

# 基于案例学SQL优化第1周

从案例中推导SQL优化的总体思路与误区

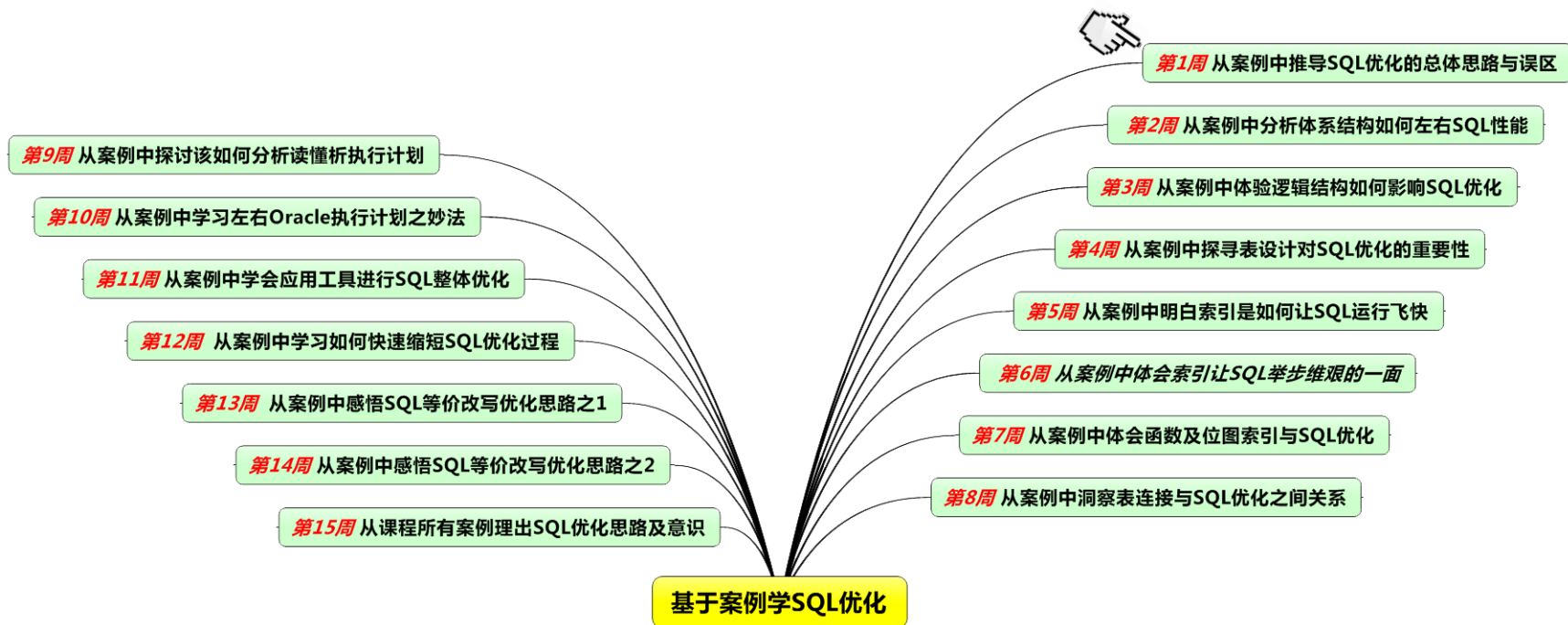
**【声明】** 本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

<http://edu.dataguru.cn>

梁敬彬，福富软件ITM产品线架构师及公司在聘数据库专家，ITPUB版主及社区专家，著有多本技术书籍，其新书《收获，不止Oracle》深受广大读者喜爱。





# 第1周课程总览之1



# 缺乏对讹传的辨知力(count讹传)



## 1. 缺乏对讹传的辨知力

记牢了，必须这么这么书写SQL！



COUNT(\*)与COUNT(列)的传言

到底是谁更快

终于明白谁快

谈SQL编写顺序之流言蜚语

表的连接顺序

表的条件顺序

IN 与 EXISTS 之争

10g


11g

.....

我抗议，这些规则不符合人性！

从产品的客户体验角度来思考  
用动手实验的方式来寻找真相

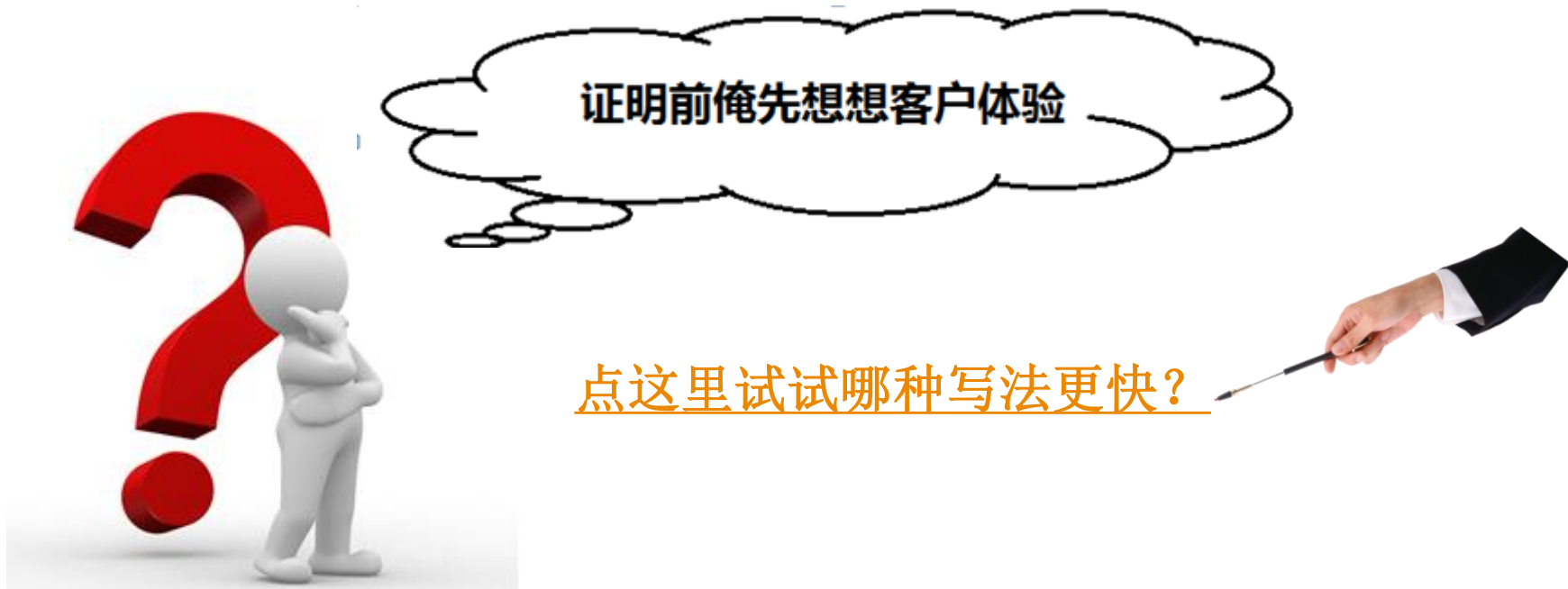
总结探讨：说某某SQL必须要这样写，不能那样写的，基本上都是错的！

- 
- 1. COUNT(\*)比COUNT(列)更慢！项目组必须用COUNT(列),不准用COUNT(\*), 谁用扣谁钱！
  - 2. COUNT(\*)用不到索引，COUNT(列)才能用到。
  - 3. COUNT(\*)是统计出全表的记录，是吞吐量的操作，肯定用不到索引。



**No no no !**这些都是谣言，转**500**次以上你是会有麻烦的！

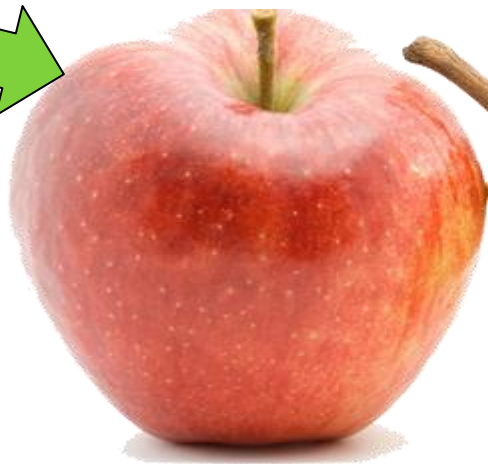
# COUNT对话正确与否的试验证明



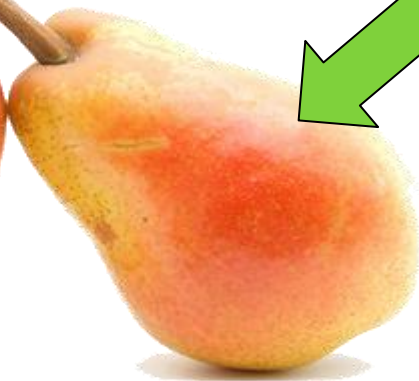
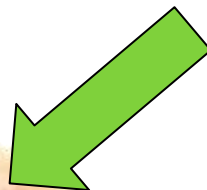


哪种写法更快？

COUNT(\*)



COUNT(列)



不等价谈何哪种写法更快呢？



COUNT每列都一样快吗？

[点这里了解更深入的信息！](#)



# 结论：更深入信息的结果图示



结论：原来优化器里的算法是这么玩的，列的偏移量决定性能，列越靠后，访问的开销越大。由于count(\*)的算法与列偏移量无关，所以**count(\*)最快，count(最后列)最慢。**

这个结论对我们开发设计，可是有启发哦，你要把不常访问的列，放在什么位置？

# 缺乏对讹传的辨知力(SQL编写顺序讹传)



## 1. 缺乏对讹传的辨知力

记牢了，必须这么这么书写SQL！

COUNT(\*)与COUNT(列)的传言

到底是谁更快

终于明白谁快

谈SQL编写顺序之流言蜚语

表的连接顺序

表的条件顺序

IN 与 EXISTS 之争

10g

11g

.....

我抗议，这些规则不符合人性！

从产品的客户体验角度来思考  
用动手实验的方式来寻找真相

总结探讨：说某某SQL必须要这样写，不能那样写的，基本上都是错的！

### 表的查询顺序(针对多表查询)

ORACLE的解析器按照从右到左的顺序处理FROM子句中的表名，因此FROM子句中写在最后的表(基础表 driving table)将被最先处理。在FROM子句中包含多个表的情况下，你必须**选择记录条数最少的表作为基础表**。当ORACLE处理多个表时，会运用排序及合并的方式连接它们。首先，扫描第一个表(FROM子句中最后的那个表)并对记录进行排序，然后扫描第二个表(FROM子句中最后第二个表)，最后将所有从第二个表中检索出的记录与第一个表中合适记录进行合并。

例如：表 TAB1 16,384 条记录

表 TAB2 1 条记录

选择TAB2作为基础表 **(最好的方法)**

select count(\*) from tab1,tab2 执行时间0.96秒

选择TAB2作为基础表 **(不佳的方法)**

select count(\*) from tab2,tab1 执行时间26.09秒

如果有3个以上的表连接查询，那就需要选择交叉表(intersection table)作为基础表，交叉表是指那个被其他表所引用的表。

?

## WHERE子句中的连接顺序

ORACLE采用自下而上的顺序解析WHERE子句,根据这个原理,当在WHERE子句中有多个表联接时, WHERE子句中排在最后的表应当是返回行数可能最少的表,有过滤条件的子句应放在WHERE子句中的最后。

如：设从emp表查到的数据比较少或该表的过滤条件比较确定，能大大缩小查询范围，则将最具有选择性部分放在WHERE子句中的最后：

```
select * from emp e,dept d
where d.deptno >10 and e.deptno =30;
```

如果dept表返回的记录数较多的话，上面的查询语句会比下面的查询语句响应快得多。

```
select * from emp e,dept d
where e.deptno =30 and d.deptno >10;
```

这些优化观点，  
你也传播出去  
了吗？

和表连接顺序有关的流言蜚语



与表条件顺序有关的以讹传讹



# 关于SQL书写顺序的试验结论

结论：基于RBO或许是如此，基于CBO时代，早就不是如此了，过时了！





# 缺乏对讹传的辨知力(in与exists之争)



## 1. 缺乏对讹传的辨知力

记牢了，必须这么这么书写SQL！



COUNT(\*)与COUNT(列)的传言

到底是谁更快

终于明白谁快

谈SQL编写顺序之流言蜚语

表的连接顺序

表的条件顺序

IN 与 EXISTS 之争

10g

11g

.....

我抗议，这些规则不符合人性！

从产品的客户体验角度来思考  
用动手实验的方式来寻找真相

总结探讨：说某某SQL必须要这样写，不能那样写的，基本上都是错的！

## 用NOT EXISTS替代NOT IN

在子查询中,NOT IN子句将执行一个内部的排序和合并. 无论在何种情况下,NOT IN都是最低效的 (因为它对子查询中的表执行了一个全表遍历). 使用NOT EXISTS子句可以有效地利用索引. 尽可能使用NOT EXISTS来代替NOT IN, 尽管二者都使用了NOT (不能使用索引而降低速度), NOT EXISTS要比NOT IN查询效率更高。

```
SELECT dname, deptno FROM dept WHERE
       deptno NOT IN (SELECT deptno FROM emp);
```

低效

```
SELECT dname, deptno FROM dept WHERE
       NOT EXISTS
       (SELECT deptno FROM emp WHERE dept.deptno =
        emp.deptno);
```

高效

第2个要比第1个的执行性能好很多。

因为1中对emp进行了full table scan,这是很浪费时间的操作。而且1中没有用到emp的index, 因为没有where子句。而2中的语句对emp进行的是缩小范围的查询。

不知道这些说法被转了多少次，我只能为你祈祷平安了！

# 关于IN与EXIST的试验证明(10g)



点这里看看10g环境的情况

```
select * from dept where deptno NOT IN ( select deptno from emp ) ;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	30	5 (0)	00:00:01
* 1	<b>FILTER</b>					
2	TABLE ACCESS FULL	DEPT	4	120	3 (0)	00:00:01
* 3	TABLE ACCESS FULL	EMP	13	169	2 (0)	00:00:01

```
select * from dept where not exists ( select deptno from emp where emp.deptno=dept.deptno) ;
```

0	SELECT STATEMENT		1	43	7 (15)	00:00:01
* 1	HASH JOIN <b>ANTI</b>		1	43	7 (15)	00:00:01
2	TABLE ACCESS FULL	DEPT	4	120	3 (0)	00:00:01
3	TABLE ACCESS FULL	EMP	14	182	3 (0)	00:00:01

```
select * from dept where deptno NOT IN ( select deptno from emp where deptno is not null) and deptno is not null;
```

0	SELECT STATEMENT		1	43	7 (15)	00:00:01
* 1	HASH JOIN <b>ANTI</b>		1	43	7 (15)	00:00:01
* 2	TABLE ACCESS FULL	DEPT	4	120	3 (0)	00:00:01
* 3	TABLE ACCESS FULL	EMP	14	182	3 (0)	00:00:01

# 关于IN与EXIST的试验证明(11g)

## 点这里瞧瞧11g环境的情况



```
select * from dept where deptno NOT IN ( select deptno from emp ) ;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		4	172	7 (15)	00:00:01
* 1	HASH JOIN <u>ANTI</u> NA		4	172	7 (15)	00:00:01
2	TABLE ACCESS FULL	DEPT	4	120	3 (0)	00:00:01
3	TABLE ACCESS FULL	EMP	14	182	3 (0)	00:00:01

```
select * from dept where not exists ( select deptno from emp where emp.deptno=dept.deptno) ;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		4	172	7 (15)	00:00:01
* 1	HASH JOIN <u>ANTI</u>		4	172	7 (15)	00:00:01
2	TABLE ACCESS FULL	DEPT	4	120	3 (0)	00:00:01
3	TABLE ACCESS FULL	EMP	14	182	3 (0)	00:00:01

```
SQL> select * from dept where deptno NOT IN ( select deptno from emp where deptno is not null) and deptno is not null;
```

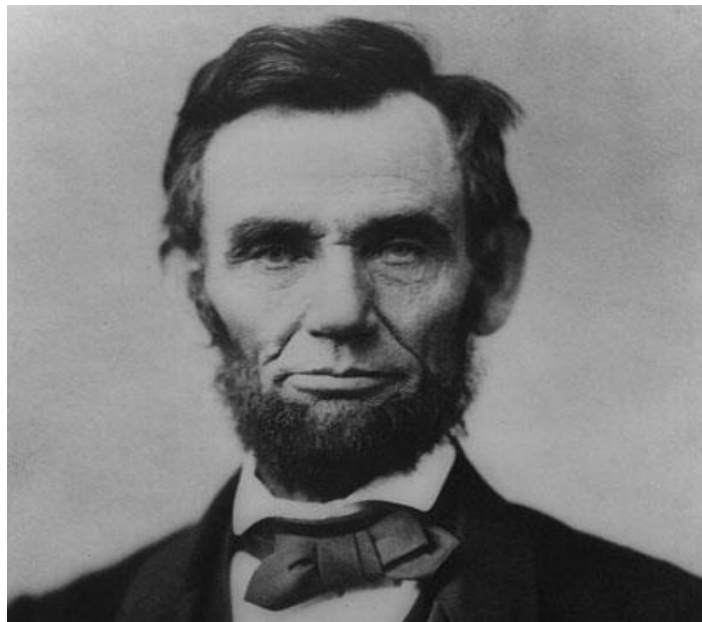
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		4	172	7 (15)	00:00:01
* 1	HASH JOIN <u>ANTI</u>		4	172	7 (15)	00:00:01
* 2	TABLE ACCESS FULL	DEPT	4	120	3 (0)	00:00:01
* 3	TABLE ACCESS FULL	EMP	14	182	3 (0)	00:00:01

结论:

一般来说, **anti**的反连接算法比**filter**更高效, 但是在10g时, Oracle的这个算法不完善, 必须要制定非空, 才可以让not in 用anti算法。

在11g的时候, 这个情况已经改变了, 无论not in 还是not exists, 无论是否列为空, 都可以走到Oracle比较先进高效的anti反连接算法。

“网上流传的名人名言**80%**以上都是假的”——亚拉伯罕·林肯



# 第1周课程总览之2





## 2. 不具备少做事的意识 M

你在设计中有考虑少做事吗？

全局临时表

自动清理数据

SESSION数据各自独立

分区表

历史数据管理机制

.....

你开发中有思考过少做事吗？

递归函数调用

避免SQL中的函数调用

减少SQL中的函数调用

集合写法

只取你所需的列

讲述催人泪下的SQL故事

.....

只是开发设计需要少做事吗？

某次数据迁移失败经历

.....

其实，你去买菜也会考虑

总结探讨：在保证能高质量完成任务的前提下尽量少做事的思想，一般都是对的。



# 不具备少做事意识（设计中的少做事）



1. 听听和全局临时表有关的故事

2. 讲述分区清理带给我们的好处

# 不具备少做事意识（设计中的少做事）

## 3. 体会分区消除带来的性能提升

```
SQL> select *
      2      from range_part_tab
      3      where deal_date >= TO_DATE('2012-09-04', 'YYYY-MM-DD')
      4            and deal_date <= TO_DATE('2012-09-07', 'YYYY-MM-DD');
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		1	2037	2 (0)	00:00:01		
1	PARTITION RANGE SINGLE		1	2037	2 (0)	00:00:01	9	9
* 2	TABLE ACCESS FULL	RANGE_PART_TAB	1	2037	2 (0)	00:00:01	9	9

```
SQL> select *
      2      from norm_tab
      3      where deal_date >= TO_DATE('2012-09-04', 'YYYY-MM-DD')
      4            and deal_date <= TO_DATE('2012-09-07', 'YYYY-MM-DD');
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		251	499K	1606 (1)	00:00:20
* 1	TABLE ACCESS FULL	NORM TAB	251	499K	1606 (1)	00:00:20



## 2. 不具备少做事的意识 M

你在设计中有考虑少做事吗？

全局临时表

自动清理数据

SESSION数据各自独立

分区表

历史数据管理机制

.....

你开发中有思考过少做事吗？

递归函数调用

避免SQL中的函数调用

减少SQL中的函数调用

集合写法

只取你所需的列

讲述催人泪下的SQL故事

.....

只是开发设计需要少做事吗？

某次数据迁移失败经历

.....

其实，你去买菜也会考虑

总结探讨：在保证能高质量完成任务的前提下尽量少做事的思想，一般都是对的。

# 不具备少做事意识（开发中的少做事）

请看与开发中少做事相关的案例说明！

## 1. 避免SQL中的函数调用有啥好处



```
SQL> select sex_id,
  2  first_name||' '||last_name full_name,
  3  get_sex_name(sex_id) gender
  4  from people;
```

已选择72821行。

已用时间: 00: 00: 07.68

统计信息

72821	recursive calls
0	db block gets
515027	consistent gets
0	physical reads

```
SQL> select p.sex_id,
  2  p.first_name||' '||p.last_name full_name,
  3  sex.name
  4  from people p, sex
  5  where sex.sex_id=p.sex_id;
```

已选择72821行。

已用时间: 00: 00: 02.30

统计信息

0	recursive calls
0	db block gets
5287	consistent gets
0	physical reads

# 不具备少做事意识（开发中的少做事）

## 2. 减少SQL中的函数调用有何思路

SQL> select name from (select rownum rn , f\_deal2(t1.object\_name) name from t1) where rn<=12;  
已选择12行。

已用时间: 00: 00: 03.49

统计信息

72824 recursive calls

0 db block gets

1043 consistent gets

0 physical reads

SQL> select f\_deal2(t1.object\_name) name from t1 where rownum<=12;

已选择12行。

已用时间: 00: 00: 00.00

统计信息

12 recursive calls

0 db block gets

5 consistent gets

0 physical reads

# 不具备少做事意识（开发中的少做事）

## 3. 集合写法能给性能提升多少



```
SQL> create table t ( x int );
```

写法1:

```
SQL> begin
2      for i in 1 .. 100000
3      loop
4          insert into t values (i);
5      end loop;
6      commit;
7  end;
8  /
```

PL/SQL 过程已成功完成。

已用时间: 00: 00: 04.00

```
SQL> drop table t purge;
```

表已删除。

已用时间: 00: 00: 00.03

```
SQL> create table t ( x int );
```

表已创建。

已用时间: 00: 00: 00.00

--写法2

```
SQL> insert into t select rownum from dual connect by level<=100000;
```

已创建100000行。

已用时间: 00: 00: 00.15

# 不具备少做事意识（开发中的少做事）

## 4. 只取你所需的列，访问视图变更快了

```
SQL> select * from v_t1_join_t2;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		8870	1039K	332 (1)	00:00:04
* 1	HASH JOIN		8870	1039K	332 (1)	00:00:04
2	TABLE ACCESS FULL	T2	8870	684K	40 (0)	00:00:01
3	TABLE ACCESS FULL	T1	69551	2784K	291 (1)	00:00:04

```
SQL> select object_id,object_name from v_t1_join_t2;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		8870	684K	40 (0)	00:00:01
* 1	TABLE ACCESS FULL	T2	8870	684K	40 (0)	00:00:01



# 不具备少做事意识（开发中的少做事）

## 5.只取你所需的列，索引读无需回表了

```
SQL> select object_id,object_type from t where object_id=28;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	24	2 (0)	00:00:01
* 1	INDEX RANGE SCAN	IDX_OBJECT_ID	1	24	2 (0)	00:00:01

```
SQL> select * from t where object_id=28;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	207	3 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T	1	207	3 (0)	00:00:01
* 2	INDEX RANGE SCAN	IDX_OBJECT_ID	1		2 (0)	00:00:01



# 不具备少做事意识（开发中的少做事）

## 6. 只取你所需的列，表连接访问提速了



```
SELECT /*+ leading(t2) use_merge(t1)*/ * FROM t1, t2 WHERE t1.id =t2.t1_id
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	lMem	Used-Mem
0	SELECT STATEMENT		1		100	00:00:00.13	1012			
1	MERGE JOIN		1	100	100	00:00:00.13	1012			
2	SORT JOIN		1	106K	101	00:00:00.13	1005	9266K	1184K	8236K (0)
3	TABLE ACCESS FULL	T2	1	106K	100K	00:00:00.02	1005			
* 4	SORT JOIN		101	100	100	00:00:00.01	7	11264	11264	10240 (0)

```
SELECT /*+ leading(t2) use_merge(t1)*/ t1.id FROM t1, t2 WHERE t1.id =t2.t1_id
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	lMem	Used-Mem
0	SELECT STATEMENT		1		100	00:00:00.08	1012			
1	MERGE JOIN		1	100	100	00:00:00.08	1012			
2	SORT JOIN		1	106K	101	00:00:00.08	1005	1895K	658K	1684K (0)

# 不具备少做事意识（开发中的少做事）

7. 催人泪下，拖垮生产系统的超长慢SQL

8. 出乎意料，SQL优化改写的飞跃性想法

9. 难以置信，让你不相信自己眼睛的SQL





## 2. 不具备少做事的意识 M

你在设计中有考虑少做事吗？

全局临时表

自动清理数据

SESSION数据各自独立

分区表

历史数据管理机制

.....

你开发中有思考过少做事吗？

递归函数调用

避免SQL中的函数调用

减少SQL中的函数调用

集合写法

只取你所需的列

讲述催人泪下的SQL故事

.....

只是开发设计需要少做事吗？

某次数据迁移失败经历

.....

其实，你去买菜也会考虑

总结探讨：在保证能高质量完成任务的前提下尽量少做事的思想，一般都是对的。


# 第1周课程总览之3





### 3. 不会依据场景选技术


住手，系统是你用还是大家用，当前是忙还是闲？

并行也会慢 

CACHE内存

.....


且慢，系统是读多还是写多，访问量是多还是少？

索引的坏处 

缓存结果集


物化视图

位图索引

BLOCK如何设计 

.....

稍等，先看表是大还是小，返回记录是多还是少？

分区也会慢 

绑定变量居然引发故障

.....

总结探讨：其实你很难像这两个案例那样，调优的如此爽快。

COUNT的性能大比拼 

从单车到飞船的优化之旅 



### 3. 不会依据场景选技术

住手，系统是你用还是大家用，当前是忙还是闲？

并行也会慢

CACHE内存

.....

且慢，系统是读多还是写多，访问量是多还是少？

索引的坏处

缓存结果集

物化视图

位图索引

BLOCK如何设计

.....

稍等，先看表是大还是小，返回记录是多还是少？

分区也会慢

绑定变量居然引发故障

.....

总结探讨：其实你很难像这两个案例那样，调优的如此爽快。

COUNT的性能大比拼

从单车到飞船的优化之旅

# 不会依据场景选择技术（关于索引坏处考虑）

## 1. 从某出账相关案例谈索引与更新



```
SQL> set timing on
SQL> create index idx_t_owner on t(owner);
已用时间: 00: 00: 00.35
SQL> create index idx_t_obj_name on t(object_name);
已用时间: 00: 00: 00.48
SQL> create index idx_t_data_obj_id on t(data_object_id);
已用时间: 00: 00: 00.21
SQL> create index idx_t_created on t(created);
已用时间: 00: 00: 00.34
SQL> create index idx_t_last_ddl on t(last_ddl_time);
已用时间: 00: 00: 00.36
SQL> insert into t select * from t;
已创建291280行。
已用时间: 00: 00: 15.03
```

```
SQL> insert into t select * from t;
已创建291280行。
已用时间: 00: 00: 02.03
SQL> create index idx_t_owner on t(owner);
已用时间: 00: 00: 00.67
SQL> create index idx_t_obj_name on t(object_name);
已用时间: 00: 00: 01.00
SQL> create index idx_t_data_obj_id on t(data_object_id);
已用时间: 00: 00: 00.45
SQL> create index idx_t_created on t(created);
已用时间: 00: 00: 00.72
SQL> create index idx_t_last_ddl on t(last_ddl_time);
已用时间: 00: 00: 00.74
```

$$0.35+0.48+0.21+0.34+0.36+15 > 2+0.67+1+0.45+0.72+0.74$$

# 不会依据场景选择技术（关于索引坏处考虑）

## 2. 建索引引发锁表带来的悲惨故事

## 3. 建索引导致排序引发的性能风波



```
SQL> select t1.name, t1.STATISTIC#, t2.VALUE
  2   from v$statname t1, v$mystat t2
  3   where t1.STATISTIC# = t2.STATISTIC#
  4   and t1.name like '%sort%';
```

NAME	STATISTIC#	VALUE
sorts (memory)	565	35
sorts (disk)	566	0
sorts (rows)	567	5170277

```
SQL> create index idx object id on t(object_id);
```

索引已创建。

```
SQL> select t1.name, t1.STATISTIC#, t2.VALUE
  2   from v$statname t1, v$mystat t2
  3   where t1.STATISTIC# = t2.STATISTIC#
  4   and t1.name like '%sort%';
```

NAME	STATISTIC#	VALUE
sorts (memory)	565	36
sorts (disk)	566	0
sorts (rows)	567	5243097





### 3. 不会依据场景选技术

住手，系统是你用还是大家用，当前是忙还是闲？

并行也会慢

CACHE内存

.....

且慢，系统是读多还是写多，访问量是多还是少？

索引的坏处

缓存结果集

物化视图

位图索引

BLOCK如何设计

.....

稍等，先看表是大还是小，返回记录是多还是少？

分区也会慢

绑定变量居然引发故障

.....

总结探讨：其实你很难像这两个案例那样，调优的如此爽快。

COUNT的性能大比拼

从单车到飞船的优化之旅

# 不会依据场景选择技术（说说分区更慢的场景）

还以为建分区一定会更快，真没想到.....



```
SQL> select * from part_tab where col2=8 ;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		1	39	13 (0)	00:00:01		
1	PARTITION RANGE ALL		1	39	13 (0)	00:00:01	1	11
2	TABLE ACCESS BY LOCAL INDEX ROWID	PART_TAB	1	39	13 (0)	00:00:01	1	11
* 3	INDEX RANGE SCAN	IDX_PAR_TAB_COL2	1		12 (0)	00:00:01	1	11

```
0 recursive calls
0 db block gets
24 consistent gets
```

```
SQL> select * from norm_tab where col2=8 ;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	39	2 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	NORM_TAB	1	39	2 (0)	00:00:01
* 2	INDEX RANGE SCAN	IDX_NOR_TAB_COL2	1		1 (0)	00:00:01

```
0 recursive calls
0 db block gets
4 consistent gets
```



### 3. 不会依据场景选技术

住手，系统是你用还是大家用，当前是忙还是闲？

并行也会慢

CACHE内存

.....

且慢，系统是读多还是写多，访问量是多还是少？

索引的坏处

缓存结果集

物化视图

位图索引

BLOCK如何设计

.....

稍等，先看表是大还是小，返回记录是多还是少？

分区也会慢

绑定变量居然引发故障

.....

总结探讨：其实你很难像这两个案例那样，调优的如此爽快。

COUNT的性能大比拼

从单车到飞船的优化之旅

# 不会依据场景选择技术（场景选择的经典案例）

统计条数语句之谁是速度之王



从单车到飞船的性能优化之旅


# 第1周课程总览之4



# 未考虑将需求最小化（催人泪下的SQL续集）

## 4. 未考虑将需求最小化

你能把复杂的需求抽象简单化吗？

催人泪下的SQL故事续集 

你接手的这些需求没有多余的吗？

这记录需要去重吗

这数据需要排序吗

这系统还需监控吗

总结探讨：需求最小化不仅能节省你的时间，还可以节省别人的时间。

# 请看SQL需求最小化的相关例子！

没想到，SQL故事的背后还引发更多故事.....




# 未考虑将需求最小化（考虑需求中多余之处）



## 4. 未考虑将需求最小化

你能把复杂的需求抽象简单化吗？

催人泪下的SQL故事续集 

你接手的这些需求没有多余的吗？

这记录需要去重吗

这数据需要排序吗

这系统还需监控吗



总结探讨：需求最小化不仅能节省你的时间，还可以节省别人的时间。



# 第1周课程总览之5



# 忽略SQL改造等价性（看似等价其实不等）



## 5. 忽略SQL改造等价性 M

看似等价的写法，其实不等价？

insert all 📄

select max(),min() 📄

in与范围 📄

count(\*)与count(列) 📄

看似不等价的写法，其实等价？

关于表是否有记录的判断 📄

总结讨论：不玄乎，等价不等价除了细心，还要关注需求。

# 请看SQL写法不等价的相关例子

## 1. Insert 多表插入的玄与机



```
insert all
  into ljb_tmp_transaction
  into ljb_tmp_session
select * from dba_objects;
```



```
insert into ljb_tmp_transaction as select *
from dba_objects;
insert into ljb_tmp_session as select *
from dba_objects;
```

# 请看SQL写法不等价的相关例子



## 2. max及min 写法的分与合

```
select min(object_id),
       max(object_id)
from t;
```

≠

```
select max(object_id) from t;
select min(object_id) from t;
```

```
select min(object_id),
       max(object_id)
from t;
```

=

```
select max, min
from (select max(object_id) max from t ) a,
     (select min(object_id) min from t ) b;
```

# 请看SQL写法不等价的相关例子

## 3. in和><写法之间的同与异

```
select /*+index(t,idx_object_id)*/ * from t where object_TYPE='TABLE' AND OBJECT_ID >= 20 AND OBJECT_ID <= 21;
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		2925	00:00:00.03	1103
1	TABLE ACCESS BY INDEX ROWID	T	1	2126	2925	00:00:00.03	1103
* 2	INDEX RANGE SCAN	IDX_OBJECT_ID	1	320	2925	00:00:00.02	730

```
select /*+index(t,idx_object_id)*/ * from t t where object_TYPE='TABLE' AND OBJECT_ID IN (20,21);
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		2925	00:00:00.01	588
1	INLIST ITERATOR		1		2925	00:00:00.01	588
2	TABLE ACCESS BY INDEX ROWID	T	2	2126	2925	00:00:00.01	588
* 3	INDEX RANGE SCAN	IDX_OBJECT_ID	2	1	2925	00:00:00.01	215

# 请看SQL写法不等价的相关例子



## 4. count 列和\*结论的对与错

```
SQL> drop table t purge;
```

表已删除。

```
SQL> create table t as select * from dba_objects;
```

表已创建。

```
SQL> update t set object_id =null where rownum<=2;
```

已更新2行。

```
SQL> set autotrace off
```

```
SQL> select count(*) from t;
```

```
COUNT(*)
```

```
-----
72822
```

```
SQL> select count(object_id) from t;
```

```
COUNT(OBJECT_ID)
```

```
-----
72819
```


# 忽略SQL改造等价性（看似不等价其实等价）



## 5. 忽略SQL改造等价性

看似等价的写法，其实不等价？

insert all 

select max(),min() 

in与范围 

count(\*)与count(列) 

看似不等价的写法，其实等价？

关于表是否有记录的判断 

总结讨论：不玄乎，等价不等价除了细心，还要关注需求。

# 看似不等价其实等价的例子

别买鱼了，就用冰箱里的牛肉来做美味晚餐吧。

```
begin
select count(*) into v_cnt from t1 ;
if v_cnt>0
then ...A逻辑...
else
then ...B逻辑...
End;
```

=

```
begin
select count(*) into v_cnt from t1 where rownum=1;
if v_cnt=1
then ...A逻辑...
else
then ...B逻辑...
End;
```

明白了吧？



# 第1周课程总览之6



# 不识需求乃顶级优化（需求优化）



## 6. 不识需求乃顶级优化

高效，需求优化需要我们不断的考虑

有趣，这些问答解决了很多性能问题

总结讨论：把握需求回归需求是性能优化之王

界面权限设计优化（并非所有操作都需做权限判定）

界面实时刷新改良(先显示汇总，再由用户点击展示)

单脚本对应多指标（不但多采集，还要去屏蔽规则）

表单展现的异步载入（界面内容自上而下逐步展现）

.....

这些排序确定真有必要吗？

此类历史数据要保留多久？

难道这个需求还没过时吗？

采集频率真要如此频繁吗？

.....

# 不识需求乃顶级优化 ( 相关问答)



## 6. 不识需求乃顶级优化

高效，需求优化需要我们不断的考虑

有趣，这些问答解决了很多性能问题

总结讨论：把握需求回归需求是性能优化之王

界面权限设计优化（并非所有操作都需做权限判定）

界面实时刷新改良(先显示汇总，再由用户点击展示)

单脚本对应多指标（不但多采集，还要去屏蔽规则）

表单展现的异步载入（界面内容自上而下逐步展现）

.....

这些排序确定真有必要吗？

此类历史数据要保留多久？

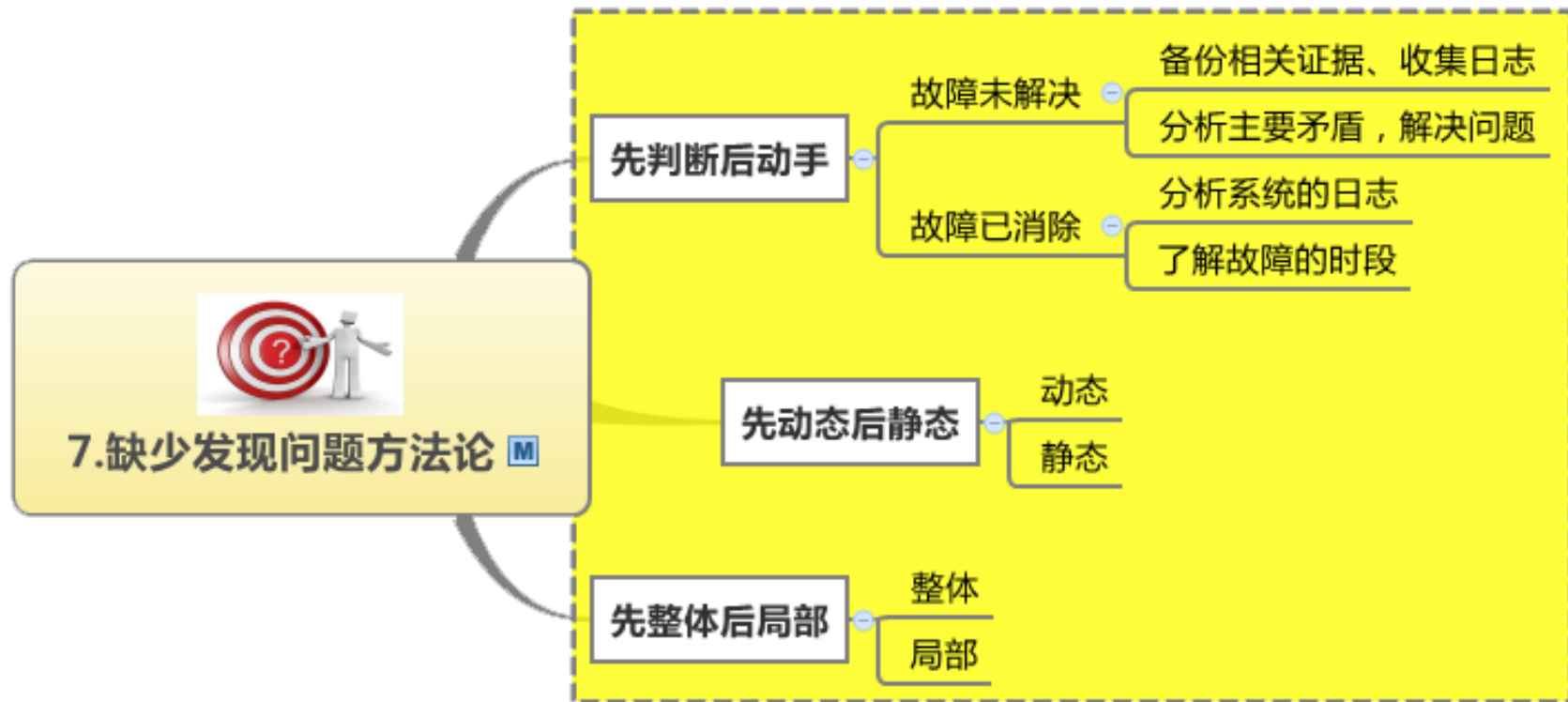
难道这个需求还没过时吗？

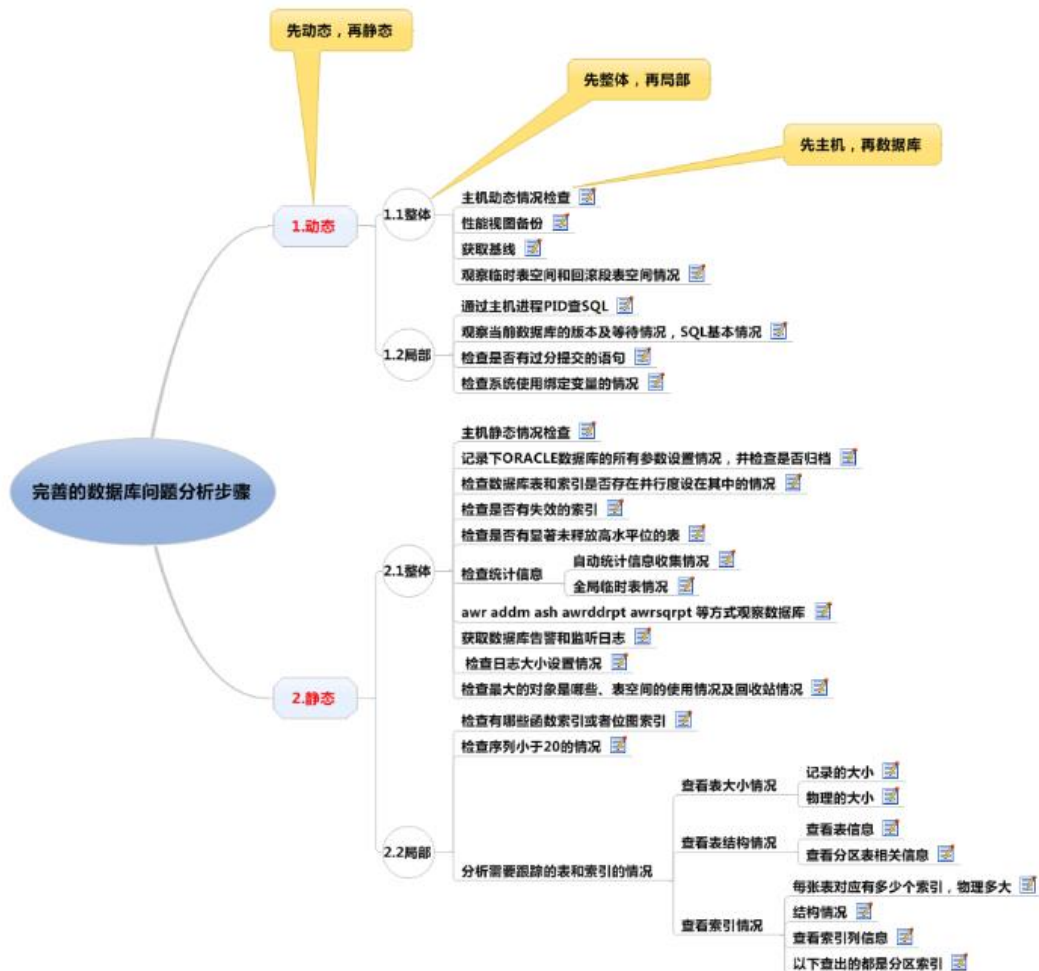
采集频率真要如此频繁吗？

.....

# 第1周课程总览之7







第一周课程到此结束，谢谢大家！



# 只上了“SQL误区”漏了“总体思路”吗

咦，老师，本周课程上完了？

嗯，是的，上完了，有收获吧？

谢谢，收获很大！不过您不是说总体思路和误区吗，怎么只说误区就结束了？

你想要听总体思路啊？

是啊！

你把误区都纠正了，总体思路不就出来了吗？

???.....



# SQL优化的总体思路

不就是这个吗，请看：



哦，我明白了，原来如此啊。

其实就是知识+意识，从下周开始，我们开始围绕各种案例，讲述优化中所涉及的各种知识和所具备的各种意识吧。本周的课程或许有些同学还无法完全理解明白，不过没关系，能有个印象就很好了，相信经过一段时间的学习后，回头再回顾老师的课堂视频，试验老师所提供的各种脚本，大家一定会豁然开朗的。

谢谢老师！

- Dataguru ( 炼数成金 ) 是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 <http://edu.dataguru.cn>

# Thanks

**FAQ时间**