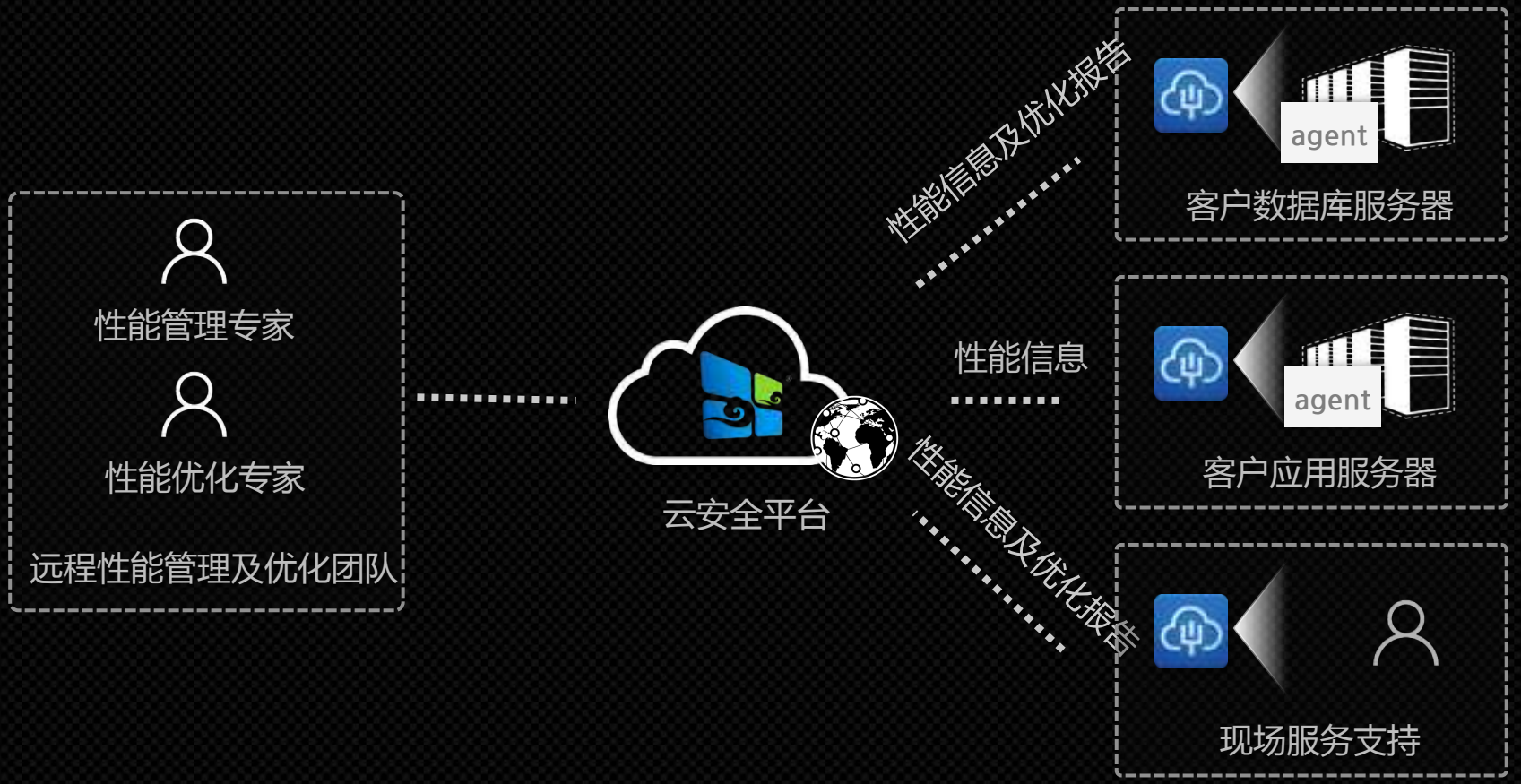
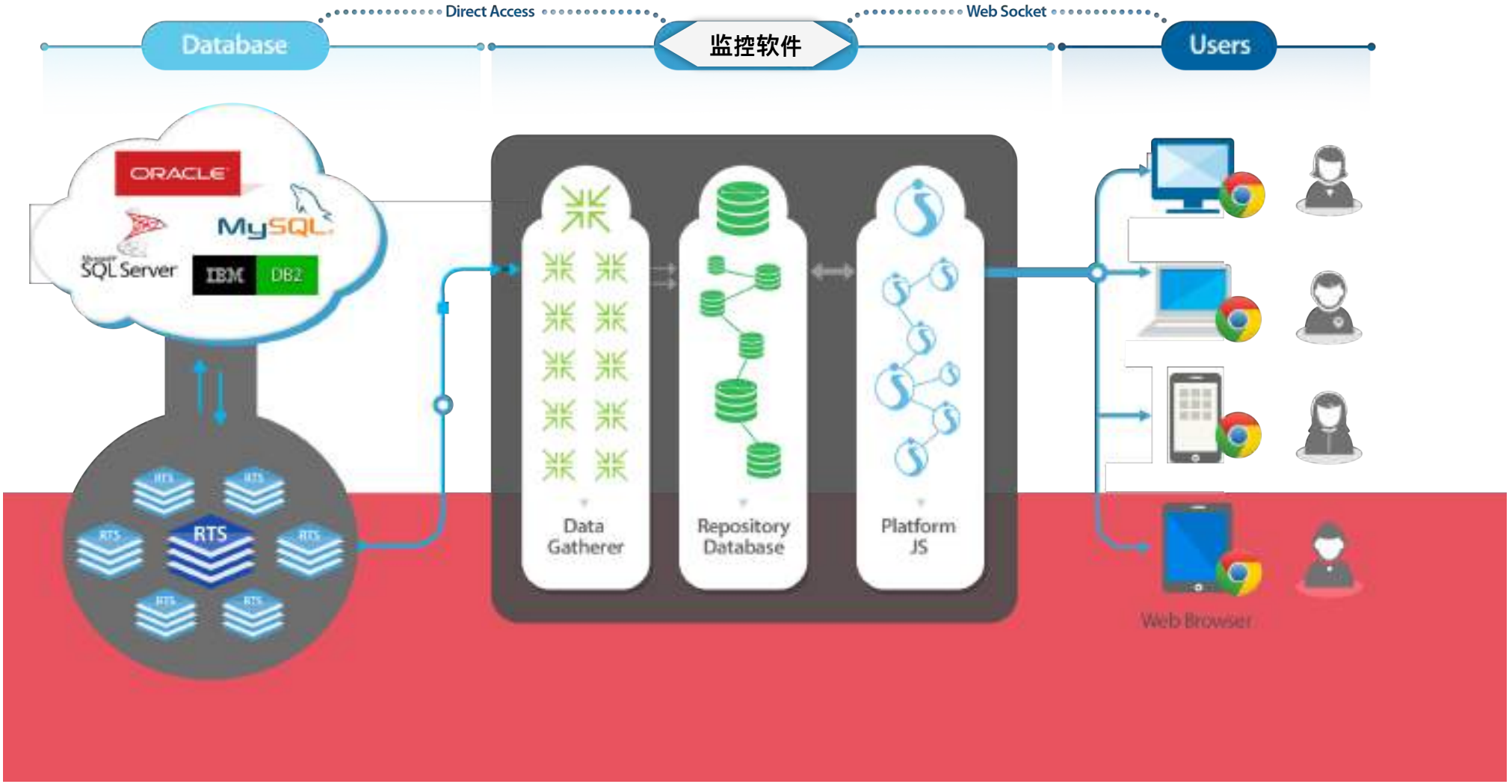
3.1 基于云的性能管理和性能服务的技术架构

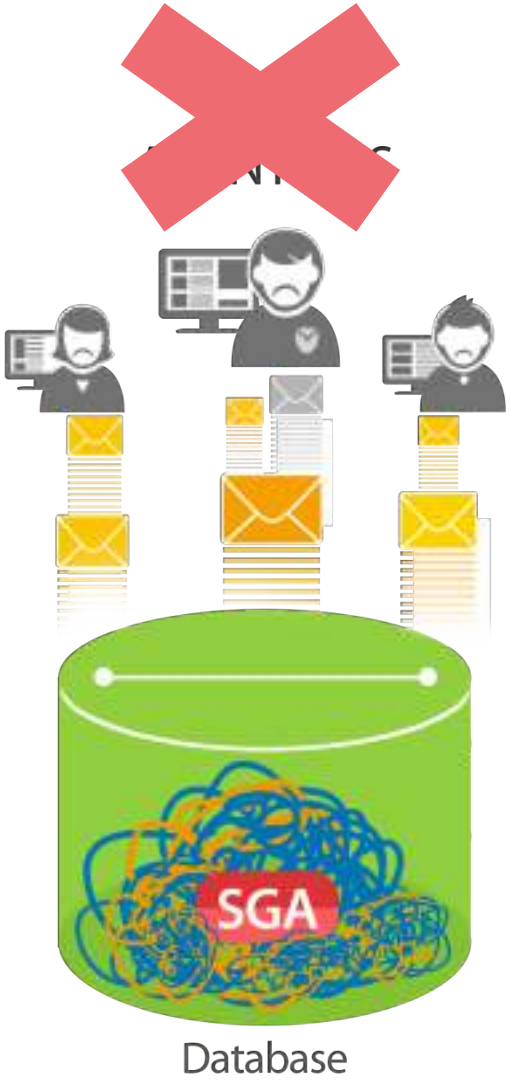


3.2 数据库性能信息监控及搜集

将性能管理或性能监控软件的AGENT部署在对象服务器上，AGENT搜集到的监控信息，可以存储在云服务平台中的知识库中，工程师通过访问知识库实现性能管理和优化。



3.2 数据库性能信息监控及搜集

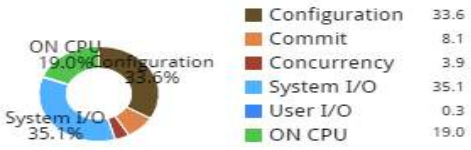


搜集各种性能指标、SQL语句、其执行计划，及日志信息。

3.2 数据库性能信息监控及搜集

1 Minute Analysis

Wait Class (%)

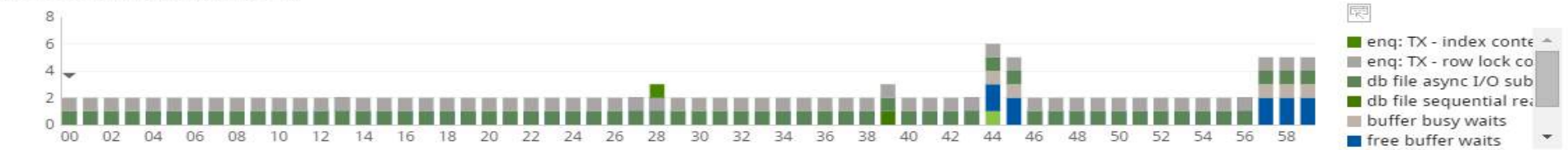


Event Name	Time (%)	Total Time (Sec)	Value/Sec	Total Waits	Wait Time/Waits (ms)	Wait Class
free buffer waits	33.6%	58	0.97	196	296.0	Configuration
db file async I/O submit	30.7%	53	0.88	11	4,823.6	System I/O
CPU Time	19.0%	32	0.55			
log file sync	8.1%	13	0.23	301	46.4	Commit
log file parallel write	3.4%	5	0.10	248	23.4	System I/O

18:42:00

Active Session Trend (Wait Class)

Active Session Trend (Wait Events)



SID	Program	Module	User Name	SPID	CPU (%)	Elapsed Time (Sec)	Event Name	Wait Time	Logical Reads/Sec (blocks)	Physica (b
384	oracle@localhost.lo...			14702	0	1,008,569	db file async I/O submit	WAITING (0)	0	
585	LitePlus.exe	summary_3	MAXGAUGE	31684	7	273,849	db file sequential read	WAITED KNOW...	1,240,017	
388	oracle@localhost.lo...	PQ_TEST	MAXGAUGE	8357	0	191,583	PX Deq Credit: send blkd	WAITING (0)	0	
772	oracle@localhost.lo...	PQ_TEST	MAXGAUGE	8359	0	191,583	PX Deq Credit: send blkd	WAITING (0)	0	

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51

可以秒单位实时搜集和分析性能状况

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 SUM

Chapter 4

性能优化方案-战略索引优化

战略索引设计及应用步骤

1) 全面搜集表的读取类型（搜集SQL语句并解析）

2) 选定索引列对象和统计分析其数据分布

3) 统计列使用频次及运算符

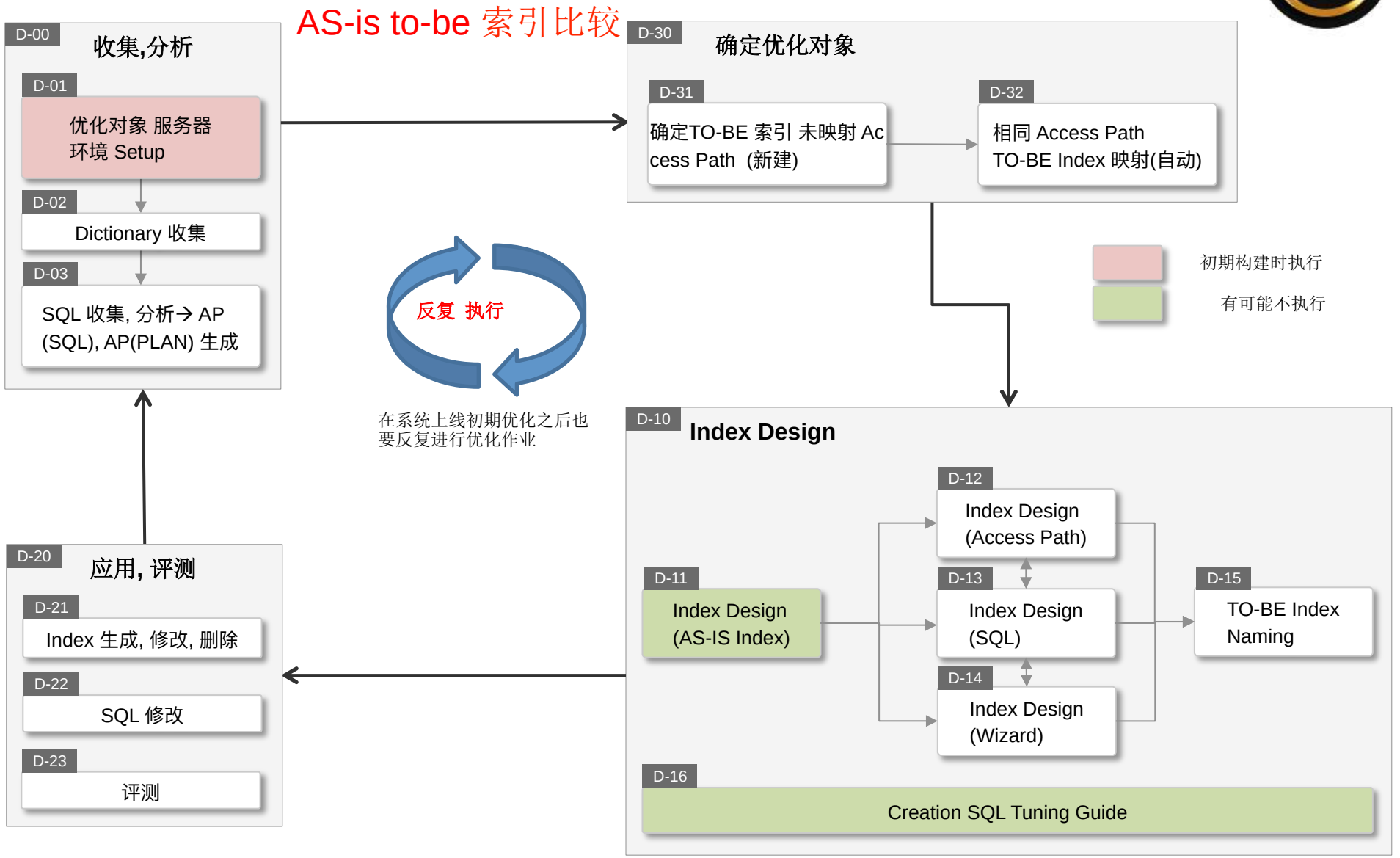
4) 组合索引列选择及列顺序设计

5) 模拟和测试

6) 调整部分SQL语句

7) 索引部署及监控

数据库性能优化-战略索引优化



数据库性能优化-战略索引优化

TABLE_NAME	SALES	销售信息管理表			TOTAL ROWS		201,500	平均月增加量		5,000	
★No	Access Pattern	Count	Range	Old	New		(1)	(2)	(3)	(4)	(5)	(6)
★1	SALENO(=), ITEM(=)	9	1	1	1	现行索引	SALENO	SALEDATE	STATUS	SALEDEPT	CUSTNO	ITEM
★2	SALEDATE(=), SALEDEPT(=)	3	180	2	2		ITEM	SALEDEPT			SALEDATE	
★3	SALEDATE(like), SALEDEPT(=)	5	3000	4	2			SALETYPE				
★4	CUSTNO(=), SALEDATE(between)	3	300	5	3							
★5	SALEDATE(like),STATUS(=60),CUSTNO(like)	2	1200	3	4							
★6	STATUS(in), [AGENTNO(like)]	4	400	3	4							
★7	ITEM(=),SALEDATE(like),SALEDEPT(like)	4	1400	2	5	新建索引	(1)	(2)	(3)	(4)	(5)	(6)
★8	SALEDATE(like),STATUS(=),group by CUSTNO	3	1000	3	4		SALENO	SALEDEPT	CUSTNO			
★9	SALEDEPT(=),SALEDATE(like),SALETYPE(=),	4	3000	4	2		ITEM	SALEDATE				
	order by SALEDATE, SALETYPE							(CLUSTER)				
★10	SALEDATE(between),CUSTNO(=),	3	300	5	3							
	(STATUS(=) or SALETYPE(=))											
★11	AGENTNO(=),SALEDATE(between), ITEM(like)	4	800		6							

主要列的分布度	列名	种类	平均	最大	备注事项
	Saleno	20,000	11	100	
	Saledate	1,500	130	800	月平均(5,000行)，月末集中
	Saledept, saledate	11,000	20	180	1个月平均(500)，使用期间为1年
	Status	25	8,000	56000	60,90时为00%，其它：平均为300
	Custno	3200	63	300	
	agentno	250	1,000	4,500	

注意事项

第6个读取类型中的STATUS为60,90时能够向部分范围处理方式引导。

第2,9个读取类型中查询范围限制在12个月以内。

数据库性能优化-战略索引优化

Sample SQL

```
SELECT COUNT( * )
FROM T_SHOP_INOUT X,
     T_SHOP_INOUT_DETAIL Y,
     T_ITEM A
WHERE X.INOUT_NO = 'N1267578'        --- (1)
  AND X.INOUT_NO = Y.INOUT_NO        --- (2)
  AND Y.ITEM_ID = A.ITEM_ID          --- (3)
ORDER BY Y.PACKING_UNIT_NO, Y.SERIAL_NO
```

自动分析

Access Path (SQL)

T_SHOP_INOUT

: **INOUT_NO (=J), INOUT_NO(=)** ← **(1),(2)**

INOUT_NO列用于常数运算以及用于条件连接运算，此为组合 Access Path.

T_SHOP_INOUT_DETAIL

: **INOUT_NO (=J), ITEM_ID(=J)** ← **(2),(3)**

INOUT_NO, ITEM_ID列应用于条件连接的Access Path.

T_ITEM

: **ITEM_ID (=J)** ← **(3)**

Index Design

TO-BE Index Design

T_SHOP_INOUT : INOUT_NO 单独作为常量条件应用的情况

➔ **INOUT_NO**

T_SHOP_INOUT_DETAIL : INOUT_NO, ITEM_ID是作为连接条件加以应用的情况. 此种情况应尽量从 T_SHOP_INOUT 表中，以常量获得 INOUT_NO 条件.

➔ **INOUT_NO**

T_ITEM : 从T_SHOP_INOUT_DETAIL表中，获得 ITEM_ID的取值.

➔ **ITEM_ID**

数据库性能优化-战略索引优化

CUST_NO	366,723	364	1	平均 40行♪

CUST_NO

CUST_NO, ARR_DT(B)
CUST_NO, PRICE_STD

CUST_NO +

CUST_NO, ARR_DT(B)
CUST_NO, PRICE_STD
PRICE_STD, BNK_CD
BNK_CD, SND_CD, ARR_DT(L)
PRICE_STD, ARR_DT(B)
BSE_CRD_NO, ACT_NO, PRICE_STD
BSE_CRD_NO, ARR_DT(L)
PRICE_STD, BNK_CD(IN), ARR_DT(L)
PRICE_STD[IN], BNK_CD(IN)
BNK_CD, ARR_DT(B)
SND_CD, BNK_CD(IN), ARR_DT(B)

~~CUST_NO, PRICE_STD~~
PRICE_STD, BNK_CD
PRICE_STD, ARR_DT(B)
~~BSE_CRD_NO, ACT_NO, PRICE_STD~~
PRICE_STD, BNK_CD(IN), ARR_DT(L)
PRICE_STD[IN], BNK_CD(IN)

PRICE_STD+BNK_C
D+ARR_DT

PRICE_STD, BNK_CD使用频率高

~~PRICE_STD, BNK_CD~~
BNK_CD, SND_CD, ARR_DT(L)
~~PRICE_STD, BNK_CD(IN), ARR_DT(L)~~
~~PRICE_STD(IN), BNK_CD(IN)~~
BNK_CD, ARR_DT(B)
SND_CD, BNK_CD(IN), ARR_DT(B)

BNK_CD+ARR_D

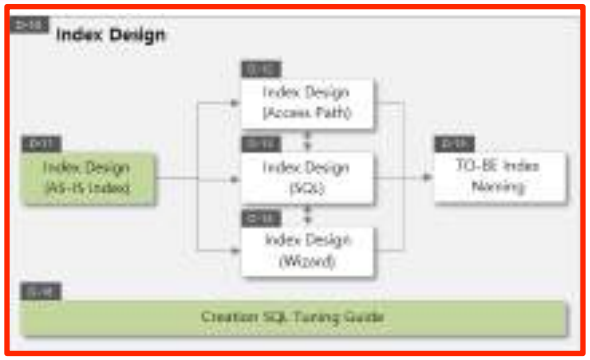
BSE_CRD_NO

BSE_CRD_NO, ACT_NO, PRICE_STD
BSE_CRD_NO, ARR_DT(L)

BSE_CRD_NO + ??

BSE_CRD_NO	402,849	320	1	平均 36行♪

数据库性能优化-战略索引优化



D-11 Index Design(AS-IS Index) 1

比较现有 AsIs-Index和收集的 AP(SQL)之间的契合度，对于契合度比较高的索引尽量复用。
对于契合度比较低的索引，需要设计新索引。

- 尽快研判AsIs Index 中能够加以复用的部分.(Optional)
- 手动对AP进行分组研判，并进行索引设计(Index Design(Access Path))或通过向导进行索引设计.
- 对ToBe Index设计的各个研判项目，依次查看SQL并加以验证. (Optional)

D-12 Index Design(Access Path) 2

对收集的SQL中提取到的 AP(SQL)正确地加以分组（Grouping）并进行 ToBe-Index设计的过程.
对索引设计技术需能够娴熟加以运用.
相比于在查验 SQL执行情况的同时进行索引设计，应倾向于仅基于AP(SQL)进行索引设计.

D-14 Index Design(Wizard) 2

将收集的 AP(SQL)按照定制的规则自动经过多次处理完成分组研判，并根据结果构建ToBe-Index的过程.
对索引设计知识没有很深积淀也能通过工具进行设计.
并非全部处理过程都采用自动方式，应按阶段适用自动处理，而部分过程应采用手动方式。

D-13 Index Design(SQL) 3

按表分别对收集的 SQL一一加以确认，并进行ToBe-Index设计的过程.
不是一开始就采用此方法，而是在上面的 Index Design(Access Path) 或者 Index Design(Wizard) 方式构建索引之后，再针对性地对SQL进行细致的分析之后，根据情况按需进行索引设计

D-15 ToBe-Index Naming 4

对上面这些构建阶段设计出的ToBe-Index加以命名，并赋予各种选项.
最终生成新增，修改这些索引的DDL

D-16 Creation SQL Tuning Guide

在设计ToBe-Index的过程中对各SQL标注改进事项，并据此编写Tuning Guide.

数据库性能优化-战略索引优化

TAB1
Total rows : 340,988

AsIs Index
TAB1_idx1 : col3

TAB1
Total rows : 340,988

AsIs Index
TAB1_idx1 : col1
TAB1-idx2 : col3, col2

反复施行

优化状态

- 周期性地反复收集和分析
- 维护好日常的优化状态
- 定期进行优化前，后性能比较
- 恶性查询的检索条件上报

D-00 搜集，解析

D-10 Index Design

D-20 测试及应用

SQL#1 (2014-4-1)
SELECT COL1, COL2
FROM TAB1
WHERE COL1 = '23'

SQL#2 (2014-4-1)
SELECT COL1, COL2
FROM TAB1
WHERE COL2 = '23'
AND COL3 = 'dkhd'

SQL#3 (2014-4-16)
SELECT COL1, COL2
FROM TAB1
WHERE COL5 = '2014'
AND COL3 = 'dkhd'

AP#1
COL1(=)

AP#2
COL2(=), COL3(=)

AP#3
COL3(=), COL5(=)

未映射状态

ToBe Index
TAB1_idx1 : COL1

ToBe Index
TAB1_idx2 : COL3, COL2

ToBe Index
TAB1_idx3 : COL5, COL3

Index 创建脚本自动生成 (自动)

DBA 执行

索引构建前后性能比较，评测

优化对象，新增构建项目的确认和处置

❖ 播放动画可以对应用IDO对索引进行优化的整个过程有全面的了解.

对未映射 AP重点加以管理可以将优化状态进行常态化运维

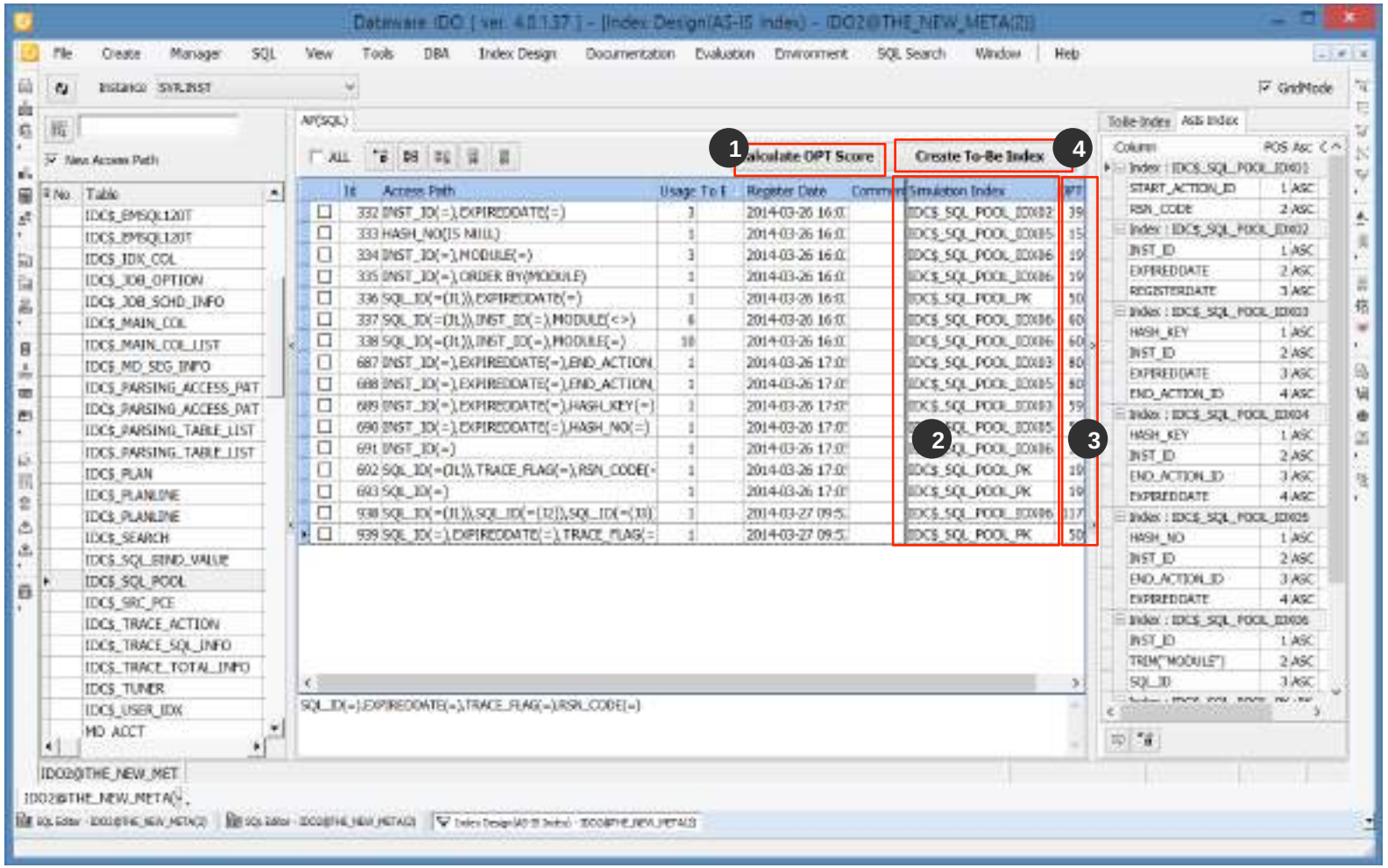
始终确认 AsIs Index – ToBe Index是否一致，并维护其一致性

对于不使用的 SQL,AP需要周期性地进行检索以及清除

数据库性能优化-战略索引优化

功能概要

- 索引设计阶段的第一个步骤.
- 从现有的AS-IS Index中甄别那些能够满足收集到的 AP(SQL)需求的索引，并将其制定为TO-BE Index的过程.
- 此过程需要对AS-IS Index和 AP(SQL)的适用度进行 Simulation 操作 。其结果会被换算为 Index OPT Score，并可根据其结果决定是否要沿用原有的索引.



1 根据选定的表的所有AP(SQL)计算其与 AsIs-Index的适用度.

2 将计算适用度分数，着重推荐那些与 AP(SQL)更加契合的AsIs-Index

3 上面步骤2中推荐的 AsIs-Index的适用度分值 (Index OPT Score) 将显示在右侧.

用户可以根据此分值将特定AsIs-Index构建为 ToBe-Index.

左侧图式就显示938号 AP(SQL) 适用度分值为 117，其OPT Score最高.

这里的 OPT Score并非绝对值，而是在一个表中的相对适用度分值

4 从AsIs-Index中选择要直接生成 ToBe-Index的 AP(SQL) 项目，并生成索引.

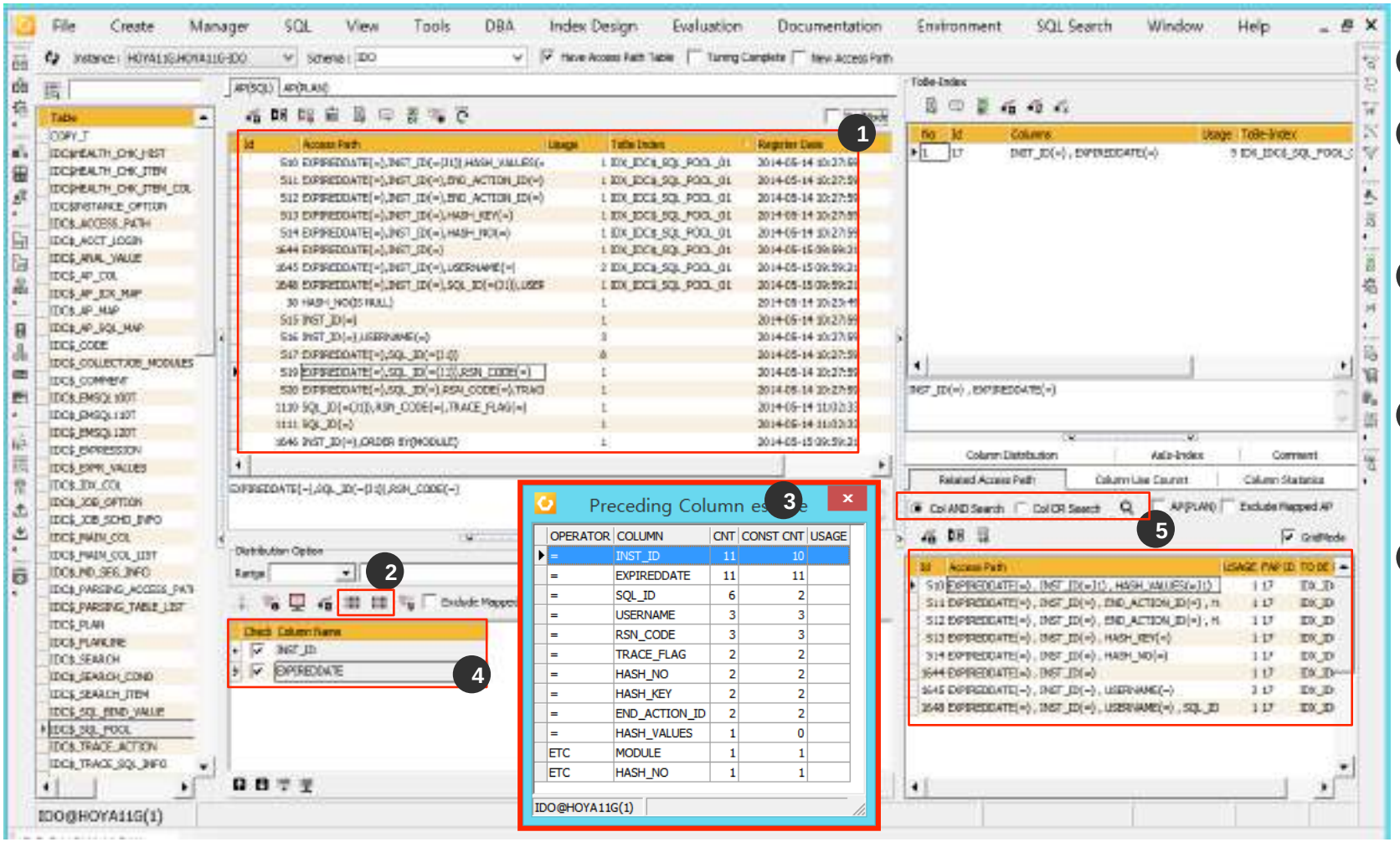
数据库性能优化-战略索引优化

Index Design Using Access

D-12

功能概要

- 按表，基于收集、分析的 AP(SQL)设计 ToBe-Index的功能.
- 此过程不是对个别SQL进行详细分析并进行索引设计，而是主要基于 AP(SQL)将主要列分组之后进行索引设计.
- 在设计过程中为了应用从DB收集的统计信息 (Cardinality 等)，可以直接对主要列进行分布度调查.



1. 按表分别进行收集、分析的 Access Path
2. 收集，分析的 AP中使用次数最多并且以 '='运算应用的条件列需要提取出来，此步骤可以针对性进行检索
3. 单击检索按钮就会显示如图所示的窗口，显示使用的运算符以及列及其适应频次
4. 从列出的检索条件中选择排在前列的列字段并将其选定为 ToBe Index对象列.
5. 对选定为Tobe Index对象列的条件，检索与其关联的 AP. 此时检索可以指定 'AND' 或 'OR'运算进行检索.

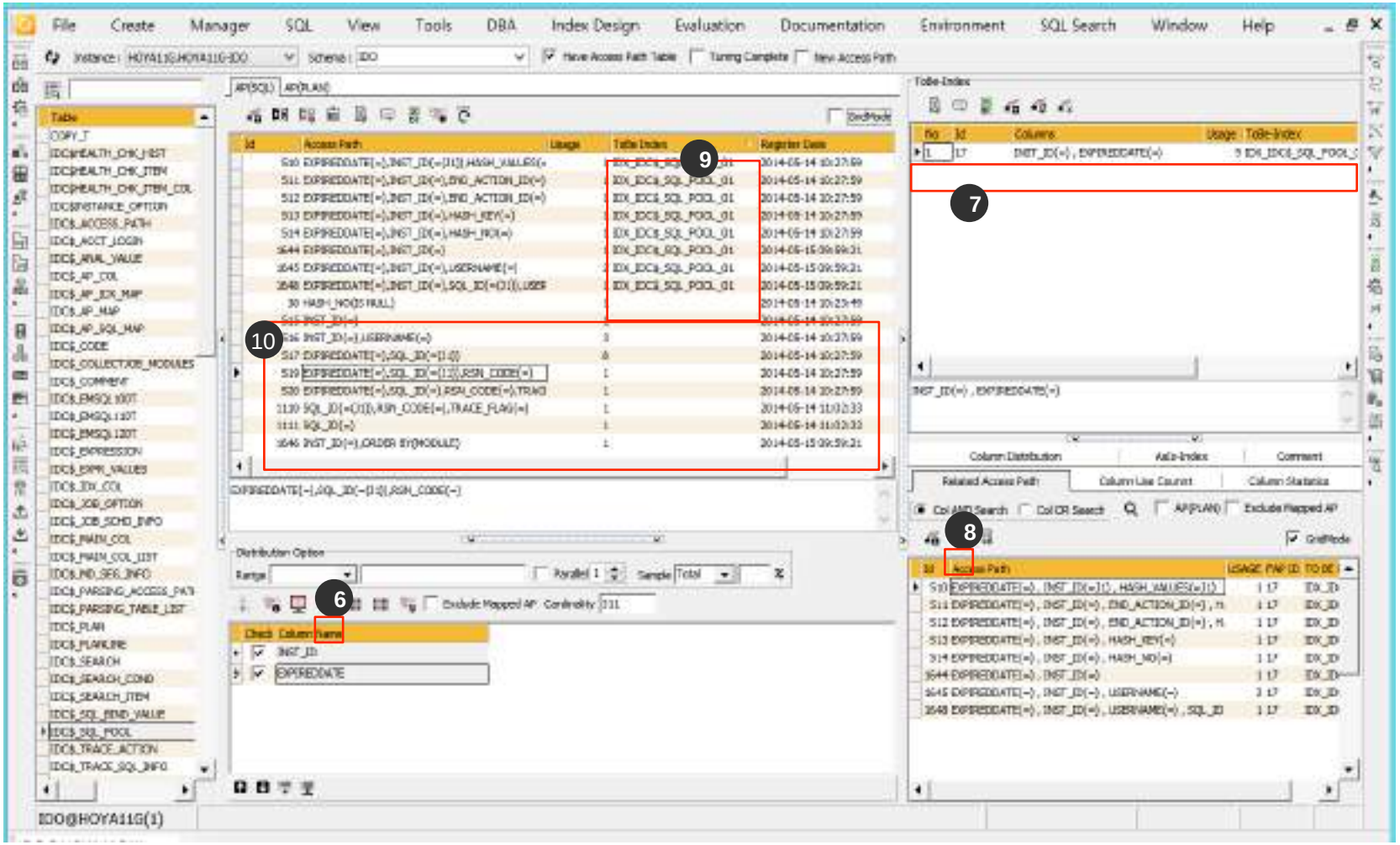
数据库性能优化-战略索引优化

Index Design Using Access

D-12

功能概要

- 按表，基于收集、分析的 AP(SQL)进行 ToBe-Index设计.
- 此过程不是对个别SQL进行详细分析并进行索引设计，而是主要基于 AP(SQL)将主要列分组之后进行索引设计.
- 在设计过程中为了应用从DB收集的统计信息 (Cardinality 等)，可以直接对主要列进行分布度调查.



6 对于选定的 ToBe Index候补列将生成索引.

7 生成索引，索引将自动生成，且索引名将按照 ‘Option’ 中指定的规则自动命名

8 将映射与 生成的ToBe Index 一致的AP.

9 执行映射之后就会产生如图所示的AP-ToBe Index 间的映射关系.

10 通过 1~9步的一系列操作，对那些与 Tobe Index尚未映射的余下的AP进行同样的分析处理，并进行 ToBe Index 设计以及完成映射操作

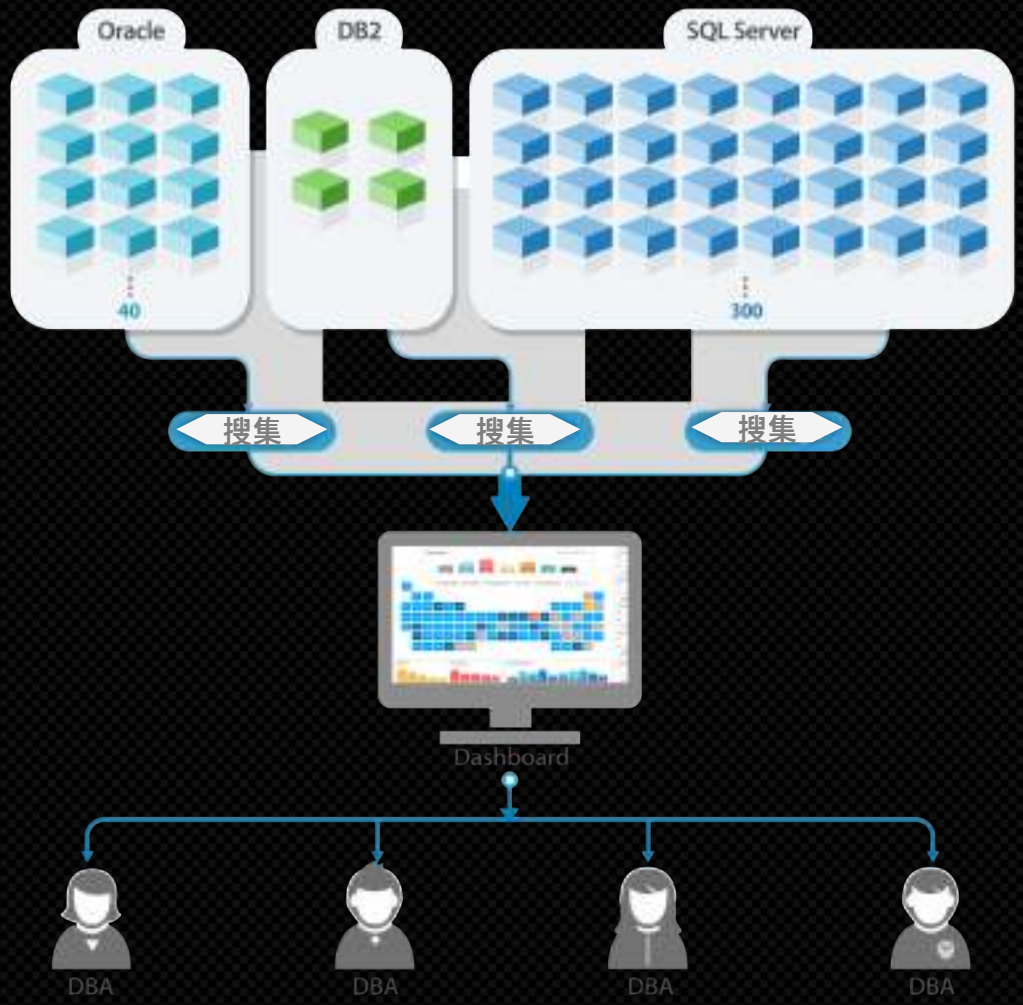
Chapter 5

成功案例

某客户案例

场景

- 实时搜集对象数据库（ORACLE，DB2， SQL SERVER）。
- 将性能信息发送至服务器短，并存储在服务器的知识库中。
- DBA，性能优化人员，实时查看系统性能问题，并及时了解系统瓶颈。
- 长时间跟踪高消耗SQL语句和SESSION,并形成分析报告，以便准确定位性能瓶颈。
- 长期累积SQL语句和执行计划，并将其解析为读取路径，以便设计战略索引。



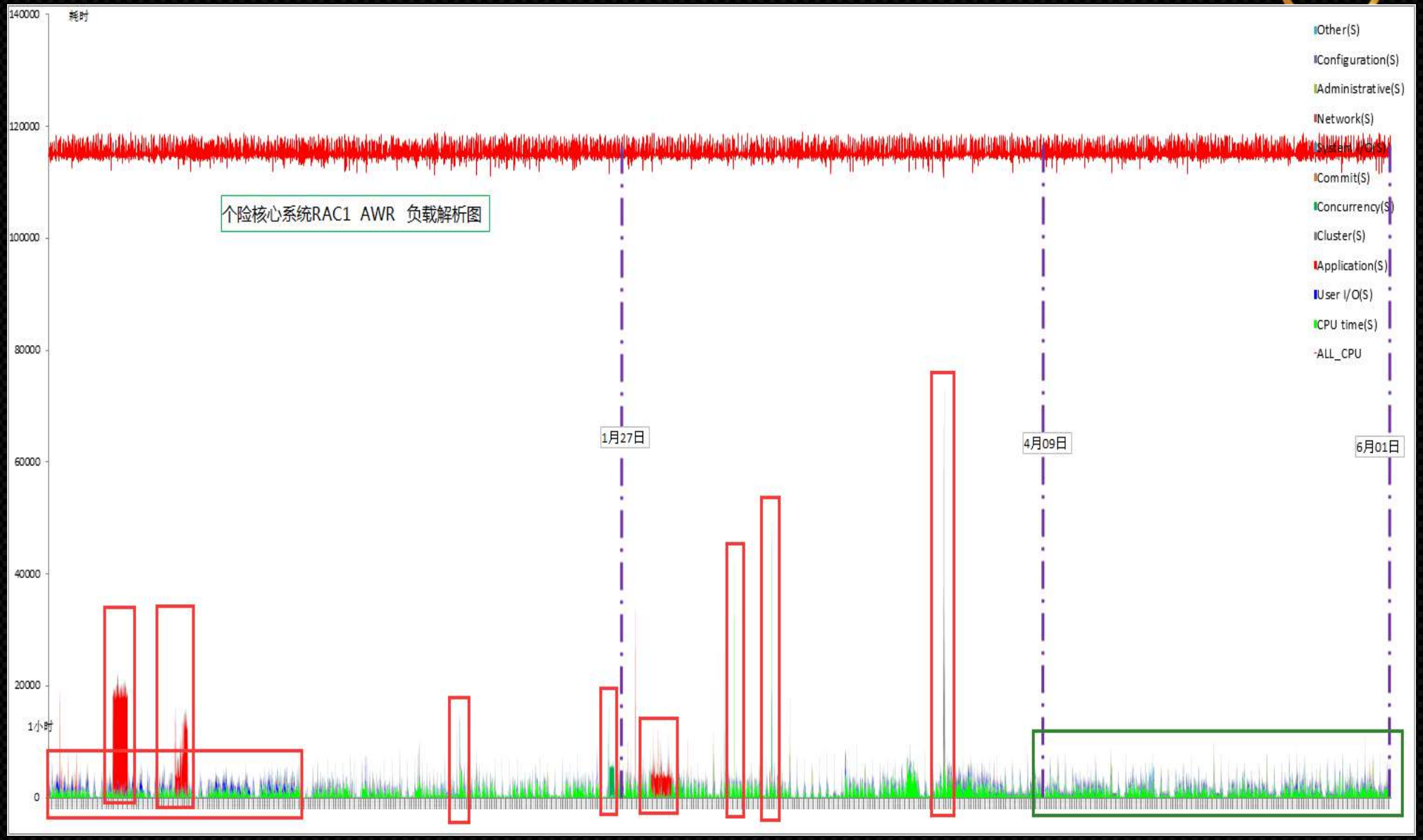
某客户案例

为提升优化效果，结合核心表索引设计，完成了125个TOPSQL优化。

优化效果如下：

- ✓ SQL 逻辑IO： 由 625,386,664 Blocks 下降到 2,687,922 Blocks,改善率231.8倍。
- ✓ SQL 耗时： 由9774.49 S 下降到 228.68 S,改善率 42.74 倍。
- ✓ TOPSQL 耗时大幅改善，核心业务性能提升。

某客户案例


耗时
140000
120000
100000
80000
60000
40000
20000
0
1小时
个险核心系统RAC1 AWR 负载解析图
1月27日
4月09日
6月01日
Other(S)
Configuration(S)
Administrative(S)
Network(S)
System I/O(S)
Commit(S)
Concurrency(S)
Cluster(S)
Application(S)
User I/O(S)
CPU time(S)
ALL_CPU

Chapter 05 # 优缺点分析

人员成本显著降低，一名工程师可以支持5~10个企业

基于云和工具化的方式实现性能信息的搜集和优化，显著降低了工程师的投入，每个工程师可以支持更多的客户服务。

企业客户无须支付昂贵的费用设立优化项目，或购买维保费用，通过该方式给企业提供的服务的持续性和质量，与现场服务没有本质区别。

企业负担降低，服务持续性好，质量有保证，效果与现场服务相近

减少工程师出差频率率，工程师获得更多归属感，降低项目实施成本，有利于提升人员稳定性。

工程师出差频次降低后，不仅可以提升工程师的归属感，而且还可以为企业获取更多利润，也有利于人员的持续性培训和管理。

该服务提供方式目前仅适用于对安全要求不是特别高，且可以通过特定网管能够允许外部访问的企业，否则难以应用。只要服务厂商可以严格遵守客户企业的安全要求，其实该方式不会对企业的数据安全带来隐患，因为搜集时，仅搜集性能有关的数据，不涉及具体业务数据。

该方式目前仅适用于安全要求不是特别高的一些企业客户。

Thank you