**# 查尋 id 最大值資料**

select \* , FROM\_UNIXTIME(DiagTimestamp/1000)

from lv\_data

where id = (select max(id) from lv\_data)

order by diagtimestamp desc,

lv\_id asc

limit 500;

# 備註 long 轉成時間 (如上) #

FROM\_UNIXTIME(DiagTimestamp/1000)

使用replace into

mysql> replace into user values (1, 'newvalue');

這種情況下邏輯是這樣的, mysql先判斷記錄是否存在, 若存在則先刪除之, 再自行insert. 所以你能看到這條語句執行後affected rows是2條(當然前提是你的資料表裡userid為1的資料只有1條)

**MySQL 給欄位元添加唯一索引**

添加唯一索引 sql：

|  |  |
| --- | --- |
|  | ALTER TABLE table\_name ADD UNIQUE (column\_list) |

注意：如果沒有唯一索引，則無法觸發 ON DUPLICATE KEY 這樣的事件，DUPLICATE 表示重複的意思，如某個欄位元沒有添加唯一索引，則不認為插入同樣的值是重複的，所以對不可重複的欄位元必須建立唯一索引。

// 修改EcuID 回傳ID 測試OK UPDATE

SET @update\_id := 0;

UPDATE vehicle\_info SET EcuID = 5555, id = (SELECT @update\_id := id)

WHERE DiagTimestamp= 3591308850 LIMIT 1;

SELECT @update\_id;

// INSERT or UPDATE ( LAST\_INSERT\_ID \*\*會呈查尋次數遞增\*\*)

INSERT INTO `read\_book` (book\_id, note, read\_time) VALUES

("14", "初次新增", "1472810252")

ON DUPLICATE KEY UPDATE

note = "後續更新", read\_id = LAST\_INSERT\_ID(read\_id),

read\_time = "1472810252",

read\_id = LAST\_INSERT\_ID(read\_id);

SELECT LAST\_INSERT\_ID() ;

不過，若只能插入不能update的原因可以檢查以下：

第一個欄位元必須是唯一索引或 unique 主鍵，

否則執行就就只會insert 而不會執行update。

INSERT INTO vehicle\_info(InterfaceSN, EcuID , DiagTimestamp, ModCode)

VALUES( '測試',777, 72, 'MODE')

ON DUPLICATE KEY UPDATE ID=LAST\_INSERT\_ID(ID),ModCode='MODE';

SELECT LAST\_INSERT\_ID() ;

INSERT INTO vehicle\_info(InterfaceSN, EcuID , DiagTimestamp, ModCode)

VALUES( '測試',777, 72, 'MODE')

ON DUPLICATE KEY UPDATE ModCode='MODE';

SELECT LAST\_INSERT\_ID(); // UPDATE無法使用 只有INSERT可用

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 資料表 login\_times | | | | | |
| **欄位** | **型態** | **屬性** | **Null** | **預設值** | **附加** |
| **user\_id** | varchar(12) |  | 否 |  |  |
| **count** | smallint(5) | UNSIGNED | 否 |  |  |

**用一般的 SQL 語法應該需要像下面這樣的三行︰**

IF(SELECT \* FROM login\_times WHERE user\_id = 'abc')

UPDATE login\_times SET count = count + 1 WHERE user\_id = 'abc';

ELSE

INSERT INTO login\_times (user\_id, count) VALUES ('abc', 1);

**用 INSERT ... ON DUPLICATE KEY UPDATE 的話就可以縮減成一行︰**

INSERT INTO login\_times (user\_id, count) VALUES ('abc', 1)

ON DUPLICATE KEY UPDATE count = count + 1;

注意：ON DUPLICATE KEY UPDATE隻是MySQL的特有語法，並不是SQL標準語法！

### 刪除重複的資料

這次又遇到一樣的問題了,key沒處理好,造成會有重複的資料  
  
搜尋了一下,有看到幾種做法  
1.直接刪掉在talbe裡重複的資料  
例:ALTER IGNORE TABLE `資料表名稱` ADD UNIQUE INDEX(`欄位1`,`欄位2`...);  
  
2.分二次,將 重複的資料 以及 未重複的資料 分別匯入另一個表,然後直接取代舊表  
  
 我採用了以下的方式(較安全,雖然慢了一點)   
1.先將"未重複"的資料寫到另一個表 INSERT INTO `table\_copy` SELECT \* FROM `table` GROUP BY `field name` HAVING count(\*)=1

2.再將"重複"的資料寫到另一個表 INSERT INTO `table\_copy` SELECT \* FROM `table` GROUP BY `field name` HAVING count(\*)>1

3.直接用新表取代舊的表

a. 刪除表：Drop table 表名;

b. 修改表名：alter table 舊表名rename 新表名;

# [MySQL] 在MySQL中找尋或刪除重複的記錄

在 MySQL 的資料庫中，若有些欄位我們在使用後才想要讓某個欄位的值不重複，讓如何刪除這個欄位中重複的值呢？例如我們在開發會員系統時，資料建置後才想到要讓電子郵件欄位不重複，就會遇到這個問題。

以下我們以一個 people 資料表來進行說明：  
如果我們想找尋資料表中多餘的重複記錄，重複記錄是根據單個欄位（peopleId）來判斷：

select \* from people where peopleId in (select peopleId from people group by peopleId having count(peopleId) > 1)

刪除表中多餘的重複記錄，重複記錄是根據單個欄位（peopleId）來判斷，只留有 rowid 最小的記錄

delete from people where peopleId in (select peopleId from people group by peopleId having count(peopleId) > 1) and rowid not in (select min(rowid) from people group by peopleId having count(peopleId )>1)

尋找資料表中多餘的重複記錄（多個欄位）

delete from people where peopleId in (select peopleId from people group by peopleId having count(peopleId) > 1) and rowid not in (select min(rowid) from people group by peopleId having count(peopleId )>1)

刪除資料表中多餘的重複記錄（多個欄位），只留有  rowid  最小的記錄

delete from vitae a where (a.peopleId,a.seq) in (select peopleId,seq from vitae group by peopleId,seq having count(\*) &gt; 1) and rowid not in (select min(rowid) from vitae group by peopleId,seq having count(\*)&gt;1)

尋找資料表中多餘的重複記錄（多個欄位），不包含rowid最小的記錄

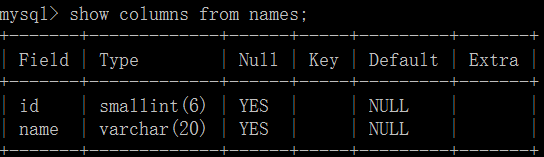
select \* from vitae a where (a.peopleId,a.seq) in (select peopleId,seq from vitae group by peopleId,seq having count(\*) &gt; 1) and rowid not in (select min(rowid) from vitae group by peopleId,seq having count(\*)&gt;1)

提供你參考。

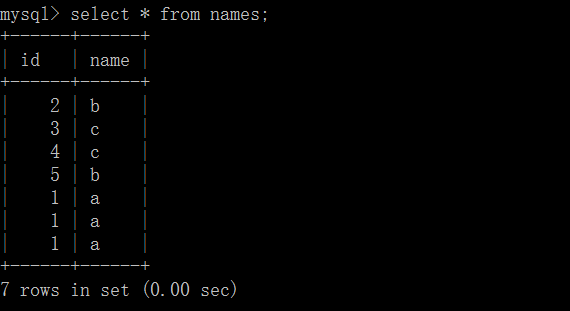
# mysql distinct 的用法

在查詢語句中加入distinct即可解決這個問題

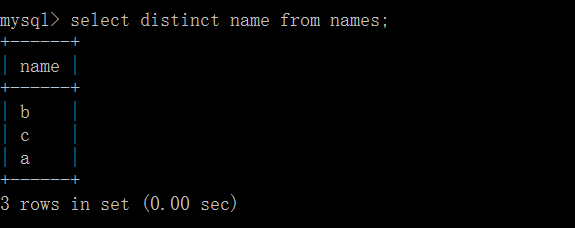
下面用實例進行說話：

創建這樣一個表：   


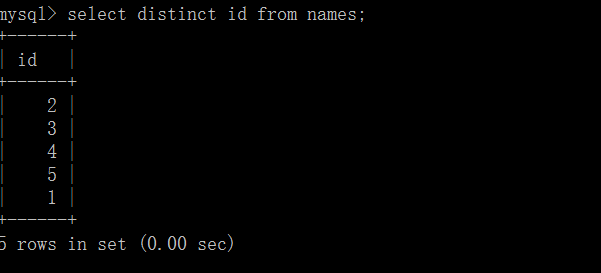
給表添加如下記錄：



distinct就是用於去除多餘記錄的：   
比如現在我要查詢表中有多少個不相同的姓名（name）



可以看到查詢結果顯示：有三條記錄

我們換一換，查一查這個表格中有多少個不同的id：   
  
我們可以看到結果與所有的記錄相比少了兩條id=1的記錄；

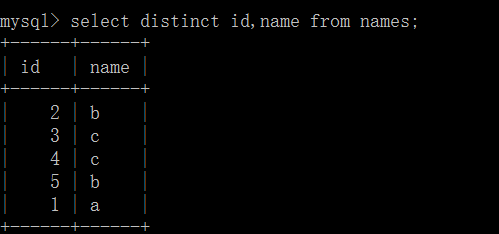
現在我們試一試用distinct來操縱多列（這裡是兩列）：

採用

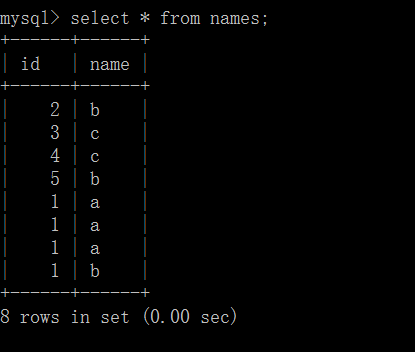
select distinct id,name from names;

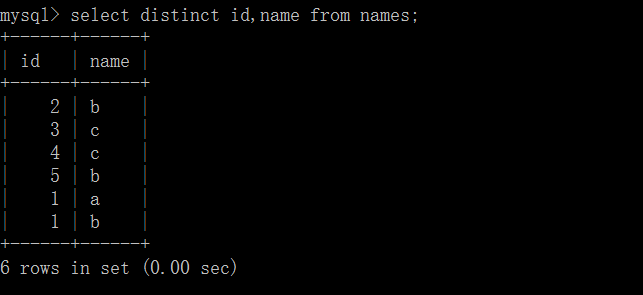
1

可以得到如下的結果：



我們認真的想一想，這個結果真實意義是什麼呢？   
對，這個查詢結果表示的是我現在要從names這張表中查找出所有的id、name都分別同時不相同的記錄（也就是說，我們的結果中的記錄就算兩條記錄的）；

為了說明情況我們在插入一條記錄id=1,name=b   


查詢結果顯示：

相信大家看到了這幾個例子，對distinct有一定的瞭解了！

# [MySQL冗余索引清理規則](http://blog.csdn.net/u010027484/article/details/53036105)

第一個索引是第二個索引的左首碼索引, 刪除第一個索引  
有主鍵, 用不到唯一索引, 刪除唯一索引  
有主鍵, 用不到普通索引, 刪除普通索引  
有唯一索引, 用不到普通索引, 刪除普通索引  
有聯合主鍵索引, 用不到普通組合索引, 刪除普通組合索引  
索引重複, 刪除其中一個索引  
普通索引預設以主鍵索引做尾碼, 重建普通索引

## MYSQL避免重複插入ON DUPLICATE KEY 只能插入無法 UPDATE的原因

如果要根據主鍵(例如：book\_id)到資料表中查詢是否已新增過、並是否存在有這樣的一筆記錄？  
若有則進行更新，若沒有則新插入一筆記錄，  
除了可以先用select count 先查一次再來處理，

|  |  |
| --- | --- |
| 1 | select \* from read\_book where read\_id =1;   //查詢資料表中是否有記錄 |
| 2 | insert into read\_book( book\_id, note)value(11 , 'xxxxx');   //沒有記錄則執行insert操作 | |

|  |  |
| --- | --- |
| 3 | update read\_book set book\_id=11, note = 'yyyyy'  where read\_id =1;   //有記錄則執行update操作 |

除了以上的方式，  
還可以用本文要介紹的方式：on duplicate key update  
  
資料表結構

|  |  |  |  |
| --- | --- | --- | --- |
| read\_id | book\_id | note | read\_time |
|  |  |  |  |

|  |  |
| --- | --- |
| 01 | CREATE TABLE `read\_book` ( |
| 02 | `read\_id` MEDIUMINT(8) UNSIGNED NOT NULL AUTO\_INCREMENT, | |

|  |  |  |
| --- | --- | --- |
| 03 | `book\_id` SMALLINT(5) UNSIGNED NOT NULL DEFAULT '0', | |
| 04 | `note` TEXT NOT NULL, |

|  |  |  |
| --- | --- | --- |
| 05 | `read\_time` INT(10) UNSIGNED NOT NULL DEFAULT '0', | |
| 06 | PRIMARY KEY (`read\_id`), |

|  |  |  |
| --- | --- | --- |
| 07 | UNIQUE INDEX `book\_id` (`book\_id`) | |
| 08 |  |

|  |  |  |
| --- | --- | --- |
| 09 | ) | |
| 10 |  |

|  |  |  |
| --- | --- | --- |
| 11 | COLLATE='utf8\_general\_ci' | |
| 12 | ENGINE=MyISAM |

|  |  |  |
| --- | --- | --- |
| 13 | AUTO\_INCREMENT=1 | |
| 14 | ; |

來看一段完整的用法：  
mysql 有就插入沒有就更新 on DUPLICATE key update批量更新

|  |  |  |
| --- | --- | --- |
| 1 | INSERT INTO `read\_book` (book\_id, note, read\_time) VALUES | |
| 2 | ("14", "初次新增", "1472810252") |

|  |  |
| --- | --- |
| 3 | ON DUPLICATE KEY UPDATE |
| 4 | note = "後續更新", read\_id = LAST\_INSERT\_ID(read\_id), | |

|  |  |
| --- | --- |
| 5 | read\_time = "1472810252", |
| 6 | read\_id = LAST\_INSERT\_ID(read\_id) | |

不過，若只能插入不能update的原因可以檢查以下：  
  
第一個字段必須是唯一索引或 unique 主鍵，  
否則執行就就只會insert 而不會執行update。

## [Mysql用grant創建及刪除使用者資訊](http://fecbob.pixnet.net/blog/post/43281511)

問題一：

用grant創建一個使用者zeros，並給它授，在mysql資料庫中的user表中和information\_schema資料庫中的USER\_PRIVILEGES表中，能夠查看到使用者zeros的資訊。

現在想刪除使用者zeros的資訊，怎麼辦？？？

解決問題一：

#1 使用root登錄mysql：

mysql無密碼的：[root@...]# mysql -u root

msyql有密碼的：[root@...]# mysql -u root -p

輸入密碼：

mysql>

#2 創建和授權使用者：

mysql> grant all on zeros.\* to zeros@'192.168.2.114' identified by

'zeros';

Query OK, 0 rows affected (0.02 sec)

#3 查看使用者zeros資訊：

#查看mysql資料庫中的user表：

mysql> use mysql;

mysql> select host, user, password from user;

+---------------+-------+------------------+

| host | user | password |

+---------------+-------+------------------+

| localhost | root | |

| 127.0.0.1 | root | |

| localhost | | |

| 192.168.2.100 | zeros | 676dfad16ab72df9 |

| 192.168.2.114 | zeros | 676dfad16ab72df9 |

+---------------+-------+------------------+

4 rows in set (0.00 sec)

#查看information\_schema資料庫中的USER\_PRIVILEGES表：

mysql> use information\_schema;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> select \* from USER\_PRIVILEGES;

+-------------------------+---------------+-----------------------+--------------+

| GRANTEE | TABLE\_CATALOG | PRIVILEGE\_TYPE |IS\_GRANTABLE |

+-------------------------+---------------+-----------------------+--------------+

................................................................................

| 'zeros'@'192.168.2.100' | Null | USAGE | NO |

| 'zeros'@'192.168.2.114' | Null | USAGE | NO | | ''@'localhost' | Null | USAGE | NO |

+-------------------------+---------------+-----------------------+--------------+

#4 刪除使用者zeros資訊：

 mysql> drop user zeros@'192.168.2.114';

 Query OK, 0 rows affected (0.00 sec)

#查看mysql資料庫中的user表：

 mysql> use mysql;

 mysql> select host, user, password from user;

+---------------+-------+------------------+

| host | user | password |

+---------------+-------+------------------+

| localhost | root | |

| 127.0.0.1 | root | |

| localhost | | |

| 192.168.2.100 | zeros | 676dfad16ab72df9 |

+---------------+-------+------------------+

4 rows in set (0.00 sec)

mysql> use information\_schema;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

#查看information\_schema資料庫中的USER\_PRIVILEGES表：

mysql> select \* from USER\_PRIVILEGES;

+-------------------------+---------------+-----------------------+--------------+

| GRANTEE | TABLE\_CATALOG | PRIVILEGE\_TYPE | IS\_GRANTABLE |

+-------------------------+---------------+-----------------------+--------------+

....................................................................................

| 'zeros'@'192.168.2.100' | Null | USAGE | NO |

| ''@'localhost' | Null | USAGE | NO | +-------------------------+---------------+-----------------------+--------------+

注意：

DROP的用法：

DROP USER語句用於刪除一個或多個MySQL帳戶。要想移除帳戶，應使用Drop USER user語 句。

DROP USER user [, user2] ...

使用與GRANT或REVOKE相同的格式為每個帳戶命名；例如，'zeros'@'localhost'。 帳戶名稱的使用者和主機部分與使用者表記錄的User和Host列值相對應。

注意：要使用DROP USER，您必須擁有mysql資料庫的全域Create USER許可權或Delete許可權。

使用DROP USER，您可以取消一個帳戶和其許可權，操作如下：

DROP USER user@ip\_address;

該語句可以刪除來自所有授權表的帳戶許可權記錄。

 drop user username@localhost

 drop user username@'%'

 MySQL 新增 User (Grant)

* GRANT ALL ON \*.\* TO 'username'@'localhost' IDENTIFIED BY 'password' WITH GRANT OPTION;
* GRANT SELECT,INSERT,UPDATE,DELETE,CREATE ON db\_name.\* TO 'username'@'localhost' IDENTIFIED BY 'password';
* GRANT ALL PRIVILEGES ON \*.\* TO 'username'@'127.0.0.%' IDENTIFIED BY 'password' WITH GRANT OPTION;
* GRANT REPLICATION SLAVE ON \*.\* TO repl@"192.168.1.%" IDENTIFIED BY 'password';
* GRANT SELECT,INSERT,UPDATE,DELETE,CREATE ON db\_name.\* TO 'username'@'%' IDENTIFIED BY 'password';
* GRANT SELECT, LOCK TABLES ON \*.\* TO  'backup'@'localhost'  
  IDENTIFIED BY 'backup\_password'; # mysqldump
* GRANT SELECT ON \*.\* TO  'backup'@'localhost'  
  IDENTIFIED BY 'backup\_password'; # mysqldump
* FLUSH PRIVILEGES;

自行把 上述的 username, password, db\_name 換掉即可.

說明

* GRANT ALL ON \*.\* TO 'username'@'localhost' IDENTIFIED BY 'passowrd'
* ALL: 授權的權限(SELECT, INSERT .... etc)
* \*.\*: Table Name, Db\_name.\*, 指定可以存取哪些Db/Table
* username: 要新增的 username
* localhost: 可從哪邊來存取, 可用 % (% 代表全部)
* password: 密碼

移除帳號

* DROP USER username@'123.123.123.%';
* DROP USER username@localhost;
* DROP USER username;

**MySQL 新增使用者及建立資料庫權限**

MySQL 安裝後只有 root 及 test 帳號, 由於未必可以用 phpMyAdmin 等 GUI 工具, 在指令模式 MySQL 新增使用者及設定權限, 在管理 MySQL 伺服器十分有用, 以下會介紹新增 MySQL 使用者, 及建立資料庫並設定權限。

首先在指令模式用 MySQL 的 root 帳號連接到 MySQL:

# mysql -u root -p

Enter password:

輪入 MySQL root 的密碼後便可以進入 MySQL 的指令模式。

然後為新帳號建立資料庫, 以下會以 “newdatabase” 為例子:

mysql> CREATE DATABASE 'newdatabase';

Query OK, 1 row affected (0.00 sec)

下一步是建立 MySQL 新帳號, 以下例子會建立“newuser”帳號, 密碼是 “newpassword”, 主機是 “localhost”:

mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'newpassword';

Query OK, 0 rows affected (0.00 sec)

然後給予新帳號 “newuser” 權限讀寫新資料庫 “newdatabase”:

mysql> GRANT ALL PRIVILEGES ON newdatabase.\* TO 'newuser'@'localhost';

Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;

Query OK, 0 rows affected (0.00 sec)

mysql> quit

現在已經完成設定, 可以測試一下新帳號“newuser”是否可以使用新資料庫 “newdatabase”:

# mysql -u newuser -p

Enter password:

輸入在上面設定的密碼, 以上例子是用“newpassword”, 如果沒有問題, 就可以用新使用者的身份連接 MySQL.

## 41 條 MYSQL / SQL 優化經驗總結

(1) 選擇最有效率的表名順序(只在基於規則的優化器中有效):   
ORACLE 的解析器按照從右到左的順序處理FROM子句中的表名,FROM子句中寫在最後的表(基礎表 driving table)將被最先處理,在FROM子句中包含多個表的情況下,你必須選擇記錄條數最少的表作為基礎表。如果有3個以上的表連接查詢, 那就需要選擇交叉表(intersection table)作為基礎表, 交叉表是指那個被其他表所引用的表.   
  
(2) WHERE子句中的連接順序.:   
ORACLE採用自下而上的順序解析WHERE子句,根據這個原理,表之間的連接必須寫在其他WHERE條件之前, 那些可以過濾掉最大數量記錄的條件必須寫在WHERE子句的末尾.   
  
(3) SELECT子句中避免使用 ‘ \* ‘:   
ORACLE在解析的過程中, 會將'\*' 依次轉換成所有的列名, 這個工作是通過查詢數據字典完成的, 這意味著將耗費更多的時間   
  
(4) 減少訪問數據庫的次數:   
ORACLE在內部執行了許多工作: 解析SQL語句, 估算索引的利用率, 綁定變量 , 讀數據塊等;   
  
(5) 在SQL\*Plus , SQL\*Forms和Pro\*C中重新設置ARRAYSIZE參數, 可以增加每次數據庫訪問的檢索數據量 ,建議值為200   
  
(6) 使用DECODE函數來減少處理時間:   
使用DECODE函數可以避免重複掃描相同記錄或重複連接相同的表.   
  
(7) 整合簡單,無關聯的數據庫訪問:   
如果你有幾個簡單的數據庫查詢語句,你可以把它們整合到一個查詢中(即使它們之間沒有關系)   
  
(8) 刪除重複記錄:   
最高效率的刪除重複記錄方法 ( 因為使用了ROWID)例子:

DELETE FROM EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID)   
FROM EMP X WHERE X.EMP\_NO = E.EMP\_NO);

(9) 用TRUNCATE替代DELETE:   
當刪除表中的記錄時,在通常情況下, 回滾段(rollback segments ) 用來存放可以被恢復的信息. 如果你沒有COMMIT事務,ORACLE會將數據恢復到刪除之前的狀態(準確地說是恢復到執行刪除命令之前的狀況) 而當運用TRUNCATE時, 回滾段不再存放任何可被恢復的信息.當命令運行後,數據不能被恢復.因此很少的資源被調用,執行時間也會很短. (譯者按: TRUNCATE只在刪除全表適用,TRUNCATE是DDL不是DML)   
  
(10) 盡量多使用COMMIT:   
只要有可能,在程序中盡量多使用COMMIT, 這樣程序的性能得到提高,需求也會因為COMMIT所釋放的資源而減少:   
COMMIT所釋放的資源:   
a. 回滾段上用於恢復數據的信息.   
b. 被程序語句獲得的鎖   
c. redo log buffer 中的空間   
d. ORACLE為管理上述3種資源中的內部花費   
  
(11) 用Where子句替換HAVING子句:   
避免使用HAVING子句, HAVING 只會在檢索出所有記錄之後才對結果集進行過濾. 這個處理需要排序,總計等操作.。  
  
如果能通過WHERE子句限制記錄的數目,那就能減少這方面的開銷. (非oracle中)on、where、having這三個都可以加條件的子句中,on是最先執行,where次之,having最後。  
因為on是先把不符合條件的記錄過濾後才進行統計,它就可以減少中間運算要處理的數據，正常來說應該速度是最快的。  
where也應該比having快點的，因為它過濾數據後才進行sum，在兩個表連接時才用on的，所以在一個表的時候,就剩下where跟having比較了。  
  
在這單表查詢統計的情況下，如果要過濾的條件沒有涉及到要計算字段，那它們的結果是一樣的，只是where可以使用rushmore技術而having就不能，在速度上後者要慢。如果要涉及到計算的字段，就表示在沒計算之前，這個字段的值是不確定的。根據上篇寫的工作流程，where的作用時間是在計算之前就完成的，而having就是在計算後才起作用的，所以在這種情況下兩者的結果會不同。  
  
在多表連接查詢時on比where更早起作用。系統首先根據各個表之間的連接條件，把多個表合成一個臨時表後，再由where進行過濾再計算，計算完後再由having進行過濾。由此可見,要想過濾條件起到正確的作用，首先要明白這個條件應該在什?時候起作用，然後再決定放在哪裡。  
  
(12) 減少對表的查詢:   
在含有子查詢的SQL語句中,要特別註意減少對表的查詢.例子:

SELECT TAB\_NAME FROM TABLES WHERE (TAB\_NAME,DB\_VER) = ( SELECT   
TAB\_NAME,DB\_VER FROM TAB\_COLUMNS WHERE VERSION = 604)

(13) 通過內部函數提高SQL效率.:   
複雜的SQL往往犧牲了執行效率. 能夠掌握上面的運用函數解決問題的方法在實際工作中是非常有意義的   
  
(14) 使用表的別名(Alias):   
當在SQL語句中連接多個表時, 請使用表的別名並把別名前綴於每個Column上.這樣一來,就可以減少解析的時間並減少那些由Column歧義引起的語法錯誤.   
如以下例子把 veryLongTableName 取成 vltn 識別的名稱。

SELECT \* FROM veryLongTableName as vltn  
WHERE vltn.title = 'title'

(15) 用EXISTS替代IN、用NOT EXISTS替代NOT IN:   
在許多基於基礎表的查詢中,為了滿足一個條件,往往需要對另一個表進行連接.在這種情況下, 使用EXISTS(或NOT EXISTS)通常將提高查詢的效率. 在子查詢中,NOT IN子句將執行一個內部的排序和合並. 無論在哪種情況下,NOT IN都是最低效率的 (因為它對子查詢中的表執行了一個全表遍歷). 為了避免使用NOT IN ,我們可以把它改寫成外連接(Outer Joins)或NOT EXISTS.   
例子:   
(高效率)

SELECT \* FROM EMP (基礎表) WHERE EMPNO > 0 AND EXISTS (SELECT 'X' FROM DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO AND LOC = 'MELB')

(低效率)

SELECT \* FROM EMP (基礎表) WHERE EMPNO > 0 AND DEPTNO IN(SELECT DEPTNO FROM DEPT WHERE LOC = 'MELB')

比較一個父結果集和子結果集，從而找到是否存在在父結果集中有而在子結果集中沒有的記錄，如：  
(低效率)

SELECT field FROM webgolds\_tb   
WHERE field NOT IN (SELECT field FROM google\_tb)

(高效率)

SELECT a.field FROM webgolds\_tb as a  
WHERE NOT EXISTS (SELECT \* FROM google\_tb b WHERE a.field = b.field)

(16) 識別'低效率執行'的SQL語句:   
雖然目前各種關於SQL優化的圖形化工具層出不窮,但是寫出自己的SQL工具來解決問題始終是一個最好的方法:

SELECT EXECUTIONS , DISK\_READS, BUFFER\_GETS,   
ROUND((BUFFER\_GETS-DISK\_READS)/BUFFER\_GETS,2) Hit\_radio,   
ROUND(DISK\_READS/EXECUTIONS,2) Reads\_per\_run,   
SQL\_TEXT   
FROM V$SQLAREA   
WHERE EXECUTIONS>0   
AND BUFFER\_GETS > 0   
AND (BUFFER\_GETS-DISK\_READS)/BUFFER\_GETS < 0.8   
ORDER BY 4 DESC;

(17) 用索引提高效率率:   
索引是表的一個概念部分,用來提高檢索數據的效率,ORACLE使用了一個複雜的自平衡B-tree結構. 通常,通過索引查詢數據比全表掃描要快. 當ORACLE找出執行查詢和Update語句的最佳路徑時, ORACLE優化器將使用索引. 同樣在聯結多個表時使用索引也可以提高效率率. 另一個使用索引的好處是,它提供了主鍵(primary key)的唯一性驗證.。  
  
那些LONG或LONG RAW數據類型, 你可以索引幾乎所有的列. 通常, 在大型表中使用索引特別有效. 當然,你也會發現, 在掃描小表時,使用索引同樣能提高效率率. 雖然使用索引能得到查詢效率的提高,但是我們也必須註意到它的代價. 索引需要空間來存儲,也需要定期維護, 每當有記錄在表中增減或索引列被修改時, 索引本身也會被修改. 這意味著每條記錄的INSERT , DELETE , UPDATE將為此多付出4 , 5 次的磁盤I/O . 因為索引需要額外的存儲空間和處理,那些不必要的索引反而會使查詢反應時間變慢.。定期的重構索引是有必要的.:

ALTER INDEX <INDEXNAME> REBUILD <TABLESPACENAME>

(18) 用EXISTS替換DISTINCT:   
當提交一個包含一對多表信息(比如部門表和雇員表)的查詢時,避免在SELECT子句中使用DISTINCT. 一般可以考慮用EXIST替換, EXISTS 使查詢更為迅速,因為RDBMS核心模塊將在子查詢的條件一旦滿足後,立刻返回結果. 例子:   
(低效率):

SELECT DISTINCT DEPT\_NO,DEPT\_NAME FROM DEPT D , EMP E   
WHERE D.DEPT\_NO = E.DEPT\_NO

(高效率):

SELECT DEPT\_NO,DEPT\_NAME FROM DEPT D WHERE EXISTS ( SELECT 'X'   
FROM EMP E WHERE E.DEPT\_NO = D.DEPT\_NO);

(19) sql語句用大寫的;因為oracle總是先解析sql語句,把小寫的字母轉換成大寫的再執行   
  
  
(20) 在java代碼中盡量少用連接符“+”連接字符串!   
  
  
(21) 避免在索引列上使用NOT 通常,    
我們要避免在索引列上使用NOT, NOT會產生在和在索引列上使用函數相同的影響. 當ORACLE”遇到”NOT,他就會停止使用索引轉而執行全表掃描.   
  
  
(22) 避免在索引列上使用計算.   
WHERE子句中,如果索引列是函數的一部分.優化器將不使用索引而使用全表掃描.   
舉例:   
低效率:

SELECT … FROM DEPT WHERE SAL \* 12 > 25000;

高效率:

SELECT … FROM DEPT WHERE SAL > 25000/12;

(23) 用>=替代>   
高效率:

SELECT \* FROM EMP WHERE DEPTNO >=4

低效率:

SELECT \* FROM EMP WHERE DEPTNO >3

兩者的區別在於, 前者DBMS將直接跳到第一個DEPT等於4的記錄而後者將首先定位到DEPTNO=3的記錄並且向前掃描到第一個DEPT大於3的記錄.   
  
  
(24) 用UNION替換OR (適用於索引列)   
通常情況下, 用UNION替換WHERE子句中的OR將會起到較好的效果. 對索引列使用OR將造成全表掃描. 註意, 以上規則只針對多個索引列有效. 如果有column沒有被索引, 查詢效率可能會因為你沒有選擇OR而降低. 在下面的例子中, LOC\_ID 和REGION上都建有索引.   
高效率:

SELECT LOC\_ID , LOC\_DESC , REGION   
FROM LOCATION   
WHERE LOC\_ID = 10   
UNION   
SELECT LOC\_ID , LOC\_DESC , REGION   
FROM LOCATION   
WHERE REGION = “MELBOURNE”

低效率:

SELECT LOC\_ID , LOC\_DESC , REGION   
FROM LOCATION   
WHERE LOC\_ID = 10 OR REGION = “MELBOURNE”

如果你堅持要用OR, 那就需要返回記錄最少的索引列寫在最前面.   
  
  
(25) 用IN來替換OR   
這是一條簡單易記的規則,但是實際的執行效果還須檢驗,在ORACLE8i下,兩者的執行路徑似乎是相同的.    
低效率:

SELECT…. FROM LOCATION WHERE LOC\_ID = 10 OR LOC\_ID = 20 OR LOC\_ID = 30

高效率

SELECT… FROM LOCATION WHERE LOC\_IN IN (10,20,30);

(26) 避免在索引列上使用IS NULL和IS NOT NULL   
避免在索引中使用任何可以為空的列,ORACLE將無法使用該索引.對於單列索引,如果列包含空值,索引中將不存在此記錄. 對於複合索引,如果每個列都為空,索引中同樣不存在此記錄. 如果至少有一個列不為空,則記錄存在於索引中.舉例: 如果唯一性索引建立在表的A列和B列上, 並且表中存在一條記錄的A,B值為(123,null) , ORACLE將不接受下一條具有相同A,B值(123,null)的記錄(插入). 然而如果所有的索引列都為空,ORACLE將認為整個鍵值為空而空不等於空. 因此你可以插入1000 條具有相同鍵值的記錄,當然它們都是空! 因為空值不存在於索引列中,所以WHERE子句中對索引列進行空值比較將使ORACLE停用該索引.   
低效率: (索引失效)

SELECT … FROM DEPARTMENT WHERE DEPT\_CODE IS NOT NULL;

高效率: (索引有效)

SELECT … FROM DEPARTMENT WHERE DEPT\_CODE >=0;

(27) 總是使用索引的第一個列:   
如果索引是建立在多個列上, 只有在它的第一個列(leading column)被where子句引用時,優化器才會選擇使用該索引. 這也是一條簡單而重要的規則,當僅引用索引的第二個列時,優化器使用了全表掃描而忽略了索引   
  
28) 用UNION-ALL 替換UNION ( 如果有可能的話):   
當SQL 語句需要UNION兩個查詢結果集合時,這兩個結果集合會以UNION-ALL的方式被合並, 然後在輸出最終結果前進行排序. 如果用UNION ALL替代UNION, 這樣排序就不是必要了. 效率就會因此得到提高. 需要註意的是,UNION ALL 將重複輸出兩個結果集合中相同記錄. 因此各位還是要從業務需求分析使用UNION ALL的可行性. UNION 將對結果集合排序,這個操作會使用到SORT\_AREA\_SIZE這塊內存. 對於這塊內存的優化也是相當重要的. 下面的SQL可以用來查詢排序的消耗量   
低效率:

SELECT ACCT\_NUM, BALANCE\_AMT   
FROM DEBIT\_TRANSACTIONS   
WHERE TRAN\_DATE = '31-DEC-95'   
UNION   
SELECT ACCT\_NUM, BALANCE\_AMT   
FROM DEBIT\_TRANSACTIONS   
WHERE TRAN\_DATE = '31-DEC-95'

高效率:

SELECT ACCT\_NUM, BALANCE\_AMT   
FROM DEBIT\_TRANSACTIONS   
WHERE TRAN\_DATE = '31-DEC-95'   
UNION ALL   
SELECT ACCT\_NUM, BALANCE\_AMT   
FROM DEBIT\_TRANSACTIONS   
WHERE TRAN\_DATE = '31-DEC-95'

(29) 用WHERE替代ORDER BY:   
ORDER BY 子句只在兩種嚴格的條件下使用索引.   
ORDER BY中所有的列必須包含在相同的索引中並保持在索引中的排列順序.   
ORDER BY中所有的列必須定義為非空.   
WHERE子句使用的索引和ORDER BY子句中所使用的索引不能並列.   
例如:   
表DEPT包含以下列:   
DEPT\_CODE PK NOT NULL   
DEPT\_DESC NOT NULL   
DEPT\_TYPE NULL  
低效率: (索引不被使用)

SELECT DEPT\_CODE FROM DEPT ORDER BY DEPT\_TYPE

高效率: (使用索引)

SELECT DEPT\_CODE FROM DEPT WHERE DEPT\_TYPE > 0

(30) 避免改變索引列的類型.:   
當比較不同數據類型的數據時, ORACLE自動對列進行簡單的類型轉換.   
假設 EMPNO是一個數值類型的索引列.

SELECT … FROM EMP WHERE EMPNO = '123'

實際上,經過ORACLE類型轉換, 語句轉化為:

SELECT … FROM EMP WHERE EMPNO = TO\_NUMBER('123')

幸運的是,類型轉換沒有發生在索引列上,索引的用途沒有被改變.   
現在,假設EMP\_TYPE是一個字符類型的索引列.

SELECT … FROM EMP WHERE EMP\_TYPE = 123

這個語句被ORACLE轉換為:

SELECT … FROM EMP WHERETO\_NUMBER(EMP\_TYPE)=123

因為內部發生的類型轉換, 這個索引將不會被用到! 為了避免ORACLE對你的SQL進行隱式的類型轉換, 最好把類型轉換用顯式表現出來. 註意當字符和數值比較時, ORACLE會優先轉換數值類型到字符類型   
  
(31) 需要當心的WHERE子句:   
某些SELECT 語句中的WHERE子句不使用索引. 這里有一些例子.   
在下面的例子里, (1)’!=' 將不使用索引. 記住, 索引只能告訴你什?存在於表中, 而不能告訴你什?不存在於表中. (2) ‘||'是字符連接函數. 就象其他函數那樣, 停用了索引. (3) ‘+'是數學函數. 就象其他數學函數那樣, 停用了索引. (4)相同的索引列不能互相比較,這將會啟用全表掃描.   
  
  
(32) a. 如果檢索數據量超過30%的表中記錄數.使用索引將沒有顯著的效率提高.   
b. 在特定情況下, 使用索引也許會比全表掃描慢, 但這是同一個數量級上的區別. 而通常情況下,使用索引比全表掃描要塊幾倍乃至幾千倍!   
  
  
(33) 避免使用耗費資源的操作:   
帶有DISTINCT,UNION,MINUS,INTERSECT,ORDER BY的SQL語句會啟動SQL引擎   
執行耗費資源的排序(SORT)功能. DISTINCT需要一次排序操作, 而其他的至少需要執行兩次排序. 通常, 帶有UNION, MINUS , INTERSECT的SQL語句都可以用其他方式重寫. 如果你的數據庫的SORT\_AREA\_SIZE調配得好, 使用UNION , MINUS, INTERSECT也是可以考慮的, 畢竟它們的可讀性很強   
  
  
(34) 優化GROUP BY:   
提高GROUP BY 語句的效率, 可以通過將不需要的記錄在GROUP BY 之前過濾掉.下面兩個查詢返回相同結果但第二個明顯就快了許多.   
低效率:

SELECT JOB , AVG(SAL)   
FROM EMP   
GROUP by JOB   
HAVING JOB = 'PRESIDENT'   
OR JOB = 'WEBGOLDS'

高效率:

SELECT JOB , AVG(SAL)   
FROM EMP   
WHERE JOB = 'PRESIDENT'   
OR JOB = 'WEBGOLDS'   
GROUP by JOB

(35) 減少在 where子句(clause)中使用 or 操作子(operator)來連接條件，否則將導致引擎放棄使用索引而進行全表掃描  
原始：

select id from t where num = 5 or num = 6

改成：

select id from t where num= 5  
union all  
select id from t where num= 6

(36)使用 in和not in也要慎用，因為IN會使系統無法使用索引，而直接搜索資料表中的數據。如：

select id from t where num in(1,2,3)

對於連續的數值，能用 between 就可以不用in了：

select id from t where num between 1 and 3

(雖然現實總是得利用到 in )  
  
(37)強制查詢使用索引-必要時強制查詢優化器使用某個索引，如在where子句中使用參數，也會導致全表掃描。因為SQL只有在運行時才會解析局部變量，但優化程序不能將訪問計劃的選擇推遲到運行時；它必須在編譯時進行選擇。然而，如果在編譯時建立訪問計劃，變量的值還是未知的，因而無法作為索引選擇的輸入項。如下面語句將進行全表掃描：

select id from t where num=@num       (沒有使用索引)

可以改為強制查詢使用索引：

select id from t with(index(索引名)) where num=@num

(38)不要在where子句中的“=”左邊進行函數、算術運算或其他表達式運算，否則係統將可能無法正確使用索引。  
避免在where子句中對字段進行表達式操作，這將導致引擎放棄使用索引而進行全表掃描。如：  
低效率：

SELECT \* FROM table WHERE num / 2 = 50

應改為：

SELECT \* FROM table WHERE num = 50 \* 2

低效率：

SELECT \* FROM RECORD WHERE SUBSTRING(CARD\_NO,1,4)='2345'

應改為:

SELECT \* FROM RECORD WHERE CARD\_NO LIKE '2345%'

低效率：

SELECT webgolds\_filed  
FROM google\_Table  
WHERE DATEDIFF( yy , query\_time ,GETDATE() ) > 21

應改為:

SELECT webgolds\_filed  
FROM google\_Table  
WHERE query\_time < DATEADD( yy , -21 ,GETDATE() )

即：任何對列的操作都將導致表掃描，包括數據庫函數、計算表達式等等，查詢時要盡可能將操作移至等號右邊。  
  
(39)在使用索引字段作為條件時，如果該索引是複合索引，那麼必須使用到該索引中的第一個字段作為條件時才能保證系統使用該索引，否則該索引將不會被使用，並且應盡可能的讓字段順序與索引順序相一致。  
例如複合索引 index(A , B ,C)，  
只用到 B 或 C 則系統不會使用索引，至少要 (A)、(A , B)、(A , B ,C) 這樣的組合。  
  
(40)查詢資料是否存在 - 如果你想校驗表裡是否存在某條紀錄，不要用count(\*) ，這樣效率不只低，而且浪費服務器資源。可以用EXISTS代替，例如：  
  
低效率：

IF (SELECT COUNT(\*) FROM table\_name WHERE column\_name = 'xxx')

可以寫成：

IF EXISTS (SELECT \* FROM table\_name WHERE column\_name = 'xxx')

(41)能用DISTINCT的就不用GROUP BY （GROUP BY 比 DISTINCT慢）  
  
低效率：

SELECT OrderID FROM table WHERE price > 10 GROUP BY OrderID

可改為：

SELECT DISTINCT OrderID FROM table WHERE price > 10

解決mylsq高併發處理之道