相忘于江湖

主要讲DB2的SQL优化，也许你在面试的时候用得着

关于DB2 SQL的优化

程序员之殇

目录

[1. 前言 2](#_Toc332210117)

[2. 为什么要进行 SQL 优化 3](#_Toc332210118)

[3. 怎样知道一个 SQL 的优劣 4](#_Toc332210119)

[3.1 图形化方案 4](#_Toc332210120)

[3.2 命令行方案 5](#_Toc332210121)

[3.2.1 Db2expln 5](#_Toc332210122)

[3.2.2 Db2batch 6](#_Toc332210123)

[4. 怎么样去优化我们的SQL语句 7](#_Toc332210124)

[4.1 改写 IN 7](#_Toc332210125)

[4.2 改写 LIKE 7](#_Toc332210126)

[4.3 改写 OR 或 <> 8](#_Toc332210127)

[4.4 合理使用Not in 和 Not Exists 9](#_Toc332210128)

[4.5 避免使用distinct 10](#_Toc332210129)

[4.6 不兼容的数据类型 10](#_Toc332210130)

[4.7 表连接 11](#_Toc332210131)

[4.8 利用子查询结果 14](#_Toc332210132)

[4.9 其他注意小点 15](#_Toc332210133)

[5. 如何建立合理的索引 18](#_Toc332210134)

[6. 避免死锁和锁等待 20](#_Toc332210135)

[7. 几个经典案例 23](#_Toc332210136)

# 前言

这篇文档综合了网上技术同仁的观点和作者的亲身实践，有不足和落伍之处还请读者明辨，这是因为计算机技术日新月异，今天看起来对的东西明天可能就不对了。

如果读者缺乏明辨的能力，请自己到网上google或baidu一下。若你要问我多年的IT工作总结出了什么经验，我只能用4个字来概括：网络搜索。

# 为什么要进行 SQL 优化

为什么要做 SQL 优化？这是不是一个无聊的工作？

那么我问你现在是面向什么编程？面对对象吗？也许吧，可网上有人说得好，现在做信息化的大部分人都在面向数据库编程，是的，我们大部分的ERP、OA、CRM中充满了SQL，没有SQL，没有数据库，就没有现在各种信息化应用。

不少人觉得查询优化是数据库管理系统的任务，与所编写的SQL语句关系不大，这种认识是不正确的，虽然现在的数据库产品在查询优化方面已经做得越来越好了，但提交的SQL语句是系统优化的基础，很难设想一个原本糟糕的查询语句经过系统的优化之后会变得高效。

据统计，SQL语句消耗了70%-90%的数据库资源，而其中读的SQL语句又占去70%-90%的资源。一个好的查询语句往往可以使程序性能提高数倍或数十倍，因此所写SQL语句的优劣至关重要！下面就与大家一起分享一下相关知识，希望对日常工作有所帮助。

# 怎样知道一个 SQL 的优劣

其实这个题目是一个伪命题。

SQL没有优劣之分，那是一个没有意义的事情。

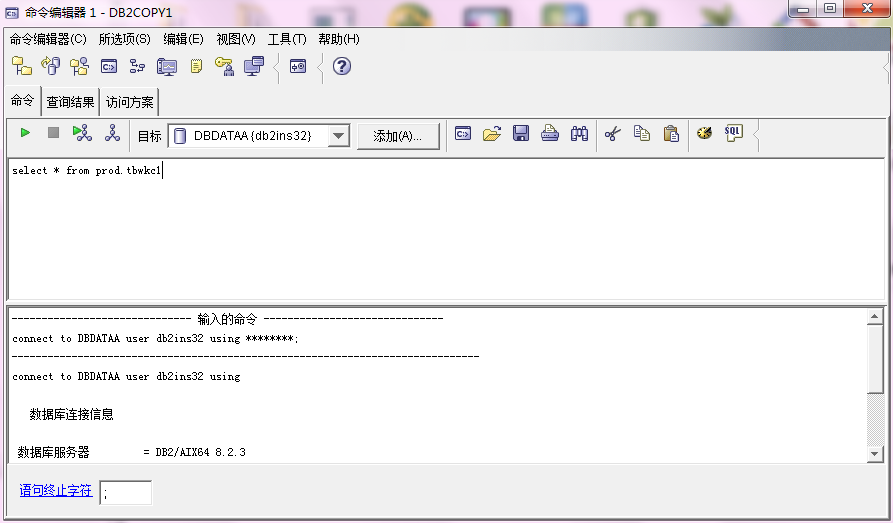
有的只是数据库性能的瓶颈。

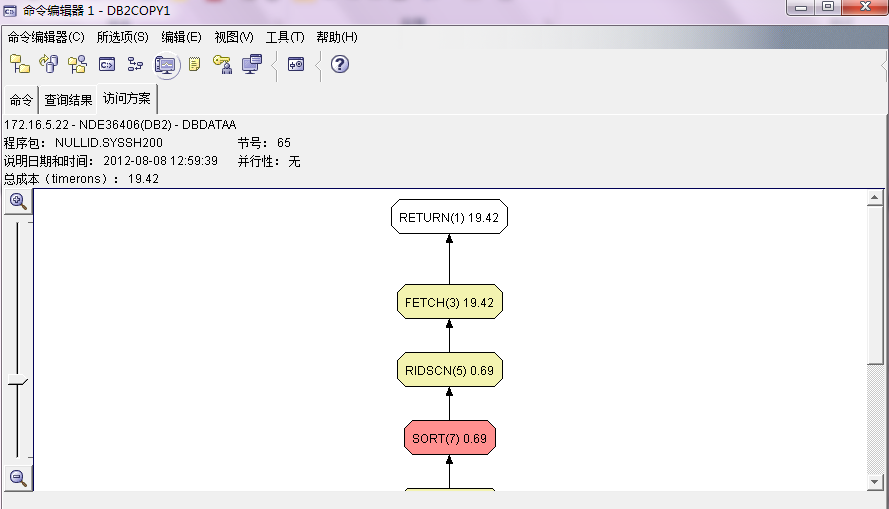
如果数据库的资源用不完，性能没有瓶颈，那么所有的SQL都不需要优化。而现实是几条含有内外连接的SQL就会把一个数据库搞挂掉。

下面我们想讨论的是——如何知道你的SQL消耗了数据库多少资源。

## 图形化方案

用DB2自带的图形工具即可知道SQL语句的查询效率，在“运行”中执行db2ce，点击下图的红圈部分显示的按钮：





可看到select \* from prod.tbwkc1 这条SQL语句的总成本为19.42.

## 命令行方案

### Db2expln

其实该命令和上面图形化方案一个意思，只是IBM将该命令集成到图形工具中而已。

a)单个SQL

db2expln -d <dbname> -q <sql语句> -t

比如：

db2expln -d sample -q "select \* from staff " -t

b)多个SQL语句

i. 将所有SQL语句放在一个文件中，以;分隔

ii. 运行命令：

db2expln -d <数据库名> -f <文件名> -t -z ;

iii. 查看db2expln的命令语法，直接打db2expln -h命令或参考db2的在线帮助

### Db2batch

刚才db2expln只能看出数据库的查询成本，但不能真实地反映出数据库消耗的时间，用db2batch命令可以查看一个或多个SQL语句的执行性能，在多SQL的程序中其实比刚才的命令更有效。

a) 将所有的SQL语句放在一个文件中，以;分隔

b) 执行命令：db2batch -d <数据库名> -f <sql文件名> -o r 0 f 0

c) 查看db2batch的帮助信息， db2batch -h或参照在线帮助

db2batch -d DBNAME -f fileName -a user/pass -r OutFile -o r 0 f 0

上面两个命令都要在db2cmd中执行哦！

# 怎么样去优化我们的SQL语句

## 改写 IN

在SQL语言中，一个查询块可以作为另一个查询块中谓词的一个操作数。因此，SQL查询可以层层嵌套。例如在一个大型分布式数据库系统中，有订单表Order、订单信息表OrderDetail，如果需要两表关联查询：

SELECT CreateUser   
FROM Order   
WHERE OrderNo IN   
 ( SELECT OrderNo   
 FROM OrderDetail   
 WHERE Price=0.5)   
可替代方案：

SELECT CreateUser FROM Order,OrderDetail WHERE Order.OrderNo=OrderDetail.OrderNo AND Praice=0.5

一个列的标签同时在主查询和where子句中的查询中出现，那么很可能当主查询中的列值改变之后，子查询必须重新查询一次。查询嵌套层次越多，效率越低，因此应当尽量避免子查询。如果子查询不可避免，那么要在子查询中过滤掉尽可能多的行。

## 改写 LIKE

在SQL语句中，LIKE关键字支持通配符匹配，但这种匹配特别耗费时间。例如：SELECT \* FROM Order WHERE CreateUser LIKE ‘M\_ \_ \_’ 。即使在CreateUser字段上建立了索引，在这种情况下也还是采用顺序扫描的方式，Order表中有1000条记录，就需要比较1000次。如果把语句改为SELECT \* FROM Order WHERE CreateUser >= ’M’ AND CreateUser <’N’，在执行查询时就会利用索引来查询，显然会大大提高速度。

## 改写 OR 或 <>

我们在编写sql时，通常都会按照程序逻辑去写，此时，当我们遇到如下场景：

我要查询企业员工表（employee）中的员工状态为实习(type=’01’)或者兼职的所有员工(type=’08’)，假设状态共有10种

此时，我们立马会写如下Sql：

Select \* from employee A where A.type=’01’ or A.type=’08’

我们假设，在type列上存在索引。而此Sql含有or运算，对于优化器来说，因为无法运用到一个范围，所以无法利用索引扫描。而通常此种情况需要遍历所有记录或者所有索引。这样会明显提高查询cost。我们希望是通过索引的方式，毕竟该表是个大表，如果出现大表扫描，多系统性能有很大的影响。那么可以采取用UNION改写OR子句，如下：

Select \* from employee A where A.type=’01’ union

Select \* from employee A where A.type=’02

改写成上述sql，优化器会分别执行两个查询子集，然后union合并。这样就可以利用到索引（type=‘01’）。当然Union包含去除重复元素的功能，即相当于distinct，这样就会有排序存在，如果业务场景允许，可以考虑使用union all，它和union不同的是，它无需排序去重，只需要两个子集合并即刻。效率要高于union。原则是：

当存在大表链接且连接条件较多，并且连接条件包含Or子句时，建议使用Union/Union all来替换。

对于不等与来说也是类似，不等于在逻辑上其实是类似于 Not 的概念。

如，对如下sql:

Sql\_stmt\_2:

Select \* from employee where type !=’01’

所以我们可以有如下改写方式：

1) 将<>改写为Not in操作，即

Select \* from employee where type not in (‘01’)

2) 将<>改写为大于和小于的结合

Select \* from employee where type >’01’ union

Select \* from employee where type <’01’(当然如果你知道一个大于已经足够，那么完全可以省略掉小于的操作，这就是分析sql的业务场景)

显然，对于1）的改法，它适用与Not in 子集中有多个值的情况；对于2）改法，要要由于1），因为它可以利用到Type列上的索引。

原则是：

当存在大表链接且连接条件较多，并且连接条件包含不等于（<>||!=）子句时，建议使用Union/Union all 联合大于小于操作来替换。

## 合理使用Not in 和 Not Exists

虽然Not in和Not exits可以实现相同的功能，但是两者本身的实现方式不同：

Not In：是自内向外操作，即先得到子查询结果，然后执行外层查询。Not in 子句的执行顺序是：首先取外部一个查询结果与内部子集比较，不管是否存在，它都要遍历整个子集，往往无法利用到索引，因而是由内向外过程。所以，当内部查询子集很大时，就会具有较高的查询代价。

Not Exists：恰恰相反，是外向内操作。即先执行外部查询结果，然后再执行内部操作，是集合操作。Not exists子句的执行顺序是：首先取外部一个查询结果与内部子集比较，若存在即刻返回，而不需要遍历整个子集，如果存在索引，就会使用索引，因而是个自外而内的过程。所以，当内部子集很大时，相对来说，性能要优于Not in。

因而，总的来说，Not exits在整体性能上要由于Not in。原则：

当子查询结果集较大时，Not exists 较 Not in 具有较高的性能提升；

当子查询结果集较小时（个数或者百数以内），两者相差不多，一般来说，此时Not in 会优于Not exists。就好像表数据小时，全表扫描总是要由于索引扫描；

当子查询具有一定的复杂度时（即sql关联关系较多，如子查询句中包含多个表查询），由于内部查询的复杂度，会导致Not exists查询具有较大的复杂度，降低性能。此时可以考虑采用Not in。

IN 与 Exists两者相差不多，这里不做比较，思路相同。

## 避免使用distinct

使用distinct是为了保证在结果集中不出现重复值，但是distinct会产生一张工作表，并进行排序来删除重复记录，这会大大增加查询和I/O的操作次数。因此应当避免使用distinct关键字。

## 不兼容的数据类型

例如float和int、char和varchar、binary和varbinary是不兼容的。数据类型的不兼容可能使优化器无法执行一些本来可以进行的优化操作。例如:

SELECT name FROM employee WHERE salary ＞ 60000

在这条语句中,如salary字段不是整型的,则优化器很难对其进行优化,因为60000是个整型数。我们应当在编程时将整型转化成为整型,而不要等到运行时转化。

## 表连接

表连接有两个要点：

1）表连接顺序

2）连接条件

Sql\_stmt\_1:

Select \* from A left join B on A.id=B.id

join C on B.id = C.C\_id

where A.con=’ ’ and B.con=’ ’

一般情况下，DB2会根据各表的JOIN顺序自顶向下处理，即从Sql来看，就是自左向右解析,先A、B做连接操作，之后会产生结果集，将会写入内存，如果内存不够，会写入临时表空间，之后会用结果集和C做连接操作。如果sql中只有两表连接，那么其前后顺序没什么关系，优化器会自己去评估。而如果sql中存在超过2个表连接时，那么表连接就会有顺序之分。那么，原则是：

如果sql中存在表A、B、C三表连接，则首先应保证最先连接的两表具有较小的子集。

在进行表连接时，需要提供连接字段（即On语法后的等价谓词，on A.id=B.id）。此时，我们需要保证，连接字段存在索引。这样当结果集小时，会走NestJoin（速度快，因为会利用到索引），当结果集大时，会走Hash join。此外，在对A、B表进行连接时，优化器需要判断采用何种连接类型，这时会先执行where 字句后的条件。也就是说，如果where字句能过滤很多的条件，那么表连接的结果集就会很小，cost自然会降低，所以适当为where字句的查询字段建立索引，能够得到更好的性能。原则：

在进行表连接时，为连接字段和查询过滤字段（where 字句后的条件）建立索引，会得到很好的性能提升。

在本次测试中，发现有的sql会在表连接时，为其指定多个连接条件，形如：

SELECT B.APPROVE\_STATUS, count ( \* ) AS NUM

FROM BIZ.WF\_TASK C

LEFT JOIN BIZ.REI\_FORM B

ON C.RECEIPT\_NO = B.REI\_FORM\_ID

WHERE C.TASK\_STATUS = '01'

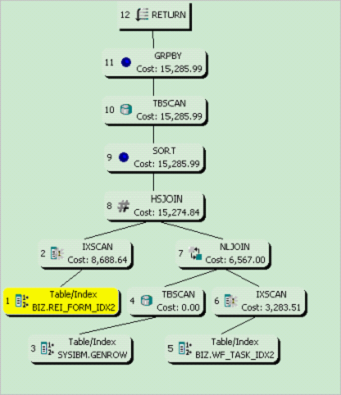
AND C.HANDLE\_ID = '1234560000102'

AND (C.RECEIPT\_TYPE = '02' OR C.RECEIPT\_TYPE = '03')

GROUP BY B.APPROVE\_STATUS

WITH UR

执行cost：



如果sql写成（增加一个表连接字段）：

SELECT B.APPROVE\_STATUS, count ( \* ) AS NUM

FROM BIZ.WF\_TASK C

LEFT JOIN BIZ.REI\_FORM B

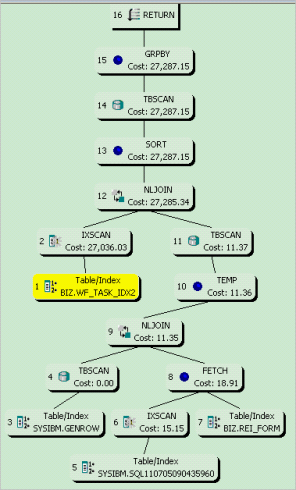
ON C.RECEIPT\_NO = B.REI\_FORM\_ID

AND (C.RECEIPT\_TYPE = '02' OR C.RECEIPT\_TYPE = '03')

WHERE C.TASK\_STATUS = '01' AND C.HANDLE\_ID = '1234560000102'

GROUP BY B.APPROVE\_STATUS

WITH UR



对比结果，我们可以看到，当连接条件存在多个时，cost会高很多，因为多做了一次表连接。如果是小表，看不出差别，如果是大表关联，则结果很明显。原则：

当进行表连接时，请确保连接条件只有一个，尤其是大表连接。

## 利用子查询结果

将查询结果作为子查询，主要是为了减少扫描的数据量，以及利用索引进行数据检索。尤其是针对大表来说。它的特点就是，在进行查询之前，先用子查询将结果集过滤到最小，并且通常这时候的过滤谓词是存在索引的。

假设如下情况：

索引：

Inst1.idx\_history\_date on inst1.history(tstmp)

Inst1.idx\_history\_acct on inst1.history(acct\_id)

查询：

Select a.name from inst1.acct a ,inst1.history h where a.acct\_id = h.acct\_id and (h.tstmp > current timestatmp – 2 days or a.balance>100);

上面的查询用于选择余额大于100元或者最近两天有过交易的账户名称。由于不存在组合索引(acct\_id,tstmp)，它们是单独字段建立索引，所以对于上述的查询无法利用索引，将会很不幸的走全表扫描。那么改写成如下方式，可以奏效：

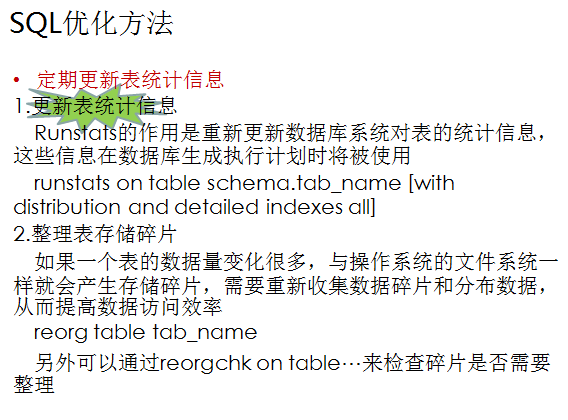
With tmp as (select acct\_id from inst1.history h where h.tstmp > current timestatmp -2 days) select name from inst1.acct a where a.acct\_id in (select acct\_id from tmp ) or a.balance >100;

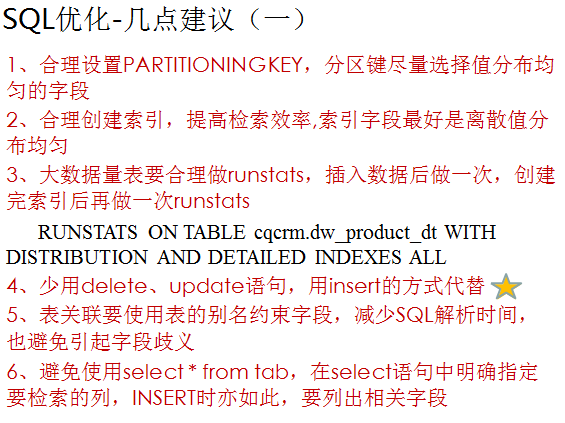
这样改写以后，子查询结果tmp就会走索引inst1.idx\_history\_date，并且会过滤掉表history一定的结果。然后再与acct表连接，走索引Inst1.idx\_history\_acct。原则：

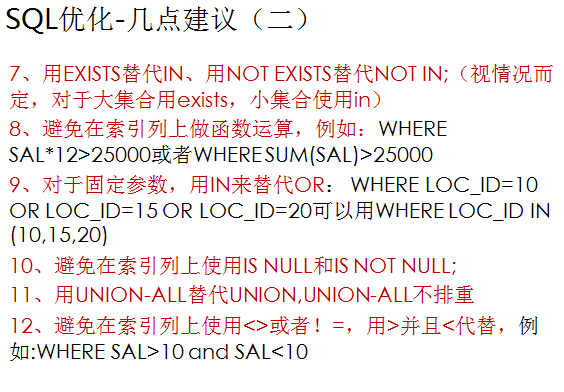
当查询谓词条件存在两个以上，并且该谓词的选择性很强，作为单独一列，未组合索引，此时可以考虑采用构建子查询，来利用索引缩减扫描的数据量。

## 其他注意小点

1. 避免select \* from……的使用，取所需的列即可。当表很小时，看不出来fetch所有列带来的开销，但是当表很大时，除了索引列外，\*号代表取出所有字段将会有很高的fetch
2. 避免在索引列上做运算，如substr分割字符串，它将会失去索引的判断性，很有可能无法利用到索引
3. 当查询返回结果集较多时，而我们却不需要这么多，那么可以采用fetch first N rows only，N为大于0的整数，这样即使你做了全表扫描，但是我只会取前100行数据，会大大降低执行时间。
4. 对于查询出来的结果集是只读的，使用select …for read only(或者fetch only)，意味着后续不会对该些数据行做update或delete操作，这可以帮助DB2提高Fetch性能。因为它允许DB2执行块操作，一个fetch操作可以请求返回多行数据
5. 根据业务逻辑选择合适粒度的隔离级别。UR对于系统有最大的并发性，但也有更多的数据问题，脏读、幻想读都会发生。如果系统可以接受这样的查询结果，那么UR是最好的选择。DB2默认是CS隔离级别，在大并发下，该隔离级别是有可能导致大量锁等待和死锁。所以在编写sql时，考虑业务场景，针对查询，为sql赋予一定的隔离级别。
6. 避免不必要的排序。排序是数据库中资源消耗比较大的一种操作。在业务允许下，通过添加索引（索引本身就是有序的，确保索引排序和业务排序相符）；在distinct、group by、order by 子句涉及的列上创建索引（注意索引的排序是升序asc还是降序desc）









# 如何建立合理的索引

在项目开发过程中，我们也可以根据需要定义索引，比如当表结构和Sql稳定后，我们便可以根据该sql执行的频率来决定是否需要为该sql建立索引。Sql中类似where 子句后就单个谓词，我们比较容易建立索引，而如果是多表关联并且谓词关系较多时，我们可以先采用Db2 提供的索引优化工具Db2Advis来帮助我们建立索引，至少它可以综合各个表的存量和各个列的占比为我们提供建议。语法如下：

db2advis -d dbname -i test1.sql -n schema\_name -t 5 > wf\_task.adv

注：将上述加粗的参数用自己的数据库参数代替；

dbname 是数据库名称

Test1.sql 是存放你待提供建议sql的文件名称

schema\_name 是你建立索引所在的视图，这里和表保持一致就好

或者用如下Sql：

db2advis -d dbname -s "SELECT \* FROM T1 ORDER BY EMPNO" -m IMCP

注：这里只需要将dbname换成自己的数据库名称，sql statement换成自己的

为了对比索引建立前后带来的不同，除了cost之外，我们还可以去查看它的执行计划，观察走索引到底比没有索引快了多少。我们仍然使用DB2提供的工具 dynexpln或者db2expln语法如下：

db2expln –d dbname –s –g –q “sql statement” –t 或者

dynexpln –d dbname –s –g –q “sql statement” -t

此外在建立索引时，需要注意以下几点：

1） 根据条件中谓词的选择度创建索引

可以简单的通过select count(\*) from tabname where col=’X’这样的方式，观察每个谓词条件过滤的总数。过滤出结果集越小，代表选择度越高，如果是建立组合索引，那么应该将该谓词放在首位

2） 避免在建有索引的列上使用函数

3） 在需要被排序的列上建立索引（注意索引的顺序与排序顺序一致），对大表很有效

4） 使用include关键词创建索引

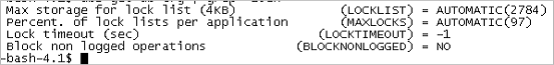
存在这种情况，当表足够大时（通常是百万级），我们需要通过一个谓词col（是唯一的）来获取列中的colA，这时候，如果将colA加入索引很浪费，因为它并没有很大选择性，而如果不加入，当表很大时，fetch cost太高。这是可以用include来将列colA包含进索引，这样不再有多余并且耗时的fetch，include列也不会影响索引的选择性。建立include类索引，必须要求索引字段是唯一的，否则无法include。

# 避免死锁和锁等待

数据库中之所以会存在死锁或者锁等待，是因为某一事务执行时间过长，导致锁没有及时释放，那么我们的解决办法就是，事务过程尽量要短，并且事务中的sql执行要快，这样才不会有过多的锁等待。还有一个原因，就是一些执行糟糕的sql，比如走了全表扫描，那么它会占据表中大量的锁，导致锁住了其他行，其他用户只能等待。

解决锁等待，要注意以下几点：

1. 优化查询 Sql，采用db2advis建立合适的索引，使得其能够走索引查询，由于索引的范围和排序，可以直接跳过许其他行，定位到符合我们需要的行。
2. 采用合适的隔离级别。由于DB2 默认是CS的隔离级别，它的原理是，游标每到一行就会锁住改行，对于一般应用来说是足够了，但是如果遇到全表扫描，那么CS模式会锁住表中大量的行，直到查询完毕。所以可以根据业务需求，将其改为UR模式，它不会对表加任何行锁。或者在JDBC中设置隔离级别(Isolation Levels)
3. 合理设置锁超时参数，它主要是用来避免事务长时间被占用，导致锁和连接无法释放，影响系统的并发。可以设置 DB参数
4. 更新操作一定要走索引，否则很容易产生死锁。（针对边更新边查的操作）
5. 避免出现锁升级现象，当锁等待达到一定程度时（行锁的个数超过loctList \*percent of lock list），就会出现行锁升级为表锁，即锁升级。因为一旦出现锁升级，那么锁住的就不再是行，而是表，那么其他事务要想访问该表中的任意行，必须等待事务将锁释放。



修改Lock timeout （-1代表不检测锁超时），一般来说，该参数默认为10s足矣。

当系统存在严重的锁等待时，可以通过以下sql，定位到锁等待Sql

db2 "select AGENT\_ID ,substr(STMT\_TEXT,1,100) as statement,STMT\_ELAPSED\_TIME\_MS

from table(SNAPSHOT\_STATEMENT('dbname',-1)) as B where AGENT\_ID in (select AGENT\_ID\_HOLDING\_LK

from table(SNAPSHOT\_LOCKWAIT(‘dbname’,-1)) as A order by LOCK\_WAIT\_START\_TIME ASC FETCH FIRST 20 ROWS ONLY ) order by STMT\_ELAPSED\_TIME\_MS DESC"

运行结果如下：

http://hi.csdn.net/attachment/201109/21/0_1316616107G52u.gif

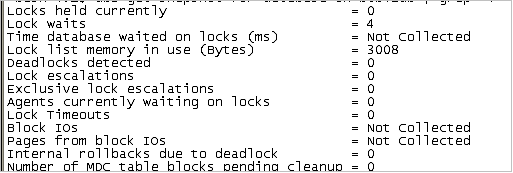
死锁比锁超时更加可怕，因为它将随机回滚一个事务，而这个不受应用程序控制，不可控的错误十分可怕，所以一旦出现死锁，必须解决掉。如何观察DB2是否存在死锁呢，有以下两种方式：

1）开启lock快照监控

db2 update monitor switches using LOCK on

执行如下命令：

db2 get snapshot for database on dbname | grep -i "LOCK" ,结果如下：

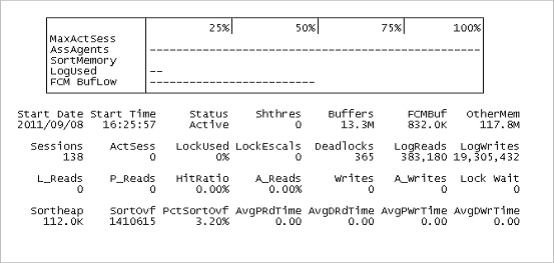


可以看到其中有DeadLocks detected

2）采用db2top 工具（db2 V9.1后才有）

Db2top –d dbname

然后键盘输入‘d’，如下：



可以看到其中有个 DeadLocks 计数。

如何定位死锁也有很多方式，如：

1）创建死锁监控器（需要针对文件分析，复杂度较高）

2）采用db2pd（最稳定，一般可以定位到）

3）采用db2top 监控组件（最快，但不一定能抓到）

# 几个经典案例

平心而论，这里面有的优化方法在SQLserver中不存在，只是DB2中有用。

