无锡移动战报邮件日报优化

|  |  |
| --- | --- |
| 客户名称 | 无锡移动 |
| 系统名称 |  |
| 数据库名 |  |
| 优化时间时间 | 2015年9月21日-2015年12月21日 |
| 工程师 | 顾屺明，王松柏，窦志浩，孙其成，代海鹏 |
| 用户确认 |  |

**上海新炬网络技术有限公司**

**2015年12月**

目录

[客户名称 i](#_Toc438125762)

[无锡移动 i](#_Toc438125763)

[目录 2](#_Toc438125764)

[1 优化工具使用 3](#_Toc438125765)

[1.1 Profiler 3](#_Toc438125766)

[1.2 DBMS\_XPLAN 3](#_Toc438125767)

[1.3 10046 sql trace 3](#_Toc438125768)

[1.4 存过依赖分析脚本 4](#_Toc438125769)

[1.5 查看TOP SQL脚本 4](#_Toc438125770)

[2. 典型优化场景解析 5](#_Toc438125771)

[2.1分区表取数 5](#_Toc438125772)

[2.1.1 场景A 建立合理的分区表 5](#_Toc438125773)

[2.1.2 场景B 使用分区键查询 7](#_Toc438125774)

[2.1.3 场景C 使用truncate独立分区 7](#_Toc438125775)

[2.1.4 场景D 合理运用11g自动分区 8](#_Toc438125776)

[2.2 远端表和本地表进行关联 10](#_Toc438125777)

[2.2.1 场景E 建立中间过程 10](#_Toc438125778)

[2.2.2 场景F dblink在CBO中的限制 11](#_Toc438125779)

[2.3 merge条件尽量写在内层 13](#_Toc438125780)

[2.3.1 场景I merge条件尽量写在内层 13](#_Toc438125781)

[2.4 CTAS代替merge 14](#_Toc438125782)

[2.4.1 场景H使用CTAS代替merge大表更新 14](#_Toc438125783)

[2.4.2 场景J避免多次merge同一张表 17](#_Toc438125784)

[2.5 merge/CTAS代替update 19](#_Toc438125785)

[2.5.1 场景K merge代替update（子查询） 19](#_Toc438125786)

[2.5.2 场景L CTAS代替update大表 20](#_Toc438125787)

[2.6 dblink 远程抽数 21](#_Toc438125788)

[2.6.1 场景M 合理使用parallel 21](#_Toc438125789)

[2.6.2 场景N 建立中间视图 23](#_Toc438125790)

[2.6.3 场景O 合理在dblink中运用函数 26](#_Toc438125791)

[2.6.4 场景P避免多次重复抽取 28](#_Toc438125792)

[2.7 nls改hash join 30](#_Toc438125793)

[2.7.1 场景P两表关联CBO估算错误 30](#_Toc438125794)

[2.8 TABLE函数CBO基数的限制 31](#_Toc438125795)

[2.8.1 场景R TABLE函数CBO基数限制 31](#_Toc438125796)

[2.9 or 与semi join触发错误的执行计划 32](#_Toc438125797)

[2.9.1 场景S or与exists触发错误执行计划 32](#_Toc438125798)

[3 总结和建议 34](#_Toc438125799)

## 1 优化工具使用

1. Profiler
2. dbms\_xplan
3. 10046 SQL TRACE
4. 存过依赖关系分析脚本
5. 查看TOP SQL脚本

1.1 Profiler

Profiler是ORACLE PL/SQL 的一个调试优化跟踪方案的应用

Profiler是一个能直观方便的调试工具，因为不需要生成和读取服务器端的跟踪文件，它是将跟踪数据全部存储的数据库表里，所以也得到了一些第三方工具的支持，如PL/SQL DEVELOPER 。

注：Profiler生成的跟踪信息远远没有trace生成的详细，它没有执行计划、没有CPU及IO 信息，它只是生成主要的执行时间信息，所以分析起来比较快，比较直观，可以让我们以最快的时间定位要优化的SQL。

还有就是可能在调用第一次时候，时间会有偏差。如果出现偏差，再执行一次就可以了。



1.2 DBMS\_XPLAN

DBMS\_XPLAN包中display\_cursor函数不同于display函数，display\_cursor用于显示SQL语句的真实的执行计划，在大多数情况下，显示真实的执行计划有助于更好的分析SQL语句的全过程，尤其是运行此SQL语句实时的I/O开销。通过对比预估的I/O与真实的I/O开销来判断SQL语句所存在问题，如缺少统计信息，SQL语句执行的次数，根据实际中间结果集的大小来选择合适的连接方式等。本文仅仅讲述display\_cursor函数的使用。



1.3 10046 sql trace

当SQL语句出现性能问题时，我们可以用SQL\_TRACE来跟踪SQL的执行情况，通过跟踪，我们可以了解一条SQL或者PL/SQL包的运行情况，SQL\_TRACE命令会将SQL执行的整个过程输出到一个trace文件中，我们可以读这个trace 文件来了解在这个SQL执行过程中Oracle 都做了哪些操作。



1.4 存过依赖分析脚本

通过分析各节点执行开始时间，及各依赖节点执行间隔时间，提取耗时较长的关键路径，进行重点优化，通过优化调度算法，节省父节点空闲等待时间，定位导致整体延迟的省公司关键依赖节点，提出调整建议。从而从SQL优化，整体调度优化，基础依赖执行时间优化等多个方面提升报表执行效率，达到效果最优！



1.5 查看TOP SQL脚本

由于share pool会将不太使用的sql执行计划刷出内存，如果我们想查看历史的sql运行状况，通过查看dba\_hist\_sqlstat来获得历史的会话中的sqlCPU,IO,BUFFER等信息，帮助我们快速找到之前可能出问题的sql



## 2. 典型优化场景解析

2.1分区表取数

A 建立合理的分区表

B. 查询分区表的时候，没有使用到分区键

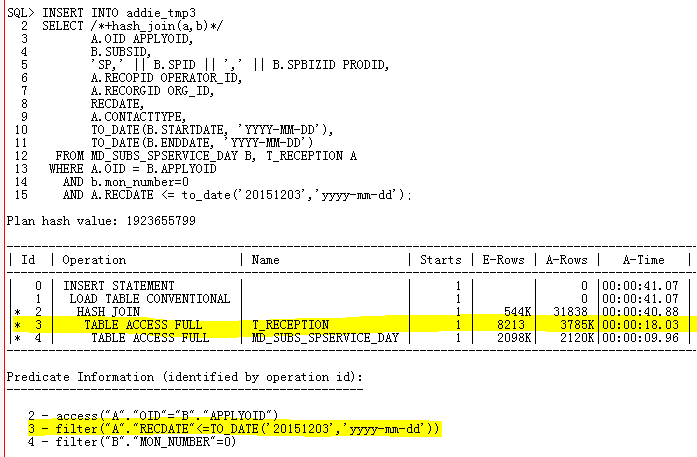
C. 删除表的某一个月的数据，使用delete方式删除

D. 没有利用到11g分区表新特性，可以按月自动进行分区

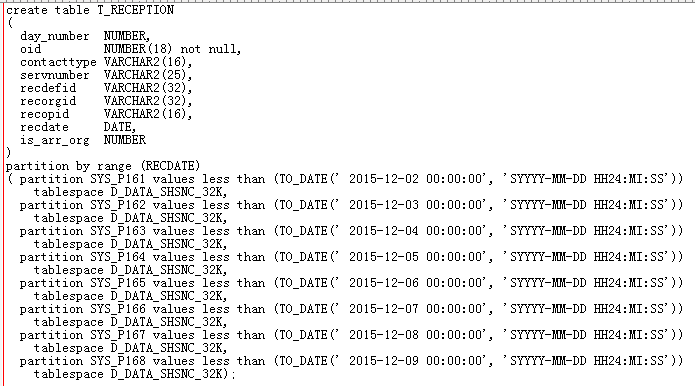
2.1.1 场景A 建立合理的分区表

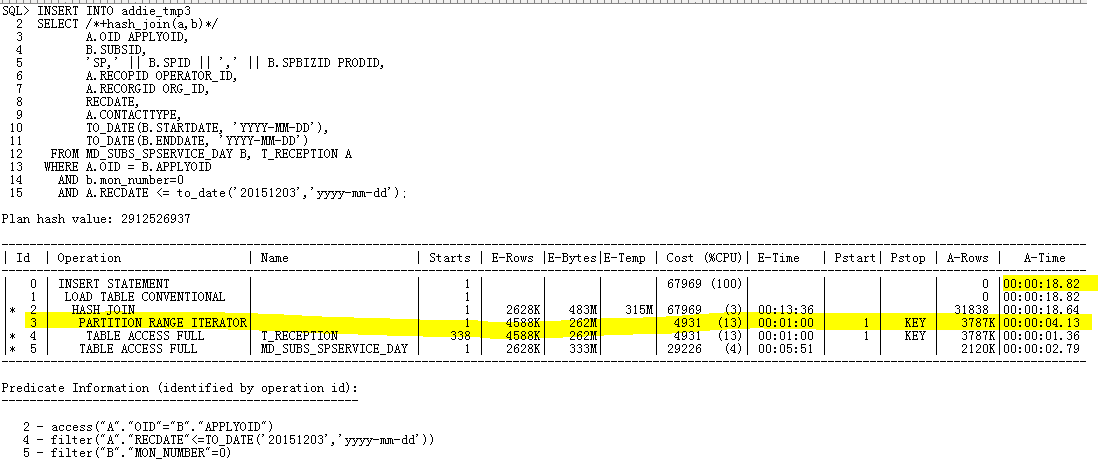
随着表中的数据量不断的增加，查询速度越来越来慢。这时候我们就需要考虑使用分区表来改善查询性能。分区能够将表、索引进一步细分为段，这些数据库对象的段叫做分区。每个分区有自己的名称，还可以选择自己的存储特性。从应用程序的角度来看，分区后的表与非分区表完全相同，使用 SQL DML 命令访问分区后的表时，无需任何修改。

例子：（该sql是对T\_RECEPTION按时间查询数据，T\_RECEPTION一共有6631496条数据，如果不使用分区将对这6631496数据进行全部的扫描）



通过改造T\_RECEPTION，注意一定要按查询条件RECDATE进行分区。可以从执行计划中看到T\_RECEPTION只用了4秒的时间，sql从原来的41秒缩短到19秒。整体效率提升2.15倍

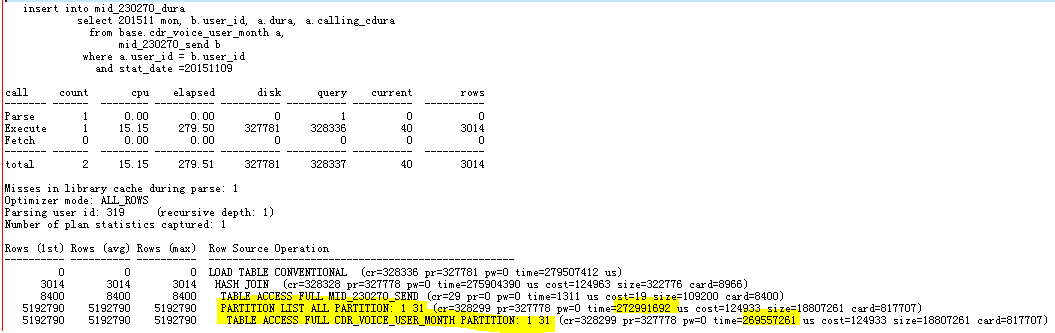




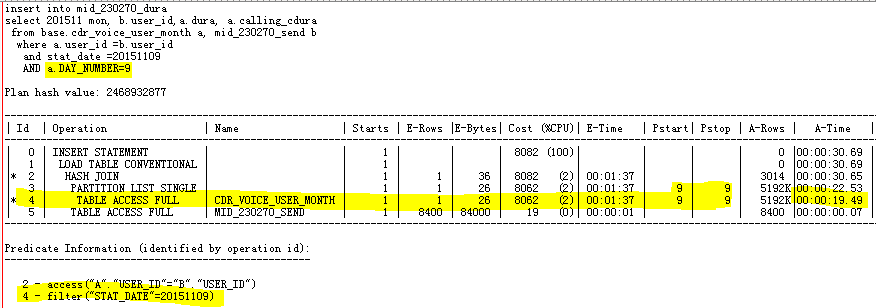
2.1.2 场景B 使用分区键查询

在部分存过中，对分区表按天来取数时，没有利用到分区表的分区键值。因此sql对全部的分区进行了扫描。建议对分区表取数时候，如果取数条件用到分区键的话，需要在where条件中要用该字段。

例子：（该条sql取的是前一天数据，用的是stat\_date这个字段，但cdr\_voice\_user\_month该表的分区是以DAY\_NUMBER为分区。因为where条件中没有使用DAY\_NUMBER这个条件所以，对cdr\_voice\_user\_month全部31个分区进行了扫描，总共用了272秒，占整个sql的97%。）



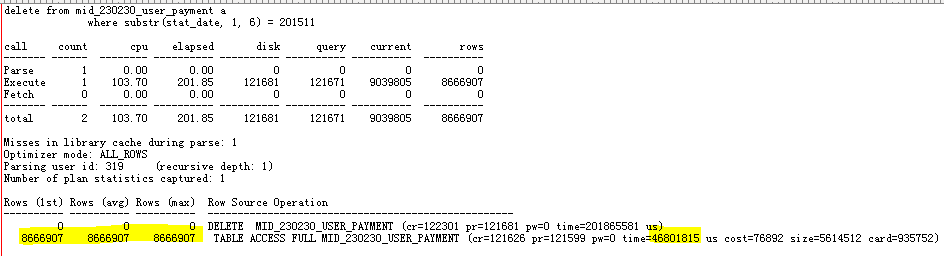
通过增加分区键的条件，可以看到执行计划只扫描了一个分区，从原来的279秒缩短到30秒。整体效果提升9.3倍



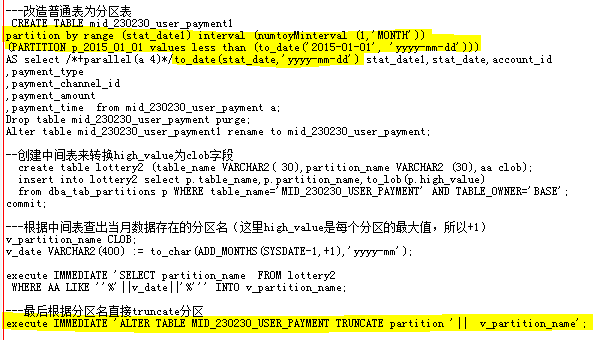
2.1.3 场景C 使用truncate独立分区

由于要删除某一个月份的数据，并且删除的数据量很大。delete大量的数据会很慢，因为要对undo做CR块用来一致读和恢复。建议这种大量删除并且是有一个时间范围的，可以对该表进行按时间范围的分区，如果是11g可以用到其特性自动分区，然后再根据dba\_tab\_partitions的high\_value对单独的分区进行truncate，由于truncate只产生很少的redo和undo所以速度要比delete快很多。

例子：（该sql中要对mid\_230230\_user\_payment删除当月的数据，一共删除了8666907条数据，耗时201秒）。



通过对该表按stat\_date做了格式转换成yyyy-mm-dd时间格式，然后以这个新的字段按月做分区（这里用到11g新特性，可以每月自动创建新的分区，但分区名称是系统自动给的），根据dba\_tab\_partitions的high\_value查询出当月的数据在哪个分区中，（由于high\_value为long数据类型，不能直接对其进行变量赋值，所以需要创建中间表来将long转换成clob字段进行PL/SQL处理），最后根据分区名直接truncate该分区。

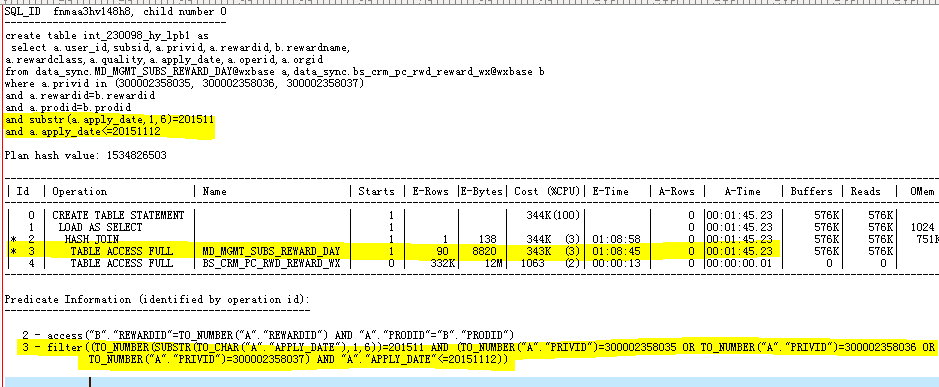


改造完后truncate只需1秒，从原来的201秒缩短到1秒，整体效果提升201倍

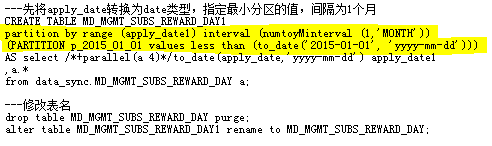
2.1.4 场景D 合理运用11g自动分区

有时候我们要查询当月的数据，可以把表按月做分区，但手工按月建分区表只能建一部分，数据库11g对这一项限制做了改进。建议如果是11g的库可以利用新特性来自动创建分区表无需人工去干涉。

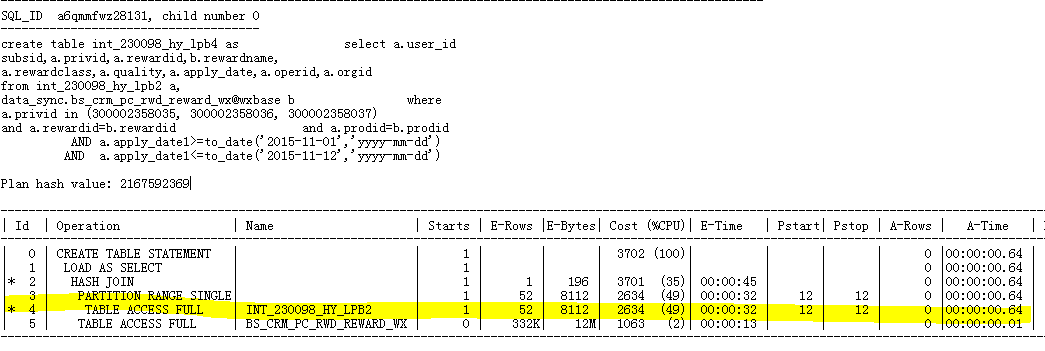
例子：（该sql是要取apply\_date为从当月1号到当天的数据，（这里由于MD\_MGMT\_SUBS\_REWARD\_DAY没有符合条件的数据，所以作为父节点ID 2控制子节点ID 4不运行，所以这里starts为0，我们主要看的是ID 3这步所消耗了105秒，占整个sql100%）。



通过将MD\_MGMT\_SUBS\_REWARD\_DAY改造成自动分区的分区表，无需人工去建立分区名和分区，只需在建表时，就指定按月和那个字段做分区就行了，



查看改造完后的sql执行计划，扫描MD\_MGMT\_SUBS\_REWARD\_DAY只扫描了一个分区。从原来的105秒缩短到45秒，整体效果提升2.3倍



2.2 远端表和本地表进行关联

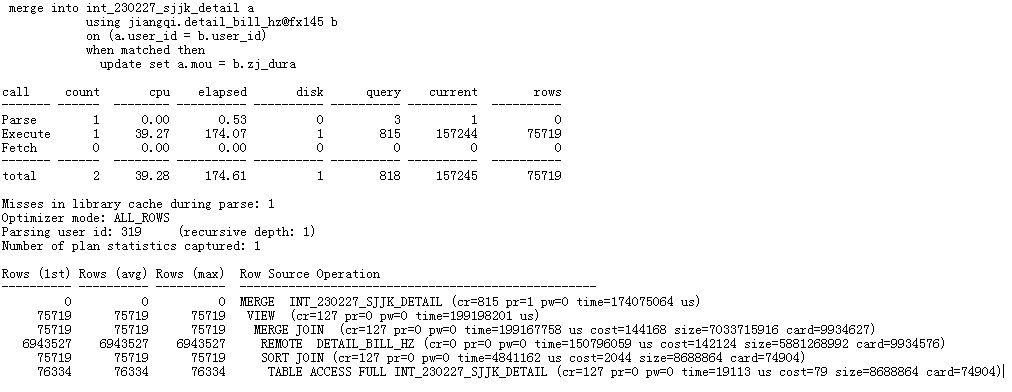
E. 远端大的表和本地小表进行关联

F 远端表和本地表或不同库的远端表进行关联

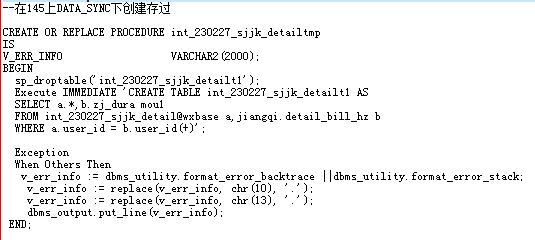
2.2.1 场景E 建立中间过程

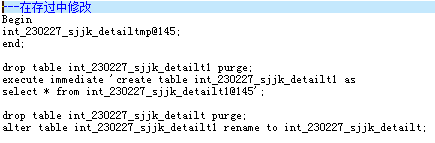
由于一些sql需要将远端的大表和本地的小表进行关联，其中大表通过条件过滤完后的数据量也很大，所以必须先从远端将这大量的数据传过来，再和本地小表关联。建议可以在远端建立中间存储过程，先将本地的小表抽到远端上，进行关联结果集，然后再将关联后的小结果集传回本地。

例子（在此sql中远端表jiangqi.detail\_bill\_hz有6943527条记录，本地表只有75719条记录。其结果集75719条，所以大部分时间消耗在远端通过网络取数）

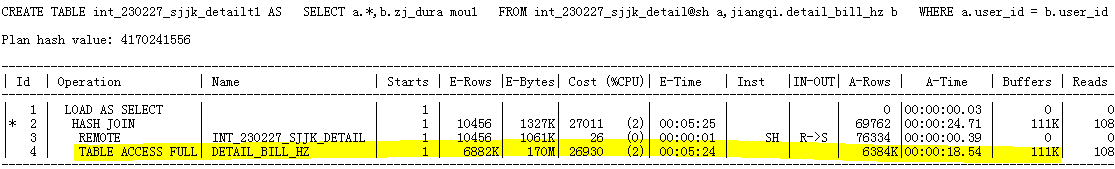


通过在fx145上建立中间存过，来避免大表通过网络来传到本地，再进行关联。





查看执行计划可以看到jiangqi.detail\_bill\_hz在本地取数只要18秒，而通过网络取到wxbase上所花费的时间为150秒

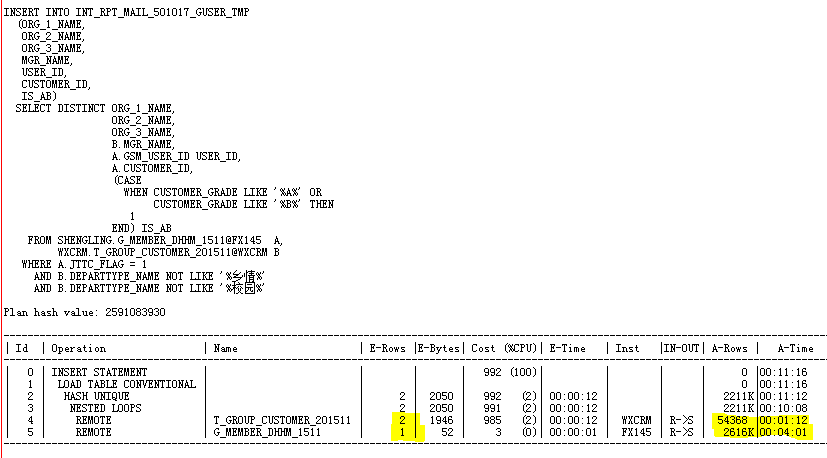


在远端执行完关联所花24秒，再传回本地，整个过程所需33秒，原来为174秒 整体效果提升5.2倍

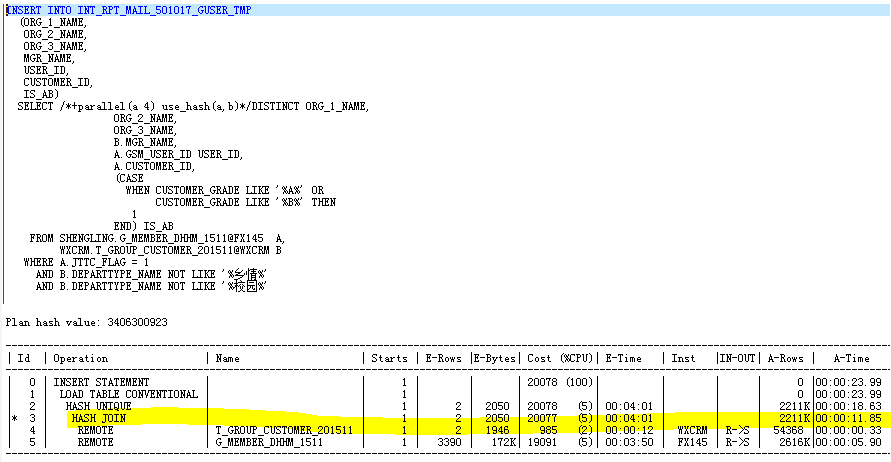
2.2.2 场景F dblink在CBO中的限制

这情况比较特殊，需要开发人员在编写完sql后，测试注意下执行计划中，remote远端表在执行计划中返回的数据量和真实select count(\*) from tab@dblink返回数据量是否差的很远，由于dblink的特殊性，oracle CBO对dblink的表有时候估算值为1，但实际上远端表有上百万条记录，这直接导致走错误的执行计划

例子：（在此sql中在估算G\_MEMBER\_DHHM\_1511和T\_GROUP\_CUSTOMER\_201511两个表的值都很小，但实际上远远大于估算的值，因此走了个nls的执行计划，这里nested loops用了5分多种，因为G\_MEMBER\_DHHM\_1511表进行了5万次的循环



根据实际返回值，用hash\_join来关联这两个表是最合适的，因此可以使用hint来强制sql走hash\_join



虽然表SHENGLING.G\_MEMBER\_DHHM\_1511添加了并行抽取的速度提升，但这边hash\_join的时间只花了11秒，和原来的5分钟相比提升了27倍。

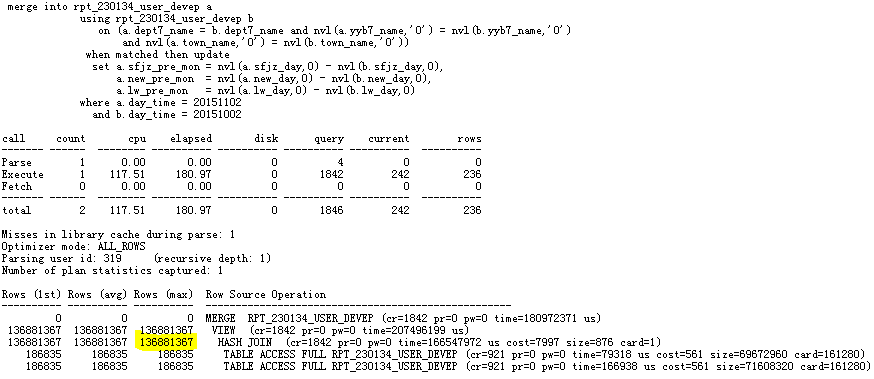
2.3 merge条件尽量写在内层

**I**. merge into条件写在内层

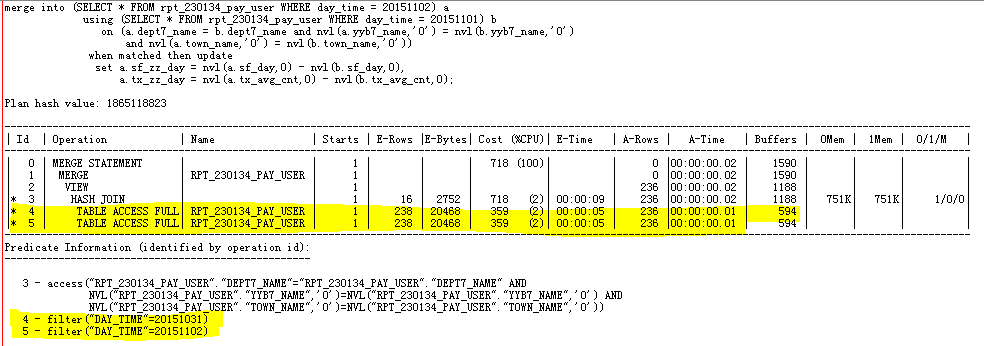
2.3.1 场景I merge条件尽量写在内层

A表中按一定的条件和B表关联，更新A表的字段，虽然可以在merge最后使用where语句进行过滤所要更新的行，但实际上oracle并没有减少扫描A表的行，而是做了类似case when的过滤。建议在merge后使用子查询将A表按条件过滤后再和B表关联更新字段，这样减少A表不必要的行扫描

例子：（这条merge语句所取的是20151102这天数据做更新，但把where过滤条件写在最后使的oracle在hash\_join后才根据条件进行选择更新，这里由于有重复值，所以关联的时候产生笛卡尔乘积现象，实际最终只更新了236条数据）

****

通过改写sql，将where条件带入子查询中，减少不必要的行扫描。再进行关联更新



在ID 4,ID 5步骤中就过滤出236条需要更新的行，时间从180秒缩短到0.02秒，整体效果提升9000倍

2.4 CTAS代替merge

**H**.一张大表merge into另一张表的某个新添加的字段

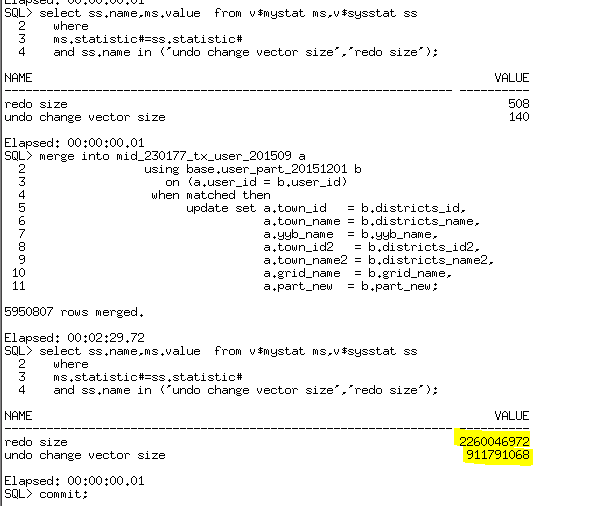
**J**.多次 merge 同一张不同字段

2.4.1 场景H使用CTAS代替merge大表更新

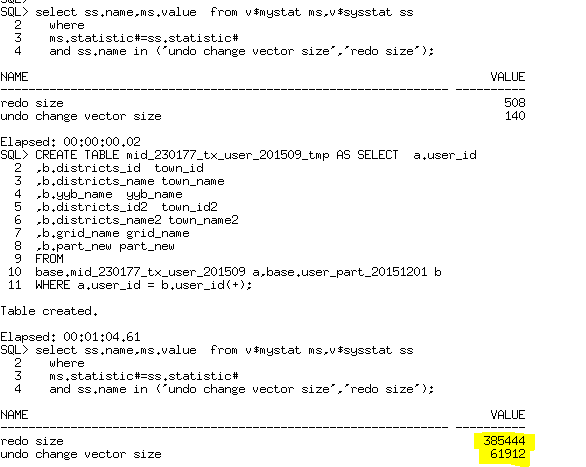
很多存过中使用了alter table add column 后，再merge into 表，在存过中大部分通过这类操作的都是临时表。merge into后都不会去回滚，直接提交，因此没有机会去使用undo，由于OLAP中处理的数据都是巨量的，因此如果使用merge会在undo中构造大量的CR块，造成大量的逻辑读。建议在这种处理数据过程中，尤其merge的是临时表，可以使用CTAS来代替merge，减少不必要的逻辑读，提高速度。

例子

首先我们来看下merge into查询所产生的undo和redo数

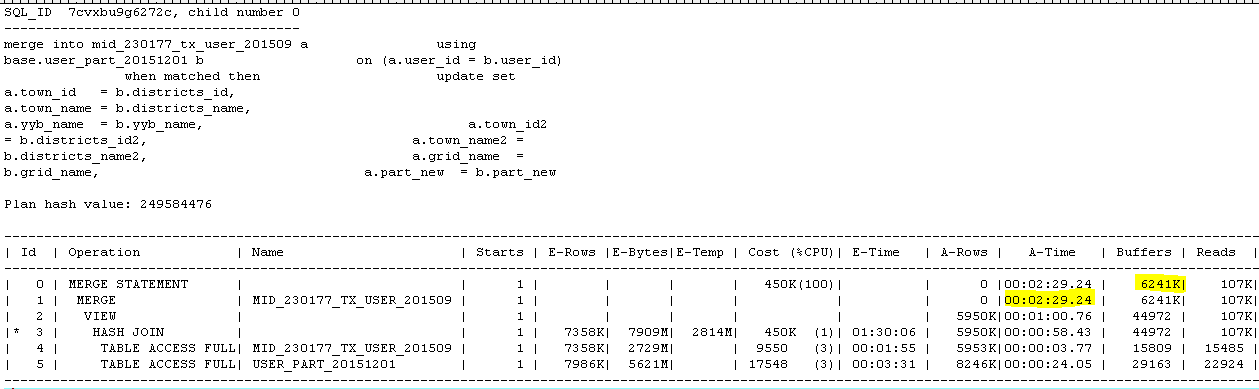


CTAS的所产生的undo和redo数，发现CTAS所产生的undo和redo远远小于merge

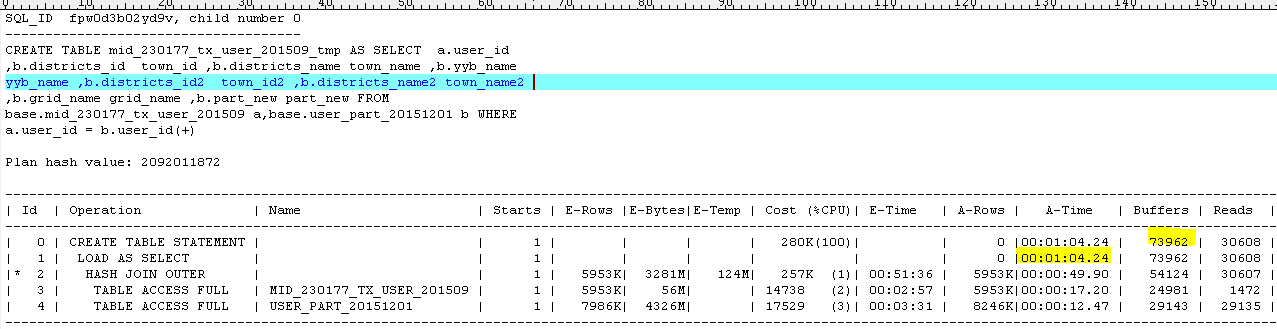


再来看下两者的执行计划

merge into



CTAS

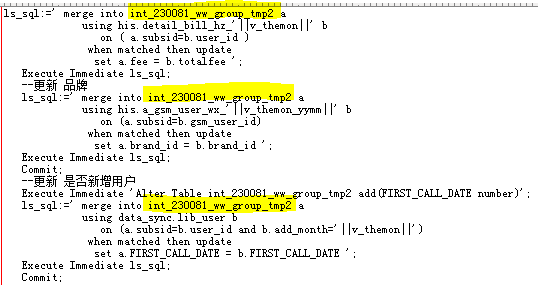


比较这两个执行计划，都是TABLE ACCESS FULL扫描，做的是hash join,唯一的不同在于ID 1这一步。由于CTAS只需产生少量的UNDO，并且以直接路径的方式插入所以逻辑读少很多，时间从149秒缩短到64秒，整体效果提升2.3倍

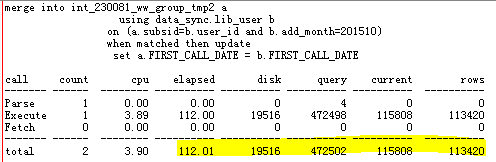
2.4.2 场景J避免多次merge同一张表

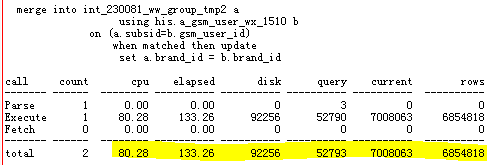
发现在存过中对同一张表merge多次，每次merge关联的表不一样。建议在这种情况下，可以把多条需要merge的sql合并成一条CTAS，这样避免了merge into的表进行了多次反复扫描。

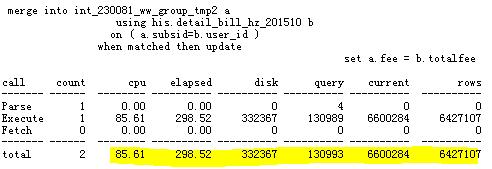
例子：（在此sql中对mid\_230032\_3w\_'||v\_themon||'这张进行了3次merge，其中关联的表有2次是同一张）



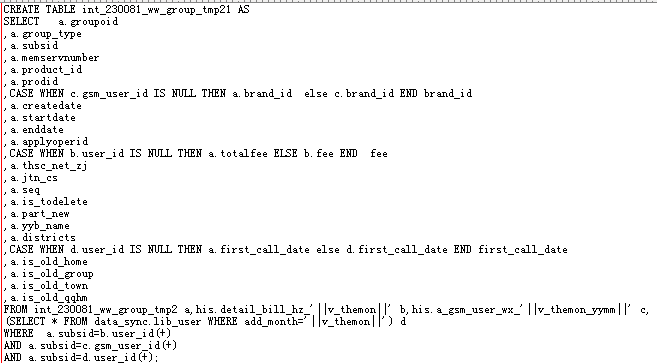
查看执行计划，发现每次的merge所花的时间和资源都不小，如果进行合并将减少不必要的资源开销。



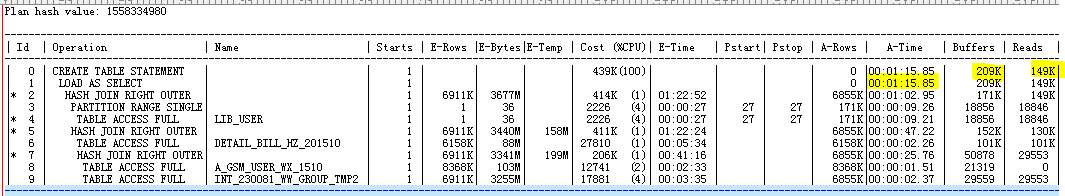




通过用ctas来一次关联4张表，使用外关联，对要更新的字段用case when 来分别更新。



查看下新sql的执行计划，合并后所需时间为75秒，原来3个sql一共消耗543秒，整体效果提升7.24倍



2.5 merge/CTAS代替update

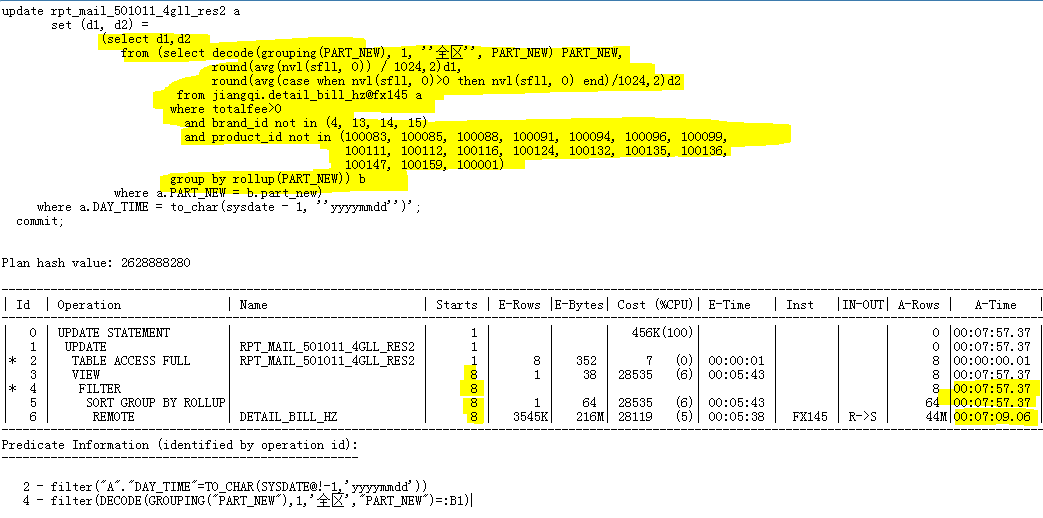
K. update 子查询耗时过长

L. update一张大表

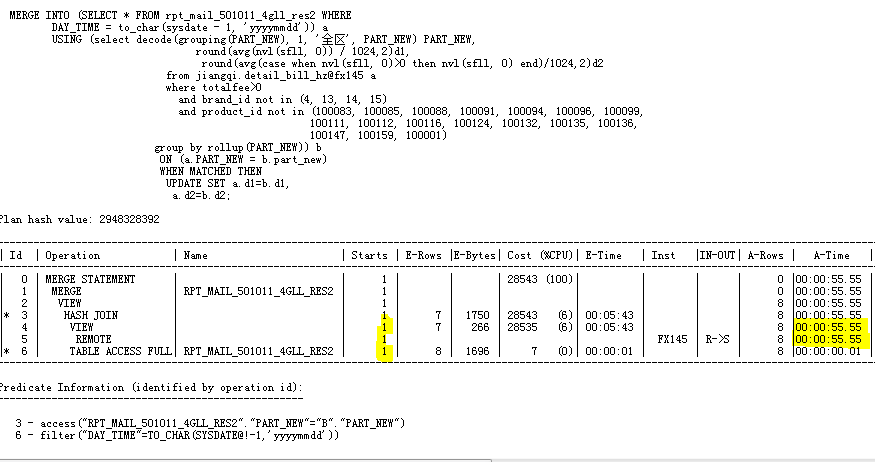
2.5.1 场景K merge代替update（子查询）

虽然update的表只有10几行，但在set中是一个按明细进行的汇总计算的子查询sql，因为数据量很庞大所以使得整个update耗时很长。使用update时会对要更新的表中每一行进行set中子查询的计算，因此。建议在此情况下，由于更新的表数据量不是很大，可以使用merge来代替update

例子：（在此sql中set里面是个按地区汇总计算的sql，每次group by需要花费近1分钟，虽然rpt\_mail\_501011\_4gll\_res2一共需要更新的只有8行，但8次计算group by的子查询所花时间就8分钟，大大的拖垮了整个sql的效率。）

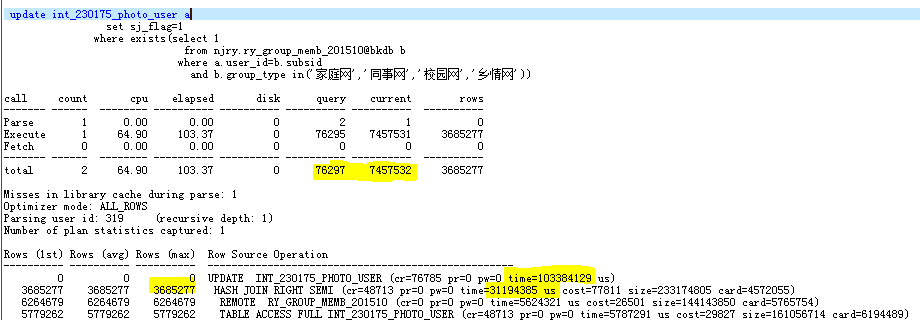


通过将update语句改造成merge，避免循环多次子查询，可以从执行计划中看出子查的表只扫描了一次。从原来的477秒缩短到55秒，整体效果提升8.6倍

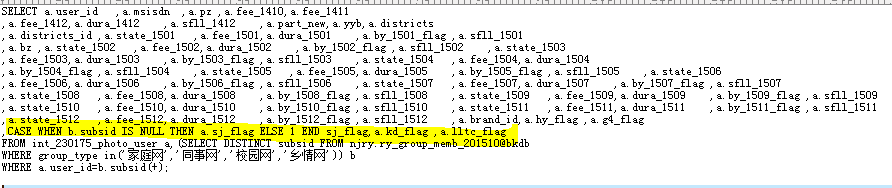


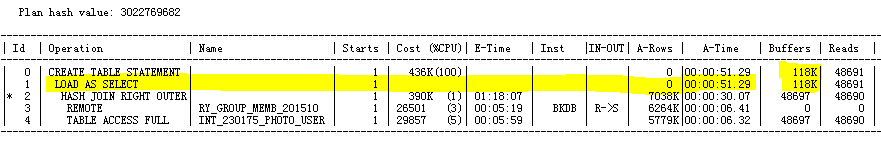
2.5.2 场景L CTAS代替update大表

update一张大表，虽然set中没有子查询是个固定值，但由于update的特殊性，会对大表中每一行都进行该字段更新，同时会在undo中构造cr块保证一致读。对于这种批量的更新操作，建议使用CTAS来代替update，CTAS处理的方式是按块处理而不基于行，效率必定会快很多。

例子：（在该sql中，可以通过time指标看出最耗时间的步骤是update INT\_230175\_PHOTO\_USER，该步骤消耗的时间为（103384129- 31194385）/1000000=72秒，占整个sql的70%，在这步骤中就进行了逐行的处理并且构造CR块）

通过改写sql，语句将update改为CTAS+case when，可以从执行计划中看出，在表扫描和关联与update几乎没有差别，但在LOAD AS SELECT这步中明显的缩短了时间和逻辑读，因为不是逐行进行处理的。从原来的103秒缩短到51秒，整体的效率提升2倍





2.6 dblink 远程抽数

M. 大表每日从省公司抽取

N. 多张远端表进行关联或者有使用group by

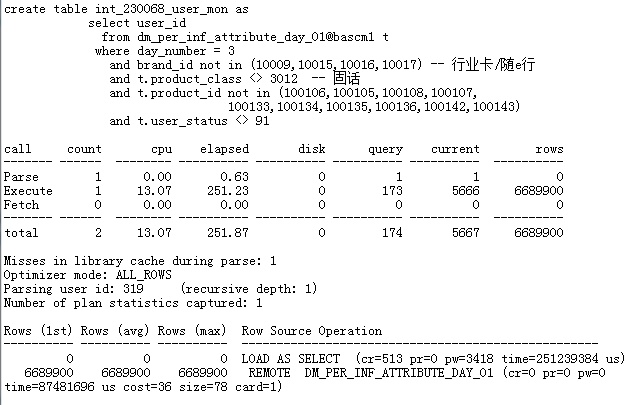
O. 抽取的表中条件有使用本地函数

P. 重复抽取同一张表

2.6.1 场景M 合理使用parallel

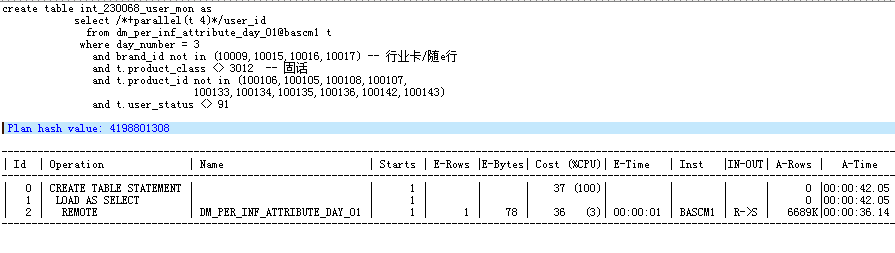
很多基础数据都要从省公司抽取，由于表都建在省公司数据库服务器上，无锡本地无法干涉省公司的表，所以只能通过添加合适的并行度来提高抽取速度。由于并行对系统有一定的开销，所以劲量不要使用，如果要开并行最多4个。（注意，单表查询或做group by计算中所需查询的数据量占总表的80%-100%并且数据量在百万级别或以上，可以使用并行，并行数一般都是以偶数增加，建议在测试一条sql使用并行时从2开始增加，同时使用top或vmstat观察操作系统idle的情况，如果使用并行使idle有明显的下降不要增加并行数或者减少并行数）

例子：



由于查询所需的数据量5173930，而且是单表的抽数，所以经测试可以添加4个并行度来提高数据抽取的速度。

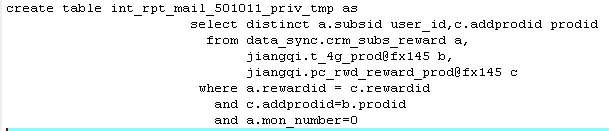
改造后从原来的251秒缩短到42秒完成。效率提升5.9倍



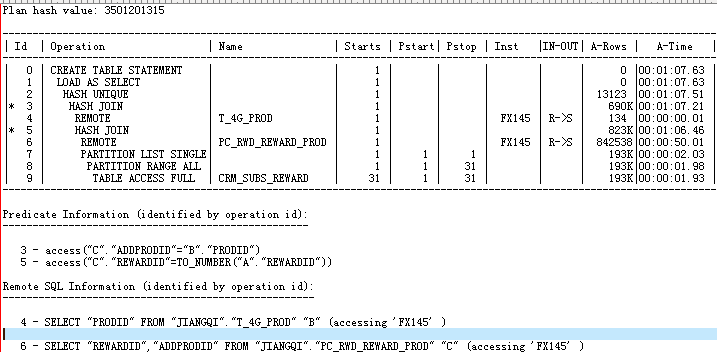
2.6.2 场景N 建立中间视图

有时候涉及到要将远端的多张表进行关联取回本地来使用。针对这种情况下建议如果要抽取同一个远端库不同表关联的数据，可以在远端库上建立视图。这样做可以将关联后的结果集通过网络传到本地，不必把原表抽过来再进行关联。（注意如果涉及的表在本地库中，建立试图不会提高效率）

例子：

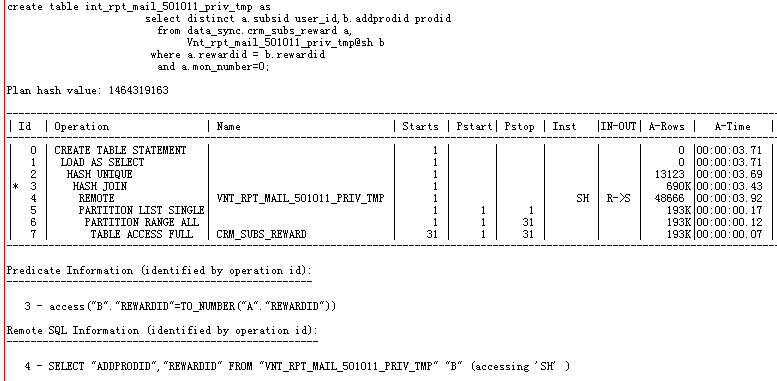


查看执行计划发现其分别抽取了2张远端表到本地再进行关联



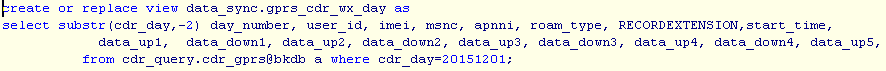
我们在fx145上创建视图后，用视图来取数，从原来的67秒到3.71秒，效果提升18倍





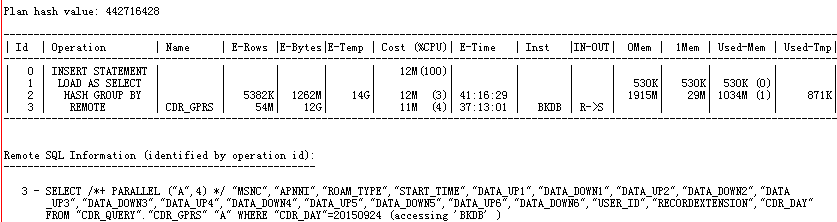
另外还有就是在远端抽取的表中有使用group by 分组函数，对于oracle分布式查询,CBO选择是低效的

例子：data\_sync.gprs\_cdr\_wx\_day是一个视图

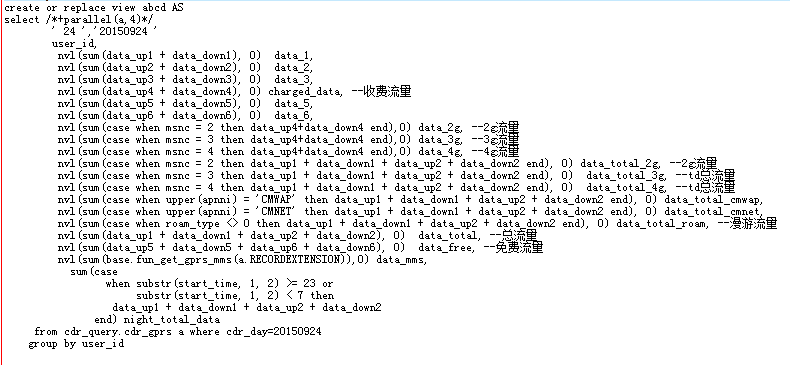


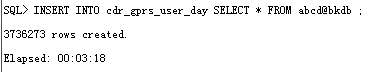


查看执行计划发现在远端执行的sql并没有进行聚合，而是将整表抽取到本地再进行group by

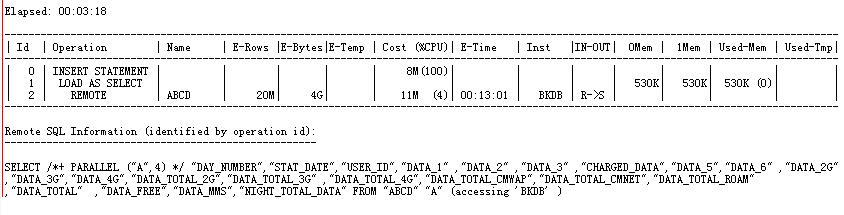


通过在bkdb上创建视图，然后用视图来直接insert into cdr\_gprs\_user\_day





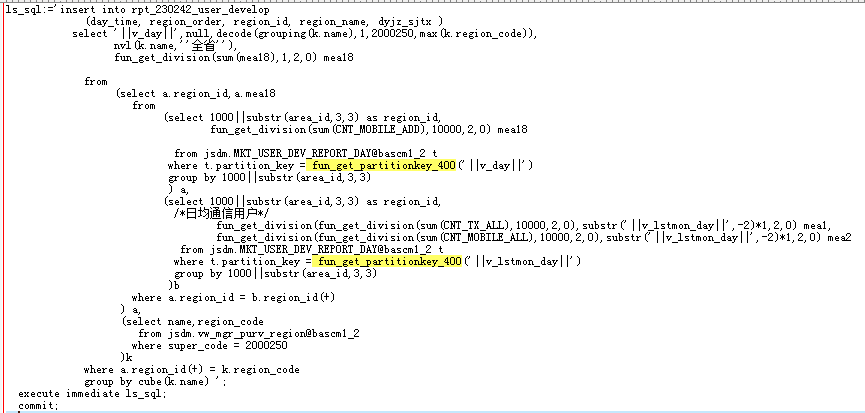
查看执行计划，sql直接查询，直接将group by后的结果集传到本地，从原来的1826秒缩短到191秒，整体效果提升9.5倍



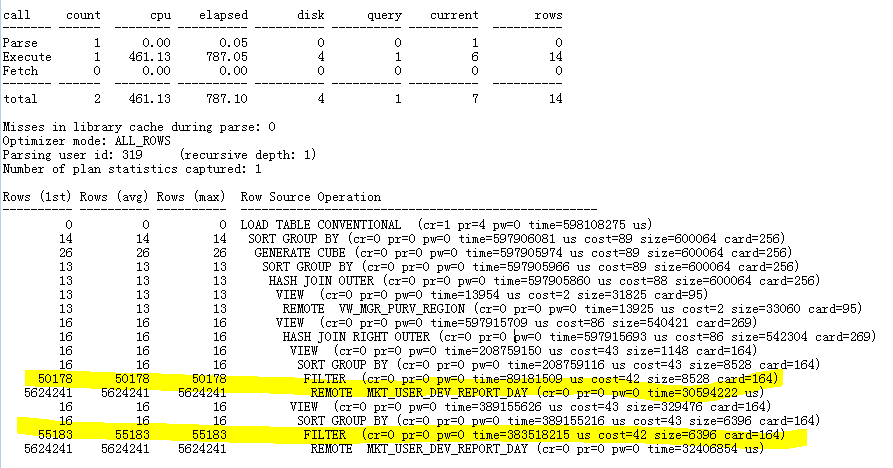
2.6.3 场景O 合理在dblink中运用函数

有时候会使用本地函数赋值带入参数到sql语句中，这在dblink中使用将会导致一个低效的执行计划，为了避免出现错误的过滤方式，建议先将要传入的参数在外层进行函数赋值，然后将赋值后的参数带入sql语句中。

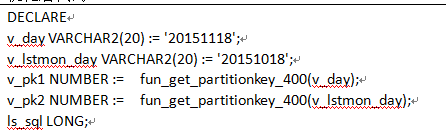
例子：此sql中运用 本地函数fun\_get\_partitionkey\_400('||v\_day||'来限止jsdm.MKT\_USER\_DEV\_REPORT\_DAY@bascm1\_2取数范围

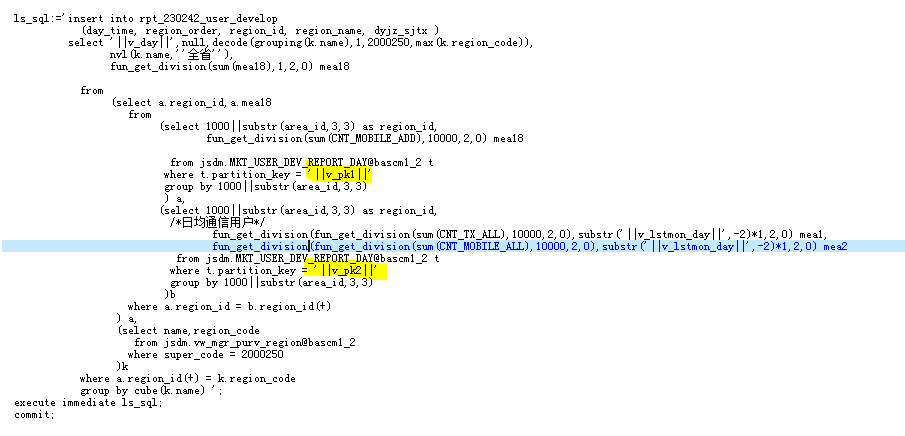


查看10046执行计划，发现黄色部分是通过本地的函数进行过滤数据，从5624241行取其中的50178行，由于远端没有该函数所以将jsdm.MKT\_USER\_DEV\_REPORT\_DAY全部抽取到本地，抽取过来的数据是没有索引，或者分区的功能，这将消耗大量的时间在逻辑读中。

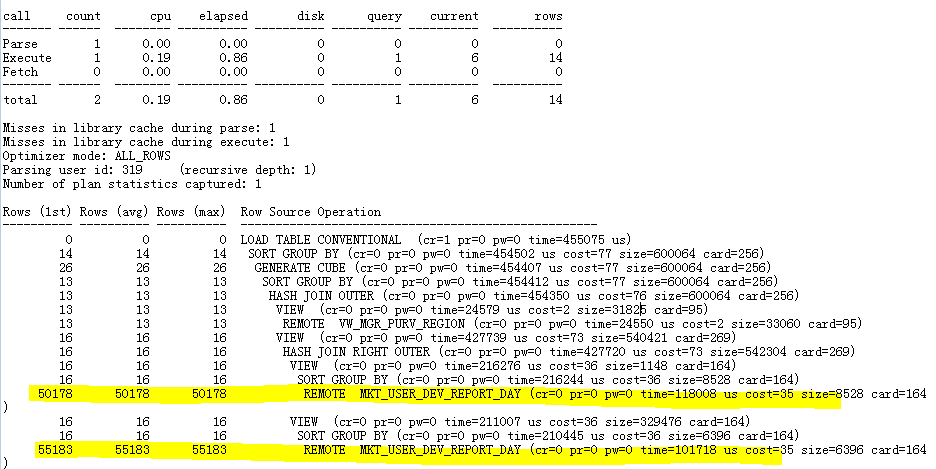


通过将函数在sql之前就进行赋值，然后将赋值后的参数带入sql中，使其走正确的执行计划。从原来的787秒缩短到0.86秒，整体效果提升915倍





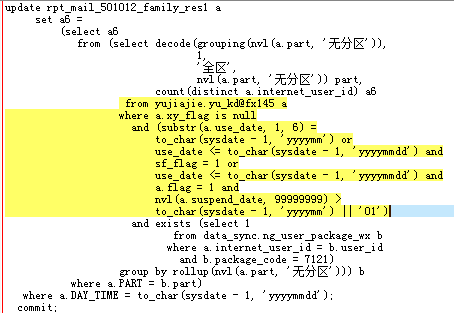
查看执行计划，过滤在远端就已经完成

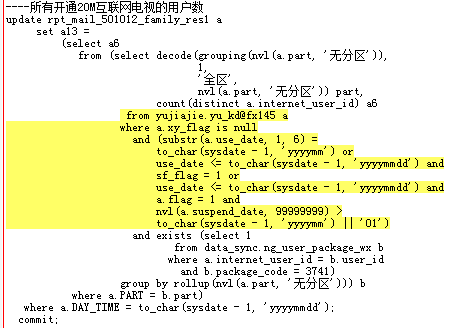


2.6.4 场景P避免多次重复抽取

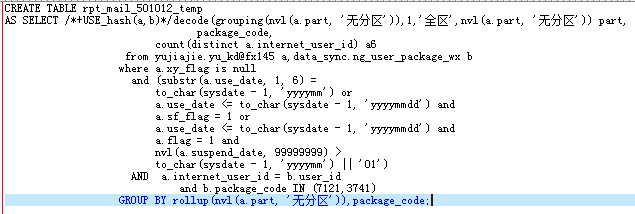
存过中很多使用了重复对一张远端表进行抽取，每个抽取过滤条件中都带有共同的部分。建议对这种重复抽取的表，可以在最初的时候按共同取值的条件一次抽取建张临时表，之后的sql直接取临时表来关联计算

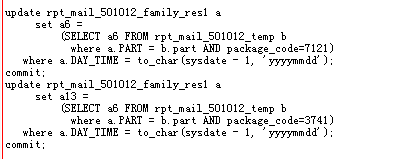
例子(有两段代码，他们都有共同取数的表和条件）





这部分共同的取数，可以建一张临时表来一次取完公共的数据





通过合并这部分代码后，从365秒缩短到18秒，整体效果提升20倍

2.7 nls改hash join

P 两表关联CBO估算错误

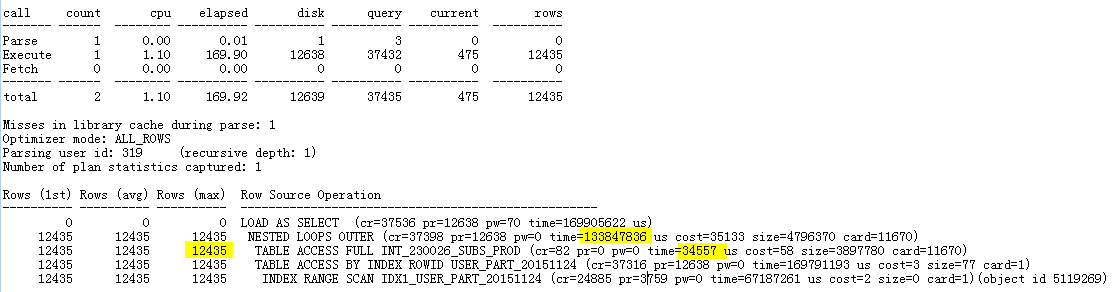
2.7.1 场景P两表关联CBO估算错误

有时在两表关联，由于统计信息不是最新的导致CBO估算不准确，使用索引+nls嵌套循环，但sql最终得到的数据远超过估算的，因此sql花了大量的时间在循环中。（注意我们在测试sql时一定要关注执行计划返回的数据量是否和我们真实的接近）

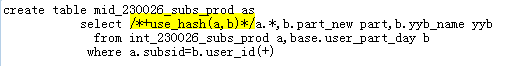
例子：是一个很简单的两表外关联的sql，INT\_230026\_SUBS\_PROD只有12435行，但该语句花了169秒的时间。

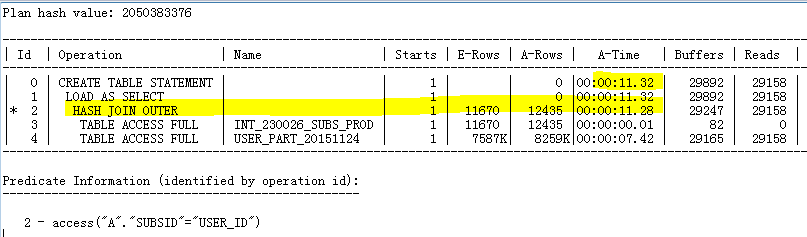


查看该语句的执行计划，发现使用了索引+nls，由于int\_230026\_subs\_prod作为内部表，外部表user\_part\_day需要循环12435次，从执行计划可以看出大部分时间都消耗在循环中（133847836- 34557）/1000000=133秒，占整个sql的79%



为了避免循环，使用hint让user\_part\_day走全表扫描，只做一次关联。时间从原来的169秒降低到11秒，整体效果提升15倍





2.8 TABLE函数CBO基数的限制

R TABLE函数CBO基数估算固定8168

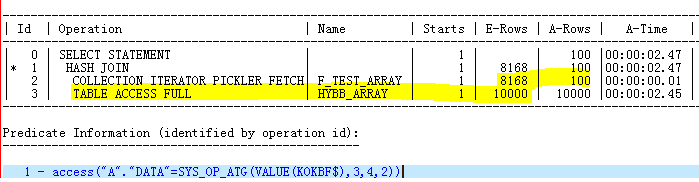
2.8.1 场景R TABLE函数CBO基数限制

在有些时候需要使用table函数来把数组类型的变量展开显示出来，但此时需要注意了oracle对table函数的基数估算有限制，这可能导致走错误的执行计划。（注意这个和统计信息无关）。

例子：SQL其实很简单，一个查询构建的A表，一个TABLE函数构建的B表关联

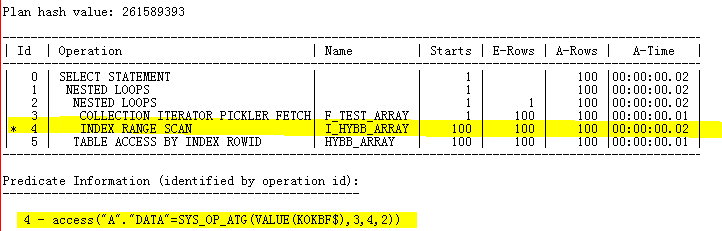


查看执行计划，发现hybb\_array表走了全表扫描，导致CBO选择full table scan是因为估算table(f\_test\_array(100))固定返回8168。但实际该函数返回的只有100多行。



通过使用hint将table函数返回的数值改为100 /\*+ cardinality(b 100) \*/，使其走nls+index，避免全表扫描。从原来的2.47秒降到0.02秒，整体效果提升123倍





2.9 or 与semi join触发错误的执行计划

S or 与semi join触发错误的执行计划

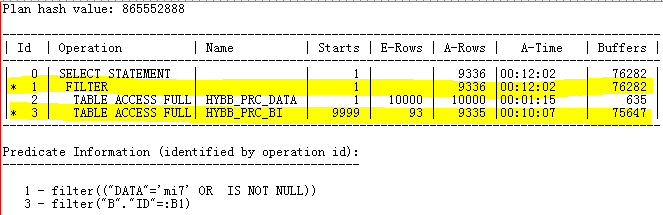
2.9.1 场景S or与exists触发错误执行计划

当OR与semi join放在一起的时候，会触发无法进行subquery unnest的问题，也就是可能会产生FILTER,导致SQL非常缓慢。

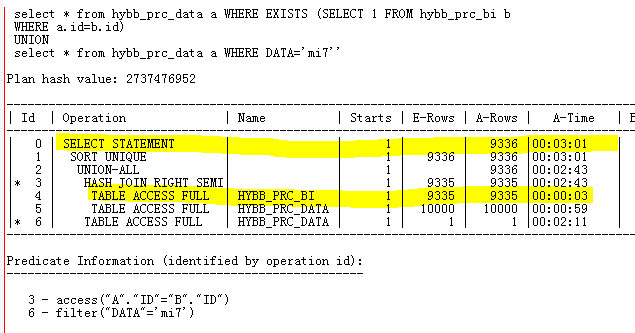
例子：



查看该sql的执行计划，发现执行计划中会发现绑定变量，父节点的行集会驱动子节点查询。FILTER与NESTED LOOPS类似，与NESTED LOOPS区别是FILTER内部会维护一HASH表，所以父节点10000行，子节点实际执行了9999次，如果JOIN条件，比如：B1这个值重复的特别多，那么第3步的STARTS就会少很多。那么FILTER还是有很大的优势



对于该语句我们只能改写sql，很显然，这里的条件是exists or ...那么改写得用UNION或UNION ALL了，为了避免有重复行，用UNION来避免循环那么多次。从原来两个分支都走HASH JOIN，虽然全部是全表扫描，但是执行效率提升很明显，执行时间从12分钟缩短到3分多种，整体效果提升4倍



3 总结和建议

此次主要针对战报和邮件日报中的15个模块进行优化，涉及200多个存过，代码行将近10万行，另外还有162个独立的存过。优化后15个模块在省公司正常下发资产数据表的情况下，都能在上午11点之前能完成。提升1倍的效率

对存过中涉及的sql建议以下几点

* 1. 访问同一个远程库的次数要尽量少,也就是同一远程库的对象应该尽量转为一条SQL运算，一次运算，运算后将结果返回给本地库
  2. 对于数据处理过程中，大数据量的更新劲量使用CTAS来代替merge减少不必要的逻辑读
  3. 适当使用并行提高效率：在存过操作中，使用并行能够迅速提高效率，注意并行度的设置，需要在测试程序运行过程中，观察CPU使用情况。
  4. 大表的更新或者涉及复杂计算的update劲量使用ctas，避免循环的次数过多
  5. 要想使用到分区扫描必须sql中使用到分区键（分区字段不能添加函数）
  6. 如果使用了分区表进行查询，不能在分区键上使用函数（to\_char)这类
  7. 在使用复合分区时，如果where只有子分区的分区键，ORACLE会对各子分区和分区做合并，效率比未分区的还要慢
  8. 避免在字段上使用函数，如果该字段上有索引，索引将失效。如果没有索引，添加函数如（to\_char）会对所有的行进行格式转换，导致查询时间变长
  9. 尽量不使用自定义函数，在sql中函数不能做块的处理只能做行的处理，如果查询数据量巨大将会拖垮整个sql的性能
  10. 在有exists 和or共同使用时，要注意执行计划是否合理
  11. 如果要取字段为null并且这部分数据量很少，可以建立bitmap索引或者函数索引
  12. 如果逻辑可以的话，尽量使用union all代替union，减少排序提高效率
  13. 如果表不会经常做DML操作，并且会经常做order by排序操作，可以在排序的字段上建索引，利用索引的有序性提高排序操作
  14. 在sql测试中，要对执行计划中返回的CBO估算值要和实际表按条件返回的值进行比对，如果CBO估算出现大的偏差，可能会导致sql走了一个错误的计划，需要对该sql执行计划进行手动调整。