任何语句使用前explain看执行计划是否用到索引

EXPLAIN

SELECT

id,

cxbm,

cxmc,

sfd,

mdd

FROM

cxxx

WHERE mdd = 210000

EXPLAIN

SELECT

id,

cxbm,

cxmc,

sfd,

mdd

FROM

cxxx

WHERE mdd = '210000'

明细统计时，只统计编码，不要关联名称等冗余字段

SELECT

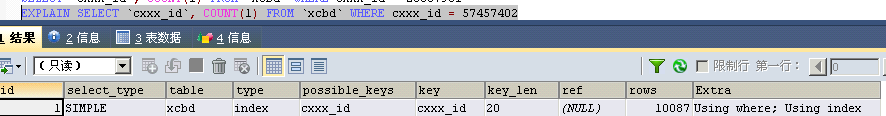
`cxxx\_id`,

COUNT(1)

FROM

`xcbd`

WHERE cxxx\_id = 23057901



使用汇总表：

建立汇总表：

CREATE TABLE `xcbd\_cxxx\_count` (

`cxxx\_id` INT 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 xcbd\_cxxx\_count;

定时插入数据进汇总表：

INSERT INTO `xcbd\_cxxx\_count` (`cxxx\_id`, `count\_num`)

SELECT

`cxxx\_id`,

COUNT(1)

FROM

`xcbd`

GROUP BY `cxxx\_id` ;

查询汇总表数据：

SELECT

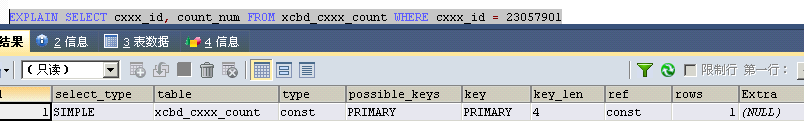
cxxx\_id,

count\_num

FROM

xcbd\_cxxx\_count

WHERE cxxx\_id = 23057901



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

xcbd

WHERE cxxx\_id = 3

正确做法：

SELECT

id,

clxx\_id,

cxxx\_id,

qsd,

zdd

FROM

xcbd

WHERE cxxx\_id = 3

明细统计时，只统计编码，不要关联名称等冗余字段

SELECT

bd.clxx\_id,

cl.cp,

COUNT(bd.clxx\_id)

FROM

`xcbd` AS bd

INNER JOIN clxx AS cl

ON bd.clxx\_id = cl.id

或者

SELECT

clxx\_id,

COUNT(1),

(SELECT

cp

FROM

clxx

WHERE id = xcbd.`clxx\_id`) AS cp

FROM

`xcbd`

正确做法

SELECT

clxx\_id,

COUNT(1)

FROM

`xcbd`

取出结果后，根据实际情况再取cp。cp, id可以考虑运用缓存存储

**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

clxx\_id,

cp,

COUNT(clxx\_id)

FROM

`xcbd` clxx

WHERE clxx\_id = id ;

正确做法：

SELECT

bd.clxx\_id,

cl.cp,

COUNT(bd.clxx\_id)

FROM

`xcbd` AS bd ;

INNER JOIN clxx AS cl

ON bd.clxx\_id = cl.id

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

id,

clxx\_id,

cxxx\_id,

qsd,

zdd

FROM

xcbd

WHERE del\_flag = 1

改写为：

SELECT

id,

clxx\_id,

cxxx\_id,

qsd,

zdd

FROM

xcbd

WHERE del\_flag = 1

AND lrsj BETWEEN (

'2012-01-01 00:00:00',

'2012-01-02 00:00:00'

)

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

语句中避免子查询，子查询无法使用索引

SELECT

clxx\_id,

COUNT(1),

(SELECT

cp

FROM

clxx

WHERE id = xcbd.`clxx\_id`) AS cp

FROM

`xcbd`

正确做法

SELECT

clxx\_id,

COUNT(1)

FROM

`xcbd`

取出结果后，根据实际情况再取cp。cp, id可以考虑运用缓存存储

语句中避免使用 GROUP BY, 可通过批量程序定期汇总

SELECT

bd.clxx\_id,

cl.cp,

COUNT(bd.clxx\_id)

FROM

`xcbd` AS bd

INNER JOIN clxx AS cl

ON bd.clxx\_id = cl.id

或者

SELECT

clxx\_id,

COUNT(1),

(SELECT

cp

FROM

clxx

WHERE id = xcbd.`clxx\_id`) AS cp

FROM

`xcbd`

正确做法

SELECT

clxx\_id,

COUNT(1)

FROM

`xcbd`

取出结果后，根据实际情况再取cp。cp, id可以考虑运用缓存存储

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)

尽量少用 TEXT、BLOB 等专有类型 (用链接代替)

字符型，数值型字段类型不能混合使用，依赖后期转换

相同字段不同表中的类型和长度要一致

字段名称不能使用关键字

不要指定字段级编码，建议全库统一

默认值要规范，例如日期不要使用 0000-00-00

不要用自增ID做主键

不要使用外键

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

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 emplyees\_firstname

ON employees (first\_name);

CREATE INDEX emplyees\_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 emplyees\_firstname,last\_name emplyees\_firstname,last\_name 16,18 420 100.00 Using union(emplyees\_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 select\_type table type possible\_keys key key\_len ref rows filtered Extra

1 PRIMARY employees ref emplyees\_firstname emplyees\_firstname 16 const 253 100.00 Using where

2 UNION employees 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

EXPLAIN SELECT \* FROM employees WHERE first\_name ='Georgi' UNION

SELECT \* FROM employees WHERE first\_name ='Bezalel' UNION

SELECT \* FROM employees WHERE first\_name ='Parto' UNION

SELECT \* FROM employees WHERE first\_name ='Chirstian' UNION

SELECT \* FROM employees WHERE first\_name ='Kyoichi' UNION

SELECT \* FROM employees WHERE first\_name ='Anneke' UNION

SELECT \* FROM employees WHERE first\_name ='Tzvetan' UNION

SELECT \* FROM employees WHERE first\_name ='Saniya' UNION

SELECT \* FROM employees WHERE first\_name ='Sumant' UNION

SELECT \* FROM employees WHERE first\_name ='Duangkaew' UNION

SELECT \* FROM employees WHERE first\_name ='Mary' UNION

SELECT \* FROM employees WHERE first\_name ='Patricio' UNION

SELECT \* FROM employees WHERE first\_name ='Eberhardt' UNION

SELECT \* FROM employees WHERE first\_name ='Berni' UNION

SELECT \* FROM employees WHERE first\_name ='Guoxiang' UNION

SELECT \* FROM employees WHERE first\_name ='Kazuhito' UNION

SELECT \* FROM employees WHERE first\_name ='Cristinel' UNION

SELECT \* FROM employees WHERE first\_name ='Kazuhide' UNION

SELECT \* FROM employees WHERE first\_name ='Lillian' UNION

SELECT \* FROM employees WHERE first\_name ='Mayuko' UNION

SELECT \* FROM employees WHERE first\_name ='Ramzi' UNION

SELECT \* FROM employees WHERE first\_name ='Shahaf' UNION

SELECT \* FROM employees WHERE first\_name ='Bojan' UNION

SELECT \* FROM employees WHERE first\_name ='Suzette' UNION

SELECT \* FROM employees WHERE first\_name ='Prasadram' UNION

SELECT \* FROM employees WHERE first\_name ='Yongqiao' UNION

SELECT \* FROM employees WHERE first\_name ='Divier' UNION

SELECT \* FROM employees WHERE first\_name ='Domenick' UNION

SELECT \* FROM employees WHERE first\_name ='Otmar' UNION

SELECT \* FROM employees WHERE first\_name ='Elvis' UNION

SELECT \* FROM employees WHERE first\_name ='Karsten' UNION

SELECT \* FROM employees WHERE first\_name ='Jeong' UNION

SELECT \* FROM employees WHERE first\_name ='Arif' UNION

SELECT \* FROM employees WHERE first\_name ='Bader' UNION

SELECT \* FROM employees WHERE first\_name ='Alain' UNION

SELECT \* FROM employees WHERE first\_name ='Adamantios' UNION

SELECT \* FROM employees WHERE first\_name ='Pradeep' UNION

SELECT \* FROM employees WHERE first\_name ='Huan' UNION

SELECT \* FROM employees WHERE first\_name ='Alejandro' UNION

SELECT \* FROM employees WHERE first\_name ='Weiyi' UNION

SELECT \* FROM employees WHERE first\_name ='Uri' UNION

SELECT \* FROM employees WHERE first\_name ='Magy' UNION

SELECT \* FROM employees WHERE first\_name ='Yishay' UNION

SELECT \* FROM employees WHERE first\_name ='Mingsen' UNION

SELECT \* FROM employees WHERE first\_name ='Moss' UNION

SELECT \* FROM employees WHERE first\_name ='Lucien' UNION

SELECT \* FROM employees WHERE first\_name ='Zvonko' UNION

SELECT \* FROM employees WHERE first\_name ='Florian' UNION

SELECT \* FROM employees WHERE first\_name ='Basil' UNION

SELECT \* FROM employees WHERE first\_name ='Yinghua' UNION

SELECT \* FROM employees WHERE first\_name ='Hidefumi' UNION

SELECT \* FROM employees WHERE first\_name ='Heping' UNION

SELECT \* FROM employees WHERE first\_name ='Sanjiv' UNION

SELECT \* FROM employees WHERE first\_name ='Mayumi' UNION

SELECT \* FROM employees WHERE first\_name ='Georgy' UNION

SELECT \* FROM employees WHERE first\_name ='Brendon' UNION

SELECT \* FROM employees WHERE first\_name ='Ebbe' UNION

SELECT \* FROM employees WHERE first\_name ='Berhard' UNION

SELECT \* FROM employees WHERE first\_name ='Breannda' UNION

SELECT \* FROM employees WHERE first\_name ='Tse' UNION

SELECT \* FROM employees WHERE first\_name ='Anoosh' UNION

SELECT \* FROM employees WHERE first\_name ='Gino' UNION

SELECT \* FROM employees WHERE first\_name ='Udi' UNION

SELECT \* FROM employees WHERE first\_name ='Satosi' UNION

SELECT \* FROM employees WHERE first\_name ='Kwee' UNION

SELECT \* FROM employees WHERE first\_name ='Claudi' UNION

SELECT \* FROM employees WHERE first\_name ='Charlene' UNION

SELECT \* FROM employees WHERE first\_name ='Margareta' UNION

SELECT \* FROM employees WHERE first\_name ='Reuven' UNION

SELECT \* FROM employees WHERE first\_name ='Hisao' UNION

SELECT \* FROM employees WHERE first\_name ='Hironoby' UNION

SELECT \* FROM employees WHERE first\_name ='Shir' UNION

SELECT \* FROM employees WHERE first\_name ='Mokhtar' UNION

SELECT \* FROM employees WHERE first\_name ='Gao' UNION

SELECT \* FROM employees WHERE first\_name ='Erez' UNION

SELECT \* FROM employees WHERE first\_name ='Mona' UNION

SELECT \* FROM employees WHERE first\_name ='Danel' UNION

SELECT \* FROM employees WHERE first\_name ='Kshitij' UNION

SELECT \* FROM employees WHERE first\_name ='Premal' UNION

SELECT \* FROM employees WHERE first\_name ='Zhongwei' UNION

SELECT \* FROM employees WHERE first\_name ='Parviz' UNION

SELECT \* FROM employees WHERE first\_name ='Vishv' UNION

SELECT \* FROM employees WHERE first\_name ='Tuval' UNION

SELECT \* FROM employees WHERE first\_name ='Kenroku' UNION

SELECT \* FROM employees WHERE first\_name ='Somnath' UNION

SELECT \* FROM employees WHERE first\_name ='Xinglin' UNION

SELECT \* FROM employees WHERE first\_name ='Jungsoon' UNION

SELECT \* FROM employees WHERE first\_name ='Sudharsan' UNION

SELECT \* FROM employees WHERE first\_name ='Kendra' UNION

SELECT \* FROM employees WHERE first\_name ='Amabile' UNION

SELECT \* FROM employees WHERE first\_name ='Valdiodio' UNION

SELECT \* FROM employees WHERE first\_name ='Sailaja' UNION

SELECT \* FROM employees WHERE first\_name ='Arumugam' UNION

SELECT \* FROM employees WHERE first\_name ='Hilari' UNION

SELECT \* FROM employees WHERE first\_name ='Jayson' UNION

SELECT \* FROM employees WHERE first\_name ='Remzi' UNION

SELECT \* FROM employees WHERE first\_name ='Sreekrishna' UNION

SELECT \* FROM employees WHERE first\_name ='Valter' UNION

SELECT \* FROM employees WHERE first\_name ='Hironobu' UNION

SELECT \* FROM employees WHERE first\_name ='Perla'

SELECT

emp\_no,

birth\_date,

first\_name,

last\_name,

gender hire\_date

FROM

employees

WHERE first\_name IN (

'Georgi',

'Bezalel',

'Parto',

'Chirstian',

'Kyoichi',

'Anneke',

'Tzvetan',

'Saniya',

'Sumant',

'Duangkaew',

'Mary',

'Patricio',

'Eberhardt',

'Berni',

'Guoxiang',

'Kazuhito',

'Cristinel',

'Kazuhide',

'Lillian',

'Mayuko',

'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',

'Breannda',

'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',

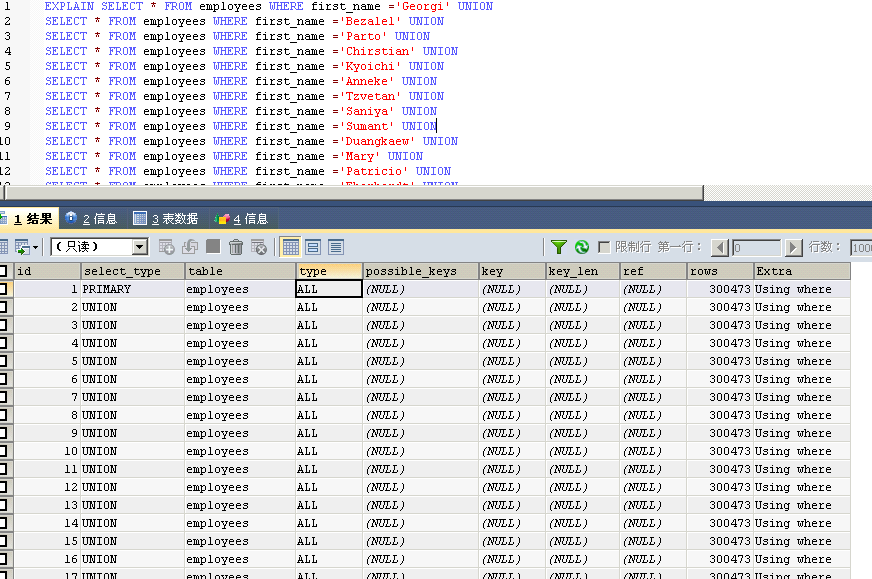
'Sreekrishna',

'Valter',

'Hironobu',

'Perla'

)

无索引表扫描１００次

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)

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测试无左右此说法

删除重复字段的索引，减少dml IO

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

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)

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)

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

id,

cxxx\_id,

gs\_bm,

gdddsj,

gdlksj

FROM

wdcx\_tjd

WHERE gs\_bm = 543001

AND xh = 1

A description...

正确做法：

EXPLAIN

SELECT

id,

cxxx\_id,

gs\_bm,

gdddsj,

gdlksj

FROM

wdcx\_tjd

WHERE cxxx\_id = 9547

AND xh = 1

或者：A description...

EXPLAIN

SELECT

id,

cxxx\_id,

gs\_bm,

gdddsj,

gdlksj

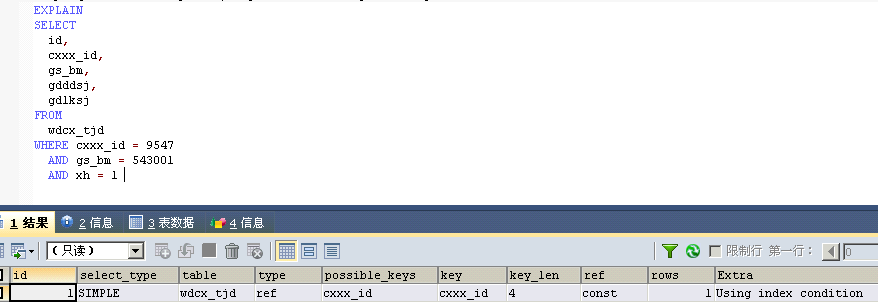
FROM

wdcx\_tjd

WHERE cxxx\_id = 9547

AND gs\_bm = 543001

AND xh = 1



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

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

id,

cxxx\_id,

gs\_bm,

gdddsj,

gdlksj

FROM

wdcx\_tjd

ORDER BY id DESC

A description...

不使用索引：

EXPLAIN

SELECT

id,

cxxx\_id,

gs\_bm,

gdddsj,

gdlksj

FROM

wdcx\_tjd

A description...

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

WHERE 条件中不要使用 NOW() 等进行判断，影响执行计划

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

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：

EXPLAIN EXTENDED

SELECT

emp\_no,

birth\_date,

first\_name,

last\_name,

gender hire\_date

FROM

employees

WHERE last\_name=100001

查询条件和字段类型不一致，用到索引：A description...

EXPLAIN EXTENDED

SELECT

emp\_no,

birth\_date,

first\_name,

last\_name,

gender hire\_date

FROM

employees

WHERE last\_name='100001'

A description...

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 ;

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 ;

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 ;

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'

存储过程要能够重复执行，执行时需要清空历史冲突记录

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 ;

远程表结构要与原始表一致，尤其是索引

远程表数据不要大于256M，远程表的 WHERE 无效

远程表一般用来全表小数据全量同步

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') ;

正确写法：

SELECT

t.emp\_no

FROM

tmp2 AS t

INNER JOIN employees e

ON t.emp\_no = e.emp\_no

WHERE e.first\_name = 'Georgi') ;

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%' ;

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;

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;

A description...

EXPLAIN SELECT

emp\_no,

first\_name,

last\_name,

hire\_date

FROM

employees

WHERE last\_name = 'Demeyer'

UNION

SELECT

emp\_no,

first\_name,

last\_name,

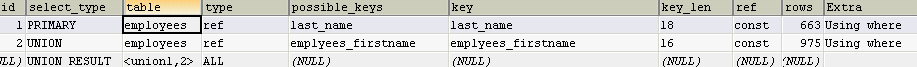
hire\_date

FROM

employees

WHERE first\_name = 'Gao'

LIMIT 0, 1000000 ;



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倍

使用type=heap的临时表

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");

PHP连接MYSQL的用户只分配SIUD权限

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);

?>