**第一周学习计划总结**

**执行计划基础知识**

云和恩墨(北京)信息技术有限公司

技术顾问 燕鑫

http://www.enmotech.com

**文档控制：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序 | 版本号 | 更改人 | 日期 | 备注 |
| 1 | 1.0版 | 燕鑫 | 2018-04-21 | 初始版本 |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编制 | 燕鑫 | （签字） | 日期 | 2018-04-21 |
| 校对 |  | （签字） | 日期 |  |
| 审核 |  | （签字） | 日期 |  |
| 批准 |  | （签字） | 日期 |  |

目录

[一． 执行计划的构成 5](#_Toc512096898)

[1. 执行计划的正文 5](#_Toc512096899)

[2. 执行计划的主体 5](#_Toc512096900)

[id，operation，name 6](#_Toc512096901)

[starts 8](#_Toc512096902)

[A-rows 8](#_Toc512096903)

[A-time 8](#_Toc512096904)

[buffers 8](#_Toc512096905)

[0mem，1mem 8](#_Toc512096906)

[3. 执行计划的Qurey Block Name 8](#_Toc512096907)

[4. 执行计划的OUTLINE部分 9](#_Toc512096908)

[5. 执行计划的predicate information 10](#_Toc512096909)

[6. 执行计划的column projection 10](#_Toc512096910)

[7. 执行计划的note 11](#_Toc512096911)

[二． 执行计划的阅读顺序 11](#_Toc512096912)

[三． 执行计划的获得方式 13](#_Toc512096913)

[1. explain plan for 13](#_Toc512096914)

[2. autotrace 14](#_Toc512096915)

[3. dbms\_xplan 14](#_Toc512096916)

[4. 10046事件 14](#_Toc512096917)

[四． 执行计划相关的视图 15](#_Toc512096918)

[1. shared cursor构成和查找方式 15](#_Toc512096919)

[2. 相关视图 16](#_Toc512096920)

[V$SQL\_PLAN\_STATISTICS\_ALL 16](#_Toc512096921)

[V$SQL\_PLAN 17](#_Toc512096922)

[V$SQL\_PLAN\_STATISTICS 17](#_Toc512096923)

[V$SQL： 17](#_Toc512096924)

[V$SQLAREA： 17](#_Toc512096925)

[V$SQLSTATS(同上) ： 18](#_Toc512096926)

[dba\_hist\_sql\_plan： 18](#_Toc512096927)

[dba\_hist\_sqlstat: 18](#_Toc512096928)

[dba\_hist\_sql\_sqltext： 19](#_Toc512096929)

[all\_source: 19](#_Toc512096930)

[dba\_hist\_optimizer\_env： 19](#_Toc512096931)

[dba\_hist\_snapshot： 19](#_Toc512096932)

[dba\_hist\_sql\_bind\_metadata： 19](#_Toc512096933)

[dba\_tables： 19](#_Toc512096934)

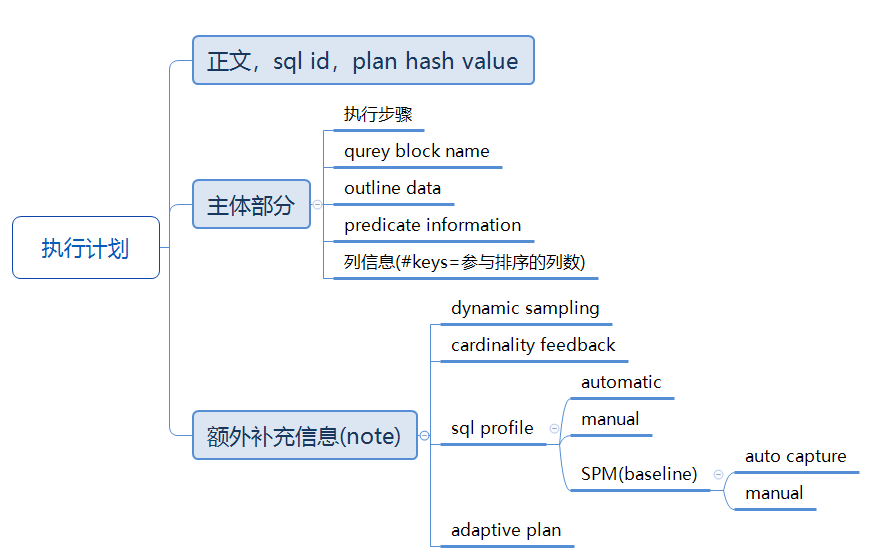
[dba\_indexes： 19](#_Toc512096935)

[dba\_ind\_columns： 20](#_Toc512096936)

[dba\_tab\_columns： 20](#_Toc512096937)

[五． 获取topsql 20](#_Toc512096938)

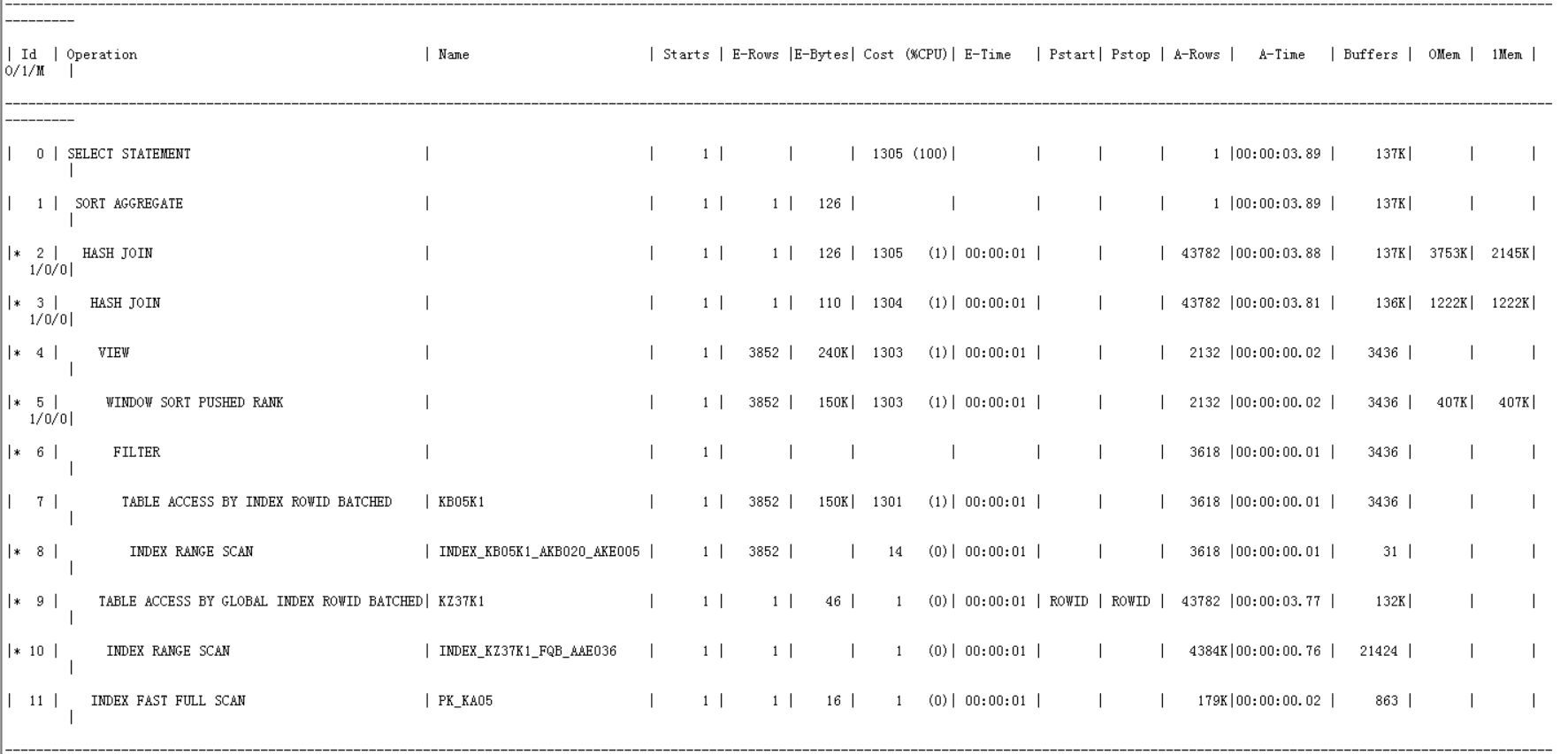
# 执行计划的构成



## 执行计划的正文

基本上就是第一部分是执行计划的开头，有sql文本，sqlid，plan hash value，child number，这就不赘述了。

## 执行计划的主体

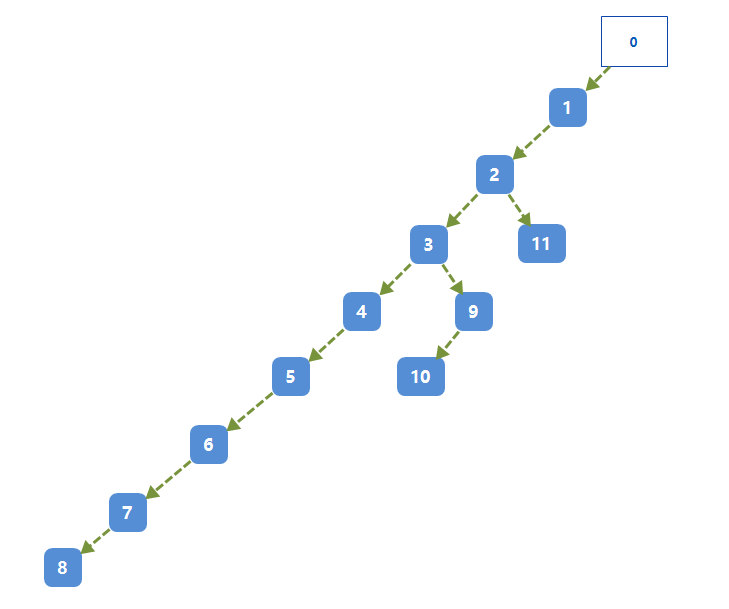


  先说一下，这个正文的由来，为了能看到starts,a-rows,a-time,buffers,0men,1mem... 所以，使用了:  
alter sesssion set statistics\_level=all;  
select \* from dbms\_xplan.display\_cursor(null,null,'advanced allstat');  
所以说我们这次主要说的是以dbms\_xplan包来看的执行计划内容。

### id，operation，name

我们先笼统的介绍一下这个3个的意思跟由来。

id：其实就是标识每一个步步骤的一个序号，就是跟个身份证似的，这个是从v$sql\_plan的id列中得到的，并不代表他们执行的顺序，不过我稍后就会说阅读执行计划的顺序及其原理。

operation：是v$sql\_plan中的operation列加上option列组合出来了，意思很浅显，说的就是这一步的操作。  
      name：就是v$sql\_plan中的object\_name，也很明白，就是这一步操作的对象名称。  
      现在我们来解释如果确定每一步的执行顺序。  
      口诀随便都能找到，就是看缩进，从左往右看，不对齐的，就是靠右的先执行；到上下平行的或者说对齐的（紧挨着的），那靠上的就比靠下的先执行，然后再继续右看，看到最右边的时候就完事儿。其实这个看不太明白我说的的，可以结合一下百度。  
我直接说原理，当然也不是最底层实现的原理，那个现在还搞不来。  
      从左往右看，靠右的先执行是因为，一个缩进就代表了一对父子，儿子把自己的步骤得到的结果集整理好，上交给老子，就这么一回事儿。所以说0-8的执行顺序是8 7 6 5 4 3 2 1 0.（我这里又要道歉，我有点儿蠢了，没给大家选到那种有俩个  
紧挨着的儿子的执行计划，所以这里先说原理，大家这块再结合下百度）  
      然后当你在找爹的路上往回走的时候，发现到了id=4这一步，往下看发现了个叔叔id=9，那这时候，4和9是并列的，也就是同辈儿的，那就先不着急再往3走，而是先去看看9有没有儿子，确实有一个，那按照靠右先执行，应该是10比9先执行。  
但是问题是，4跟9谁先执行呢？因为并列就是上面的先执行，所以是4先执行。然后又有人问了，那9跟3又是怎么个顺序呀，9跟4是兄弟，4是3儿子，也就是说9也是3儿子，当然是儿子先做收集结果集喽。那现在得到的顺序就成了 8-7-6-5-4-10-9-3-2-1-0。  
      然而，当找到id=3得时候，又发现，这个爸爸也有个同辈儿的，id=11，但是这个11没正好就是个光棍儿，没儿子。那就排出3比11先执行。后面2 1 0都没兄弟了，所以最终的执行顺序是8-7-6-5-4-10-9-3-11-2-1-0。  
     这个说起来很枯燥，读起来简单其实，现在我就说下重点，这个到底是怎么个实现的。  
      其实这就是个二叉树，这里没有把v$sql\_plan中的position跟parent\_id，还有树的深度截图出来，很不好意思，所以口述下这俩个的图形表现。就是他们在上面这张执行计划图片表现：同一个缩进度的就是position值一样的，只查一个缩进度的就是父子关系，也就是说id=3和id=11的parent\_id都是2。  
     现在我们来画一下这个二叉树，二叉树就是只有俩个叉，所以你不可见到说谁不遵守计划生育偏要整3个孩子的。  


     看，每个块块里的数字就是id。画图可以让我们正着看执行计划，按id画出来图，然后再去理出执行计划的执行顺序。  
     首先，第一个孩子都是往左边画，第二个孩子才是挨着往右边画，找其中典型的部分来说明下，3下面的孩子是4，但是发现仅比3少一个缩进的，或者说与4同缩进的还有9。那先发现的3所以，3画左边，9画右边。而9下面的有10，所以10画9下面的左边。  
那有人问了，照这么说，10跟5也是平行的，都比4少一个缩进，为啥不把10画4下面，因为紧挨着的是直系，往上找没有紧挨着的爸爸了才去找老王。所以就按照这个原则画，就画出了这个执行计划的本质。  
     然后，就是怎么遍历这棵树，后序遍历，别问为什么。后序遍历：先左后右再根。也就是说一直找到最左边的节点（找到第一个紧挨着的并行的亲兄弟的时候的最靠右的步骤，也就是这对亲兄弟靠上的哥哥，执行计划的入口），然后往右找（这就是找亲弟弟，这就是为啥靠上的先执行），  
没有就往上找（找爹，缩进靠右的步骤先执行的原因）。  
     最后，我们来看上面这副图，是不是8-7-6-5-4-10-9-3-11-2-1-0？！

### starts

      这个就是真实的这一个步骤的总的执行次数，你看上面所有步骤都只执行了一次。

### A-rows

     这个就是这一个步骤真实返回的行数，actual rows。也就是说，最上面id=0那一步的a-rows就是这条查询最总返回来的行数。

### A-time

      这一步真实执行的时间。最上面id=0那一步的a-time就是这条查询的真实执行时间。

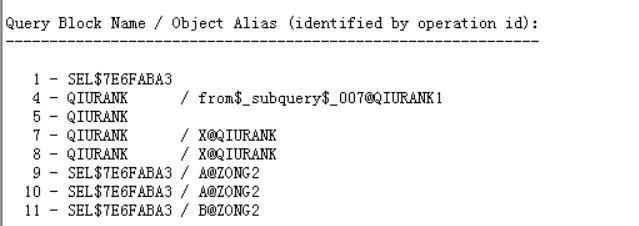
### buffers

      逻辑读，这里没显示出reads，但是还是说一下，reads就是物理读。

### 0mem，1mem

      这是说的hash join，sort，group by使用的PGA的内存大小，具体我还没验证清楚，好像0mem是用的PGA内存，1mem是用的硬盘空间。

## **执行计划的Qurey Block** Name



      所谓查询块，说白了就是select的个数。这部分内容，- 前面的数字是步骤id。  
      我在这里用/\*+ qb\_name() \*/把查询块名固定了，就是方便自己阅读，其实要是不用这个，oracle会自己给查询块取名，而且也挺好懂的，都是以SEL$开头的，后面跟个数字。举个例子，大家看10 - 这一行，A@zong2，我们回执行计划找一找，id=10走了一个索引。  
你可能上来就看执行计划的时候会说，诶我去，这个索引是谁的呀。这不qurey block这部分就告诉你了嘛！这查询块zong2上的a表的。这个zong2我要是取名儿，oracle会给它取名SEL$2,为啥？！因为简单的说，这是第二个select。

对了，这里我们发现一个在语句中并没有的别名，from$\_subquery$\_007。我们回步骤4找一下，发现是VIEW，再看看它是qiurank1块儿的，那就很明白了，这个就是整个qiurank1这个子查询整体的一个view，说白了就是这么一整块，它并不是一个真实存在再oracle中的视图，  
提醒大家一下，执行计划中别看见view就是视图。子查询不展开，也是被当view来对待的。  
     综上，从这部分内容中，可以发现具体操作的对象是谁的，是属于那个select的。

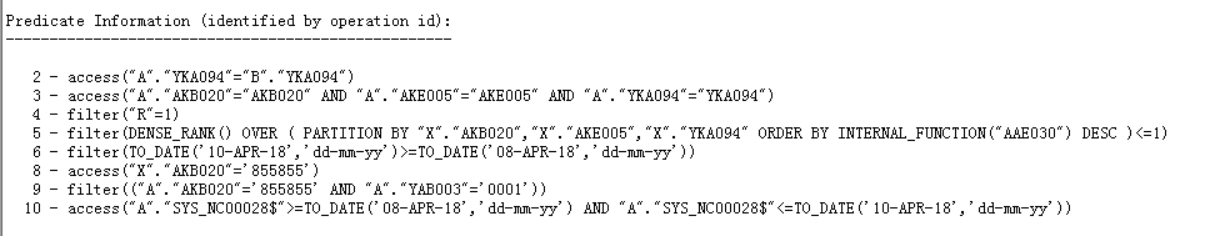
## 执行计划的OUTLINE部分



      是不是看到这个/\*+    \*/有点儿眼熟，没错，这就这条语句执行使用的hint，其实使用hint也就是来控制这部分。这部分其实是真的告诉你执行计划里面到底干了些什么，当然，有一些我还不太确定，比如merge(qba>qbb),no\_access,这些目前就只能大致猜。

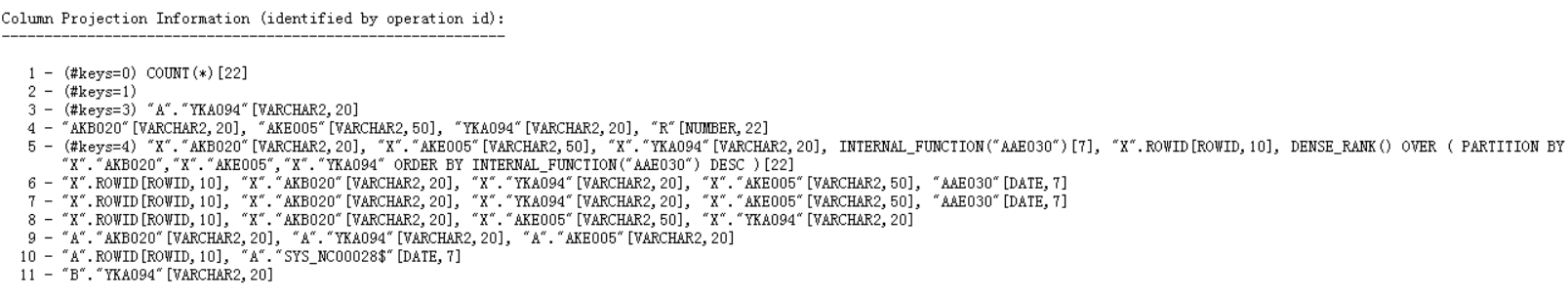
      那有人可能要问了，执行计划正文加上后面要介绍的谓词条件，不就已经告诉我们这条sql到底是怎么运作的吗？还看这个干嘛。关键就是，一般我们去关注执行计划，就是因为执行计划有问题了。所以，我们光知道执行计划是怎么个流程还不够，我们是不是还要知道怎么修改它，让它按照我们理想的姿势走正确的道路？！   
      就这个图片我举个例子，看到倒数第三行，index\_rs\_asc那一行，这一行明确告诉你有一步是走了索引，这一步就是走了qiurank块里的x表的...索引，而且这个索引是RS\_ASC：range scan升序的，说明走这个索引是正常从左往右走的而且还是个index range scan。  
     这一行下面，说了个啥？！是不是再说qiurank块里的x表索引扫描完了，然后用rowid回表是用batched这种多块扫描的方式回的表，自己可以去执行计划里面找找，就是id=8，7做的事情。还有往上看一行，leading那一行，这个hash join的顺序是不是跟执行计划里面体现的一毛一样，  
那我们把这里后面的做一个顺序调整，然后加入到sql的hint当中，就像这条sql的hint所做的那样，是不是就能轻易的把hash join的顺序进行调整，你说你调整A和B不用看这个outline，那你调整子查询那一整块儿在hash join中的顺序，你准备咋调呀？！当然，肯定有别的办法，但是这样做是不是就很方便，后面你调整好这块，然后在固定profile偷梁换柱的时候也方便呀。

## 执行计划的predicate information



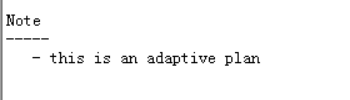
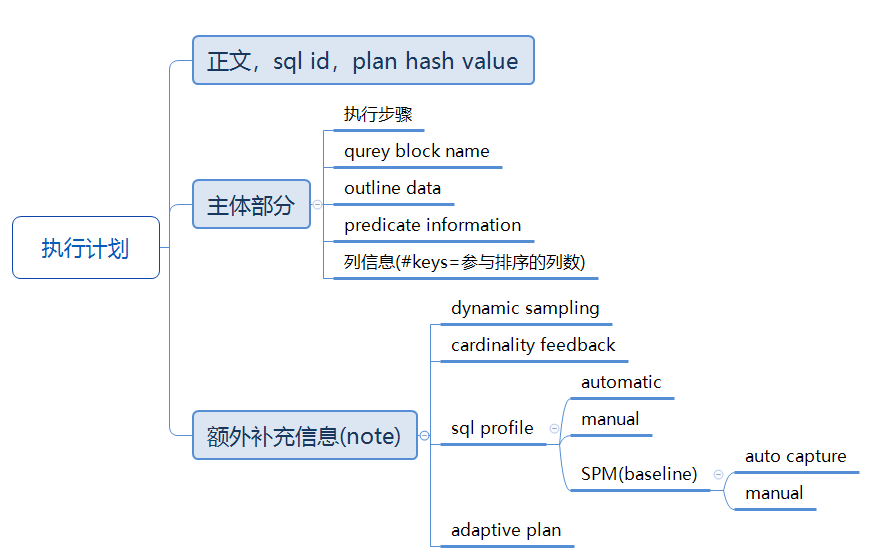
     这个正经没啥可说的，就很直白。前面的数字还是id，access代表用了索引，意思就是进入这个索引块跑了一圈儿。filter才是真正的过滤条件，也就是说这是回了表以后在表上的过滤条件。自己对这执行计划一步一步的看。  
    这里虽然很直白，但是很重要，如果你不看outline跟qurey block也就算了，这里你要是再不看，那读执行计划其实还挺费劲的，因为你要把所有涉及的表的列和索引的基本信息放在旁边，对着sql自己分析，就很累。而且有的时候，你觉得是这个索引走对了，比如说这条sql最后的条件，你要是有一边日期笔误写错了，比如多加了个空格，出来的执行计划还是一毛一样，但是predicate information就不一样了。所以说，如果能看到这部分，还是要好好看一下。

## 执行计划的column projection

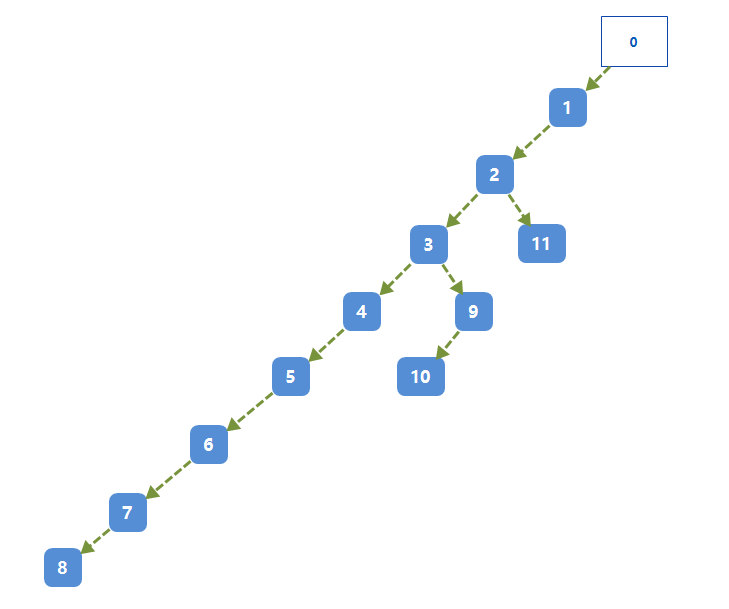


      这里我还是好好说一下，虽然内容不多，前面的数字还是id，keys就是说明这里有排序（这部分我后面学习明白了我再补充）。  
      这部分就是每一个步骤操作的列以及该列的数据类型。道理很简单，但是我们能学到一些东西，我们能通过这部分和执行计划正文了解到，每一步到底是在操作哪些列，我们看到你真正的排序操作，是在8 7 6一系列过滤后才开始在5进行  
分析函数的排序，在完成最后的形成一个整体view之前，8 7 6 5都是带着rowid在操作的。

## 执行计划的note

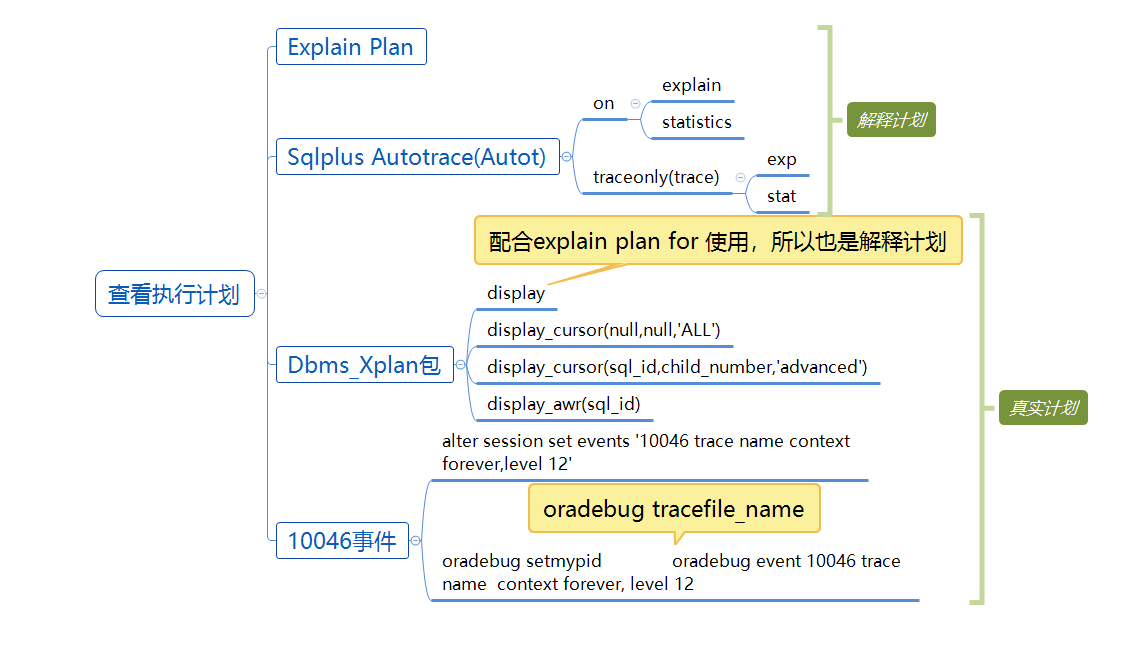
       
     这不多说了，直接上个我总结的的图。  


# 执行计划的阅读顺序

现在我们来解释如果确定每一步的执行顺序。  
      口诀随便都能找到，就是看缩进，从左往右看，不对齐的，就是靠右的先执行；到上下平行的或者说对齐的（紧挨着的），那靠上的就比靠下的先执行，然后再继续右看，看到最右边的时候就完事儿。其实这个看不太明白我说的的，可以结合一下百度。  
我直接说原理，当然也不是最底层实现的原理，那个现在还搞不来。  
      从左往右看，靠右的先执行是因为，一个缩进就代表了一对父子，儿子把自己的步骤得到的结果集整理好，上交给老子，就这么一回事儿。所以说0-8的执行顺序是8 7 6 5 4 3 2 1 0.（我这里又要道歉，我有点儿蠢了，没给大家选到那种有俩个  
紧挨着的儿子的执行计划，所以这里先说原理，大家这块再结合下百度）  
      然后当你在找爹的路上往回走的时候，发现到了id=4这一步，往下看发现了个叔叔id=9，那这时候，4和9是并列的，也就是同辈儿的，那就先不着急再往3走，而是先去看看9有没有儿子，确实有一个，那按照靠右先执行，应该是10比9先执行。  
但是问题是，4跟9谁先执行呢？因为并列就是上面的先执行，所以是4先执行。然后又有人问了，那9跟3又是怎么个顺序呀，9跟4是兄弟，4是3儿子，也就是说9也是3儿子，当然是儿子先做收集结果集喽。那现在得到的顺序就成了 8-7-6-5-4-10-9-3-2-1-0。  
      然而，当找到id=3得时候，又发现，这个爸爸也有个同辈儿的，id=11，但是这个11没正好就是个光棍儿，没儿子。那就排出3比11先执行。后面2 1 0都没兄弟了，所以最终的执行顺序是8-7-6-5-4-10-9-3-11-2-1-0。  
     这个说起来很枯燥，读起来简单其实，现在我就说下重点，这个到底是怎么个实现的。  
      其实这就是个二叉树，这里没有把v$sql\_plan中的position跟parent\_id，还有树的深度截图出来，在下面的视图介绍中会有说明，所以这里先口述下这俩个的图形表现。就是他们在上面这张执行计划图片表现：同一个缩进度的就是position值一样的，只查一个缩进度的就是父子关系，也就是说id=3和id=11的parent\_id都是2。  
     现在我们来画一下这个二叉树，二叉树就是只有俩个叉，所以你不可见到说谁不遵守计划生育偏要整3个孩子的。  


     看，每个块块里的数字就是id。画图可以让我们正着看执行计划，按id画出来图，然后再去理出执行计划的执行顺序。  
     首先，第一个孩子都是往左边画，第二个孩子才是挨着往右边画，找其中典型的部分来说明下，3下面的孩子是4，但是发现仅比3少一个缩进的，或者说与4同缩进的还有9。那先发现的3所以，3画左边，9画右边。而9下面的有10，所以10画9下面的左边。  
那有人问了，照这么说，10跟5也是平行的，都比4少一个缩进，为啥不把10画4下面，因为紧挨着的是直系，往上找没有紧挨着的爸爸了才去找老王。所以就按照这个原则画，就画出了这个执行计划的本质。  
     然后，就是怎么遍历这棵树，后序遍历。后序遍历：先左后右再根。也就是说一直找到最左边的节点（找到第一个紧挨着的并行的亲兄弟的时候的最靠右的步骤，也就是这对亲兄弟靠上的哥哥，执行计划的入口），然后往右找（这就是找亲弟弟，这就是为啥靠上的先执行），没有就往上找（找爹，缩进靠右的步骤先执行的原因）。  
     最后，我们来看上面这副图，是不是8-7-6-5-4-10-9-3-11-2-1-0？！

# 执行计划的获得方式



## explain plan for

把计划步骤写到PLAN\_TABLE$(on commit preserve rows)，dbms\_xplan.display把这张全局临时表中的内容格式化显示出来。

## autotrace

本质也是explain plan，所以就算语句执行了，但显示出来的还是解释计划，尤其对于绑定变量，极可能不准。

注意：set autot trace exp DML语言还是会真的执行。

## dbms\_xplan

display\_cursor要求shared\_pool里还有目标sql执行计划。

display\_awr要求awr快照包含目标sql，但是只有执行计划，没有谓词信息。

Format:

basic

typical

serial

all

advanced

adaptive(12c新特性)

allstats=iostats+memstats(PGA使用了多少，往硬盘写了多少字节，主要是对hash join,sort有用)---->能看到e-rows,A-rows(实际返回行数)

LAST (to see only the statistics for the last execution）

PEEKED\_BINDS （使用了绑定变量才会展现，展现绑定变量的具体值）

**常用查比较全面的执行计划命令如下：**

Alter session set statistics\_level=all;

select \* from table(dbms\_xplan.display\_cursor(null,null,'ADVANCED ALLSTATS LAST PEEKED\_BINDS'));

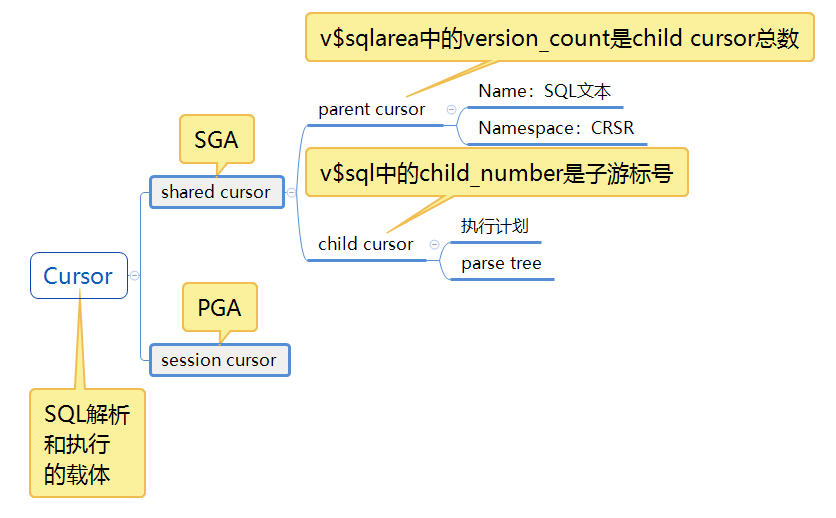
## 10046事件

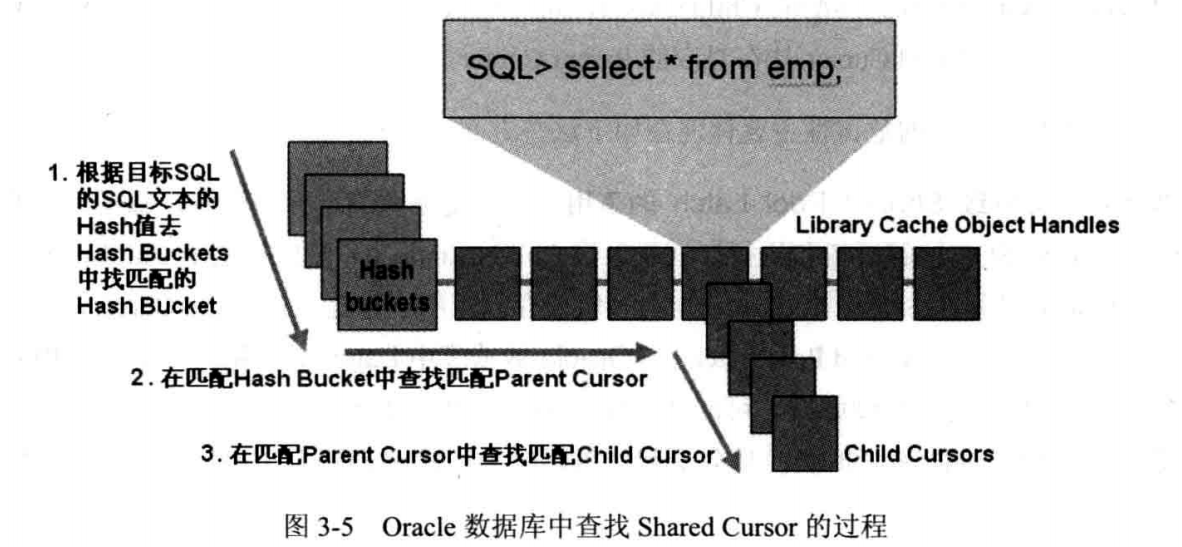
每一步的逻辑读(cr)、物理读(pr)、消耗时间、基数都有记录(card)，可用tkprof格式化日志后查看。

# 执行计划相关的视图

## shared cursor构成和查找方式

这里展示了shared cursor的构成和查找流程，主要是为了明确后面出现在视图中的parent\_id,child\_number是什么。





## 相关视图

### V$SQL\_PLAN\_STATISTICS\_ALL

address,hash\_value/sql\_id :parent cursor的句柄地址，以及其中的sql文本的哈希值/sql文本的ID，sql\_id等同于hash\_value

full\_plan\_hash\_value(新特性adptive)：该sql下由全部plan生成的一个hash值

plan\_hash\_value,child\_adress,child\_number：

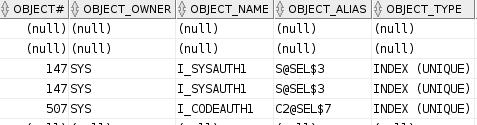
plan\_hash\_value是单一执行计划的hash值，包含该执行计划child cursor的地址和子游标号确定唯一child cursor

timestamp:计划生成的时间

operation---options:步骤执行的操作和对操作的描述，table access---FULL

object\_owner,object\_name,object\_type,object\_alias:

该步骤涉及的对象所在用户，对象名，对象类型，对象别名（执行计划的column projection）



optimizer:这个计划中优化器选择的模式 rule,choose,all rows,first rows

ID,parent\_id,postion,depth:

每一步的id，每一步的父id（靠右的先执行），同一个父id下的左右位置（并列靠上先执行），树的深度

这里在说一下执行计划的顺序原理所在，缩进就是父子关系，子步骤所得传递给父步骤，并列的子步骤依照position从左向右排排坐（一样的深度），遍历的时候先找最左边的叶子，即后序遍历

cost,cardinality,bytes:CBO估计的这一步的消耗，返回的结果集行数，产生的字节数

partition\_start,partition\_stop,partition\_id:对分区表范围访问时候涉及的开始的分区，结束的分区以及这俩个分区对应的id

cpu\_cost,io\_cost:CBO的cost评估的具体依据，cpu\_cost+io\_cost=cost

temp\_space:sort,hash join用到的temp表空间大小

access\_predicates,filter\_predicates:定位行的谓语和处理行前过滤行的谓语

projection,qblock\_name:被查询块的名字，俩个都是执行计划里的部分，advaced就能看到

executions:这个child cursor被执行的次数

disk\_reads，elapsed\_time：物理读次数，消耗时间（1/1e6s）

con\_id,con\_dbid：容器id，PDB的dbid

### V$SQL\_PLAN

### V$SQL\_PLAN\_STATISTICS

同上，没上面多

### V$SQL：

一个child cursor一行，每次执行完更新，长查询则5秒更新一次

sql\_text:varchar2(1000) 前一千个字儿

sql\_fulltext:clob 全部sql文本

sql\_id:sql标识

sql\_profile,sql\_plan\_baseline:用到的sql profile和baseline，如果有的话

SHARABLE\_MEM，PERSISTENT\_MEM，RUNTIME\_MEM：该child\_cursor总共用的，lifetime内用的，运行用的shared memory

\*\_wait\_time:应用、并发、集群、用户io等待时间

rows\_processed:最总结果集行数

child\_number

con\_id

is\_reoptimizable：下次执行触发了优化的话，这里就是y

### V$SQLAREA：

同上，一条sql一行

### V$SQLSTATS(同上) ：

也是一条sql一行，但是比上面俩个存的数据多，因为有些可能不在shared pool里的还可以在这里查到

### dba\_hist\_sql\_plan：

（V$SQL\_PLAN+V$DBA\_HIST\_SQLSTAT）

### dba\_hist\_sqlstat:

（The total value is the value of the statistics since instance startup.

The delta value(1/1e6s) is the value of the statistics from the BEGIN\_INTERVAL\_TIME to the END\_INTERVAL\_TIME in the DBA\_HIST\_SNAPSHOT view.

This view is used with the DBA\_HIST\_OPTIMIZER\_ENV, DBA\_HIST\_SQLTEXT, and DBA\_HIST\_SQL\_PLAN views to provide a complete picture of historical SQL statistics.）

dbid:database id,CDB的dbid

snap\_id,instance\_number:快照id（唯一），快照的instance number

module

version\_count:子游标的个数

PARSING\_SCHEMA\_NAME：最初导致产生这个子游标的用户名

sorts\_total,sorts\_delta:

自实例起累计为这个子游标而产生的排序，BEGIN\_INTERVAL\_TIME to the END\_INTERVAL\_TIME in the DBA\_HIST\_SNAPSHOT产生的排序

bind\_data

dbms\_sqltune.extract\_bind(bind\_data,position).VALUE\_string--->dba\_hist\_sqlbind(只能查到最近一次使用的绑定变量的值)

**剩下的列下面查询topsql的查询中有，义如其名**

### dba\_hist\_sql\_sqltext：

就是历史的sql文本，对应dbid，con\_id

### all\_source:

owner,name,type,line,text,origin\_con\_id：

对象所在用户，对象名，对象类型（存储过程呀，函数呀，各种），行号，每一行的文本，数据来源的con\_id

### dba\_hist\_optimizer\_env：

优化器环境变量设置

### dba\_hist\_snapshot：

dbid,snap\_id,instance\_number,con\_id

startup\_time:实例启动的时间

BEGIN\_INTERVAL\_TIME ,END\_INTERVAL\_TIME：快照的开始和结束时间，结束时间是真正的产生的快照时间点

### dba\_hist\_sql\_bind\_metadata：

结合dba\_hist\_sqlstat的bind\_data可以得到该sql使用过的所有绑定变量的值

name,datatype\_string:绑定变量的名字，数据类型

position:绑定变量在sql中的位置

### dba\_tables：

num\_rows(收集统计信息才有）

last\_analyzed

### dba\_indexes：

clustering\_factor

### dba\_ind\_columns：

### dba\_tab\_columns：

num\_distinct(现在也在[TAB|PART}\_COL\_STATISTICS这个视图里)

num\_nulls

num\_buckets(直方图里的桶数)

histogram

# 获取topsql

select a.\*,row\_number() over( order by els desc) as r\_els

,row\_number() over( order by elsp desc) as r\_elsp

,row\_number() over( order by phy desc) as r\_phy

,row\_number() over( order by phy desc) as r\_phyp

,row\_number() over( order by iow desc) as r\_iow

,row\_number() over( order by iow desc) as r\_iowp

,row\_number() over( order by exe desc) as r\_exe

,row\_number() over( order by rwo desc) as r\_rwo

,row\_number() over( order by rowp desc) as r\_rowp

,row\_number() over( order by iow desc) as r\_cpu

,row\_number() over( order by iow desc) as r\_cpup

,row\_number() over( order by iow desc) as r\_get

,row\_number() over( order by iow desc) as r\_getp

from (select /\*+ ordered\*/

min(to\_char(begin\_interval\_time, 'MMDD-HH24MI')) btime,

max(to\_char(begin\_interval\_time, 'MMDD-HH24MI')) etime,

min(sn.snap\_id) minsnap,

max(sn.snap\_id) maxsnap,

max(distinct PARSING\_SCHEMA\_NAME) USR,

max(distinct module) module,

plan\_hash\_value,

sql\_id,

sql.dbid,

con.name PDBNAME,

con.con\_id con\_id,

sql.instance\_number inst\_num,

sum(executions\_delta) exe,

round(sum(elapsed\_time\_delta ) / 1e6, 2) els,

round(sum(cpu\_time\_delta ) / 1e6, 2) cpu,

round(sum(iowait\_delta ) / 1e6, 2) iow,

sum(buffer\_gets\_delta) get,

sum(disk\_reads\_delta) phy,

sum(rows\_processed\_delta) RWO,

round(sum(elapsed\_time\_delta ) / greatest(sum(executions\_delta), 1) / 1e6,4) elsp,

round(sum(cpu\_time\_delta ) / greatest(sum(executions\_delta), 1) / 1e6, 4) cpup,

round(sum(iowait\_delta ) / greatest(sum(executions\_delta), 1) / 1e6, 4) iowp,

round(sum(buffer\_gets\_delta ) / greatest(sum(executions\_delta), 1), 2) getp,

round(sum(disk\_reads\_delta ) / greatest(sum(executions\_delta), 1), 2) phyp,

round(sum(rows\_processed\_delta ) / greatest(sum(executions\_delta), 1), 2) ROWP

from dba\_hist\_snapshot sn, dba\_hist\_sqlstat sql,v$containers con

where sn.snap\_id = sql.snap\_id

and sn.dbid = sql.dbid

and sn.instance\_number = sql.instance\_number

and sql.snap\_id between :minsnap and :maxsnap

and sql.dbid = :dbid

AND sql.parsing\_schema\_nAME not LIKE 'SYS%'

and sn.dbid = sql.dbid

and sql.plan\_hash\_value != 0

and sql.con\_id = con.con\_id

group by sql.dbid,

sql.plan\_hash\_value,

sql.sql\_id,

sql.instance\_number,

con.name,

con.con\_id) a