DTCC

2017第八届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2017

SQL审核与经典案例细数 - Oracle的DevOps实战

罗海雄

Who Am I

罗海雄，网名： RollingPig

- 现任云和恩墨公司性能管理总监，负责Oracle数据库性能相关的产品及服务交付。
- 2012 ITPUB全国SQL大赛冠军得主
- ITPUB 论坛数据库管理版版主资深版主
- ChinaUnix 论坛 Oracle版资深版主
- 曾经服务于Oracle亚太区开发者计划部门和大型制造企业中芯国际，从开发到性能管理，有超过10年的企业级系统设计和优化经验。

1 案例1: .. Not In .. 的优化

案例1: .. Not In .. 的 优化 -- 问题

某保险客户，ETL 耗时数个小时，压力主要在其中一个SQL上。

@?/rdbms/admin/awrsqrpt.sql

Stat Name	Statement Total	Per Execution	% Snap Total
Elapsed Time (ms)	5,788,881	5,788,881.42	33.00
CPU Time (ms)	6,049,105	6,049,105.06	44.37
Executions	1		
Buffer Gets	105,294,814	105,294,814.00	57.05
Disk Reads	1,933	1,933.00	0.00
Parse Calls	583,899	583,899.00	95.34
Rows	213,632	213,632.00	
User I/O Wait Time (ms)	4,934		
Cluster Wait Time (ms)	156		
Application Wait Time (ms)	0		
Concurrency Wait Time (ms)	21,769		
Invalidations	0		
Version Count	4		
Sharable Mem(KB)	1,400		

单次执行时间：
5788(秒)
单次逻辑读：
10亿(块)
单次返回行数:
21万(行)

案例1: .. Not In .. 的 优化 --SQL Text

SQL很复杂，限于篇幅，不全贴

```
INSERT INTO /*+ APPEND parallel(4) */ APP_xxx_PREM
SELECT .... FROM TMP_xxx_FEE T
WHERE (
 T.ORGAN_ID, T.PRODUCT_ID, T.POLICY_TYPE, T.CHANNEL_TYPE, T.HEAD_BANK_ID,
T.CIRC_SELL_WAY, T.PAY_PERIOD, T.INSURANCE_PERIOD, T.
 CHARGE_PERIOD, T.CHARGE_YEAR, T.COVERAGE_PERIOD, T.COVERAGE_YEAR,
T.FEE_CODE ) NOT IN (
 SELECT ORGAN_ID, PRODUCT_ID, POLICY_TYPE, CHANNEL_TYPE, HEAD_BANK_ID,
CIRC_SELL_WAY, PAY_PERIOD, INSURANCE_PERIO D,
  CHARGE_PERIOD, CHARGE_YEAR, COVERAGE_PERIOD, COVERAGE_YEAR,
FEE_CODE
 FROM TMP_APP_xxx_PREM A
 WHERE A.MONTH_ID = :B6  AND A.HEAD_BANK_ID IS NOT NULL  AND
A.FINISH_TIME BETWEEN :B5  AND :B4 )
UNION ALL
。 。 。
UNION ALL
。 。 。
```

案例1: .. Not In .. 的 优化 --SQL Plan

SQL Plan也很复杂，限于篇幅，不全贴

FILTER	
PX COORDINATOR	
PX SEND QC (RANDOM)	:TQ70000
PX BLOCK ITERATOR	
TABLE ACCESS FULL	TMP_APP_FINANCE_PREM_FEE
PX COORDINATOR	
PX SEND QC (RANDOM)	:TQ10000
FILTER	
PX BLOCK ITERATOR	
TABLE ACCESS FULL	TMP_APP_FINANCE_PREM

案例1: .. Not In .. 的 优化 --致命 Filter

多年的经验告诉我，两个全表扫组成的Filter ，问题很严重, 因为涉及数据逐条处理。 而这个执行计划里，被驱动表还是全表扫。

Not In/In 操作有时候的确会产生 Filter 操作

但在11g版本中，优化器可以自动把Not in操作从昂贵的Filter转换成Null-Aware-Anti-Join

案例1: .. Not In .. 的 优化 --Null-Aware-Anti-Join

11g之前的版本，要把not in 语句转换成反连接，not in 条件的列必须有Not null 属性, 或者语句中带入了not null的限制，否则只能采用Filter，逐条过滤.

```
CREATE TABLE T_OBJ AS SELECT
OBJECT_ID,OWNER,OBJECT_NAME,OBJECT_TYPE  FROM DBA_OBJECTS
WHERE OWNER != 'SEROL';
=========================
CREATE TABLE T_TABLE AS SELECT OWNER,TABLE_NAME  FROM
DBA_TABLES WHERE OWNER!='SEROL';
```

```
SQL> DESC T_OBJ
 名称                                      是否为空? 类型
 ----------------------------------------- -------- ----------------------------
 OBJECT_ID                                          NUMBER
 OWNER                                              VARCHAR2(30)
 OBJECT_NAME                                        VARCHAR2(128)
```

案例1: .. Not In .. 的 优化 --Null-Aware-Anti-Join

伪装成10G的优化器。

```
SQL> alter session set optimizer_features_enable="10.2.0.5";
============================
SQL> set autotrace trace exp
SQL> SELECT * FROM T_TABLE WHERE TABLE_NAME NOT IN(SELECT OBJECT_NAME FROM T_OBJ);
```

Id	Operation	Name
0	SELECT STATEMENT	
* 1	FILTER	
2	TABLE ACCESS FULL	T_TABLE
* 3	TABLE ACCESS FULL	T_OBJ

案例1: .. Not In .. 的 优化 --Null-Aware-Anti-Join

加个Not null 条件或者栏位属性设为not null

```
SQL> alter table T_OBJ modify(OBJECT_NAME NOT NULL);
==============================
SQL> SELECT * FROM T_TABLE WHERE TABLE_NAME
NOT IN(SELECT OBJECT_NAME FROM T_OBJ
WHERE OBJECT_NAME IS NOT NULL);
```

Id	Operation	Name
0	SELECT STATEMENT	
* 1	HASH JOIN ANTI	
2	TABLE ACCESS FULL	T_TABLE
* 3	TABLE ACCESS FULL	T_OBJ

案例1: .. Not In .. 的 优化 --Null-Aware-Anti-Join

11g里，允许not in列没有not null 限制也可以转换Anti-Join.

```
SQL> alter session set optimizer_features_enable="11.2.0.4";
==========================================
SQL> alter table T_OBJ modify(OBJECT_NAME NULL);
==========================================
SQ>  SELECT * FROM T_TABLE  WHERE TABLE_NAME
NOT IN (SELECT OBJECT_NAME FROM T_OBJ);
```

Id	Operation	Name
	HASH JOIN ANTI NA	
2	TABLE ACCESS FULL	T_TABLE
3	TABLE ACCESS FULL	T_OBJ

案例1: .. Not In .. 的 优化 --Null-Aware-Anti-Join

这个特性, 可通过优化器参数控制

SQL>alter session set "_optimizer_null_aware_antijoin"=FALSE;

==========

SQL> SELECT * FROM T_TABLE WHERE TABLE_NAME NOT IN (SELECT OBJECT_NAME FROM T_OBJ);

Id	Operation	Name
0	SELECT STATEMENT	
* 1	HASH JOIN ANTI NA	
2	TABLE ACCESS FULL	T_TABLE
3	TABLE ACCESS FULL	T_OBJ

但经过验证，不是这个参数设置问题

案例1: .. Not In .. 的 优化 --改写

Not in 的逻辑，就是结果集之间的互斥，其实有多种改写的方式（区别在于not in 是会排斥空值）：

-- Not exists

-- Outer Join + is null

-- Minus

。 。 。

案例1: .. Not In .. 的 优化 --改写

```
INSERT /*+ APPEND parallel(4) */
INTO APP_XXX_PREM NOLOGGING
SELECT 。。。  FROM TMP_APP_XXX_PREM_FEE T
WHERE NOT EXISTS (SELECT 1 FROM TMP_APP_XXX_PREM A
WHERE T.PRODUCT_ID = A.PRODUCT_ID AND T.POLICY_TYPE = A.POLICY_TYPE
AND T.CHANNEL_TYPE = A.CHANNEL_TYPE AND T.HEAD_BANK_ID = A.HEAD_BANK
AND T.CIRC_SELL_WAY = A.CIRC_SELL_WAY AND T.PAY_PERIOD = A.PAY_PERIOD
AND T.INSURANCE_PERIOD = A.INSURANCE_PERIOD AND T.CHARGE_PERIOD = A.
AND T.CHARGE_YEAR = A.CHARGE_YEAR AND T.COVERAGE          .COVERAGE
AND T.FEE_CODE = A.FEE_CODE)
UNION ALL
```

奇迹发生了~~

Oh, no .... 语句报错了~~

```
ERROR at line 62:
ORA-00904: "A"."FEE_CODE": invalid identifier
```

案例1: .. Not In .. 的 优化 --再看看SQL Text

```
INSERT INTO /*+ APPEND parallel(4) */ APP_xxx_PREM
SELECT .... FROM TMP_xxx_FEE T
WHERE (
  T.ORGAN_ID, T.PRODUCT_ID, T.POLICY_TYPE, T.CHANNEL_TYPE, T.HEAD_BANK_ID,
T.CIRC_SELL_WAY, T.PAY_PERIOD, T.INSURANCE_PERIOD, T.
  CHARGE_PERIOD, T.CHARGE_YEAR, T.COVERAGE_PERIOD, T.COVERAGE_YEAR,
T.FEE_CODE ) NOT IN (
  SELECT ORGAN_ID, PRODUCT_ID, POLICY_TYPE, CHANNEL_TYPE, HEAD_BANK_ID,
CIRC_SELL_WAY, PAY_PERIOD, INSURANCE_PERIO D,
  C
A.F
  F
  WHERE A.MONTH_ID = :B6   AND A.HEAD_BANK_ID IS NOT NULL   AND
A.FINISH_TIME BETWEEN :B5   AND :B4 )
UNION ALL
。 。 。
UNION ALL
。 。 。
```

ORA-00904: "A"."FEE_CODE": invalid identifie

TMP_APP_xxx_PREM A 中并没有 FEE_CODE字段，
所以，Not in 无法自动改成Null Aware ANTI JOIN

案例1: .. Not In .. 的 优化 -- 提炼优化规则?

规则：注意执行计划中的带两个子操作的Filter

让我们的工具扫描SQL的执行计划...

规则：多表关联中，每个表应该有别名，且每个栏位应该写清楚表的别名

让我们的工具对SQL文本进行分析，找出不符合规则的

规则： 关注执行时间特别长的SQL

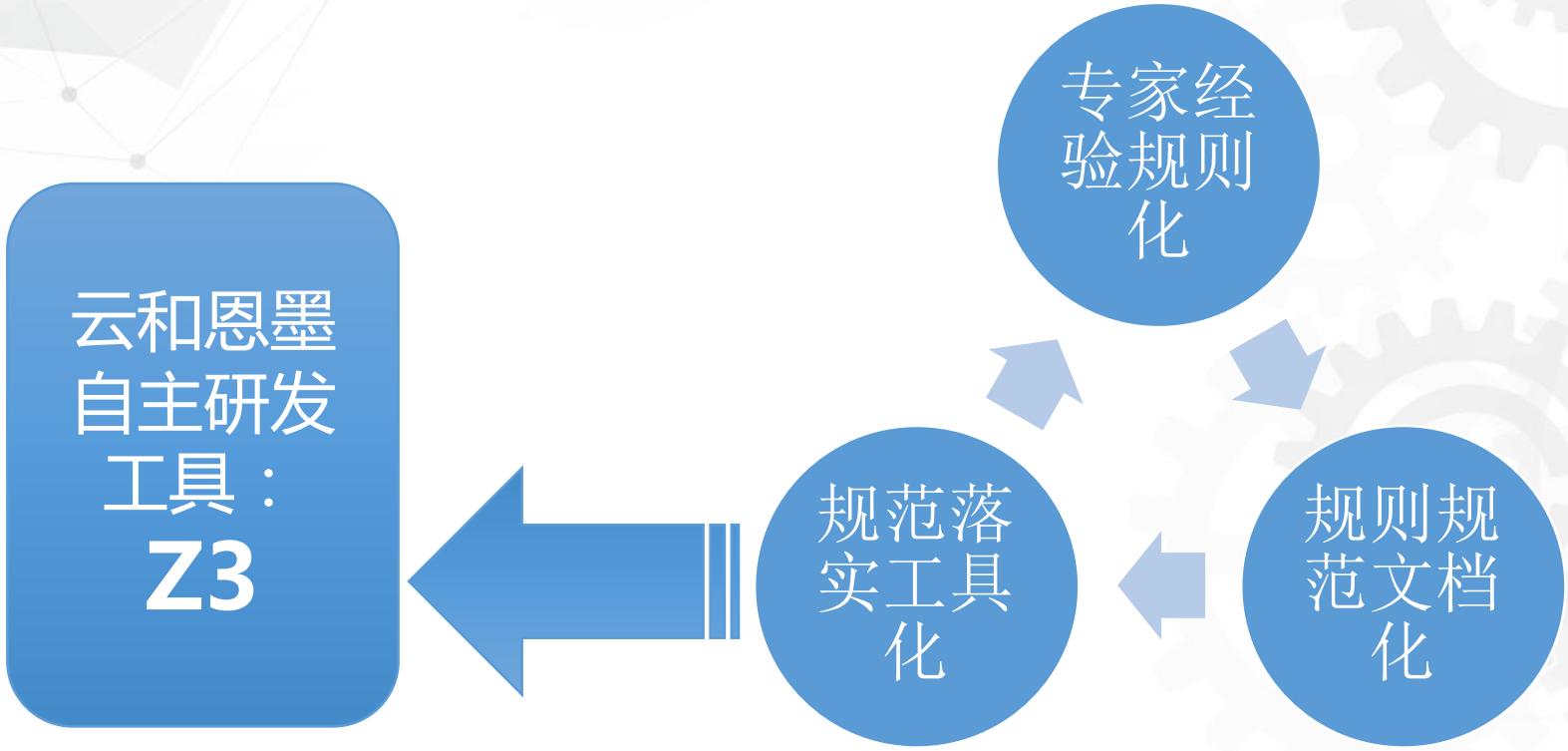
让我们的工具主动抓取执行时间超标的SQL

2 SQL审核工具：Z3 介绍

SQL 优化的DevOps -- SQL审核

同样的问题：

- ✓ 专家DBA一眼看出来
- ✓ 普通DBA花几个小时
- ✓ 开发人员把问题搁置,成为系统的定时炸弹



Z3是什么？


1. 配置Z3连接数据库与采集频率
2.Z3定时收集与分析SQL与表结构信息
网页版SQL审核工具
3.根据规则找出问题，形成审核报告
4. 根据审核结果对系统进行整改和提升

最重要的是规则 – 丰富规则

优化大师的日常积累

- 盖国强
- 杨廷琨
- 罗海雄
- 怀晓明
- 熊军
- 张乐奕
- 侯圣文
- …

行业客户的优化经验

- 通讯行业
- 金融行业
  - 保险
  - 银行
  - 互联网金融
- 制造业
- 互联网
- …

全方位覆盖

- SQL语句
- 执行计划
- 表
- 索引
- 序列
- …

最重要的是规则 – 灵活规则

可调参数	可调分值	自定义规则	模板化管理
• 表大小 • 绑定变量个数 • 统计信息时间 • …	• 区分严重程度 • 区分关注程度	• 预留接口 • 无需发布新版本 • 反应迅速	• OLTP • OLAP • 生产 • 测试 • 开发 • …

3 案例2：还是子查询

案例2: 还是子查询

```
SELECT


  FROM uop_act1.tf_b_batch_info a
 WHERE a.trade_time between to_date('20141101123000', 'y
       to_date('20141101123000', 'yyyymmddhh24miss')
   and a.trade_staff_id = 'E04Y0155'
   and a.trade_depart_id = '034b079'
   and a.trade_depart_id in
       (select a.depart_id
          from uop_act1.td_m_depart a,
               uop_act1.td_chl_kingdef b
         where a.depart_kind_code = b.chnl_kind_id
           and b.standard_kind_code like '2%')
```

案例2: 还是子查询

执行计划 (请关注下面红色标记部分)

ID	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		0	0
1	VIEW	VM_NWVW_2	1	547
2	HASH UNIQUE		1	150
3	FILTER		0	0
4	NESTED LOOPS		1	150
5	NESTED LOOPS		1	150
6	MERGE JOIN CARTESIAN		1	138
7	TABLE ACCESS BY INDEX RO...	TF_B_BATCH_INFO	1	126
8	INDEX RANGE SCAN	IDX_TF_B_BATCH_ST...	1	0
9	BUFFER SORT		21	252
10	TABLE ACCESS FULL	TD_CHL_KINDDEF	21	252
11	INDEX RANGE SCAN	PK_TD_M_DEPART	1	0
12	TABLE ACCESS BY GLOBAL INDE...	TD_M_DEPART	1	12

案例2: 还是子查询

```
SELECT


  FROM uop_act1.tf_b_batch_info a
 WHERE a.trade_time between to_date('20141101123000', 'y
       to_date('20141101123000', 'yyyymmddhh24miss')
   and a.trade_staff_id = 'E04Y0155'
   and a.trade_depart_id = '034b079'
   and a.trade_depart_id in
       (select a.depart_id
          from uop_act1.td_m_depart a,
               uop_act1.td_chl_kingdef b
         where a.depart_kind_code = b.chnl_kind_id
           and b.standard_kind_code like '2%')
```

子查询和父查询使用了相同的表别名，导致查询歧义，将造成结果集错误

案例2: 还是子查询

现有规则：注意执行计划中的笛卡尔积(Merge Join Cartesian)

让我们的工具扫描SQL的执行计划...

新的规则：多表关联中，每个表应该有别名，且每个表的别名必须不一样

让我们的工具对SQL文本进行分析，找出不符合规则的

4 案例3：数据库对象属性

案例3：数据库对象属性

- When & Where
  - 2016/01/01
  - 某保险客户开门红

- 发生了什么？

  高并发下，用户系统突然运行缓慢。

案例3：数据库对象属性

- 处理过程
  - 检查系统CPU内存 –正常
  - 检查session, 大量等待在 row cache lock

```
SQL> Select event,count(*) from v$session
  2  where status = 'ACTIVE' and wait_class!= 'Idle'
  3  group by event;

EVENT                                                            COUNT(*)
---------------------------------------------------------------- ----------
SQL*Net message to client                                                 1
row cache lock                                                           92
db file sequetial read                                                    2
```

```
SQL> select sql_id,count(*) from v$session
  2  where event =  'row cache lock'
  3  group by sql_id
  4  /

SQL_ID          count(*)
------------- ----------
3mmcwuyvjx2d9         92
```

案例3：数据库对象属性

- 处理过程
  - 检查当前SQL, 发现为sequence

```
SQL> select sql_text from v$sql where sql_id = '3mmcwuyvjx2d9';

SQL_TEXT
--------------------------------------------------------------------------------
select hibernate_sequence.nextval from dual
```

  - 人肉规则引擎启动.....
  - Sequence主要问题集中在cache 和 order选项

```
SQL> select SEQUENCE_NAME,ORDER_FLAG,CACHE_SIZE
  2  from dba_sequences where SEQUENCE_NAME=upper('hibernate_sequence')

SEQUENCE_NAME                  O CACHE_SIZE
------------------------------ - ----------
HIBERNATE_SEQUENCE             N          0
```

案例3：数据库对象属性

- 处理过程
  - 果然是cache=0搞的鬼
  - 迅速改为cache=100
  - 系统恢复正常

```
SQL> alter SEQUENCE hibernate_sequence cache 100;

Sequence altered.

SQL> select SEQUENCE_NAME,ORDER_FLAG,CACHE_SIZE
  2  from dba_sequences where SEQUENCE_NAME=upper('hibernate_sequence');

SEQUENCE_NAME                  O CACHE_SIZE
------------------------------ - ----------
HIBERNATE_SEQUENCE             N        100
```

案例3：数据库对象属性

规则：注意数据库里的序列，CACHE至少应为200(默认值20)

让我们的工具扫描数据库里的序列，以及其他可能造成性能问题的对象及其属性...

5 案例4：匪夷所思的CBO

案例4：匪夷所思的CBO

某运营商，刚刚做完小版本升级(11.2.0.3-11.2.0.4)

其中某个SQL突然性能变差很多

@?/rdbms/admin/awrsqrpt.sql

Stat Name	Statement Total	Per Execution
Elapsed Time (ms)	34,660,383	17,330,191.25
CPU Time (ms)	1,195,946	597,972.77
Executions	2	
Buffer Gets	27,267,471	13,633,735.50
Disk Reads	2,073,349	1,036,674.50
Parse Calls	2	1.00
Rows	0	0.00

执行时间:

5小时

逻辑读:

136亿块，100G

物理读:

10亿块, 8G

返回行数:

0

案例4：匪夷所思的CBO -- SQL Text

SQL挺简单，3个查询UNION ALL，其中两个带有not exists

```
SELECT nvl(round(sum(ALL_TIME) * 24 * 3600), 0)
FROM (
 SELECT SUM(FINISH_TIME - exec_time) ALL_TIME  FROM TI_C_XXX A
 WHERE A.exec_time >= ... AND A.exec_time < ...
  AND NOT EXISTS (SELECT 1   FROM Ti_C_YYYY B  WHERE B.Olcom_Work_Id = A.trade_id)
  AND A.STATE = '3'
 UNION ALL
 SELECT SUM(FINISH_TIME - exec_time) ALL_TIME FROM TI_CH_ZZZ A
 WHERE A.exec_time >= ... AND A.exec_time < ...
  AND NOT EXISTS (SELECT 1   FROM Ti_C_YYYY B   WHERE B.Olcom_Work_Id = A.trade_id)
  AND A.STATE = '3'
 UNION ALL
 SELECT SUM(AUTO_FINISH_TIME - exec_time) ALL_TIME
 FROM Ti_C_YYYY A
 WHERE A.exec_time >= ... AND A.exec_time < ...
  AND A.OLCOM_state = '3'
 )
```

案例4：匪夷所思的CBO -- 执行计划

但看执行计划没有明显问题，COST都很低

Execution Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT				5 (100)
1	SORT AGGREGATE		1	13	
2	VIEW		3	39	5 (0)
3	UNION-ALL				
4	SORT AGGREGATE		1	38	
5	FILTER				
6	PARTITION LIST ALL		1	38	1 (0)
7	TABLE ACCESS BY LOCAL INDEX ROWID	TI_C_OLCOMORDER	1	38	1 (0)
8	INDEX RANGE SCAN	IDX_TI_C_OLCOMORDER_2	1		1 (0)
9	INDEX RANGE SCAN	PK_TI_C_OLCOMWORKKJ	1	18	1 (0)
10	SORT AGGREGATE		1	40	
11	FILTER				
12	PARTITION LIST ALL		1	40	1 (0)
13	TABLE ACCESS BY LOCAL INDEX ROWID	TI_CH_OLCOMORDER	1	40	1 (0)
14	INDEX RANGE SCAN	IDX_TI_CH_ORDER_STATE	50		1 (0)
15	INDEX RANGE SCAN	PK_TI_C_OLCOMWORKKJ	1	18	1 (0)
16	SORT AGGREGATE		1	22	
17	TABLE ACCESS BY INDEX ROWID	TI_C_OLCOMWORKKJ	1	22	1 (0)
18	INDEX RANGE SCAN	IDX_TI_C_OLCOMWORKKJ_STATE	2		1 (0)

案例4：匪夷所思的CBO -- 统计信息？

COST 很低，执行很慢，绝大多数是统计信息不准确造成。

几个表的统计信息，都是几天内搜集的。

案例4：匪夷所思的CBO -- 抽丝剥茧

复杂问题简单化，把多个Union All 拆分小段来分析。
检查后发现问题在第二段。

```
SELECT SUM(FINISH_TIME - exec_time) ALL_TIME
FROM TI_CH_ZZZ A
 WHERE A.exec_time >= ... AND A.exec_time < ...


 AND A.STATE = '3'
```

问题并不在Not EXISTS

案例4：匪夷所思的CBO -- 可疑的rows

TI_CH_ZZZ表两个索引，分别是STATE/EXEC_TIME，运行中选择了STATE上的索引

Execution Plan

Id	Operation	Name	Rows	B
12	PARTITION LIST ALL		1	4
13	TABLE ACCESS BY LOCAL INDEX ROWID	TI_CH_OLCOMORDER	1	4
14	INDEX RANGE SCAN	IDX_TI_CH_ORDER_STATE	50	

STATE索引，CBO评估为 50 rows.

STATE	COUNT(*)
1	898,000
3	160,738,000
4	1,434,000

STATE=3的占了绝大多数记录

案例4：匪夷所思的CBO -- 10053

做个10053看看

```
Table Stats::
  Table: TI_CH_OLCOMORDER  Alias: TI_CH_OLCOMORDER  (Using
    #Rows: 164585900  #Blks:  17570473  AvgRowLen:  376.00
```

```
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for TI_CH_OLCOMORDER[TI_CH_OLCOMORDER]
  Column (#5):
    NewDensity:0.000000, OldDensity:0.000000 BktCnt:1645859, PopBktCnt:164585
  Column (#5): STATE(
    AvgLen: 2 NDV: 5 Nulls: 0 Density: 0.000000
    Histogram: Freq  #Bkts: 5  UncompBkts: 1645859  EndPtVals: 5
  Table: TI_CH_OLCOMORDER  Alias: TI_CH_OLCOMORDER
    Card: Original: 164585900.000000  Rounded: 50  Computed: 50.00  Non Adjust
```

案例4：匪夷所思的CBO --直方图EPV转换

Number: 基本不变，最多在精度上做调整

Date: 和公元前4712年1月1日的天数之差，支持非整数天

Select to_date('-4712-01-01','sYYYY-MM-DD')
+ endpoint_value
From dba_histograms where …

案例4：匪夷所思的CBO --直方图EPV转换

char/varchar2: 16进制前14位的RAW格式，为了避免乱码，有时只取前12位。

```
Select utl_raw.cast_to_varchar2(substr(lpad
(to_char(endpoint_value,'fmxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxx'),30,'0'),1,14)) …
```

Nchar/Nvarchar2: 16进制前16位的RAW格式

```
utl_raw.cast_to_nvarchar2(…,16))
```

案例4：匪夷所思的CBO --直方图EPV转换

检查直方图，看看值是否正常

```
select endpoint_number epn,
substr(to_char(ENDPOINT_VALUE,rpad('fm',38,'9'),1,14) EPV,
utl_raw.cast_to_varchar2(substr(lpad(to_char(
endpoint_value, rpad('fm',38 ,'x') ),30,'0'),1,12)) EPV_char
From dba_histograms where ....
```

ENDPOINT_NUMBER	EP_RAW	EP_CHAR
1	302020202020	0
448	312020202020	1
1645161	332020202020	3
1645854	342020202020	4
1645859	382020202020	8

案例4：匪夷所思的CBO -- 空格之谜

细看这段16进制值 332020202020
33='3'
20=' '　空格
所以这个值是 '3　　' (跟着5个空格)
和想象中的'3' 有点不一致。

案例4：匪夷所思的CBO -- 空格之谜

数据库是升级过来，而且，该栏位是char类型。
找了两个数据库，分别是 11.2.0.4/11.2.0.3.

```
Create table test_char_epv(id number,state char(1)) ;
==============================
Insert into test_char_epv
Select rownum , substr(to_char(rownum),1,1)) from
dual connect by level < 1000;
==============================
Analyze table test_char_epv compute statistics for all
columns ;
```

案例4：匪夷所思的CBO -- 空格之谜

检查结果

EP_RAW_11_2_0_3	EP_RAW_11_2_0_4	EP_HEX_11_2_0_3	EP_HEX_11_2_0_4
11.2.0.3		11.2.0.4	
312020202020		310000000000	
1<补齐空格	>	原始值	

案例4：匪夷所思的CBO --算法改变

几经周折，发现问题的根源是这样的:
Oracle 11.2.0.3 之前，char/nchar 在计算直方图的EPV时，都是补全到7位空格的。

后来发现一个BUG:15898932 , 补全空格的方式可能导致一些直方图计算不准确，于是就改变了算法，改成不补全空格。

最终产生不一致: 11.2.0.3 取得值 !=11.2.0.4取得值

案例4：匪夷所思的CBO -- 解决

把算法改回原来的

alter system set "_fix_control" = '12555499:0';

重新搜集可能存在问题的表的统计信息

.. dba_tab_columns(data_type = Char/Nchar) …

升级前预打Patch:18255105

6 案例n-m：成功客户

江苏移动：SQL审核保障性能0故障

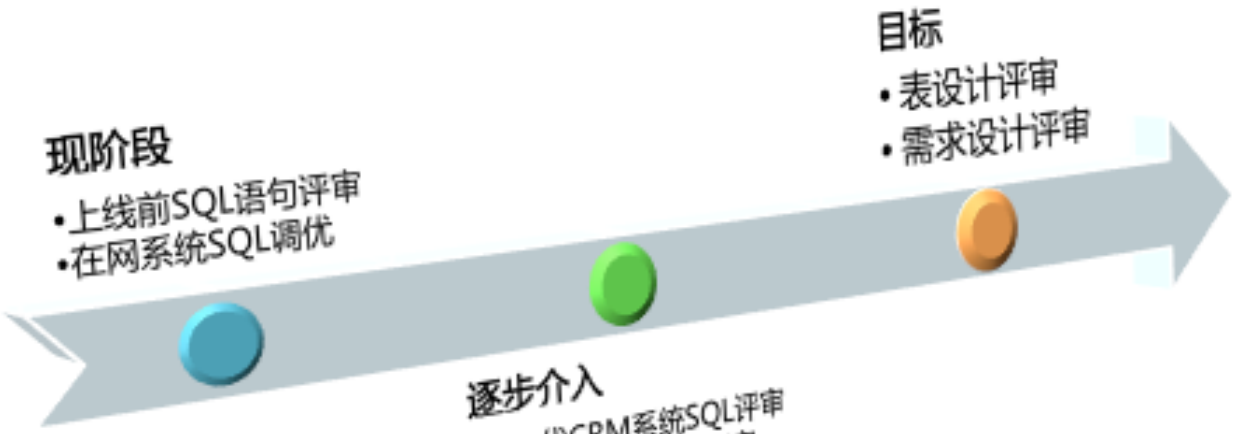
- **新业务上线**- 40次大上线，确保近800个新SQL平稳上线，无一出现性能问题
- **工具** -Z3发现近100个TOP SQL,主动优化、消除了现网系统中性能瓶颈
- **主机资源**-CPU、IO保持平稳，保证了在业务高峰期、账期有充足的硬件资源
- **ORACLE数据库**-除日常维护之外数据库零停机
- **重大调整**-调整前严密的测试、调整中二线专家保障、调整后实时跟踪

其实在生产中，绝大多数Oracle的业务系统出现问题都是SQL导致的。但是大多DBA，尤其是偏运维的DBA对SQL并不擅长，这些DBA承担着数据库运维和维护稳定性的职责，而他们对这些问题可能又无能为力。原本SQL的质量应该是开发层负责的问题，但目前的现状是，开发人员管不了，运维人员不擅长。所以当系统出现问题的时候，就需要专业人员“救火”，而事发或事后救火往往是业务已经遭受了损失。

- 江苏移动资深专家 戴建东

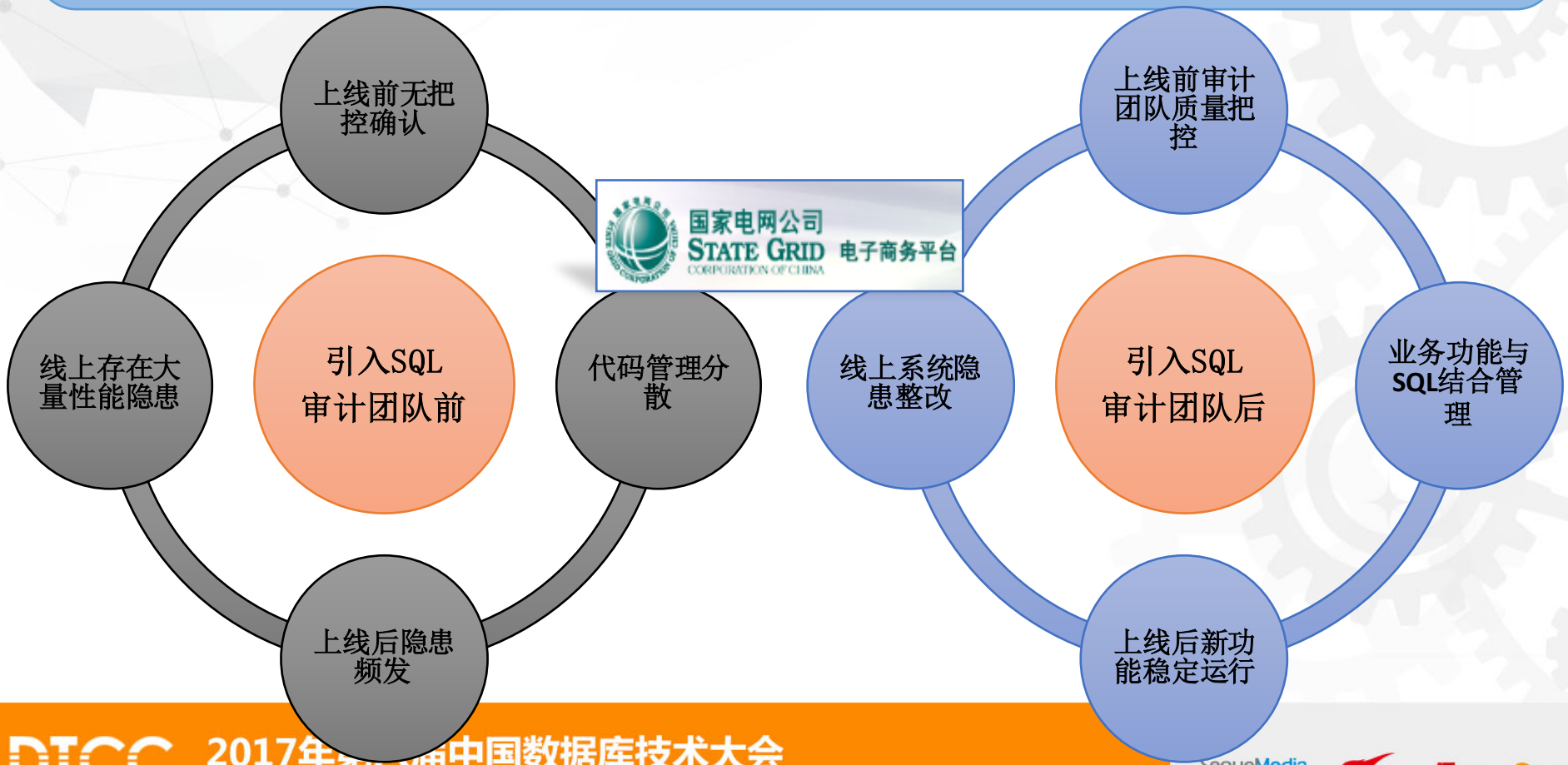
江苏移动：SQL审核保障性能0故障

- **新业务上线**- 40次大上线，确保近800个新SQL平稳上线，无一出现性能问题
- **工具** -Z3发现近100个TOP SQL,主动优化、消除了现网系统中性能瓶颈
- **主机资源**-CPU、IO保持平稳，保证了在业务高峰期、账期有充足的硬件资源
- **ORACLE数据库**-除日常维护之外数据库零停机
- **重大调整**-调整前严密的测试、调整中二线



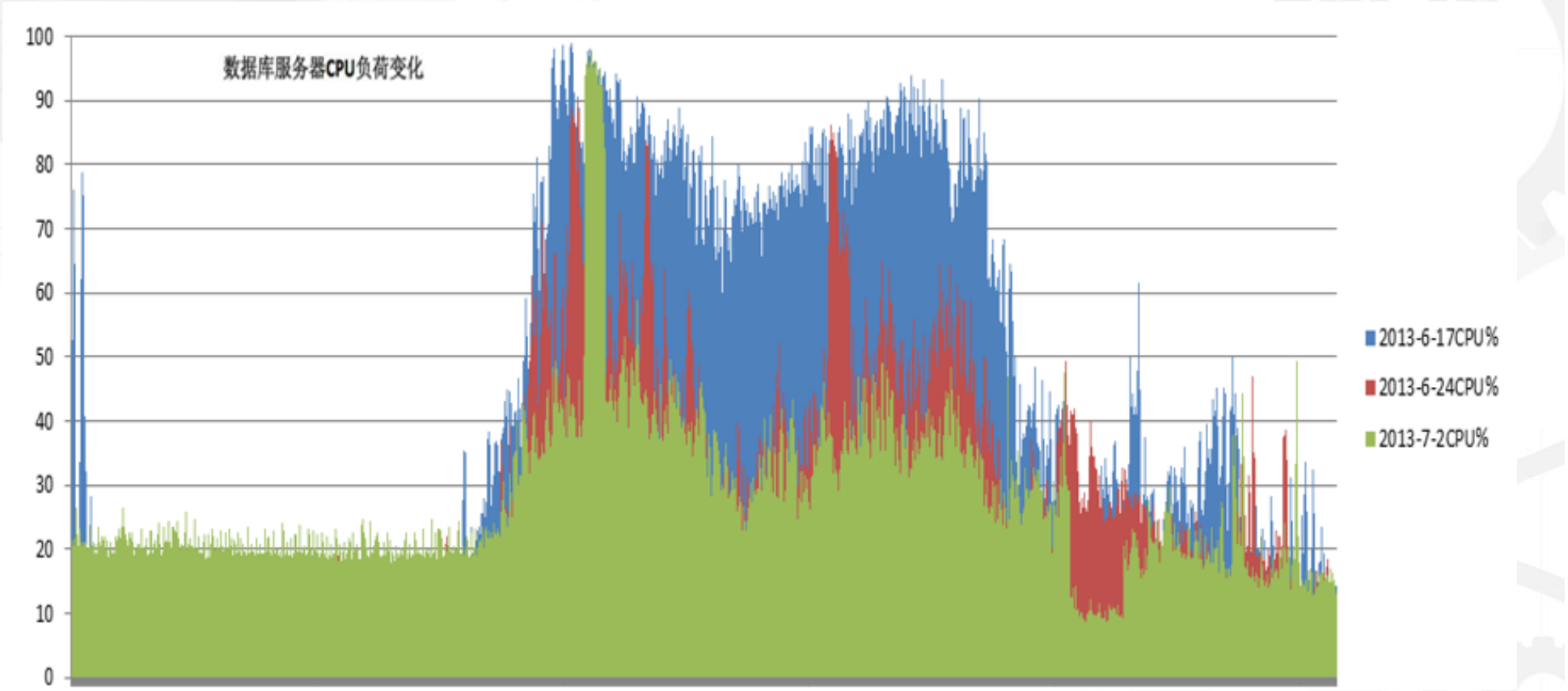
国家电网SQL审核

国家电网电子商务平台是一级部署平台，支撑全国网超过1万亿的物资招标采购项目，并提供相关物资类多种服务。
电子商务平台整体开发团队约80人左右，各业务模块功能点总和超过2000个。经过评估，云和恩墨为其定制了3人的SQL审计团队，进行业务系统SQL代码的审核把控。

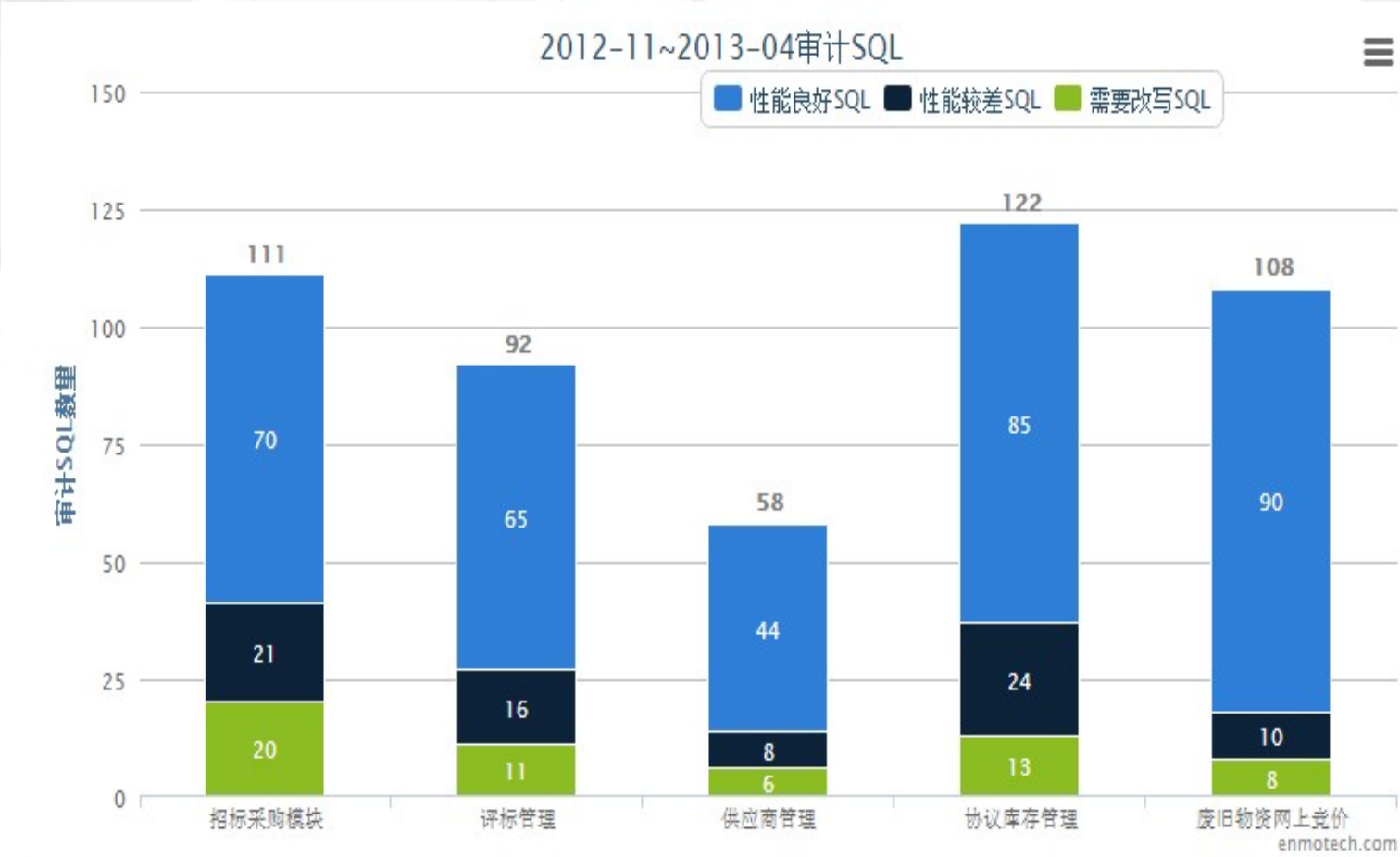


国家电网SQL审核

2013年6月至7月，经过不断优化，三个时间节点的CPU负载情况对比



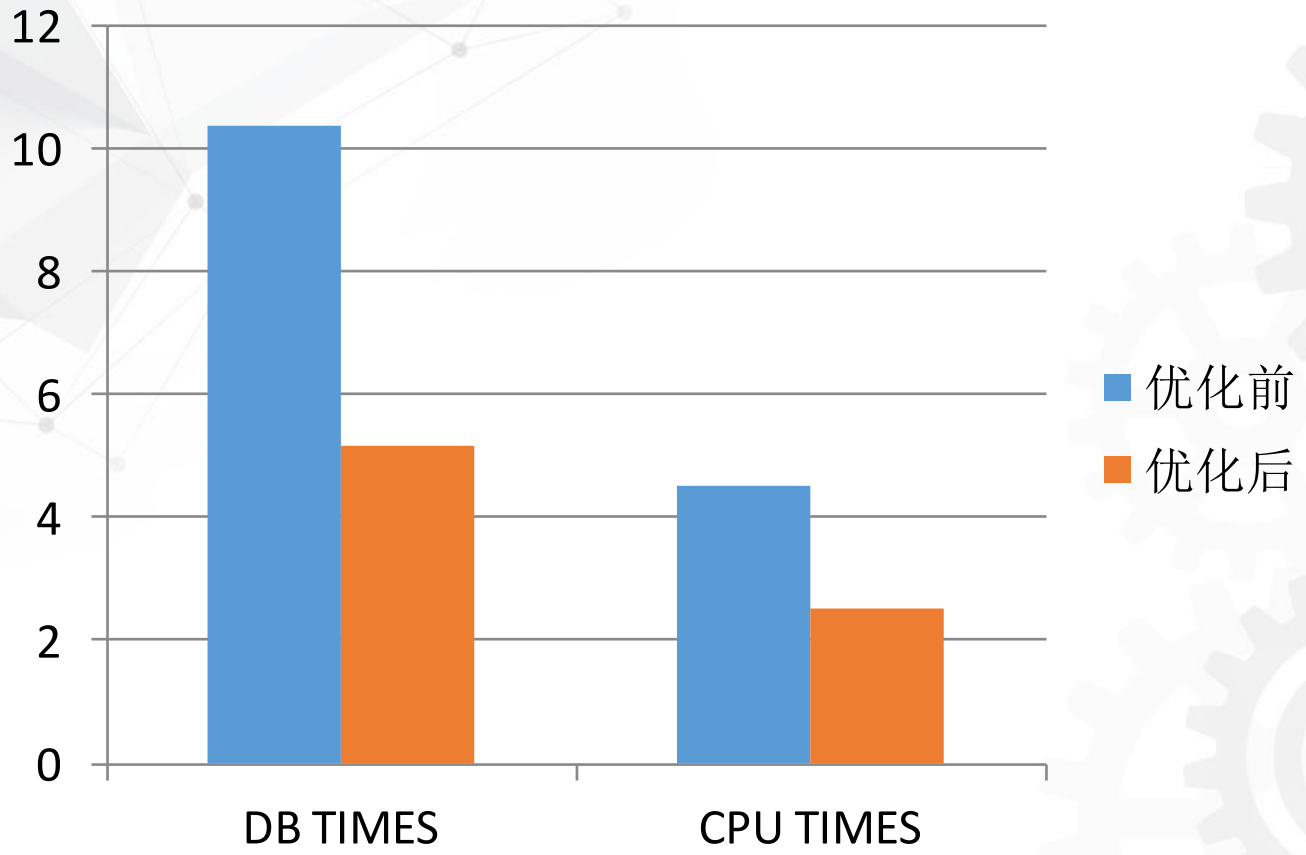
国家电网SQL审核


2012-11~2013-04审计SQL
性能良好SQL
性能较差SQL
需要改写SQL
审计SQL数量
150
125
100
75
50
25
0
111
70
21
20
92
65
16
11
58
44
8
6
122
85
24
13
108
90
10
8
招标采购模块
评标管理
供应商管理
协议库存管理
废旧物资网上竞价
enmotech.com

华北某财险公司系统

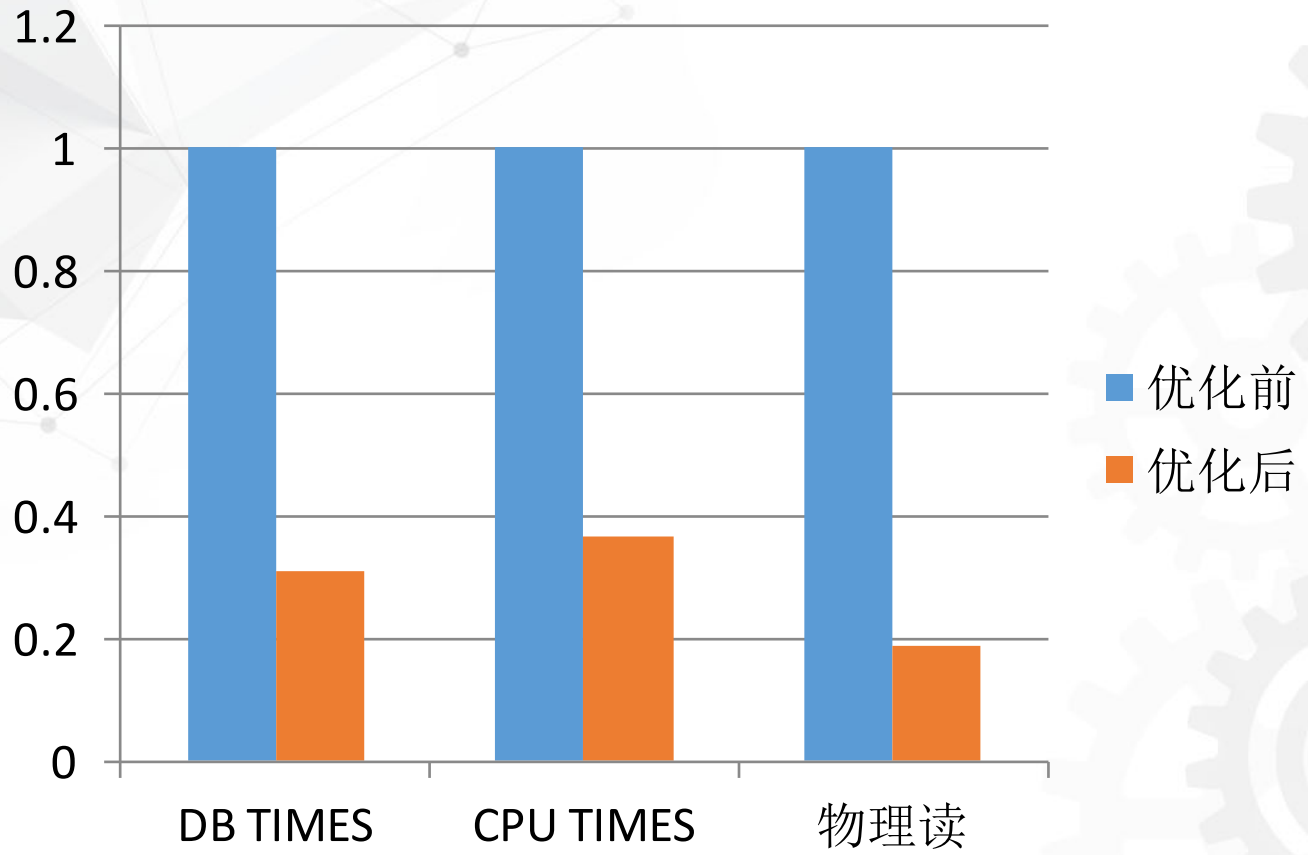
- 不仅仅是提高了系统响应速度
  - 经过我方优化，虚拟化环境下的两节点的RAC，**CPU从每节点12颗直接降低到每节点8颗**，大大节约了宝贵的硬件资源。
  - 而这部分节省出的资源，客户又将其划分给其他系统使用，无异于节省了昂贵的硬件成本。

华北某财险公司系统



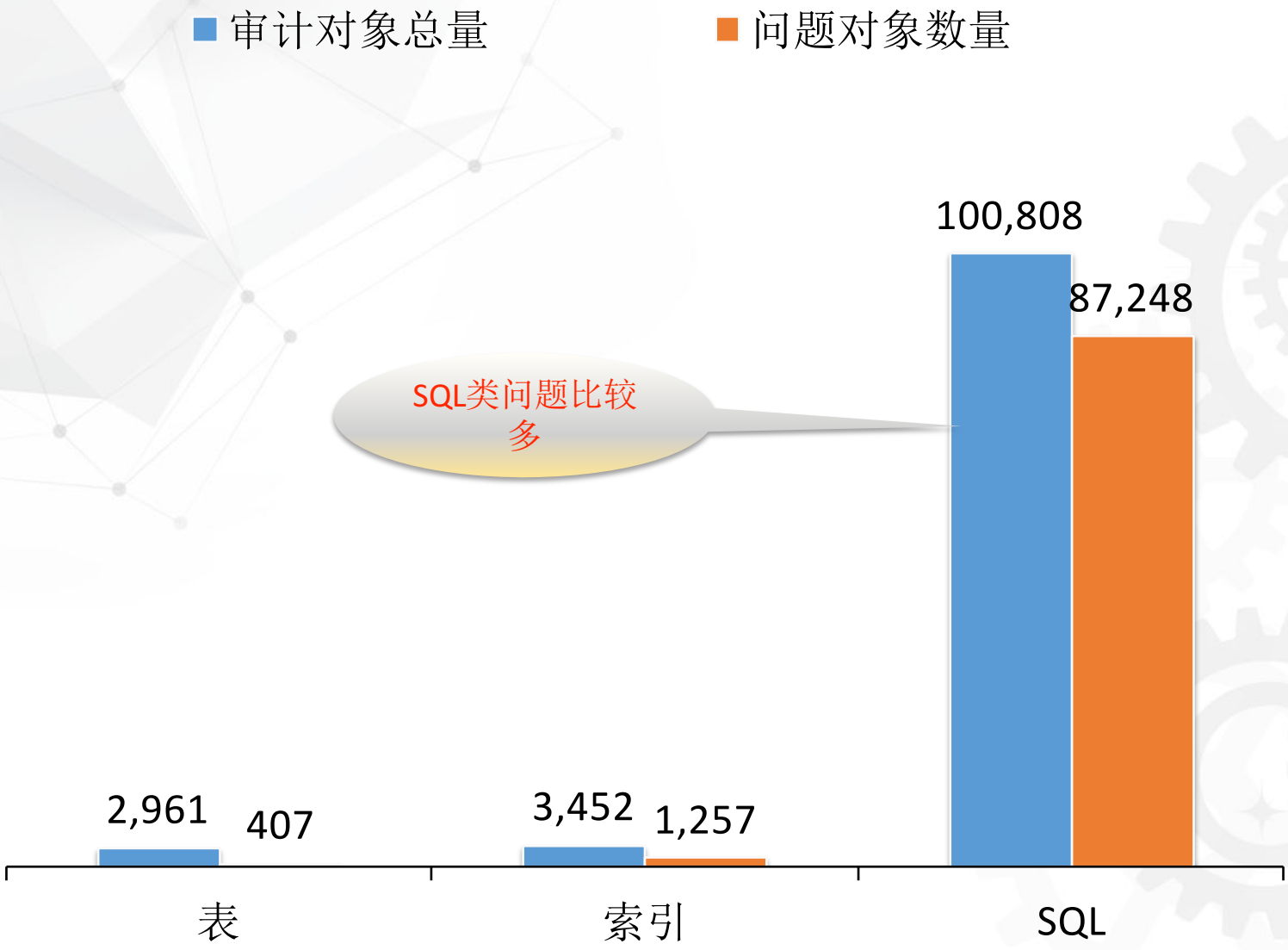
注：TIMES表示该指标为正常时间的倍数

华北某寿险公司核心系统



注：优化前的设为比较基准，为1

上海某金融系统


审计对象总量
问题对象数量
100,808
87,248
SQL类问题比较多
2,961
407
3,452
1,257
表
索引
SQL

上海某金融系统

审计对象	审计项	问题数量	问题率(%)
表	使用long、long raw类型字段	1	0.03
	无主键表	337	11.38
	外键列无索引	69	2.33
索引	无SQL使用的索引	1257	36.41
SQL	索引跳扫	3363	3.34
	全表扫描	28004	27.78
	过多嵌套	15368	15.24
	未使用绑定变量	80955	80.31
	查询条件上做运算	1198	1.19
	笛卡尔积	239	0.24
	过多表连接	2443	2.42

上海某金融系统

用户	审核对象(个)			问题对象(个)			评分(0~100)			
	表	索引	SQL	表	索引	SQL	表	索引	SQL	综合
FPS	204	244	0	54	215	0	97.59	96.88	100	97.2
IFM30	585	722	2613	115	181	1715	99.48	99.11	**64.1**	**75.83**
LCZGDBA	723	828	74053	79	243	65462	99.54	98.9[illegible]	**78.89**	**79.31**
LCZGDBB	723	828	21468	79	1	17479	99.54	100	**78.33**	**79.78**
LCZGDBC	726	830	2674	80	617	2592	[illegible].54	97.37	80.41	87.02

这段时间内用户IFM30的SQL问题比较多


THANKS
SequeMedia
盛拓传媒
ITPUB
ChinaUnix