETWEEN A AND B，建议大表按可小时操作	4
1.7	页面查询在 10 秒内要返回结果	5
1.8	联合查询时，每个表必须加别名以提高 SQL 解析效率	5
1.9	语句中避免使用 GROUP BY，可通过批量程序定期汇总	5
2	字段设计	7
2.1	尽可能使用更小的数据类型，如 TINYINT、smallint、MEDIUMINT、INT、BIGINT (如 int(11) 的 11 代表客户端显示宽度，并不是取值范围，tinyint -2 ⁸ -2 ⁸ -1, smallint -2 ¹⁵ -2 ¹⁵ -1 int -2 ³¹ -2 ³¹ -1 bigint -2 ⁶³ -2 ⁶³ -1)	7
2.2	尽量少用 TEXT、BLOB 等专有类型 (用链接代替)	7
2.3	字符型，数值型字段类型不能混合使用，依赖后期转换	7
2.4	相同字段不同表中的类型和长度要一致	7
2.5	字段名称不能使用关键字	7
2.6	不要指定字段级编码，建议全库统一	7
2.7	默认值要规范，例如日期不要使用 0000-00-00	8
2.8	不要用自增 ID 做主键	8
2.9	不要使用外键	8
2.10	事务相关记录保留时间戳，建议只增不改；在必须对记录进行修改的时候，保留更改时间戳	8
3	索引使用	9
3.1	一般情况下，一次查询只会用到一个索引 (特定情况出现 merge index 的情况，如下可能出现 (a=1 or b=2) 会合并 a 和 b 的索引，或者使用 union all)	9
3.2	每个表索引越少越好	13
3.3	每个查询必须用到索引	13
3.4	建立组合索引时，WHERE 条件中用到等于的字段放前边，用到范围的字段放后边	13
3.5	删除重复字段的索引，减少 dml IO	13
3.6	除了主键外，避免建立其他唯一性索引	13
3.7	索引中重复的记录数越少，效率越高，效率最高的是主键	13
3.8	索引字段最好不要存在 NULL	13
3.9	组合索引可以只使用第一个，或者前两个，或者前几个	14
4	查询条件	17

4.1	SQL 语句的 WHERE 条件避免无效条件和无效括号	17
4.2	SQL 语句中不要加用不到的排序	17
4.3	WHERE 条件中最好不要用 IN 和 LIKE	17
4.4	WHERE 条件中不要使用 NOW() 等进行判断	17
4.5	索引要使用的字段不要使用函数或者进行运算	18
4.6	禁止字段格式转换	18
5	存储过程	19
5.1	存储过程中操作的记录数超过 1000 条时不能使用游标	19
5.2	在存储过程的关键步骤开始和结束都要记录信息到日志表，用于监控和调试	20
5.3	字符变量使用单引号，不要使用双引号	23
5.4	存储过程要能够重复执行，执行时需要清空历史冲突记录	23
6	远程表	25
6.1	远程表结构要与原始表一致，尤其是索引	25
6.2	远程表数据不要大于 256M，远程表的 WHERE 无效	25
6.3	远程表一般用来全表小数据全量同步	25
7	查询技巧	27
7.1	SQL 语句不要太长	27
7.2	避免使用 LIKE	27
7.3	WHERE 多个 OR 条件不走一个索引时可通过 UNION	28
8	性能优化	31
8.1	编译器、文件格式、磁盘、索引结构	31
8.2	使用 type=heap 的临时表	31
9	引擎使用	33
9.1	innodb 引擎，在过程结尾提交，避免过度 commit	33
10	权限控制	35
10.1	PHP 连接 MYSQL 的用户只分配 SIUD 权限	35
10.2	所有提交变量经过 mysql_real_escape_string 进行转义	35

设计原则

1.1 任何语句使用前 **explain** 看执行计划是否用到索引

```
EXPLAIN
SELECT
  `emp_no`,
  `birth_date`,
  `first_name`,
  `last_name`,
  `gender`,
  `hire_date`
FROM
  `employees`
WHERE `first_name` = 'Georgi'
```

1.2 不要从明细表查统计结果，定期统计插入到汇总表

1.2.1 一般做法

```
EXPLAIN
SELECT
  `first_name`,
  COUNT(1) AS emps
FROM
  `employees`
WHERE `first_name` = 'Georgi'
GROUP BY `first_name`
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employees	ALL	NULL	NULL	NULL	NULL	300016	Using where

1.2.2 使用汇总表

建立汇总表：

```
CREATE TABLE `name_emps` (
  `first_name` VARCHAR(14) PRIMARY KEY,
  `count_num` INT
);
```

定时插入数据进汇总表之前先清空汇总表：

```
TRUNCATE TABLE name_emps;
```

定时插入数据进汇总表：

```
INSERT INTO `name_emps` (`first_name`, `count_num`)
SELECT
    `first_name`,
    COUNT(1) AS emps
FROM
    `employees`
GROUP BY `first_name`
```

查询汇总表数据：

```
EXPLAIN
SELECT
    `first_name`,
    `count_num`
FROM
    `employees`.`name_emps`
WHERE `first_name` = 'Georgi'
```

Query Favorites ▾ Query History ▾ Run									
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	name_emps	const	PRIMARY	PRIMARY	16	const	1	

1.3 禁止使用 **SELECT ***，必须指定字段名称，包括 **insert table** 后边加字段列表

```
SELECT
    *
FROM `employees`.`employees`
LIMIT 0, 1000;
```

正确做法：

```
SELECT
    `emp_no`,
    `birth_date`,
    `first_name`,
    `last_name`,
    `gender`,
    `hire_date`
FROM `employees`.`employees`
LIMIT 0, 1000;
```

1.4 明细统计时，只统计编码，不要关联名称等冗余字段

```
SELECT
    t.`emp_no`,
    e.`first_name`,
    COUNT(1) AS nums
```

```
FROM
  `employees`.`titles` AS t
  LEFT JOIN employees AS e
    ON e.`emp_no` = t.`emp_no`
GROUP BY t.emp_no
```

或者

```
SELECT
  t.`emp_no`,
  (SELECT
    `first_name`
  FROM
    `employees`
  WHERE emp_no = t.emp_no
  LIMIT 1) AS first_name,
  COUNT(1) AS nums
FROM
  `employees`.`titles` AS t
GROUP BY t.emp_no
```

正确做法

```
SELECT
  `emp_no`,
  COUNT(1) AS nums
FROM
  `employees`.`titles`
GROUP BY emp_no
```

取出结果后，根据实际情况再取 first_name。部分基础数据可以考虑运用缓存存储

1.5 联合查询时，每个表必须加别名以提高 SQL 解析效率，如 SELECT T1.BM FROM GS T1 LEFT JOIN GSJJ T2 ON T1.BM=T2.BM

```
SELECT
  t.`emp_no`,
  e.`first_name`,
  COUNT(1) AS nums
FROM
  `employees`.`titles` AS t,
  employees AS e
WHERE e.`emp_no` = t.`emp_no`
GROUP BY t.emp_no
```

正确做法：

```
SELECT
  t.`emp_no`,
  e.`first_name`,
  COUNT(1) AS nums
FROM
  `employees`.`titles` AS t
  INNER JOIN employees AS e
    ON e.`emp_no` = t.`emp_no`
GROUP BY t.emp_no
```

1.5. 联合查询时，每个表必须加别名以提高 SQL 解析效率，如 SELECT T1.BM FROM GS T1 LEFT JOIN GSJJ T2 ON T1.BM=T2.BM

1.6 每个查询结果集使用的内存量不要超过 256M，可以通过时间范围控制，如 **RK BETWEEN A AND B**，建议大表按可小时操作

```
SELECT
  `emp_no`,
  `birth_date`,
  `first_name`,
  `last_name`,
  `gender`,
  `hire_date`
FROM `employees`.`employees`
```

改写为：

```
SELECT
  `emp_no`,
  `birth_date`,
  `first_name`,
  `last_name`,
  `gender`,
  `hire_date`
FROM
  `employees`.`employees`
WHERE birth_date BETWEEN '1989-01-01 00:00:00'
  AND '1990-01-01 00:00:00'
```

1.7 页面查询在 10 秒内要返回结果

页面查询在 10 秒内要返回结果, 服务器超时限制默认是 65 秒 (查看 query cache 是否足够大和命中率, show variables like '%cache%') ;

1.8 联合查询时，每个表必须加别名以提高 SQL 解析效率

如 SELECT T1.BM FROM GS T1 LEFT JOIN GSJJ T2 ON T1.BM=T2.BM

```
SELECT
  `title`
FROM
  `titles`,
  `employees`
WHERE titles.`emp_no` = employees.`emp_no`
```

正确做法：

```
SELECT
  t.`title`
FROM
  `employees`.`titles` AS t,
  `employees` AS e
WHERE e.`emp_no` = t.`emp_no`
GROUP BY t.`emp_no`
```


1.9 语句中避免使用 **GROUP BY**, 可通过批量程序定期汇总

参考不要从明细表查统计结果，定期统计插入到汇总表

字段设计

2.1 尽可能使用更小的数据类型

int(11) 的 11 代表客户端显示宽度，并不是取值范围

类型	取值范围
tinyint	$-2^8, 2^8-1$
smallint	$-2^{15}, 2^{15}-1$
int	$-2^{31}, 2^{31}-1$
bigint	$-2^{63}, 2^{63}-1$

2.2 尽量少用 TEXT、BLOB 等专有类型

用单独的表来保存 TEXT 列内容

```
CREATE TABLE titles (  
    id INT PRIMARY KEY,  
    title VARCHAR (50),  
    db_time TIMESTAMP,  
    author INT  
);
```

```
CREATE TABLE contents (  
    title_id INT PRIMARY KEY,  
    content TEXT  
);
```

用文件链接来保存 TEXT 列内容,content 列保存文件 path

```
CREATE TABLE titles (  
    id INT PRIMARY KEY,  
    title VARCHAR (50),  
    content VARCHAR(60),  
    db_time TIMESTAMP,  
    author INT  
)
```

2.3 字符型，数值型字段类型不能混合使用，依赖后期转换

试验：表 employees 增加字符列，存储数字, 并建立索引。

```
ALTER TABLE employees
  ADD COLUMN var1 VARCHAR (6);
CREATE INDEX var1
ON employees (var1);
```

把 emp_no 更新到 var1 列。

```
UPDATE
  employees
SET
  var1 = emp_no
```

用 explain 查询，注意 2 幅截图的”key” 字段。

```
EXPLAIN
SELECT
  `emp_no`,
  `birth_date`,
  `first_name`,
  `last_name`,
  `gender`,
  `hire_date`
FROM `employees`.`employees`
WHERE var1=10001;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employees	ALL	var1	NULL	NULL	NULL	300016	Using where

```
EXPLAIN
SELECT
  `emp_no`,
  `birth_date`,
  `first_name`,
  `last_name`,
  `gender`,
  `hire_date`
FROM `employees`.`employees`
WHERE var1='10001';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employees	ref	var1	var1	9	const	1	Using where

2.4 相同字段不同表中的类型和长度要一致

如 employees 库中各个表中：

emp_no	int
from_date	date
dept_no	char(4)

2.5 字段名称不能使用关键字

如 count, group, int, select 等, 避免使用关键字, 减少出错机率.

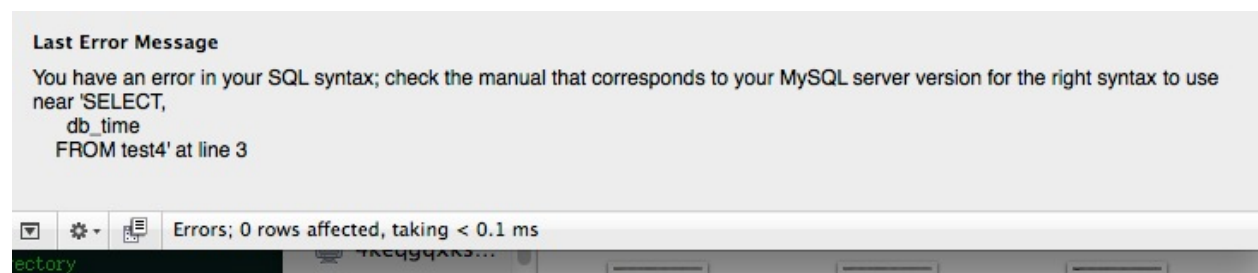
```
CREATE TABLE test1 (
  `id` INT PRIMARY KEY,
  `select` INT,
  `db_time` TIMESTAMP
)
```

查询时列名加“`”会得到正确结果：

```
SELECT
  `id`,
  `select`,
  `db_time`
FROM `test1`
```

如果忘了加“`”号, 则下面语句会出错

```
SELECT
  id,
  SELECT,
  db_time
FROM test1
```



2.6 不要指定字段级编码，建议全库统一

如下载的 employess 库的 sql 文件，并没有单独指定表、列的字符集，而是全部统一使用服务器的默认字符集。

2.7 默认值要规范

例如日期不要使用 0000-00-00，有的数据库不支持 ‘0000-00-00 00:00:00’ 的日期格式，在数据传输过程中可能需要做多余的转换工作

mysql 的 boolean 和有的数据库也不一样，mysql 是以 0 和 1 来代表 false 和 true。

2.8 不要用自增 ID 做主键

留空

2.9 不要使用外键

留空

2.10 事务相关记录保留时间戳，建议只增不改

在必须对记录进行修改的时候，保留更改时间戳

索引使用

3.1 一般情况下，一次查询只会用到一个索引 (特定情况出现 merge index 的情况, 如下可能出现 (a=1 or b=2) 会合并 a 和 b 的索引, 或者使用 union all)

explain mysql 测试合并索引

建立索引:

```
CREATE INDEX employees_firstname
ON employees (first_name);
```

```
CREATE INDEX employees_lastname
ON employees (last_name);
```

a=1 or b=2 情况下 :

```
EXPLAIN
SELECT
  emp_no,
  birth_date,
  first_name,
  last_name,
  gender hire_date
FROM
  employees
WHERE first_name = 'Georgi'
OR last_name = 'Simmel' ;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees	index_merge	employees_firstname,employees_lastname	employees_firstname,employees_lastname	16,18		420	100.00	Using union(employees_firstname,last_name); Using where

```
EXPLAIN EXTENDED
SELECT
  emp_no,
  birth_date,
  first_name,
  last_name,
  gender hire_date
FROM
```

```
employees
WHERE first_name = 'Georgi'
UNION
ALL
SELECT
  emp_no,
  birth_date,
  first_name,
  last_name,
  gender hire_date
FROM
  employees
WHERE last_name = 'Simmel' ;
```

id	se-lect_type	ta-ble	type	possi-ble_keys	key	key_len	ref	rows	fil-tered	Extra
1	PRI-MARY	em-ploy-ees	ref	emply-ees_firstname	emply-ees_firstname	16	const	253	100.00	Using where
2	UNION	em-ploy-ees	ref	last_name	last_name	18	const	167	100.00	Using where

- in 与 union: 当条件参数为大量的时候，union all 明显慢于 in

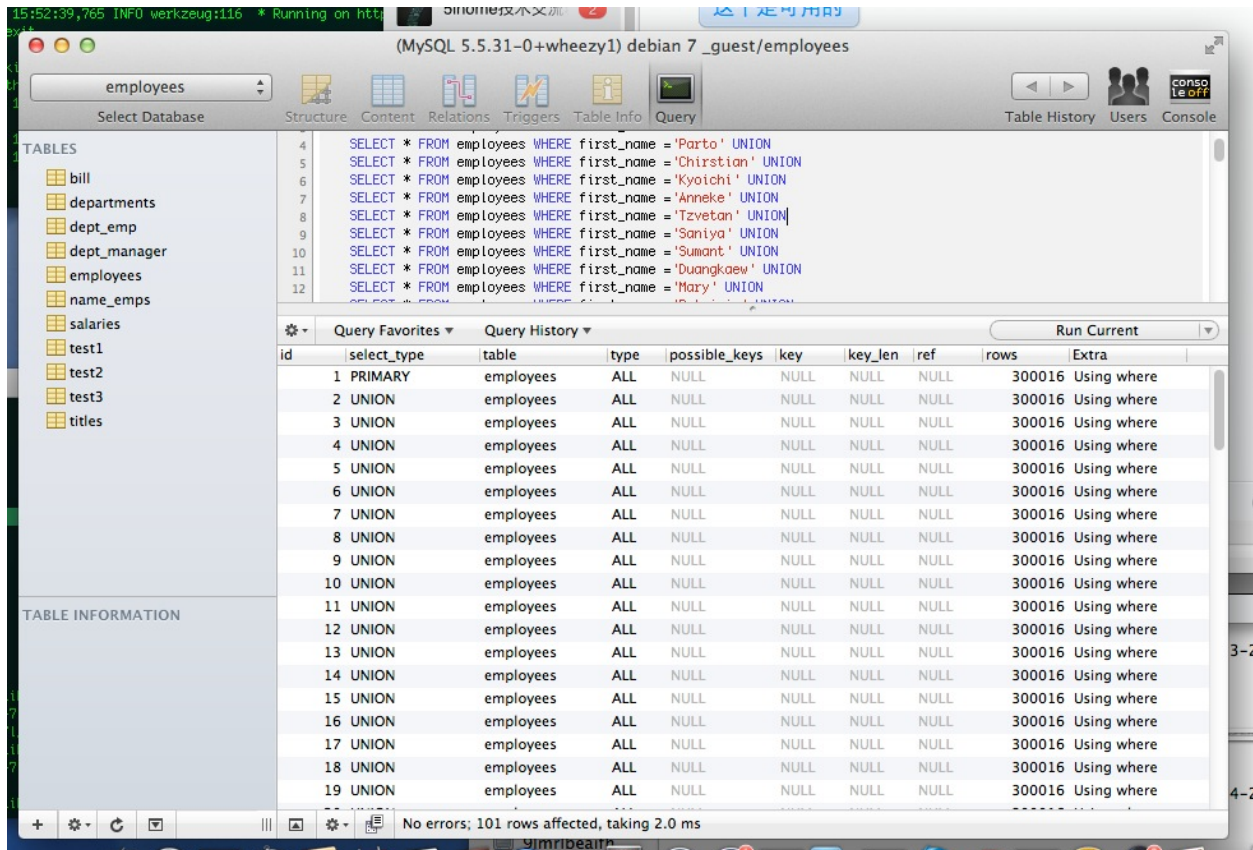
30W 数据:

- 无索引情况下：执行 100 条语句 union 耗时 0.795 秒，用 in 条件 0.001 秒
- 有索引情况下：执行 100 条语句 union 耗时 0.005 秒，用 in 条件 0.002 秒

90W 数据

- 有索引情况下：执行 100 条语句 union 耗时 0.028 秒，用 in 条件 0.000 秒

无索引时扫描表 1 0 0 次



```

SELECT
  emp_no,
  birth_date,
  first_name,
  last_name,
  gender hire_date
FROM
  employees
WHERE first_name IN (
  'Georgi',
  'Bezael',
  'Parto',
  'Chirstian',
  'Kyoichi',
  'Anneke',
  'Tzvetan',
  'Saniya',
  'Sumant',
  'Duangkaew',
  'Mary',
  'Patricio',
  'Eberhardt',
  'Berni',
  'Guoxiang',
  'Kazuhide',
  'Cristinel',
  'Kazuhide',
  'Lillian',
  'Mayuko',

```

3.1. 一般情况下，一次查询只会用到一个索引（特定情况出现 **merge index** 的情况，如下可能出现 $a=13$ or $b=2$ ）会合并 a 和 b 的索引，或者使用 **union all**）

```
'Ramzi',
'Shahaf',
'Bojan',
'Suzette',
'Prasadram',
'Yongqiao',
'Divier',
'Domenick',
'Otmar',
'Elvis',
'Karsten',
'Jeong',
'Arif',
'Bader',
'Alain',
'Adamantios',
'Pradeep',
'Huan',
'Alejandro',
'Weiyi',
'Uri',
'Magy',
'Yishay',
'Mingsen',
'Moss',
'Lucien',
'Zvonko',
'Florian',
'Basil',
'Yinghua',
'Hidefumi',
'Heping',
'Sanjiv',
'Mayumi',
'Georgy',
'Brendon',
'Ebbe',
'Berhard',
'Breannnda',
'Tse',
'Anoosh',
'Gino',
'Udi',
'Satosi',
'Kwee',
'Claudi',
'Charlene',
'Margareta',
'Reuven',
'Hisao',
'Hironoby',
'Shir',
'Mokhtar',
'Gao',
'Erez',
'Mona',
'Danel',
'Kshitij',
```

```

'Premal',
'Zhongwei',
'Parviz',
'Vishv',
'Tuval',
'Kenroku',
'Somnath',
'Xinglin',
'Jungsoon',
'Sudharsan',
'Kendra',
'Amabile',
'Valdiodio',
'Sailaja',
'Arumugam',
'Hilari',
'Jayson',
'Remzi',
'Sreekrishna',
'Valter',
'Hironobu',
'Perla'
)

```

The screenshot shows a SQL IDE interface. On the left, a sidebar lists tables: bill, departments, dept_emp, dept_manager, employees, name_emps, salaries, test1, test2, test3, and titles. The main window displays a query starting with 'EXPLAIN' followed by a 'SELECT' statement. The query selects columns emp_no, birth_date, first_name, last_name, gender, and hire_date from the 'employees' table, filtered by a 'WHERE first_name IN' clause listing 20 names. Below the query, a 'Query History' table is visible, showing the execution of the query on the 'employees' table, which has 300016 rows.

```

1 EXPLAIN
2 SELECT
3   emp_no,
4   birth_date,
5   first_name,
6   last_name,
7   gender hire_date
8 FROM
9   employees
10 WHERE first_name IN (
11   'Georgi',
12   'Bezael',
13   'Parto',
14   'Chirstian',
15   'Kyoichi',
16   'Anneke',
17   'Tzvetan',
18   'Saniya',
19   'Sumant',
20   'Duangkaew',
21   'Mary',
22   'Patricio',
23   'Eberhardt',
24   'Berni',
25   'Guoxiang',
26   'Kazuhiro',
27   'Cristinel',
28   'Kazuhide',
29   'Lillian',

```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employees	ALL	NULL	NULL	NULL	NULL	300016	Using where

3.2 每个表索引越少越好

每个表索引越少越好，建议 1-3 个，最多 5 个 (oltp 1-5, olap 5 以上)

表的索引影响 INSERT ,UPDATE 的速度，每次操作要更新索引。

索引越多影响越大。

3.3 每个查询必须用到索引

每个查询必须用到索引 (小表可能全表更好, 视数据量决定)

建立合适的索引, 根据数据量来配合 explain 分析

3.4 建立组合索引时, **WHERE** 条件中用到等于的字段放前边, 用到范围的字段放后边

建立组合索引时, WHERE 条件中用到等于的字段放前边, 用到范围的字段放后边, 如 DD=100000 AND SJ BETWEEN A AND B 例子 (见以上)

explain mysql 测试无左右条件左右说法

3.5 删除重复字段的索引, 减少 dml IO

删除重复字段的索引, 减少 dml IO

3.6 除了主键外, 避免建立其他唯一性索引

除了主键外, 避免建立其他唯一性索引

插入数据时增加额外开销

3.7 索引中重复的记录数越少, 效率越高, 效率最高的是主键

索引中重复的记录数越少, 效率越高, 效率最高的是主键

如果同一记录超过 50%, 全表扫描定期 analyze table 收集统计信息和直方图, 如果可以加 not null 或者 unique 的最好加上

3.8 索引字段最好不要存在 **NULL**

索引字段最好不要存在 NULL, NULL 可用 0 替代, 建议把默认值设置为 0

也可以 myisam_stats_method 和 innodb_stats_method 取值 nulls_equal, 在 null 远多于非 null 的情况下, 建议表设计 default 0

3.9 组合索引可以只使用第一个，或者前两个，或者前几个

组合索引可以只使用第一个，或者前两个，或者前几个，不能从第二个开始用，也不能跳着使用
索引使用从前缀开始，多字段索引到 between 或者 <, > 等以后字段不会使用, 排序最好在索引中实现

```
EXPLAIN SELECT
  `emp_no`,
  `salary`,
  `from_date`,
  `to_date`
FROM `employees`.`salaries`
WHERE from_date = '2000-07-15'
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	salaries	ALL	NULL	NULL	NULL	NULL	2844513	Using where

用到索引, 注意下面 2 幅截图并不一样：

```
EXPLAIN SELECT
  `emp_no`,
  `salary`,
  `from_date`,
  `to_date`
FROM `employees`.`salaries`
WHERE emp_no = 100001
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	salaries	ref	PRIMARY,emp_no	PRIMARY	4	const	4	

或者：

```
EXPLAIN SELECT
  `emp_no`,
  `salary`,
  `from_date`,
  `to_date`
FROM `employees`.`salaries`
WHERE emp_no = 100001 AND from_date = '2000-07-15'
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	salaries	const	PRIMARY,emp_no	PRIMARY	7	const,const	1	

查询条件

4.1 SQL 语句的 WHERE 条件避免无效条件和无效括号

SQL 语句的 WHERE 条件避免无效条件和无效括号

如 SELECT BM FROM GS WHERE (1=1) 、 order by

4.2 SQL 语句中不要加用不到的排序

排序使用主键索引，比不排序多了读取索引的步骤

```
EXPLAIN SELECT
  `emp_no`,
  `birth_date`,
  `first_name`,
  `last_name`,
  `gender`,
  `hire_date`,
  `var1`
FROM `employees`.`employees`
ORDER BY emp_no DESC
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employees	index	NULL	PRIMARY	4	NULL	300016	

不使用索引：

```
EXPLAIN SELECT
  `emp_no`,
  `birth_date`,
  `first_name`,
  `last_name`,
  `gender`,
  `hire_date`,
  `var1`
FROM `employees`.`employees`
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employees	ALL	NULL	NULL	NULL	NULL	300016	

4.3 WHERE 条件中最好不要用 IN 和 LIKE

WHERE 条件中最好不要用 IN 和 LIKE
in 的效率低于 =。
无索引情况下，in 效率要高于 union。

4.4 WHERE 条件中不要使用 NOW() 等进行判断

WHERE 条件中不要使用 NOW() 等进行判断，影响执行计划

4.5 索引要使用的字段不要使用函数或者进行运算

索引要使用的字段不要使用函数或者进行运算，如 field1 + 1 = field2、adddate(field1,...、CAST

```
EXPLAIN EXTENDED
SELECT
  (emp_no + 100) AS emp_no100,
  birth_date,
  CONCAT(first_name
    ,last_name) AS fullname,
  gender hire_date
FROM
  employees
```

运算方法，字符串连接，如无必要，在客户端中计算，连接。

4.6 禁止字段格式转换

禁止字段格式转换，如 SELECT * FROM GS WHERE BM=200000，数值两边不要加引号
大多数字段使用函数不会使用索引，只有形如 left(BM)='200000' 等可以使用)
查询条件和字段类型不一致，没有用到索引：

var1 列参考字符型，数值型字段类型不能混合使用，依赖后期转换

```
EXPLAIN EXTENDED
SELECT
  emp_no,
  birth_date,
  first_name,
  last_name,
  gender hire_date
FROM
  employees
WHERE var1=100001
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees	ALL	NULL	NULL	NULL	NULL	300016	100.00	Using where

查询条件和字段类型不一致，用到索引：


```
EXPLAIN EXTENDED
SELECT
    emp_no,
    birth_date,
    first_name,
    last_name,
    gender hire_date
FROM
    employees
WHERE var1='100001'
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees	ref	var1	var1	9	const	1	100.00	Using where

存储过程

5.1 存储过程中操作的记录数超过 1000 条时不能使用游标

存储过程中操作的记录数超过 1000 条时不能使用游标

禁止游标，用临时表代替

```
DELIMITER //
CREATE PROCEDURE `curdemo`()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE a INT;
    DECLARE b DATE;
    DECLARE cur1 CURSOR FOR SELECT emp_no,hire_date FROM employees WHERE first_name='Georgi';
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
    OPEN cur1;
    REPEAT
        FETCH cur1 INTO a, b;
    IF NOT done THEN
        IF hire_date < '1989-10-01' THEN
            INSERT INTO tmp1 (emp_no) VALUES (vemp_no);
        ELSE
            INSERT INTO tmp2 (emp_no) VALUES (vemp_no)
        END IF;
    END IF;
    UNTIL done END REPEAT;
    CLOSE cur1;
END //
DELIMITER ;
```

建立临时表：

```
DELIMITER //
CREATE PROCEDURE `curdemo`()
BEGIN
    INSERT INTO tmp_table (emp_no, hire_date) SELECT emp_no,hire_date FROM employees WHERE first_name='Georgi';

    INSERT INTO tmp1 (emp_no) SELECT emp_no FROM tmp_table WHERE hire_date < '1989-10-01' ;
    INSERT INTO tmp2 (emp_no) SELECT emp_no FROM tmp_table WHERE hire_date >='1989-10-01' ;
END //
DELIMITER ;
```

5.2 在存储过程的关键步骤开始和结束都要记录信息到日志表，用于监控和调试

在存储过程的关键步骤开始和结束都要记录信息到日志表，用于监控和调试

建立日志表：

```
CREATE TABLE `mylogs` (  
  `produce_name` CHAR(20) DEFAULT NULL,  
  `dt` DATETIME DEFAULT NULL,  
  `step` SMALLINT(6) DEFAULT NULL,  
  `msg` VARCHAR(100) DEFAULT NULL,  
  KEY `produce_name` (`produce_name`,`dt`)  
)  
  
DELIMITER //  
CREATE PROCEDURE `curdemo` ()  
BEGIN  
  INSERT INTO mylogs (`produce_name`, `dt`, `step`, `msg`)  
  VALUES  
  (  
    procedure_name,  
    NOW(),  
    0,  
    ' 程序开始'  
  ) ;  
  DECLARE done INT DEFAULT 0 ;  
  DECLARE procedure_name CHAR(20) ;  
  SET procedure_name = 'curdemo' ;  
  DECLARE a INT ;  
  DECLARE b DATE ;  
  DECLARE cur1 CURSOR FOR  
  SELECT  
    emp_no,  
    hire_date  
  FROM  
    employees  
  WHERE first_name = 'Georgi' ;  
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1 ;  
  INSERT INTO mylogs (`produce_name`, `dt`, `step`, `msg`)  
  VALUES  
  (  
    procedure_name,  
    NOW(),  
    1,  
    ' 开始打开游标'  
  ) ;  
  OPEN cur1 ;  
  INSERT INTO mylogs (`produce_name`, `dt`, `step`, `msg`)  
  VALUES  
  (  
    procedure_name,  
    NOW(),  
    2,  
    ' 结束打开游标'  
  ) ;  
  INSERT INTO mylogs (`produce_name`, `dt`, `step`, `msg`)
```

```

VALUES
(
    procedure_name,
    NOW(),
    3,
    ' 开始处理数据'
);
REPEAT
    FETCH cur1 INTO a,
    b ;
    IF NOT done
    THEN IF hire_date < '1989-10-01'
    THEN
    INSERT INTO tmp1 (emp_no)
    VALUES
        (vemp_no) ;
    ELSE
    INSERT INTO tmp2 (emp_no)
    VALUES
        (vemp_no) END IF ;
    END IF ;
    UNTIL done
END REPEAT ;
INSERT INTO mylogs (`produce_name`, `dt`, `step`, `msg`)
VALUES
(
    procedure_name,
    NOW(),
    4,
    ' 结束处理数据'
);
INSERT INTO mylogs (`produce_name`, `dt`, `step`, `msg`)
VALUES
(
    procedure_name,
    NOW(),
    5,
    ' 关闭游标'
);
CLOSE cur1 ;
INSERT INTO mylogs (`produce_name`, `dt`, `step`, `msg`)
VALUES
(
    procedure_name,
    NOW(),
    6,
    ' 程序结束'
);
END //

DELIMITER ;

```

`INSERT INTO mylogs` 可以写一个存储过程，以增加可读性。

```

DELIMITER //
CREATE PROCEDURE `writelogs` (
    IN produce_name CHAR(20),
    IN step SMALLINT,

```

```
    IN msg CHAR(100)
)
BEGIN
    INSERT INTO mylogs (`produce_name`, `dt`, `step`, `msg`)
    VALUES
        (produce_name, NOW(), step, msg) ;
END //
DELIMITER ;
```

程序改写后：

```
DELIMITER //
CREATE PROCEDURE `curdemo` ()
BEGIN
    DECLARE procedure_name CHAR(20) DEFAULT 'curdemo' ;
    DECLARE done INT DEFAULT 0 ;
    DECLARE a INT ;
    DECLARE b DATE ;
    DECLARE cur1 CURSOR FOR
    SELECT
        emp_no,
        hire_date
    FROM
        employees
    WHERE first_name = 'Georgi' ;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1 ;
    CALL writelogs (procedure_name, 0, ' 程序开始') ;
    CALL writelogs (
        procedure_name,
        1,
        ' 开始打开游标'
    ) ;
    OPEN cur1 ;
    CALL writelogs (
        procedure_name,
        2,
        ' 结束打开游标'
    ) ;
    CALL writelogs (
        procedure_name,
        3,
        ' 开始处理数据'
    ) ;
    REPEAT
        FETCH cur1 INTO a,
        b ;
        IF NOT done
        THEN IF hire_date < '1989-10-01'
        THEN
            INSERT INTO tmp1 (emp_no)
            VALUES
                (vemp_no) ;
        ELSE
            INSERT INTO tmp2 (emp_no)
            VALUES
                (vemp_no) ;
        END IF ;
    END IF ;
```

```

    UNTIL done
  END REPEAT ;
  CALL writelogs (
    procedure_name,
    4,
    ' 结束处理数据'
  ) ;
  CALL writelogs (procedure_name, 5, ' 关闭游标') ;
  CLOSE cur1 ;
  CALL writelogs (procedure_name, 6, ' 程序结束') ;
END //

DELIMITER ;

```

5.3 字符变量使用单引号，不要使用双引号

字符变量使用单引号，不要使用双引号，【"2012-09-23 00:00:00"】可改为【'2012-09-23 00:00:00'】

```

SELECT
  emp_no,
  first_name,
  last_name,
  hire_date
FROM
  `employees`
WHERE hire_date = "1986-06-26"

```

用单引号写成：

```

SELECT
  emp_no,
  first_name,
  last_name,
  hire_date
FROM
  `employees`
WHERE hire_date = '1986-06-26'

```

5.4 存储过程要能够重复执行，执行时需要清空历史冲突记录

存储过程要能够重复执行，执行时需要清空历史冲突记录, 比如程序重新执行计算，或者上次执行未到中途取消...

```

DELIMITER //
CREATE PROCEDURE `curdemo` ()
BEGIN
  DECLARE procedure_name CHAR(20) DEFAULT 'curdemo' ;
  DECLARE done INT DEFAULT 0 ;
  DECLARE a INT ;
  DECLARE b DATE ;
  DECLARE cur1 CURSOR FOR
  SELECT
    emp_no,
    hire_date

```

```
FROM
    employees
WHERE first_name = 'Georgi' ;
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1 ;
CALL writelogs (procedure_name, 0, ' 程序开始') ;
CALL writelogs (
    procedure_name,
    1,
    ' 开始清除原有记录'
) ;
DELETE
FROM
    tmp1
WHERE emp_no IN
    (SELECT
        emp_no
    FROM
        employees
    WHERE first_name = 'Georgi') ;
DELETE
FROM
    tmp2
WHERE emp_no IN
    (SELECT
        emp_no
    FROM
        employees
    WHERE first_name = 'Georgi') ;
CALL writelogs (
    procedure_name,
    2,
    ' 结束清除原有记录'
) ;
CALL writelogs (
    procedure_name,
    3,
    ' 开始处理数据'
) ;
OPEN cur1 ;
REPEAT
    FETCH cur1 INTO a,
    b ;
    IF NOT done
    THEN IF hire_date < '1989-10-01'
    THEN
        INSERT INTO tmp1 (emp_no)
        VALUES
            (vemp_no) ;
    ELSE
        INSERT INTO tmp2 (emp_no)
        VALUES
            (vemp_no) ;
    END IF ;
    END IF ;
UNTIL done
END REPEAT ;
CALL writelogs (
    procedure_name,
```



```
        4,  
        ' 结束处理数据'  
    );  
    CALL writelogs (procedure_name, 5, ' 关闭游标' );  
    CLOSE cur1 ;  
    CALL writelogs (procedure_name, 6, ' 程序结束' );  
END //  
  
DELIMITER ;
```


远程表

6.1 远程表结构要与原始表一致，尤其是索引

远程表结构要与原始表一致，尤其是索引

在另一台机器建立远程表：

```
CREATE TABLE `employees` (  
  `emp_no` int(11) NOT NULL,  
  `birth_date` date NOT NULL,  
  `first_name` varchar(14) NOT NULL,  
  `last_name` varchar(16) NOT NULL,  
  `gender` enum('M','F') NOT NULL,  
  `hire_date` date NOT NULL,  
  `var1` varchar(6) DEFAULT NULL,  
  PRIMARY KEY (`emp_no`),  
  KEY `var1` (`var1`)  
) ENGINE=FEDERATED DEFAULT CHARSET=latin1  
CONNECTION='mysql://user:password@host:port/employess/employees';
```

6.2 远程表数据不要大于 256M，远程表的 WHERE 无效

远程表数据不要大于 256M，远程表的 WHERE 无效

6.3 远程表一般用来全表小数据全量同步

远程表一般用来全表小数据全量同步

查询技巧

7.1 SQL 语句不要太长

SQL 语句不要太长，如果 IN 列表太多必须改为 LEFT JOIN，且关联字段主键索引

```
SELECT emp_no
FROM
    tmp2
WHERE emp_no IN
    (SELECT
        emp_no
    FROM
        employees
    WHERE first_name = 'Georgi') ;
```

可以写为：

```
SELECT
    t.emp_no
FROM
    tmp2 AS t
    INNER JOIN employees e
        ON t.emp_no = e.emp_no
WHERE e.first_name = 'Georgi') ;
```

7.2 避免使用 LIKE

避免使用 LIKE，【lrsj like “2012-09-23%”】可改为【LRSJ BETWEEN ‘2012-09-23 00:00:00’ AND ‘2012-09-23 23:59:59’】或者 left,right 函数

```
SELECT
    emp_no,
    first_name,
    last_name,
    hire_date
FROM
    employees
WHERE hire_date LIKE '1989%' ;
```

可以写为：

```

SELECT
    emp_no,
    first_name,
    last_name,
    hire_date
FROM
    employees
WHERE hire_date BETWEEN '1989-01-01' AND '1989-12-31' ;

```

left 与 *like* 结果有出入：

以下 2 条语句，*like* 速度明显快于 *left* 函数.

```

SELECT
    emp_no,
    first_name,
    last_name,
    hire_date
FROM
    employees
WHERE last_name LIKE 'D%' LIMIT 0, 1000000;

```

```

SELECT
    emp_no,
    first_name,
    last_name,
    hire_date
FROM
    employees
WHERE LEFT(last_name,1) = 'D' LIMIT 0, 1000000;

```

7.3 WHERE 多个 OR 条件不走一个索引时可通过 UNION

WHERE 多个 OR 条件不走一个索引时可通过 UNION，如【bm1=953016 or bm2=953016】改为【SELECT ...WHERE BM1=953016 UNION ALL SELECT ...WHERE BM2=953016】(merge index,explain 的结果是 using union(idx_name,idx_name))

以下 2 条语句在 90W 表中速度相当。

```

EXPLAIN SELECT
    emp_no,
    first_name,
    last_name,
    hire_date
FROM
    employees
WHERE last_name = 'Demeyer' OR first_name='Gao' LIMIT 0, 1000000;

```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employees	ALL	NULL	NULL	NULL	NULL	300016	Using where

```

EXPLAIN SELECT
    emp_no,
    first_name,
    last_name,
    hire_date

```

```

FROM
    employees
WHERE last_name = 'Demeyer'
UNION
SELECT
    emp_no,
    first_name,
    last_name,
    hire_date
FROM
    employees
WHERE first_name = 'Gao'
LIMIT 0, 1000000 ;

```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	employees	ALL	NULL	NULL	NULL	NULL	300016	Using where
2	UNION	employees	ALL	NULL	NULL	NULL	NULL	300016	Using where
NULL	UNION RESULT	<union1,2>	ALL	NULL	NULL	NULL	NULL	NULL	

性能优化

8.1 编译器、文件格式、磁盘、索引结构

编译器改为 ICC 可以提升 5%，文件格式改为 XFS 可以提升 5%，增加 1/6 磁盘可以提升 1/6，优化索引和结构一般可以提升 100-1000

8.2 使用 `type=heap` 的临时表

使用 `ENGINE = MEMORY` 建立内存表，（`type=heap` 我用的 mysql 5.6 不支持）关机后数据会全部丢失，表结果不变。

```
CREATE TABLE `test2` (  
  `id` INT(11) NOT NULL,  
  `count` INT(11) DEFAULT NULL,  
  `db_time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  PRIMARY KEY (`id`)  
) ENGINE = MEMORY DEFAULT CHARSET=latin1
```


引擎使用

9.1 innodb 引擎, 在过程结尾提交, 避免过度 commit

innodb 引擎, 在过程结尾提交, 避免过度 commit

```
<?php
$conn = mysql_connect('localhost','root','root') or die ("数据连接错误!!!");
mysql_select_db('test',$conn);
mysql_query("set names 'GBK'"); //使用 GBK 中文编码;
//开始一个事务
mysql_query("BEGIN"); //或者 mysql_query("START TRANSACTION");
$sql = "INSERT INTO `employees`.`employees` (
    `emp_no`,
    `birth_date`,
    `first_name`,
    `last_name`,
    `hire_date`,
)
VALUES
(
    99999,
    '1989-01-01',
    'first_name',
    'last_name',
    '1989-01-01'
) ;";
$sql2 = "delete from employees where emp_no= -1"; //这条我故意写错
$res = mysql_query($sql);
$res1 = mysql_query($sql2);
if($res && $res1){
    mysql_query("COMMIT");
    echo ' 提交成功。';
}else{
    mysql_query("ROLLBACK");
    echo ' 数据回滚。';
}
mysql_query("END");
```


权限控制

10.1 PHP 连接 MYSQL 的用户只分配 SIUD 权限

PHP 连接 MYSQL 的用户只分配 SIUD 权限, 即 : select, insert, update, delete

10.2 所有提交变量经过 `mysql_real_escape_string` 进行转义

所有提交变量经过 `mysql_real_escape_string` 进行转义, 防止注入

```
<?php
$con = mysql_connect("localhost", "hello", "321");
if (!$con)
{
    die('Could not connect: ' . mysql_error());
}
```

// 获得用户名和密码的代码

// 转义用户名和密码, 以便在 SQL 中使用

```
$user = mysql_real_escape_string($user);
$pwd = mysql_real_escape_string($pwd);
```

```
$sql = "SELECT * FROM users WHERE
user='" . $user . "' AND password='" . $pwd . "'"
```

// 更多代码

```
mysql_close($con);
```