



海量数据学院  
VASTDATA EDUCATION



ORACLE  
ACE

VDOUG  
VastData Oracle User Group  
海量数据Oracle用户组

OMOOCU  
Oracle Massive Open Online Courses Union  
Oracle 慕课联盟

OCM  
之家

# 讲师：崔旭

网名：DBstyle

海量数据学院首席讲师，院长兼总经理，Oracle ACE，中国OCM之家(OCMH)发起人，同时创办了Oracle慕课联盟(OMOOCU)，用互联网+的形式推广线上免费课程。获得Oracle 10g 11g 12c OCM认证。从业10年以上，资深Oracle数据库专家，51CTO认证讲师，在数据库领域有丰富的经验。拥有Oracle数据库，SQL Server数据库，RHCE, F5, Cisco等十余种相关技术认证。曾任职于北京神州泰岳软件股份有限公司、北京电信发展有限公司，云和恩墨的恩墨学院教学总监。负责运维全国各省客户的海量数据库，负责高可用数据库的部署实施、故障处理、性能优化，教育培训等工作。现任海量数据学院首席讲师兼院长，为多家大中型企业和多所国家211工程高等学校，提供过Oracle相关课程培训以及技术分享活动。讲课富有亲和力和感染力，擅长理论联系实际，通过华丽的操作将枯燥的技术展现出来，使学员理解技术在真实生产中的应用。至今培养OCP和OCM数千人，培训经验丰富，致力于推广和分享ORACLE技术。技术博客地址：<http://www.dbstyle.net>



ORACLE®

Certified Master

Oracle Database 12c  
Administrator

ORACLE®

Certified Master

Oracle Database 11g  
Administrator

ORACLE®

Certified Master

Oracle Database 10g  
Administrator



海量数据学院  
VASTDATA EDUCATION

# *Examination time*

- |   |       |         |
|---|-------|---------|
| 1. General Database and<br>Network Administration,<br>and Backup Strategy | ..... | 120 min |
| 2. Data and Performance<br>Management                                     | ..... | 90 min  |
| 3. Data Guard   | ..... | 90 min  |
| 4. Grid Infrastructure and<br>Real Application Clusters                   | ..... | 90 min  |

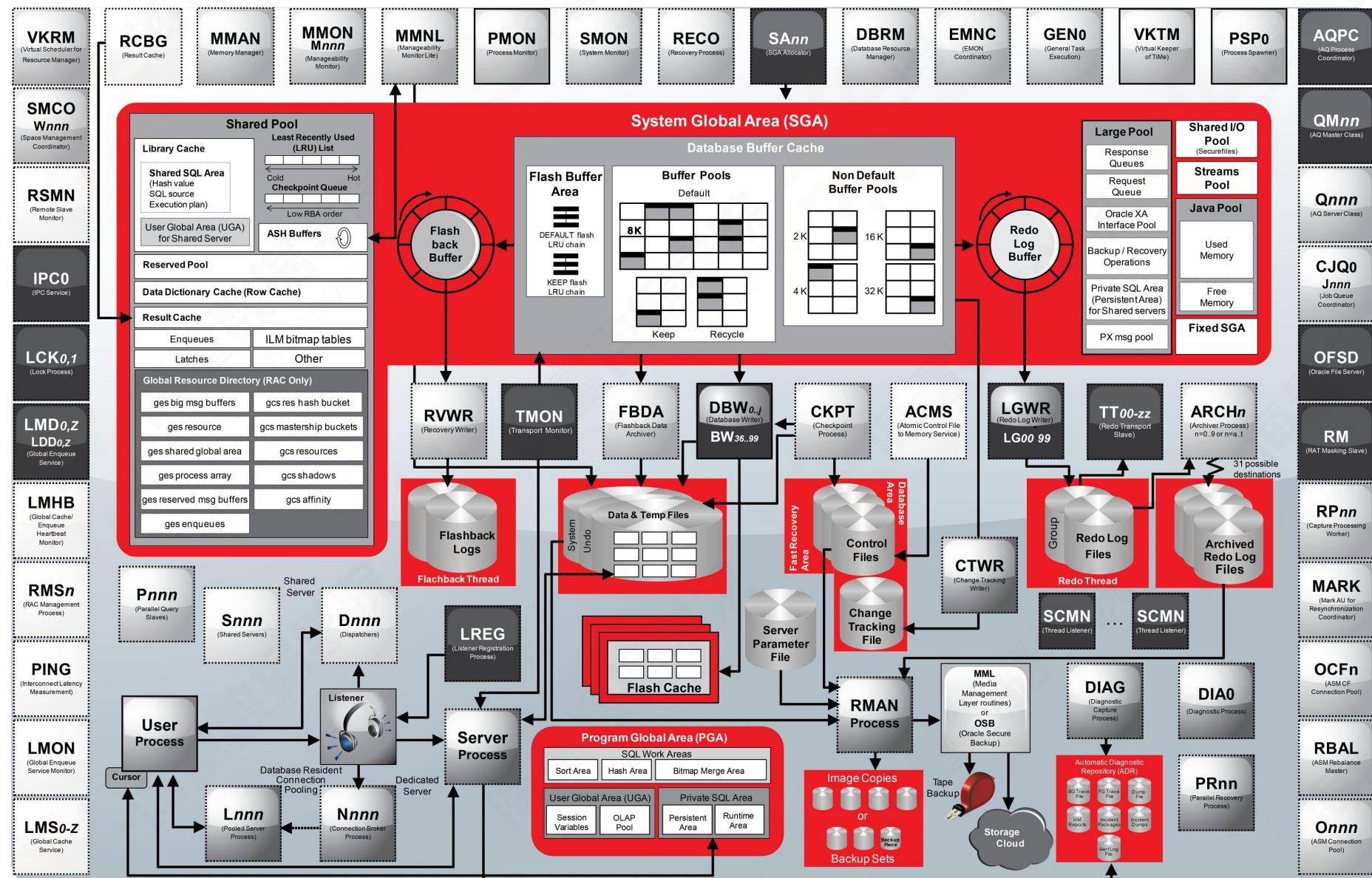


海量数据  
VASTDATA

海量数据学院  
VASTDATA EDUCATION

# Oracle Database 12c Certified Master Upgrade Exam





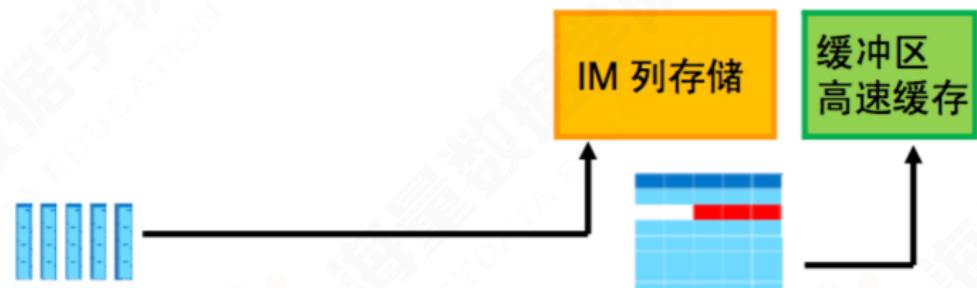
# 内存中列存储的目标

- 即时查询响应：
  - 对非常大的表中任何列进行查询均较快（100 倍）
  - 使用扫描、联接和聚集
  - 不使用索引
  - 最适合分析：列少/行多
- DML 更快：删除大多数分析索引（3 到 4 倍）

- 对应用程序完全透明



- 易于设置：
  - 内存中列存储配置
  - 段属性

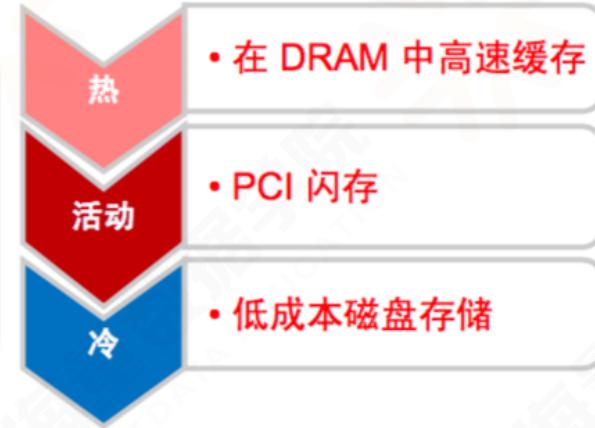


# 优点

- 内存中列存储中的活动数据



- 填充到 IM 列存储中
- IM 列存储的专门压缩算法：  
2 至 10 倍：  
FOR QUERY LOW / HIGH  
FOR CAPACITY / HIGH

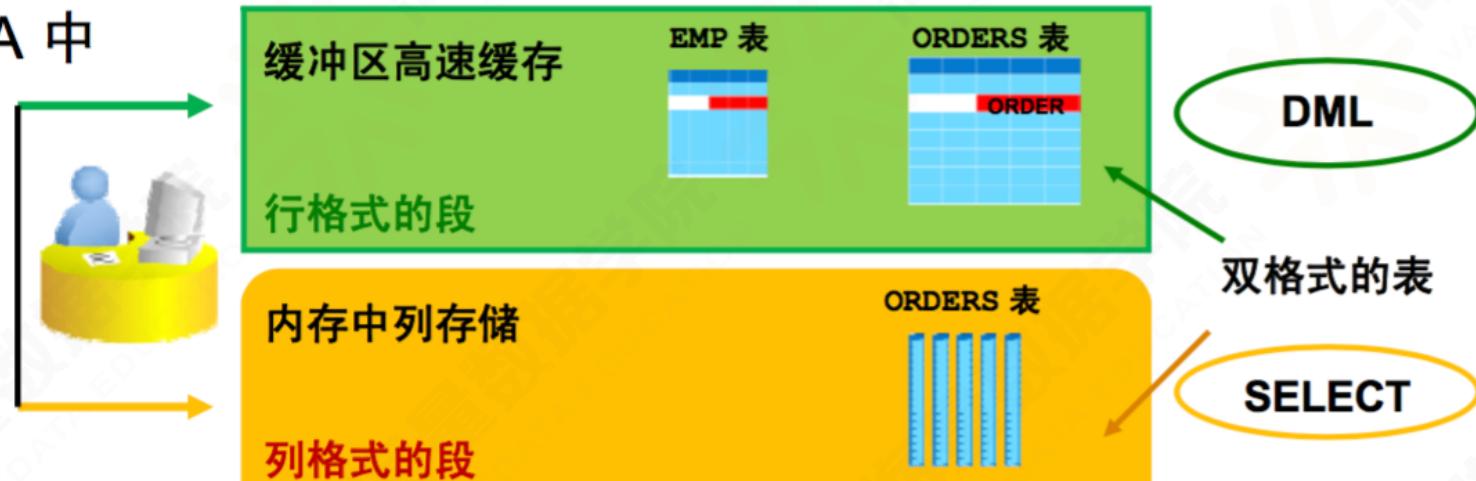


- 在 RAC 上全面支持
- 在多租户体系结构上全面支持
- 不再需要 SQL 优化
- 对 SQL 无限制
- 执行备份和恢复操作时无需更改
- 无需移植数据

# 概览

- SGA 中的新池：内存中列存储
  - 填充到 IM 列存储中的段将转换为列格式
  - 内存中段与缓冲区高速缓存的事务处理一致
- 只有一个段在磁盘上并采用行格式

在 SGA 中

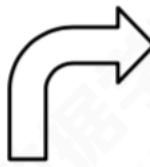


# 行存储与列存储：2维视角

SALES 表

	PRODID	CUSTID	TIMEID	CHANID	QTT	AMOUNT
row1	123	ABC	04/02	C1	12	3500
row2	357	CDE	12/05	C4	1	2000
row3	50	GHI	06/17	C1	5	4765
row4	243	PQR	05/24	C2	9	1350

行存储格式



col1: PRODID

col2: CUSTID

col3: TIMEID

col4: CHANID

col5: QTT

col6: AMOUNT

	order1	order2	order3	order4
123	357	50	243	
ABC	CDE	GHI	PQR	
04/02	12/05	06/17	05/24	
C1	C4	C1	C2	
12	1	5	9	
3500	2000	4765	1350	

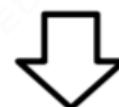
列存储格式

# 内存中列单元

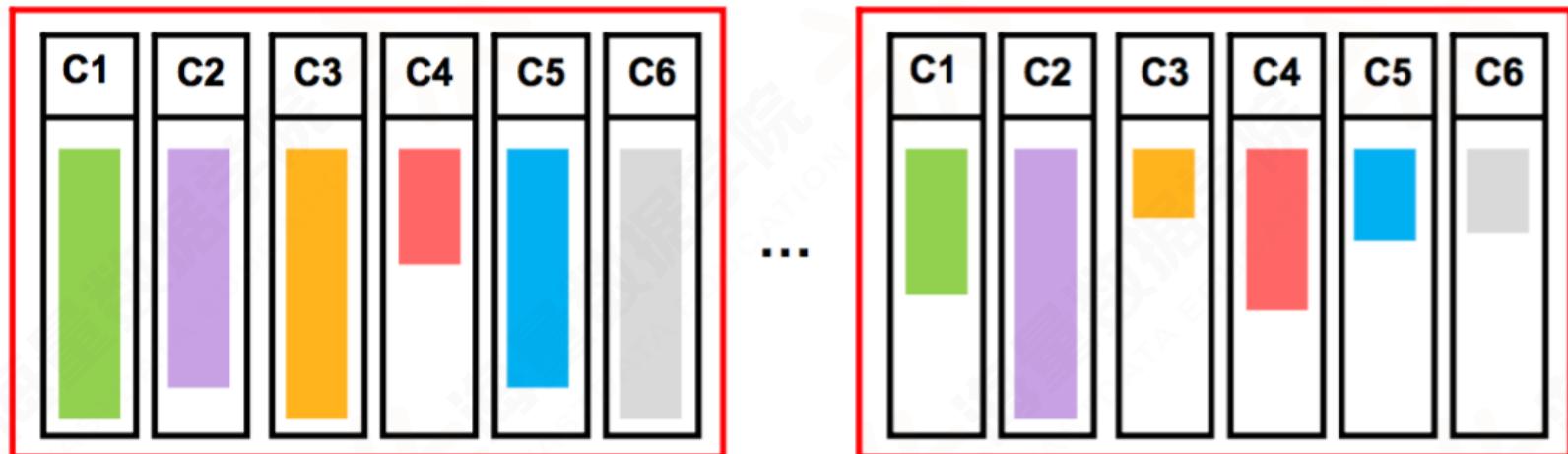
## 列格式

**SALES 表**

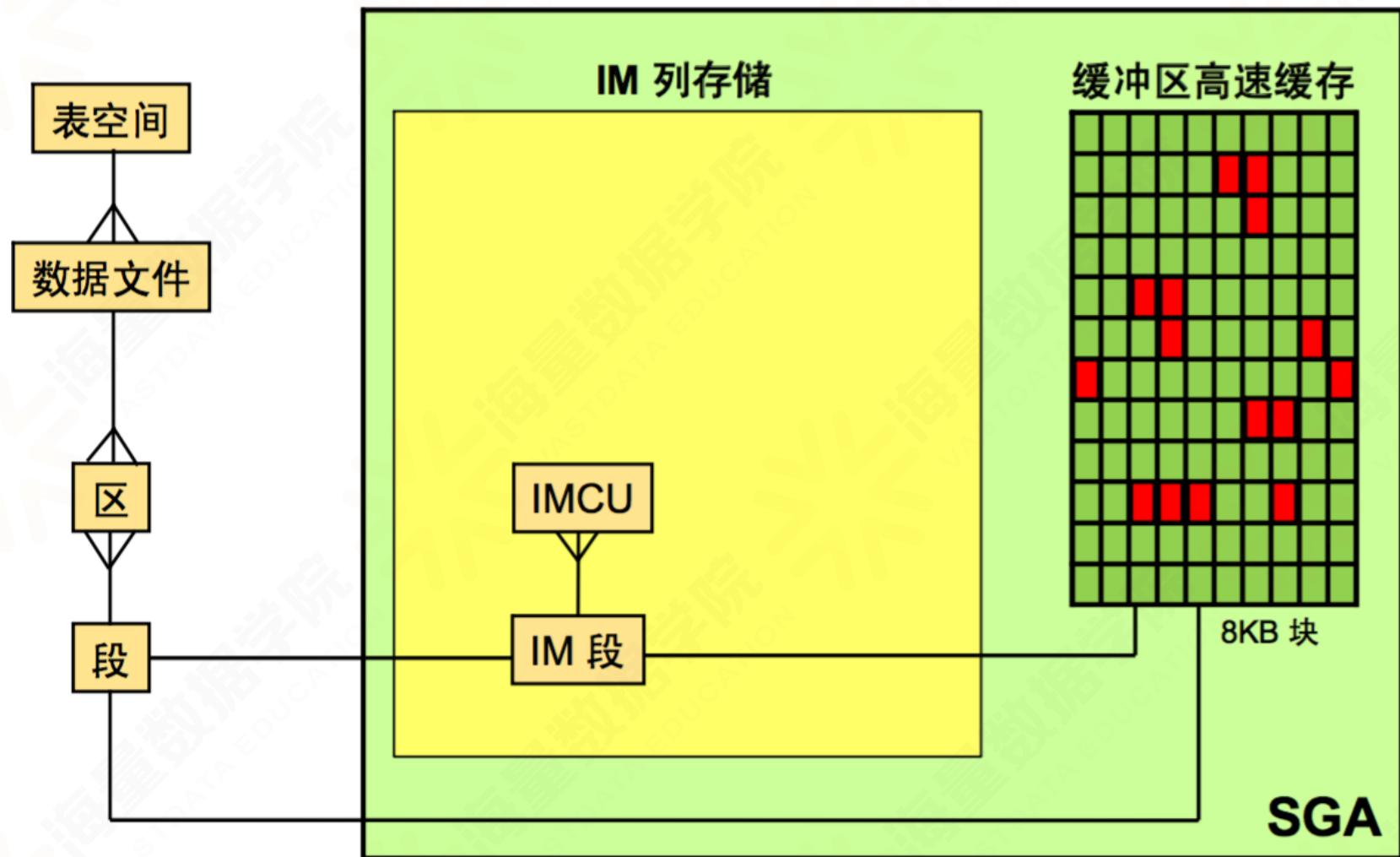
```
123:1,357:2,50:3,243:4 \| ABC:1,CDE:2,GHI:3,PQR:4 \|  
04/02:1,12/05:2,06/17:3,05/24:4 \| C1:1;3,C4:2,C2:4 \|  
12:1,1:2,5:3,9:4 \| 3500:1,2000:2,4765:3,1350:4 \|
```



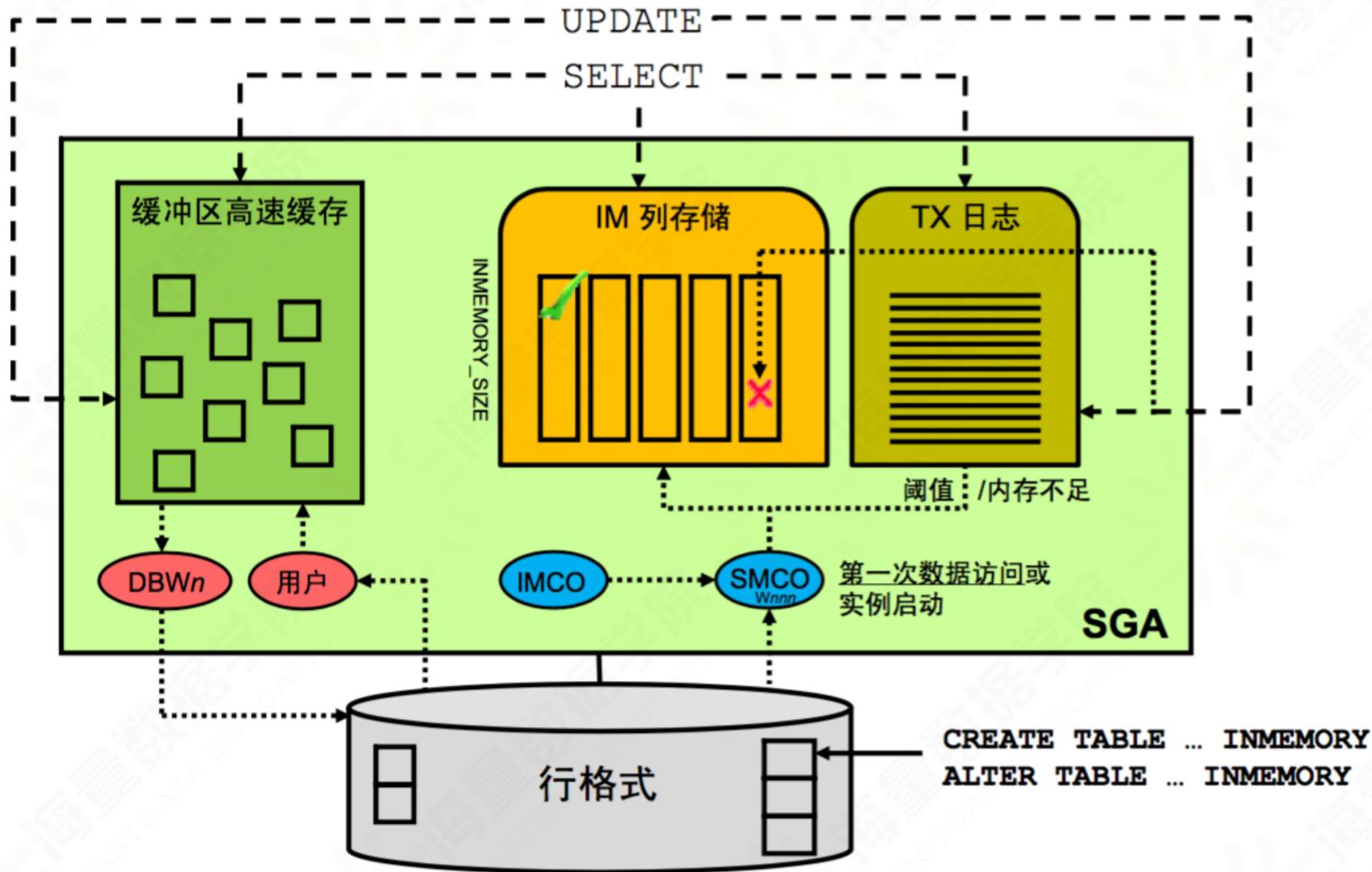
## 内存中列单元



# 内存中列存储高速缓存与缓冲区高速缓存



# 内存中的双格式



# 不再出现索引问题

- 为哪些列编制索引？



对于 OLTP: 1 到 3 个  
对于分析: 10 或 20 个

不使用 IM 列存储



- 由于缺少索引导致性能较低
- DML 期间维护任务繁重
- 需要存储空间

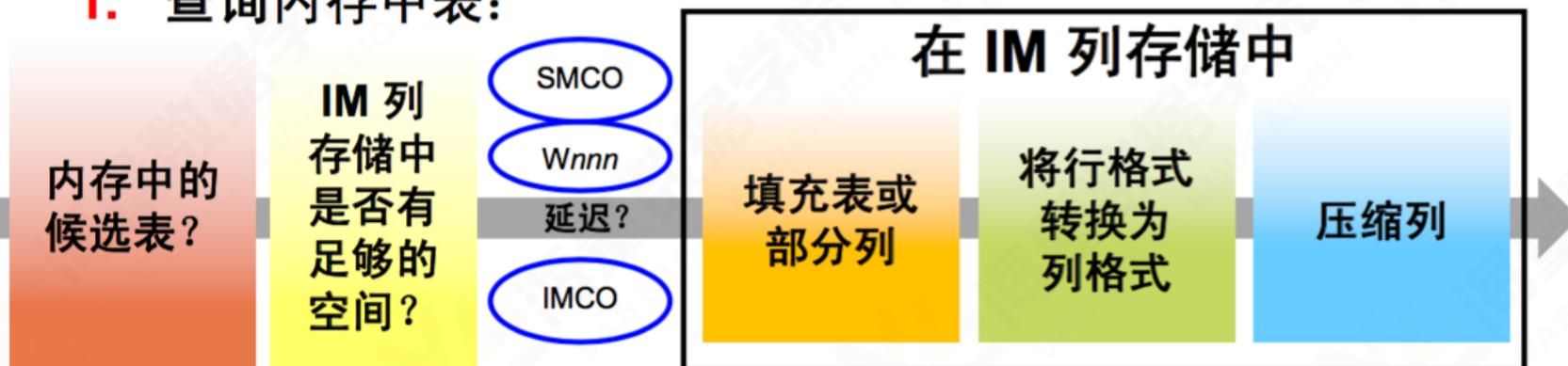
使用 IM  
列存储

- 不再有内存中表的分析索引
- 预定义的分析查询和即席分析查询均可快速运行
- OLTP 和批处理运行速度最多提高 300%

# 流程

## IM 列存储初始化。

### 1. 查询内存中表:



### 2. 结果的显示速度比采用行格式时快得多：



# 部署 IM 列存储

1. 验证数据库兼容性值。

```
COMPATIBLE = 12.1.0.0.0
```

2. 配置 IM 列存储大小。

```
INMEMORY_SIZE = 100G
```



海量数据  
VASTDATA

海量数据学院  
VASTDATA EDUCATION

# 部署 IM 列存储：对象设置

## 3. 启用/禁用要填充到 IM 列存储中的对象。

- 启用/禁用整个段：

- IMCU 仅在查询访问时进行初始化和填充。

```
SQL> CREATE TABLE large_tab (c1 ...) INMEMORY; —> 双格式
```

```
SQL> ALTER TABLE t1 INMEMORY ; —> 双格式
```

```
SQL> ALTER TABLE sales NO INMEMORY; —> 仅行格式
```

```
SQL> CREATE TABLE countries_part ... PARTITION BY LIST ...
  ( PARTITION p1 .. INMEMORY, PARTITION p2 .. NO INMEMORY);
```

- IMCU 可以在打开数据库时进行初始化。

```
SQL> CREATE TABLE test (...) INMEMORY PRIORITY CRITICAL;
```

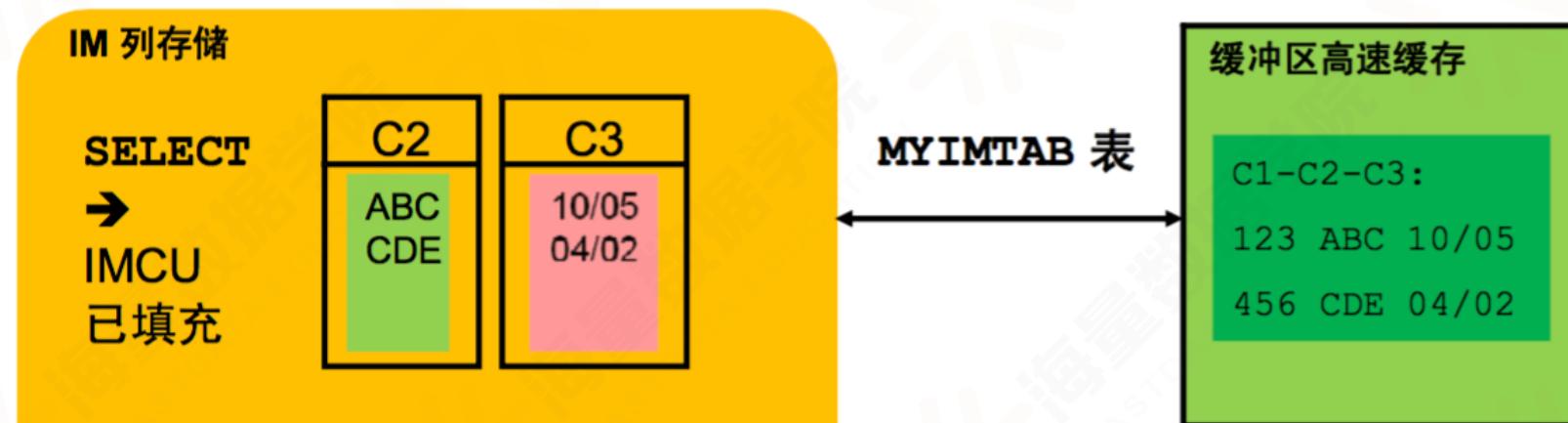
# 部署 IM 列存储：列设置

- 启用/禁用部分列：

```
SQL> CREATE TABLE myimtab (c1 NUMBER, c2 CHAR(2), c3 DATE)  
      INMEMORY NO INMEMORY (c1);
```

```
SQL> ALTER TABLE myimtab NO INMEMORY (c2);
```

```
SQL> CREATE TABLE t (c1 NUMBER, c2 CHAR(2)) NO INMEMORY INMEMORY (c2);  
ORA-64361: column INMEMORY clause may only be specified for an inmemory  
table
```



# IM 列存储的候选对象

DBA\_TABLES 视图显示 IM 列存储的候选表：

```
SQL> SELECT table_name tab, inmemory_compression Comp,
     inmemory_priority Priority,
     inmemory_distribute RAC, inmemory_duplicate DUP
   FROM dba_tables;
```

TABLE	COMP	PRIORITY	RAC	DUP
TEST1	FOR QUERY HIGH	NONE	AUTO	DUPLICATE ALL
TEST2	NO MEMCOMPRESS	CRITICAL	AUTO	DUPLICATE
EMP	FOR DML	LOW	BY PARTITION	NO DUPLICATE

- NO MEMCOMPRESS:  
未压缩
- FOR QUERY / FOR CAPACITY:  
已压缩

- NONE: 按需填充
- 其他值: 根据优先级填充

- AUTO: Oracle 在 RAC 节点的 IM 列存储间分布对象
- BY PARTITION: 分区在 RAC 节点间分布对象

- DUPLICATE ALL
- DUPLICATE
- NO DUPLICATE

# IM 列存储的候选列

V\$IM\_COLUMN\_LEVEL 视图还显示内存中表中并非 IM 列存储候选项的列：

```
SQL> SELECT obj_num, segment_column_id, inmemory_compression  
FROM v$im_column_level;
```

内存中列  
覆盖父表的压缩子句

obj_num	segment_column_id	inmemory_compression
98202	1	DEFAULT
98202	2	NO MEMCOMPRESS
98202	3	DEFAULT
98202	4	NO INMEMORY

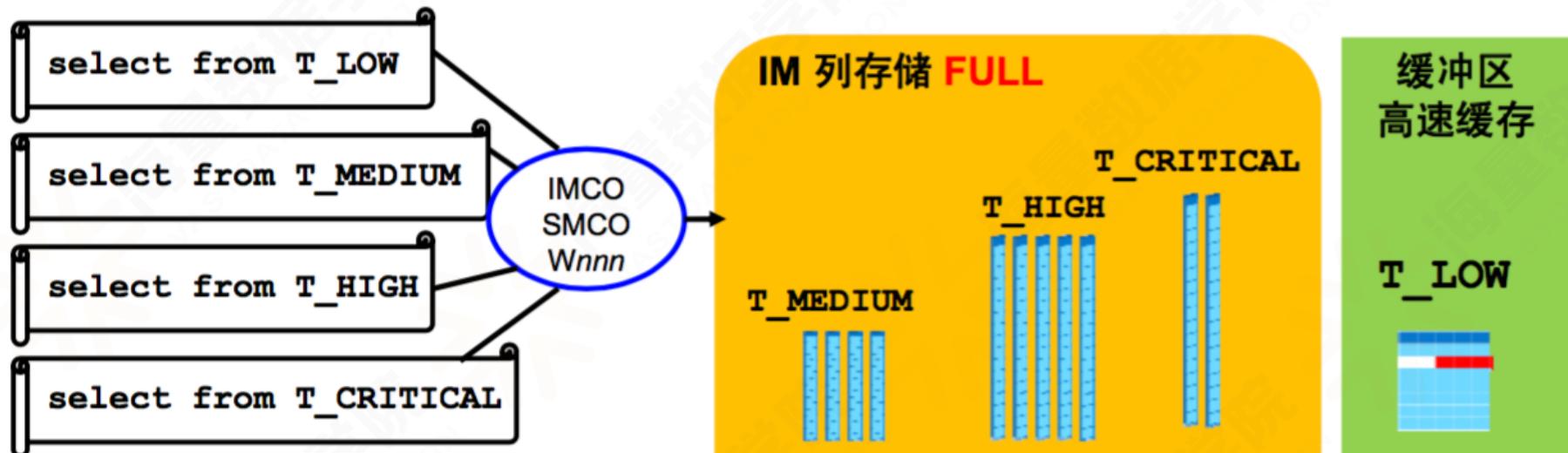
内存中表的内存中列  
继承父表的压缩子句

内存中表中的非内存中列

# 定义 IM 列存储优先级

使用 **PRIORITY** 可以定义填充优先级。

```
SQL> CREATE TABLE t_low (code NUMBER) INMEMORY PRIORITY LOW;
```



```
SQL> CREATE TABLE countries_part ... PARTITION BY LIST ...
( PARTITION p1 .. INMEMORY PRIORITY HIGH,
  PARTITION p2 .. INMEMORY MEMCOMPRESS FOR CAPACITY LOW,
  PARTITION p3 .. , .. );
```

# 填充到 IM 列存储中的段

- 查看填充到 IM 列存储中的段：

```
SQL> SELECT owner, segment_name name FROM v$im_segments  
      WHERE segment_name = 'SALES';
```

优先级为 NONE →  
尚未填充任何行

```
SQL> SELECT /*+ full(s) noparallel (s) */ count(*) FROM sales s;
```

- 检查段是否已完全填充到 IM 列存储中：
- 计算磁盘上的块与 IM 列存储中的块：

```
SQL> SELECT segment_name, bytes, inmemory_size,  
          bytes_not_populated  
    FROM v$im_segments;
```

SEGMENT_NAME	BYTES	INMEMORY_SIZE	BYTES_NOT_POPULATED
SALES	228231	36299	0
T1	11475615744	4664262656	5004262678

字节数

IM 列存储中的字节数

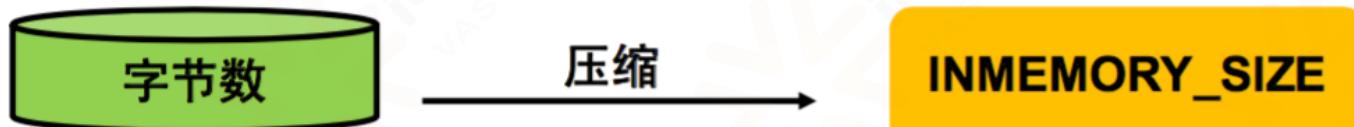
# 定义 IM 列存储压缩

- 使用 **MEMCOMPRESS** 可以定义压缩级别。

```
SQL> CREATE MATERIALIZED VIEW mv1 INMEMORY NO MEMCOMPRESS;
```

```
SQL> CREATE TABLE large_tab (c1 ...) INMEMORY  
      MEMCOMPRESS FOR QUERY;
```

- 将磁盘上存储的字节数与列格式压缩的字节数进行比较：



- 将 **MEMCOMPRESS** 更改为较高的压缩级别：

```
SQL> ALTER TABLE t1 INMEMORY MEMCOMPRESS FOR CAPACITY HIGH;
```

SEGMENT_NAME	BYTES	INMEMORY_SIZE
T1	11475615744	4664262656



无自动  
重新填充

# IM 列存储压缩指导

压缩指导可以分析要使用不同级别为表启用 INMEMORY MEMCOMPRESS, IM 列存储中需要多少空间。

```
BEGIN  
    DBMS_COMPRESSION.GET_COMPRESSION_RATIO (  
        'TS_DATA', 'SSB', 'LINEORDER', NULL,  
        DBMS_COMPRESSION.COMP_INMEMORY_QUERY_LOW,  
        blkcnt_cmp, blkcnt_ncmp, row_cmp, row_ncmp, cmp_ratio,  
        comptype_str,10000,1);  
    DBMS_OUTPUT.PUT_LINE('Block count uncompressed = ' || blkcnt_ncmp);  
    DBMS_OUTPUT.PUT_LINE('Row count per block uncompressed=' || row_ncmp);  
    DBMS_OUTPUT.PUT_LINE('Compression type = ' || comptype_str);  
    DBMS_OUTPUT.PUT_LINE('Comp ratio= '|| cmp_ratio );  
    DBMS_OUTPUT.PUT_LINE('      ');  
    DBMS_OUTPUT.PUT_LINE('The IM column store space needed is about 1 byte  
in IM column store for '|| cmp_ratio ||' bytes on disk.' );  
END;  
/
```

# 计算压缩率

使用 V\$IM\_SEGMENTS 视图可以显示压缩率：

```
SQL> SELECT segment_name, bytes Disk, inmemory_size,  
       inmemory_compression COMPRESSION,  
       bytes / inmemory_size comp_ratio  
  FROM v$im_segments;
```

OBJECT_NAME	DISK	INMEMORY_SIZE	COMPRESSION	COMP_RATIO
SALES	212284915	200104115	FOR QUERY LOW	1.06
EMPLOYEES	3456956	17984	FOR CAPACITY HIGH	192.22



越高越好： 20



越低越差： 1.06

1 字节

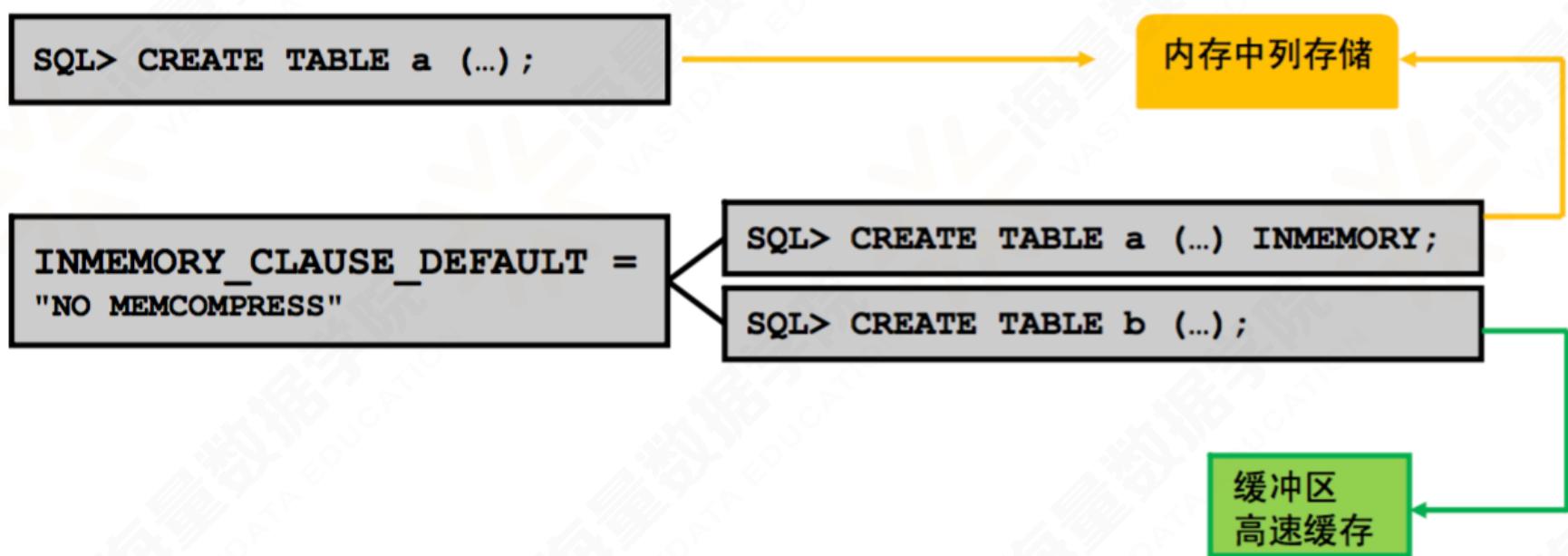
对应于



# 默认内存中设置

分配新的表默认内存中属性：

```
INMEMORY_CLAUSE_DEFAULT = "INMEMORY MEMCOMPRESS FOR QUERY LOW"
```

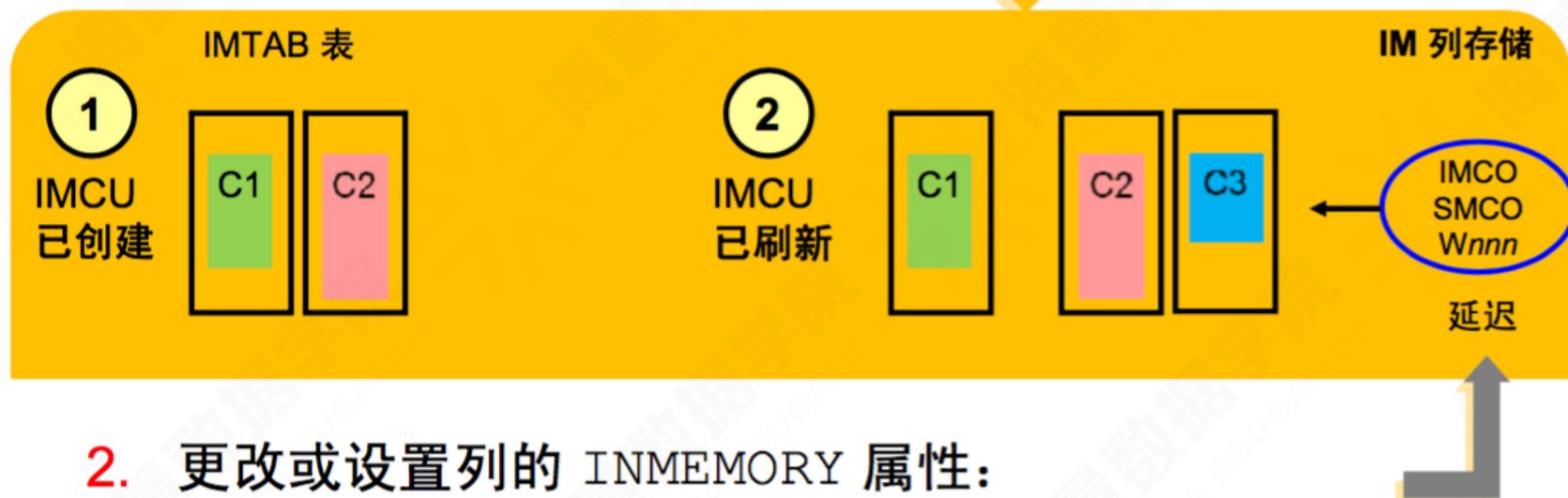


# 更改内存中属性的影响

## 1. 创建了一个内存中表:

```
SQL> CREATE TABLE imtab (c1 number, c2 VARCHAR2(30), c3 DATE)  
      INMEMORY NO INMEMORY(c3);
```

```
SQL> INSERT INTO imtab VALUES (...)  
SQL> SELECT c1, c2 FROM imtab;
```



## 2. 更改或设置列的 INMEMORY 属性:

```
SQL> ALTER TABLE imtab INMEMORY(c3);
```

# 移动或拆分内存中段

## 1. 创建了一个内存中段:

```
SQL> CREATE TABLE imt (c1 number, c2 VARCHAR2(30), c3 DATE) INMEMORY;  
SQL> INSERT INTO imt ...      SQL> SELECT * FROM imt;
```



## 2. 移动内存中表或拆分内存中分区:

```
SQL> ALTER TABLE imt MOVE;  
SQL> SELECT * FROM imt;
```



# INMEMORY 继承

表空间: DEFAULT INMEMORY / COMPRESS / PRIORITY 属性

DBA\_TABLESPACES

所有表: INMEMORY / 其他 COMPRESS / 其他 PRIORITY

NO INMEMORY 表除外

DBA\_TABLES

所有分区: INMEMORY / 其他 COMPRESS / 其他 PRIORITY

NO INMEMORY 分区除外

DBA\_PART\_TABLES

所有列: INMEMORY / 其他 COMPRESS / 其他 PRIORITY

NO INMEMORY 列除外

```
SQL> CREATE TABLESPACE IMCS ... DEFAULT INMEMORY  
      MEMCOMPRESS FOR CAPACITY HIGH PRIORITY LOW;
```

# 对象设置之后

删除分析索引。

```
SQL> DROP INDEX i_sales_timeid PURGE;
```

```
SQL> DROP INDEX i_sales_chanid PURGE;
```



# 检索内存中对象的 CREATE DDL 语句

```
SQL> SELECT DBMS_METADATA.GET_DDL ('TABLE','CUSTOMER','SSB')  
      FROM dual;  
  
DBMS_METADATA.GET_DDL('TABLE','CUSTOMER','SSB')  
-----  
CREATE TABLE "SSB"."CUSTOMER"  
  ( "C_CUSTKEY" NUMBER,    "C_NAME" VARCHAR2(25),  
    "C_ADDRESS" VARCHAR2(25),   "C_CITY" CHAR(10),  
    "C_NATION" CHAR(15),    "C_REGION" CHAR(12),  
    "C_PHONE" CHAR(15),    "C_MKTSEGMENT" CHAR(10)  
  ) SEGMENT CREATION IMMEDIATE  
    PCTFREE 10 PCTUSED 40 INITTRANS 1 MAXTRANS 255  
  NOCOMPRESS LOGGING ...  
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE  
  DEFAULT) TABLESPACE "TS_DATA"  
  INMEMORY PRIORITY MEDIUM MEMCOMPRESS FOR QUERY LOW  
    AUTO DISTRIBUTE  
  CACHE
```

# 查询获益

适用于：

- 扫描大量行并应用过滤器
- 查询表中的部分列
- 将小型表（维）与大型表（事实）联接
  - 在事实表中利用重复的维值
- 聚集数据

# 测试和比较查询性能

- 测试在 IM 列存储中对内存中表执行的查询:

```
SQL> SET timing ON  
SQL> SELECT max(lo_ordtotalprice) most_expensive_order  
      FROM lineorder;
```



用时: 00:00:01.80

对应用程序透明

- 针对缓冲区高速缓存重新执行相同的查询:

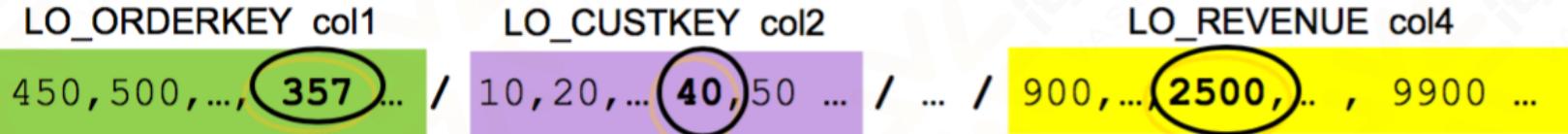
```
SQL> ALTER SESSION SET INMEMORY_QUERY="DISABLE";  
SQL> SELECT max(lo_ordtotalprice) most_expensive_order  
      FROM lineorder;
```



用时: 00:12:21.90

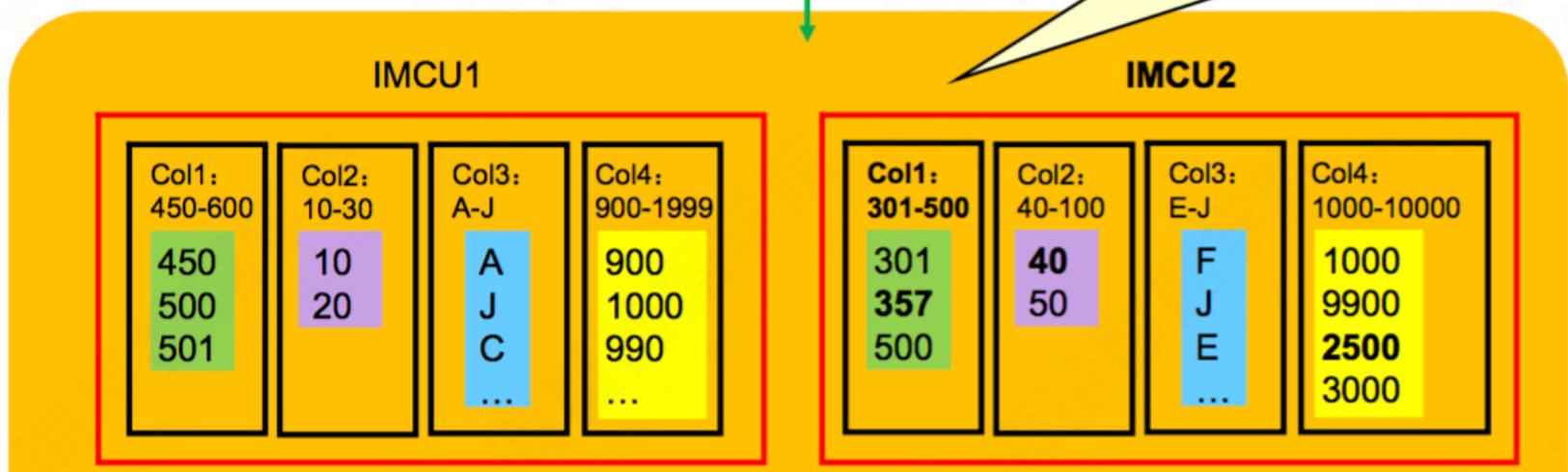
使用 INMEMORY\_QUERY 会话参数针对缓冲区  
高速缓存执行查询

# 对内存中表的查询：简单谓词



```
SQL> SELECT lo_orderkey, lo_custkey, lo_revenue  
  FROM lineorder  
 WHERE lo_orderkey = 357;
```

IMCU 修剪：使用每个 IMCU 中存储的最小值和最大值



# MINMAX 修剪统计信息

是否会发生 MINMAX 修剪？

MINMAX 修剪针对包含等式和不等式的 WHERE 子句进行。

```
SQL> SELECT display_name, value
  FROM v$mystat m, v$statname n
 WHERE m.statistic# = n.statistic#
 AND display_name IN (
    'IM scan segments minmax eligible',
    'IM scan CUs pruned',
    'IM scan CUs optimized read',
    'IM scan CUs predicates optimized');
```

DISPLAY_NAME	VALUE
-----	
IM scan segments minmax eligible	250
IM scan CUs pruned	249
IM scan CUs optimized read	0
IM scan CUs predicates optimized	249

# IM 列存储统计信息

- 使用 V\$MYSTAT 和 V\$STATNAME: %IM% 统计信息。

```
SQL> SELECT display_name, value
      FROM v$mystat m, v$statname n
     WHERE m.statistic# = n.statistic#
       AND display_name IN (
          'session logical reads - IM',
          'IM scan rows', 'IM scan rows valid',
          'IM scan blocks cache',
          'IM scan CUs columns accessed' );
```

DISPLAY_NAME	VALUE
session logical reads - IM	3063604
IM scan rows	211307905
IM scan rows valid	879059
IM scan blocks cache	0
IM scan CUs columns accessed	3

注：如果未在 IM 列存储中执行查询，则所有统计信息均显示为 0。

# 执行计划: TABLE ACCESS IN MEMORY FULL

在 IM 列存储中还是在缓冲区高速缓存中执行?

```
SQL> SELECT * FROM table(dbms_xplan.display_cursor());  
-----  
| Id  | Operation          | Name      |  
-----  
...  
|*  4 | TABLE ACCESS INMEMORY FULL    | LINEORDER |  
-----  
Predicate Information (identified by operation id):  
-----  
4 - inmemory ("LO_ORDERKEY"=357) filter("LO_ORDERKEY"=357)
```

在 IM 列存储中执行

Predicate Information (identified by operation id):

4 - inmemory ("LO\_ORDERKEY"=357) filter("LO\_ORDERKEY"=357)

针对缓冲区高速缓存执行

```
...  
|*  4 | TABLE ACCESS FULL  
-----  
4 - access (:Z>=:Z AND :Z<=:Z) filter("LO_ORDERKEY"=357)
```

# 对内存中表的查询：联接

查询多个内存中表：

```
SQL> SELECT SUM(lo_extendedprice * lo_discount) revenue  
      FROM lineorder l, date_dim d  
     WHERE l.lo_orderdate = d.d_datekey  
       AND l.lo_discount BETWEEN 2 AND 3  
       AND l.lo_quantity < 24  
       AND d.d_date='December 24, 1996';
```



用时：00:00:00.28

- 比较针对缓冲区高速缓存的相同查询的性能：

```
SQL> SELECT SUM(lo_extendedprice * lo_discount) revenue  
      FROM lineorder l, date_dim d  
     WHERE l.lo_orderdate = d.d_datekey  
       AND l.lo_discount BETWEEN 2 AND 3  
       AND l.lo_quantity < 24  
       AND d.d_date='December 24, 1996';
```



用时：00:02:28.85

# 执行计划： JOIN FILTER CREATE / USE

Id	Operation	
0	SELECT STATEMENT	
1	SORT AGGREGATE	
* 2	HASH JOIN	
3	JOIN FILTER CREATE	
* 4	TABLE ACCESS INMEMORY FULL	:BF0000 DATE_DIM
5	JOIN FILTER USE	:BF0000
* 6	TABLE ACCESS INMEMORY FULL	LINEORDER

```
2 - access("L"."LO_ORDERDATE"="D"."D_DATEKEY")
4 - inmemory ("D"."D_DATE"='December 24, 1996')
      filter ("D"."D_DATE"='December 24, 1996')
6 - inmemory ((("L"."LO_DISCOUNT" <=3 AND L"."LO_QUANTITY" < 24
                AND "L"."LO_DISCOUNT" >= 2
                AND SYS_OP_BLOOM_FILTER(:BF0000, "L"."LO_ORDERDATE")))
```

如果针对缓冲区高速缓存  
查询表: access

如果针对缓冲区高速缓存  
查询表:  
**TABLE ACCESS FULL**

# 对内存中表和非内存中表进行查询

```
SQL> SELECT (lo_extendedprice * lo_discount) revenue  
  FROM inmem_lineorder l, noinmem_date_dim d  
 WHERE l.lo_orderdate=d.d_datekey AND d.d_date='December 24, 1996';
```



Id   Operation	Name
0   SELECT STATEMENT	
* 1   HASH JOIN	
2   JOIN FILTER CREATE	
* 3   TABLE ACCESS FULL	:BF0000
4   JOIN FILTER USE	NOINMEM_DATE_DIM
* 5   TABLE ACCESS INMEMORY FULL	:BF0000
	INMEM_LINEORDER

# 对内存中列和非内存中列进行查询

- 对内存中表的非内存中列进行查询：

```
SQL> SELECT noinmem_lo_revenue  
      FROM inmem_lineorder;    → 缓冲区  
                                         高速缓存 → IM 统计信息 = 0 
```

- 对内存中表的非内存中列和内存中列进行查询：

```
SQL> SELECT noinmem_custkey,  
           inmemlo_revenue  
      FROM inmem_lineorder;    → 缓冲区  
                                         高速缓存 → IM 统计信息 = 0 
```

DISPLAY_NAME	VALUE
<hr/>	
IM scan rows	0
IM scan rows valid	0
IM scan blocks cache	0
IM scan CUs columns accessed	0

# DML 和内存中列存储

- 内存中列存储可以管理 DML 事务处理。

```
SQL> SELECT display_name, value FROM v$mystat m, v$statname n
      WHERE m.statistic# = n.statistic#
      AND     display_name like 'IM transactions%';
```

DISPLAY_NAME	VALUE
IM transactions	1
IM transactions rows journaled	10224
...	

- 内存中列存储可以管理批量加载。
  - 表的列式表示形式
  - 同一表的行格式表示形式
    - 如果删除了分析索引
    - 如果保留了分析索引
- 与 OLTP 完全兼容。



# 建议

查询的性能提升如下：

- IM 列存储和并行化
- 仅 IM 列存储
- 无 IM 列存储但有并行化
- 无 IM 列存储也无并行化



可能无需优化

启用 IM 列存储意味着：

- 查找有多少内存可供 IM 列存储使用
- 计算要在 IM 列存储中加载的  $\Sigma$  (对象的估计大小)
- 不在表空间级别设置 DEFAULT INMEMORY  
PRIORITY xxx



# 视图

- V\$IM\_SEGMENTS / V\$IM\_USER\_SEGMENTS /  
V\$IM\_SEGMENTS\_DETAIL: 所有用户段的内存中状态
- V\$IM\_COLUMN\_LEVEL: 内存中段的所有列的内存中状态
- V\$IM\_TBS\_EXT\_MAP: 提供 TSN 和数据对象 ID 的磁盘上 DBA 范围的内存中区
- V\$IM\_SEG\_EXT\_MAP: 通过 TSN、磁盘上 DBA 和数据对象编号映射，该视图将列出所有内存中区
- V\$IM\_COL\_CU: 每一行都包含有关特定列的内存中列单元的信息（包括最小值和最大值）

# 与其他产品的交互

优化程序

RAC 环境

内存中并行执行

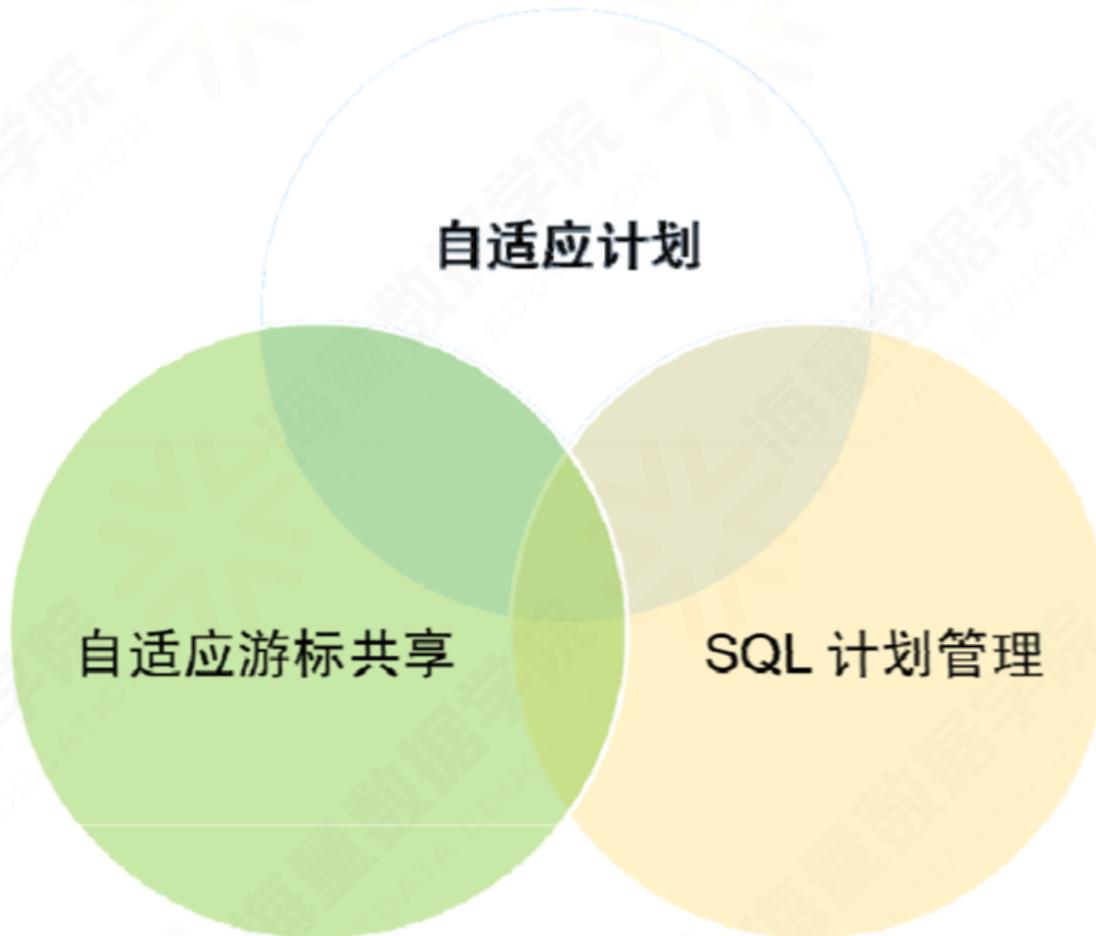
数据泵



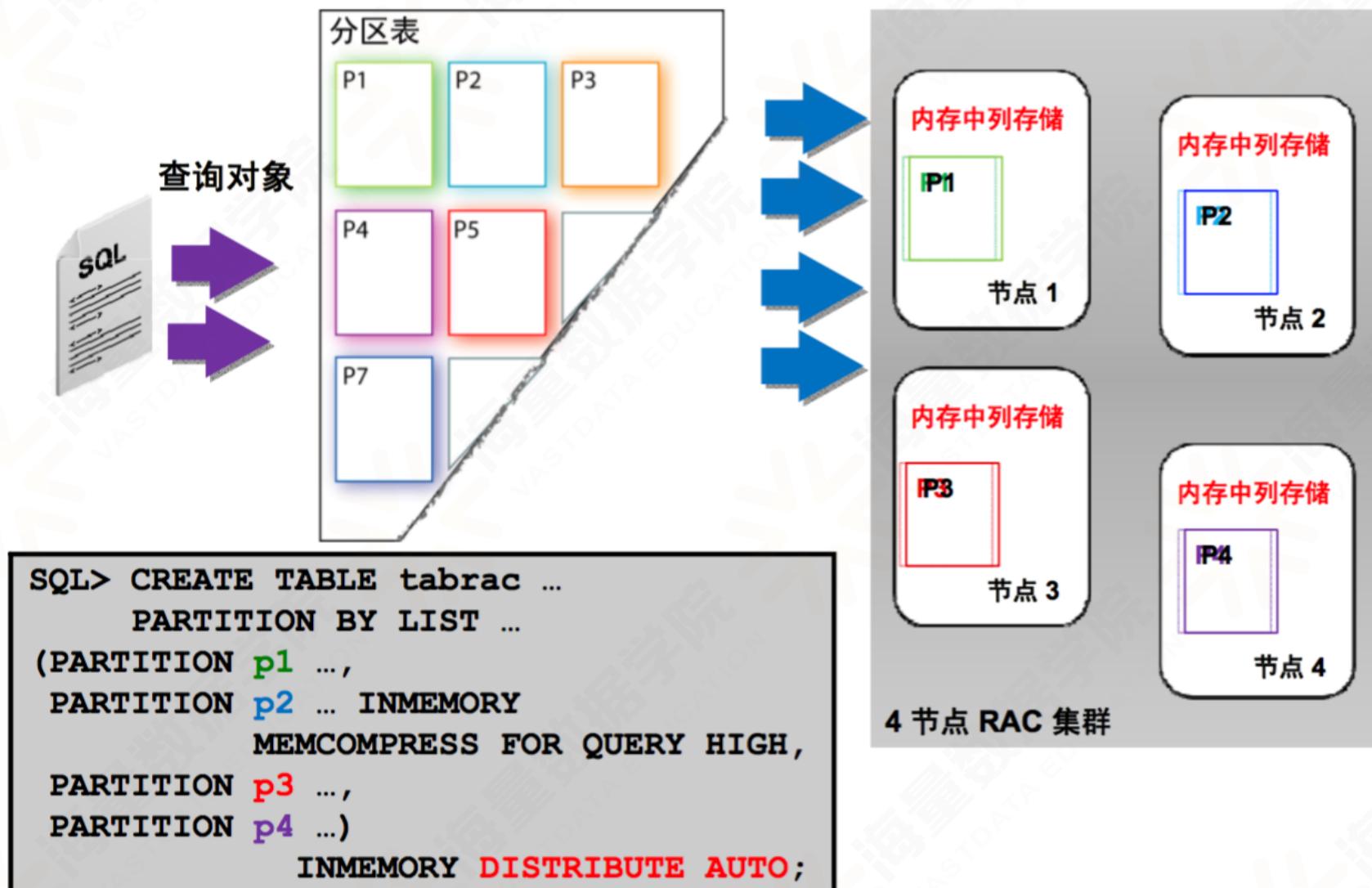
海量数据  
VASTDATA

海量数据学院  
VASTDATA EDUCATION

# 优化程序



# IM 列存储和 RAC



# IM 列存储和数据泵

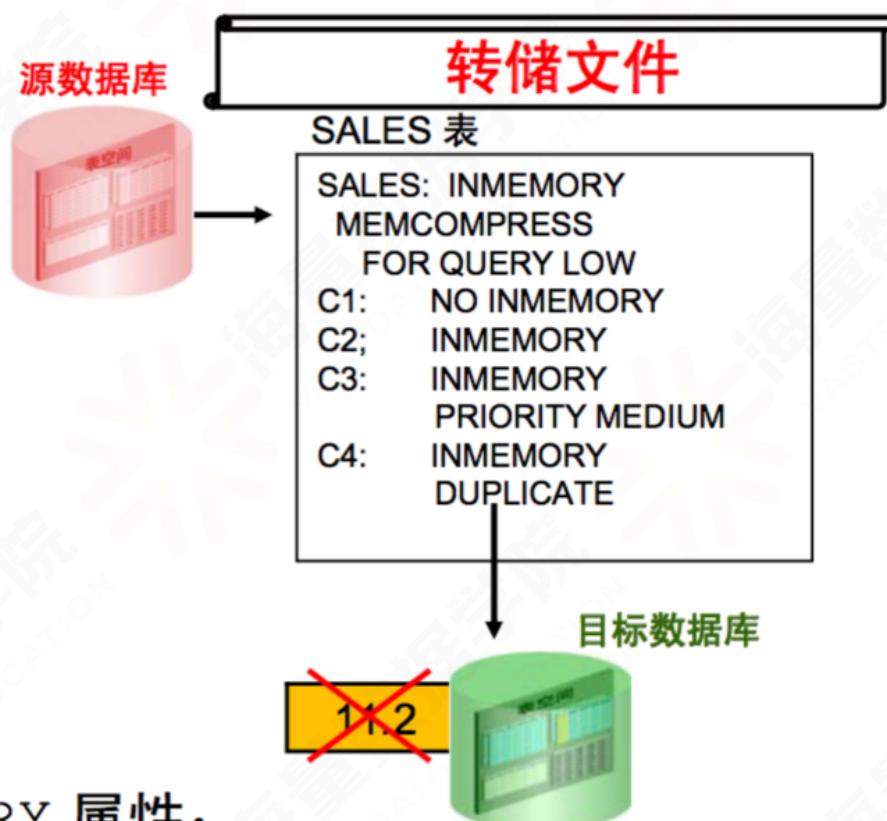
- 导出时将传输 INMEMORY 属性:

DBA\_TABLESPACES

DEFAULT\_INMEMORY\_COMPRESSION  
DEFAULT\_INMEMORY\_PRIORITY  
DEFAULT\_INMEMORY\_DISTRIBUTE  
DEFAULT\_INMEMORY\_DUPLICATE

DBA\_TABLES

DBA\_TAB\_PARTITIONS  
DBA\_TAB\_SUBPARTITIONS  
INMEMORY\_COMPRESSION  
INMEMORY\_PRIORITY  
INMEMORY\_DISTRIBUTE  
INMEMORY\_DUPLICATE



- 导入时重新生成 INMEMORY 属性:

# 数据泵 TRANSFORM 名称

- 默认情况下，数据泵导入时将传输 INMEMORY 属性。
- 导入 INMEMORY 对象时可保留/不保留 INMEMORY 属性：

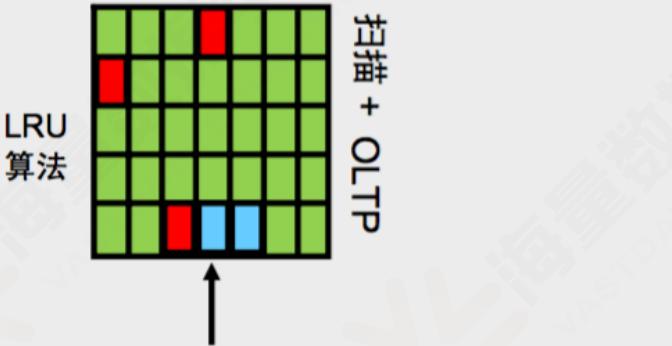
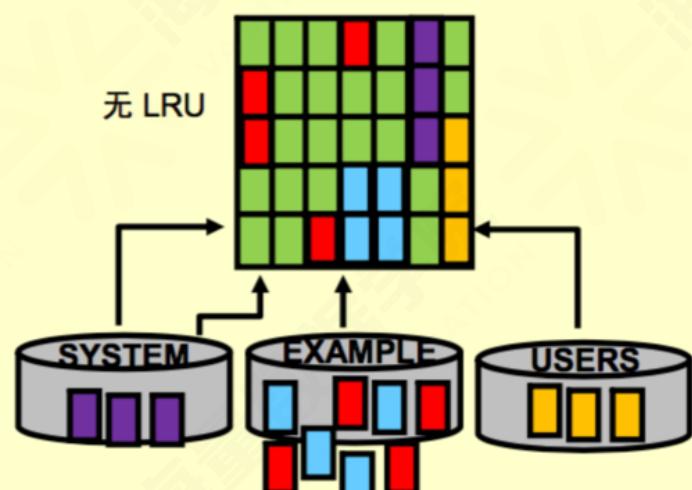
```
$ impdp ... TRANSFORM=INMEMORY:Y|N
```

- 导入 INMEMORY 对象时将传输 INMEMORY 属性：

```
$ impdp ... TRANSFORM=INMEMORY_CLAUSE:\  
"INMEMORY MEMCOMPRESS FOR CAPACITY HIGH  
PRIORITY MEDIUM\"
```

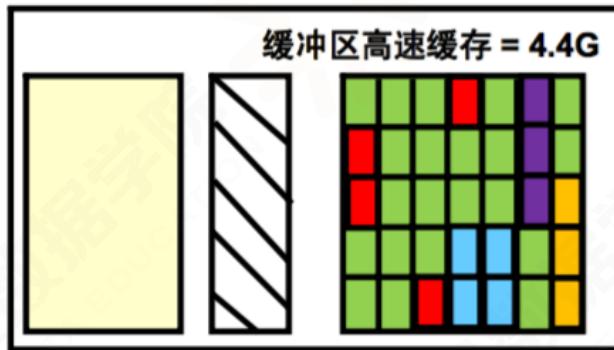
```
$ impdp ... TRANSFORM=INMEMORY_CLAUSE:\\"NO INMEMORY\\"
```

# 整个数据库内存中高速缓存

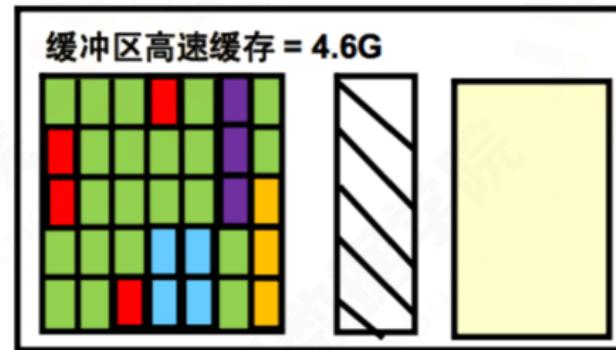
传统缓冲区高速缓存使用情况	整个数据库内存中高速缓存
<p><b>DB_CACHE_SIZE = 10g</b></p>  <p>LRU 算法</p> <p>加载到缓冲区高速缓存中 如果表大小 &lt; 缓冲区 高速缓存大小的很小百分比</p> <p>表 HR.EMPLOYEES 表 SH.SALES</p>	<p>将整个数据库加载到缓冲区高速缓存中：</p> <ul style="list-style-type: none"> <li>• 巨大的性能优势</li> <li>• 两种模式             <ul style="list-style-type: none"> <li>- 整个数据库高速缓存</li> <li>- 强制整个数据库高速缓存</li> </ul> </li> </ul>  <p>无 LRU</p>

# 设置强制整个数据库高速缓存

实例 1



实例 2



所用数据库空间 = 10g

- 所用数据库空间的逻辑大小 < 每个实例的缓冲区高速缓存
- 逻辑数据库大小 < 80% ( $\Sigma$  所有实例的缓冲区高速缓存大小)

```
SQL> STARTUP MOUNT  
SQL> ALTER DATABASE FORCE FULL DATABASE CACHING;  
SQL> SELECT force_full_db_caching FROM v$database;
```



备份控制文件

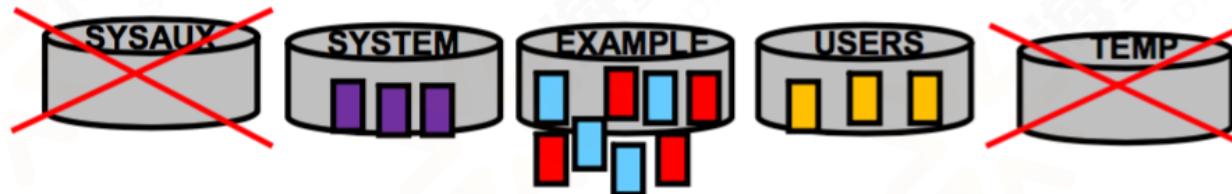
# 监视整个数据库内存中高速缓存

1. 针对 FORCE\_FULL\_DB\_CACHING 选择了数据库。
2. 数据库是否整个加载到缓冲区高速缓存中?
  - a. 当前缓冲区高速缓存大小是多少?

```
SQL> SELECT sum(cnum_set * blk_size) BC FROM X$KCBWDS;
```

12G

- b. 存储的数据大小是多少?



使用的空间 = 10g

$$\Sigma (\text{所有数据文件} - \text{SYSAUX 数据文件} - \text{TEMP 临时文件})$$

- c. 设置 SGA\_SIZE 或 DB\_CACHE\_SIZE, 以使存储的数据大小 < 80% 的缓冲区高速缓存:

```
SQL> ALTER SYSTEM SET SGA_TARGET = 13G SCOPE=SPFILE;
```

- d. 重新启动实例。

# 自动大表高速缓存之前的内存中并行查询

SQL 语句



确定正在查看  
的表的大小。

表非常小。



表非常适合内存中  
并行执行。



将表的碎片读取到每个  
节点的缓冲区高速缓存。



读取到任意节点上的  
缓冲区高速缓存。



始终使用从磁盘直接读取。

# 自动大表高速缓存

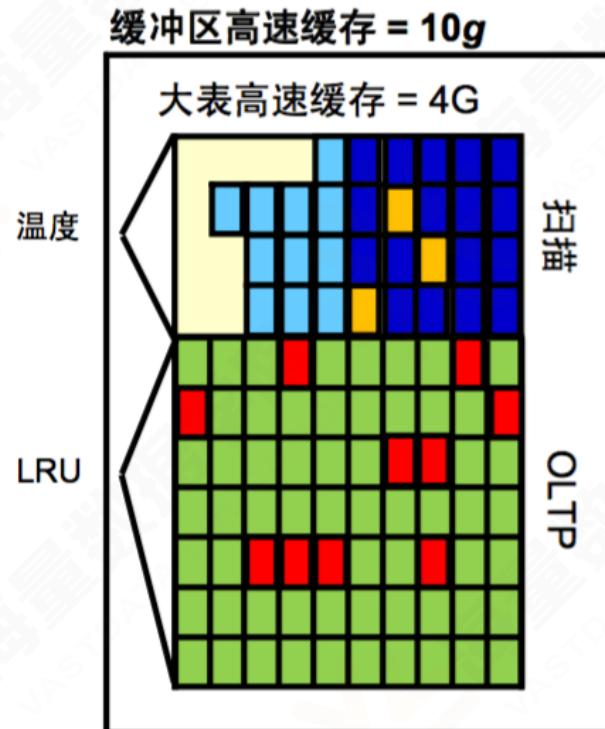
- 提高大小无法完全存入缓冲区高速缓存的大对象的内存中 PX 性能
  - 消除大对象的磁盘读取
  - 促进大对象在数据仓库环境中的高效高速缓存
  - 将扫描的对象存储在大表高速缓存中而非缓冲区高速缓存中
    - 非分区表
    - 分区表的分区和子分区
- 使用不同的高速缓存替换算法

	替换基础	替换级别
9i、10g、11g、12.1.0.1	基于访问	块级别
12.1.0.2	基于温度	对象级别

# 配置自动大表高速缓存

配置数据库缓冲区高速缓存的一部分：

- PARALLEL\_DEGREE\_POLICY=AUTO 或 ADAPTIVE
- DB\_BIG\_TABLE\_CACHE\_PERCENT\_TARGET=40



- 选择“热”对象：

- 表 HR.EMPLOYEES: 热对象  $30^{\circ}$
- 表 SH.SALES: 热对象  $80^{\circ}$
- 表 SH.BIG: 热对象  $40^{\circ}$

当大表高速缓存中需要空间时，  
HR.EMPLOYEES 表将替换为  
温度更高的新对象。

- 避免发生崩溃：高速缓存部分对象

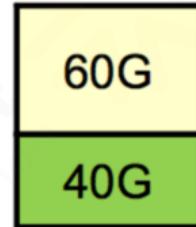
# 使用自动大表高速缓存

DB\_CACHE\_SIZE=100G

DB\_BIG\_TABLE\_CACHE\_PERCENT\_TARGET=60

→ 大表扫描使用 60G。

→ OLTP 使用 40G。



工作量更改... → 增加大表高速缓存

```
SQL> ALTER SYSTEM SET db_big_table_cache_percent_target=80;
```

DB\_BIG\_TABLE\_CACHE\_PERCENT\_TARGET=80

→ 大表扫描使用 80G。

→ OLTP 使用 20G。



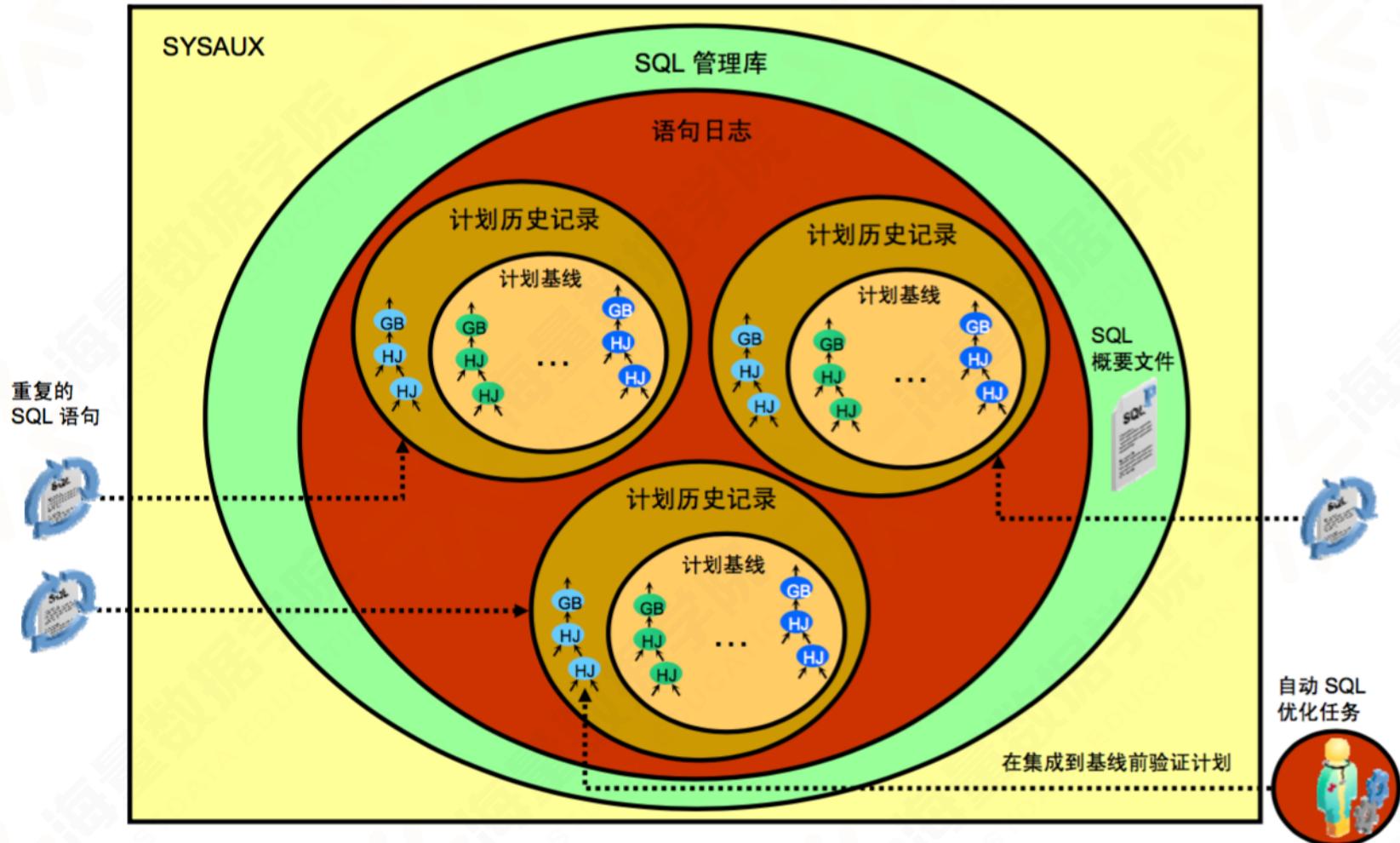
# 监视自动大表高速缓存

下面两个视图提供了有关自动大表高速缓存的详细信息：

- V\$BT\_SCAN\_CACHE
- V\$BT\_SCAN\_OBJ\_TEMPS



# SQL 计划基线：体系结构



# SQL 计划管理：概览

SQL 计划管理可以确保运行时性能不因执行计划的更改而下降。

## 优化程序自动管理“执行计划”

- 仅使用已知的计划
- 可以手动或自动捕获计划

## 计划更改已验证

- 仅使用性能更好的计划
- 可通过 EM 或在 Oracle Database 11g 中运行 DBMS\_SPM.EVOLVE\_SQL\_PLAN\_BASELINES 执行手动验证
- 通过夜晚在 Oracle Database 11g 中运行 DBMS\_SPM.EVOLVE\_SQL\_PLAN\_BASELINES 的自动 SQL 优化 (Automatic SQL Tuning, AST) 作业执行自动验

## SPM 由以下部分组成：

- SQL 计划基线捕获
- SQL 计划基线选择
- SQL 计划基线演化

# 自适应 SQL 计划管理

- 新的演化自动任务在夜间维护窗口运行。
- 它会对所有未接受的计划进行排名，并为其运行演化过程。新发现的计划排名最高。
- 执行较差的计划在 30 天内不重试。此后，仅当语句处于活动状态时，才会重试这些计划。
- 新任务为 `SYS_AUTO_SPM_EVOLVE_TASK`。
- 有关任务的信息在 `DBA_ADVISOR_TASKS` 中。
- 使用 `DBMS_SPM.REPORT_AUTO_EVOLVE_TASK` 可查看自动作业的结果。

# 自动演化 SQL 计划基线

## SPM 演化指导

- 演化功能现在是一个指导任务。
- 通过新函数，DBA 可以创建任务、任务中的一个或多个演化计划等。

## DBMS\_SPM 基于任务的函数

- DBMS\_SPM.CREATE\_EVOLVE\_TASK
- DBMS\_SPM.EXECUTE\_EVOLVE\_TASK
- DBMS\_SPM.REPORT\_EVOLVE\_TASK
- DBMS\_SPM.IMPLEMENT\_EVOLVE\_TASK

## 永久存储计划演化报告

- 这样可以随时实施指导任务基础结构。



# SQL 管理库增强功能

- 在 Oracle Database 12c 之前，  
DBMS\_XPLAN.DISPLAY\_SQL\_PLAN\_BASELINE 通过  
以基线编译语句显示执行计划。12c 之前版本
- 在 Oracle Database 12c 中，将新计划添加到 SQL 语  
句的计划历史记录中时，计划行也会存储在 SQL 管理  
库 (SQL management base, SMB) 中。执行  
DBMS\_XPLAN.DISPLAY\_SQL\_PLAN\_BASELINE 后，  
可从 SMB 提取计划数据。12c
- 无法复制计划时更易诊断。

# 自适应执行计划

- 由于运行时条件指示优化程序估计值不准确，因此在执行过程中查询计划会发生更改。
- 所有自适应执行计划都依赖于在查询执行期间收集的统计信息。
- 自适应计划技术有两个：
  - 动态计划
  - 重新优化
- 将 OPTIMIZER\_FEATURES\_ENABLE 设置为 12.1.0.1 或更高版本，并且将 OPTIMIZER\_ADAPTIVE\_REPORTING\_ONLY 初始化参数设置为默认值 FALSE 时，数据库会使用自适应执行计划。

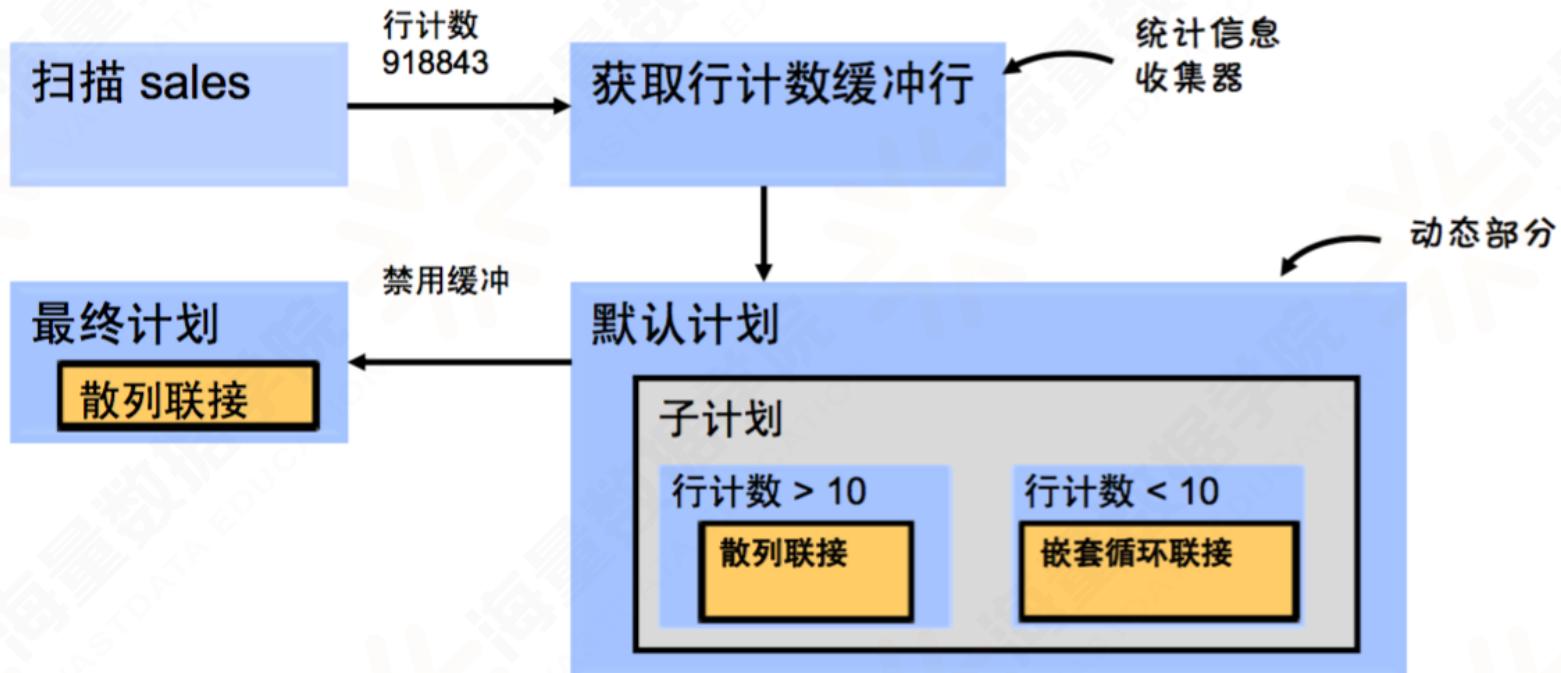


# 动态计划

- 最终决策基于在执行期间收集的统计信息。
- 替代子计划已预先计算并存储在游标中。
- 会在计划的关键点插入统计信息收集器。
- 如果统计信息证实超出范围，则可以交换子计划。
- 这要求在临近交换点时进行缓冲以避免将行返回给用户。
- 仅联接方法和分布方法可以更改。

# 动态计划：自适应过程

```
SQL> SELECT s.cust_id, c.cust_last_name  
  FROM sh.sales s, sh.customers c  
 WHERE s.cust_id = c.cust_id;
```



# 动态计划：示例

## A.1. The default plan show NLJ

```
SQL> explain plan for
select /*Q10*/ product_name
from order_items o, product_information p
where o.unit_price = 15
and quantity > 1
and p.product_id = o.product_id;
```

Explained.

```
SQL> select * from table(dbms_xplan.display(format=>'basic +note'));
```

### PLAN\_TABLE\_OUTPUT

Plan hash value: 384646879

Id   Operation	Name
0   SELECT STATEMENT	
1   NESTED LOOPS	
2   NESTED LOOPS	
3   TABLE ACCESS FULL	ORDER_ITEMS
4   INDEX UNIQUE SCAN	PRODUCT_INFORMATION_PK
5   TABLE ACCESS BY INDEX ROWID	PRODUCT_INFORMATION

## A.2. The final plan shows HJ

```
SQL> select /*Q10*/ product_name
from order_items o, product_information p
where o.unit_price = 15
and quantity > 1
and p.product_id = o.product_id;
```

### PRODUCT\_NAME

Screws <B.28.S>

13 rows selected.

```
SQL> select * from table(dbms_xplan.display_cursor(format=>'basic +note'));
```

### PLAN\_TABLE\_OUTPUT

#### EXPLAINED SQL STATEMENT:

```
select /*Q10*/ product_name from order_items o, product_information p
where o.unit_price = 15 and quantity > 1 and p.product_id =
o.product_id
```

Plan hash value: 2615131494

Id   Operation	Name
0   SELECT STATEMENT	
1   HASH JOIN	
2   TABLE ACCESS FULL  ORDER_ITEMS	
3   TABLE ACCESS FULL  PRODUCT_INFORMATION	

## 重新优化：统计信息反馈

- 统计信息反馈（以前称为“基数反馈”）是在 Oracle Database 11g 发行版 2 中引入的。
- 统计信息反馈对于要处理的数据卷在一段时间内处于稳定状态的查询十分有用。
- 在查询执行期间，会将优化程序估计值与执行统计信息进行比较：如果二者差异很大，将选择新的计划用于后续执行。

注：CURSOR\_SHARING 参数为 EXACT 或 FORCE。



# 统计信息反馈：监视查询执行

```
SQL> SELECT /*+ gather_plan_statistics */ product_name
  FROM order_items o, product_information p
 WHERE o.unit_price = 15
   AND quantity > 1
   AND p.product_id = o.product_id;
```

Id	Operation	Name	Starts	E-Rows	A-Rows
0	SELECT STATEMENT		1	13	13
1	NESTED LOOPS		1	13	13
2	NESTED LOOPS		1	4	13
3	TABLE ACCESS FULL	ORDER_ITEMS	1	4	13
4	INDEX UNIQUE SCAN	PRODUCT_INFORMATION_PK	13	1	13
5	TABLE ACCESS BY INDEX ROWID	PRODUCT_INFORMATION	13	1	13

Predicate Information (identified by operation id):

```
3 - filter(("O"."UNIT_PRICE"=15 AND "QUANTITY">>1))
4 - access("P"."PRODUCT_ID"="O"."PRODUCT_ID")
```

初始基数估计值缩减了  
十分之一还多。

谓词信息

# 统计信息反馈：重新分析语句

```
SQL> SELECT /*+ gather_plan_statistics */ product_name
   FROM order_items o, product_information p
 WHERE o.unit_price = 15
   AND quantity > 1
   AND p.product_id = o.product_id;
```

Id	Operation	Name	Starts	E-Rows	A-Rows
0	SELECT STATEMENT		1	13	13
* 1	HASH JOIN		1	13	13
* 2	TABLE ACCESS FULL	ORDER_ITEMS	1	13	13
3	TABLE ACCESS FULL	PRODUCT_INFORMATION	1	288	288

Predicate Information (identified by operation id):

```
1 - access("P"."PRODUCT_ID"="O"."PRODUCT_ID")
2 - filter(("O"."UNIT_PRICE"=15 AND "QUANTITY">>1))
```

Note

```
- cardinality feedback used for this statement
```

基数估计值现在正确，并将触发联接方法更改。

统计信息反馈显示在计划的注释部分中。

# 自动重新优化

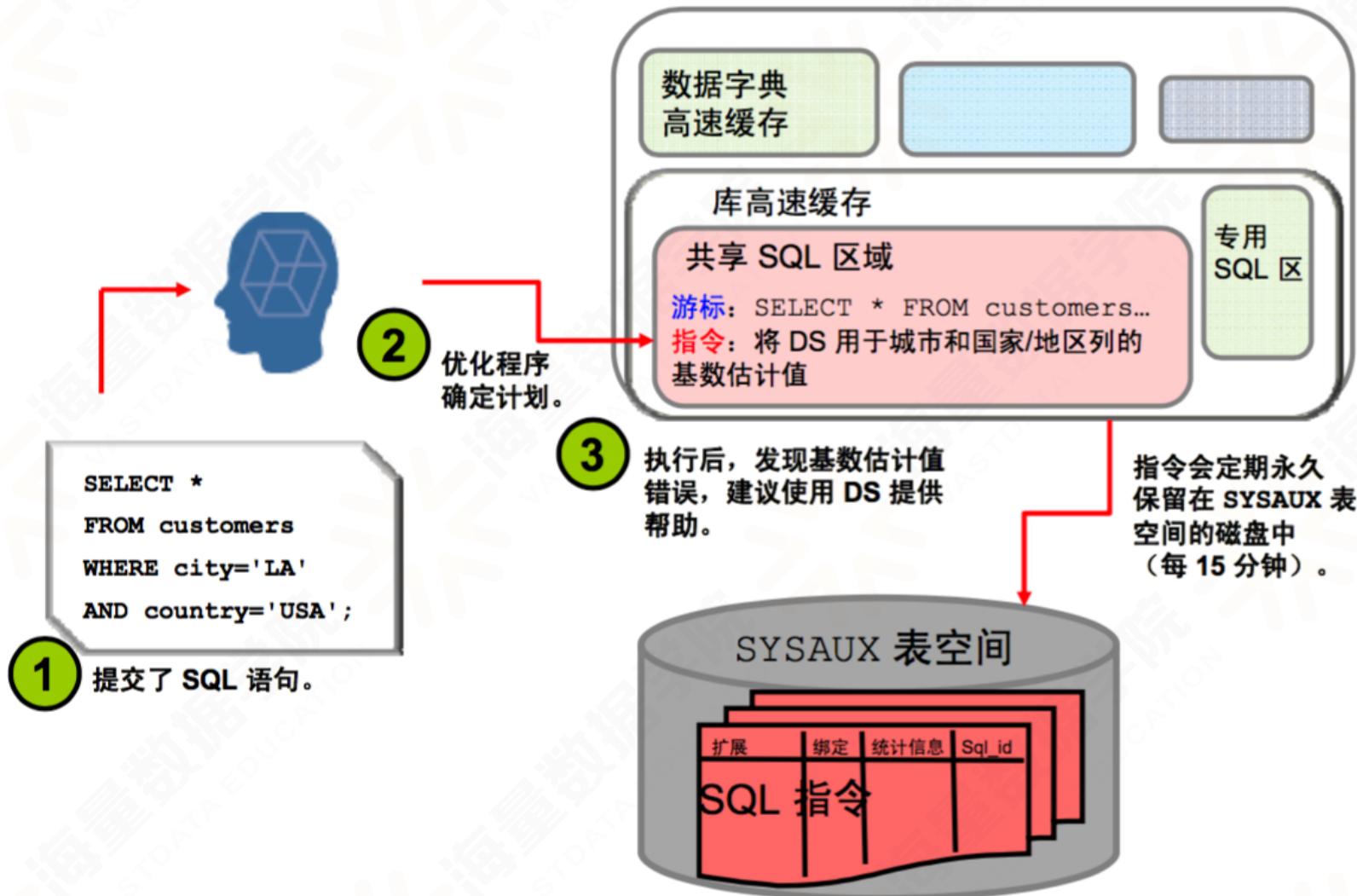
- 在后续执行 SQL 语句时，优化程序会自动更改计划。
- 联接统计信息也包括在内。
- 将持续监视语句以查看每次执行时统计信息是否有波动。
- 对于包含绑定变量的语句，将使用自适应游标共享。
- 在 V\$SQL 中添加了 IS\_REOPTIMIZABLE 列。
- 在执行时发现的信息将作为 SQL 计划指令保存。



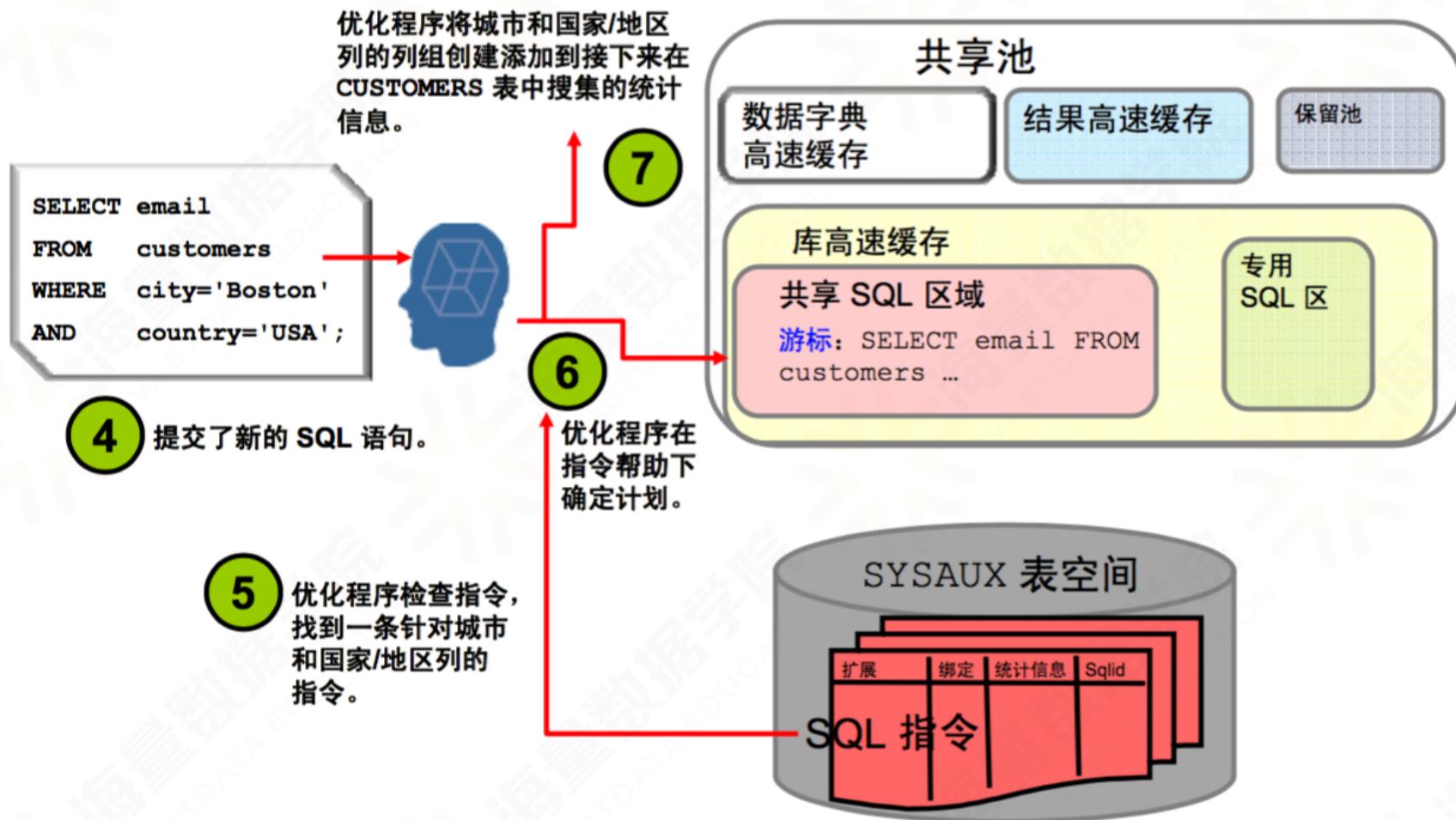
# SQL 计划指令

- SQL 计划指令是指优化程序可用于生成更好的计划的其他信息和指令：
  - 收集缺少的统计信息。
  - 创建列组统计信息。
  - 执行动态采样。
- 指令可用于多条语句：
  - 指令在查询表达式中进行收集。
- 这些指令会永久保留在 SYSAUX 表空间的磁盘中。
- 指令是自动维护的：
  - 在编译或执行过程中根据需要创建：
    - 缺少统计信息，基数估计错误
    - 如果在一年后未使用，则会清除。

# 创建 SQL 计划指令



# 使用 SQL 计划指令



# SQL 计划指令：示例

B.1. Manually Flush Directives (auto flush done every 15 minutes)

```
=====
```

```
SQL> exec dbms_spd.flush_sql_plan_directive
```

PL/SQL procedure successfully completed.

B.2. Display Directives

```
=====
```

```
SQL> col state format a5
```

```
SQL> col subobject_name format a11
```

```
SQL> col object_name format a13
```

```
SQL> select object_name, SUBOBJECT_NAME, type, state, reason from dba_sql_plan_directives d,  
and object_name in ('ORDER_ITEMS', 'PRODUCT_INFORMATION');
```

OBJECT_NAME	SUBOBJECT_N	TYPE	STATE	REASON
-------------	-------------	------	-------	--------

ORDER_ITEMS	UNIT_PRICE	DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
ORDER_ITEMS	QUANTITY	DYNAMIC_SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE
ORDER ITEMS		DYNAMIC SAMPLING	NEW	SINGLE TABLE CARDINALITY MISESTIMATE

B.3. Use Directives

```
=====
```

```
SQL> explain plan for  
select /*Q10*/ product_name  
from order_items o, product_information p  
where o.unit_price = 15  
and quantity > 1  
and p.product_id = o.product_id;
```

Explained.

```
SQL> select * from table(dbms_xplan.display(format=>'basic +note'));
```

PLAN\_TABLE\_OUTPUT

```
Plan hash value: 2615131494
```

Id   Operation	Name
0   SELECT STATEMENT	
1   HASH JOIN	
2   TABLE ACCESS FULL  ORDER_ITEMS	
3   TABLE ACCESS FULL  PRODUCT_INFORMATION	

Note

```
- dynamic sampling used for this statement (level=2)  
- 1 Sql Plan Directive used for this statement
```

# 批量加载的联机统计信息搜集

- 在 Oracle Database 12c 中，数据库将在下面几种批量加载期间自动搜集表统计信息：
  - CREATE TABLE AS SELECT
  - INSERT INTO ... SELECT 空表中 — 使用直接路径插入。
- 加载后立即获得统计信息。
- 无需额外的表扫描即可搜集统计信息。
- 使用 CTAS (MV 刷新) 的所有内部维护操作都将受益。

# Oracle Database 12c 中的并发统计信息增强功能

- 自动统计信息搜集作业使用并发操作。
- 通过仔细规划资源使用率上限可防止因并发搜集统计信息而导致系统使用过度。
- 允许并发搜集索引统计信息。
- 允许并发搜集多个分区表的统计信息。



# 全局临时表的统计信息

- 在 Oracle Database 12c 中，对于每个会话，全局临时表可以有一组不同的优化程序统计信息。
- GLOBAL\_TEMP\_TABLE\_STATSv 程序包的 DBMS\_STATS 首选项控制是否为全局临时表搜集会话或共享统计信息。
- DBMS\_STATS 将更改提交到特定于会话的全局临时表，而不是特定于事务处理的全局临时表。
- 使用特定于会话的统计信息的游标不会由其他会话共享。

# 直方图增强功能

- 直方图是一个表示表中每列数据分布的结构。
- 优化程序使用直方图计算过滤器和联接谓词的选择性。
- 在 Oracle Database 12c 中，引入了两种新的直方图：
  - 最高频率直方图
  - 混合直方图
- 在将采样设置为 AUTO 的情况下，仅创建频率和混合直方图。



# 最高频率直方图

- 过去，仅当 NDV 小于指定的直方图存储桶数量时，才会创建频率直方图。
- 创建最高频率直方图的条件如下：
  - 采样百分比设置为 AUTO
  - 列包含超过  $nb$  个相异值
  - 前  $nb$  个频繁值的百分比大于行的  $p\%$
- 最高频率直方图可以提供更准确的基数估计值。
- 最高频率直方图由 DBA\_TAB\_COL\_STATISTICS.HISTOGRAM 列中的 TOP-FREQUENCY 值指明。

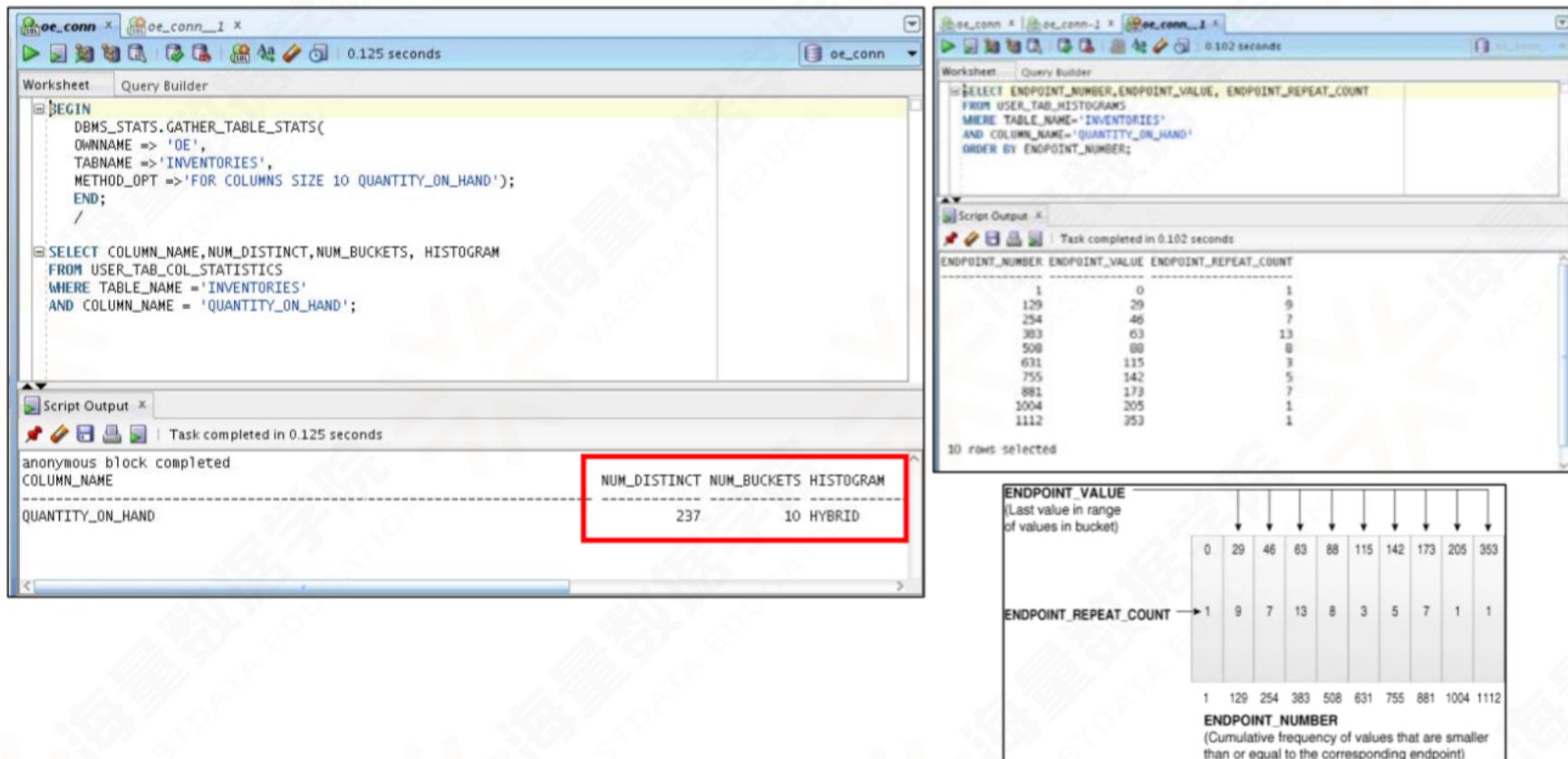


# 混合直方图

- 混合直方图将基于高度的直方图和频率直方图的特征集于一体。
- 在直方图中存储每个端点的端点重复计数值，该值是端点值重复的次数。
- 在 Oracle Database 12c 中，数据库将在符合以下条件时创建混合直方图：
  - 采样百分比为 AUTO。
  - 最高频率直方图的条件不适用。
  - NDV 大于  $nb$ ，其中  $nb$  是用户指定的存储桶数量。如果未指定任何数量，则  $nb$  默认为 254。

# 混合直方图：示例

可以使用 USER\_TAB\_COL\_HISTOGRAMS 表查看混合直方图。



# 扩展统计信息增强功能

- 使用 DBMS\_STATS 可以识别和创建扩展统计信息。
- 有两种类型的扩展统计信息：
  - 列组统计信息
  - 表达式统计信息
- 在表上搜集统计信息时，将自动维护扩展统计信息。
- Oracle Database 12c 可以根据工作量为您检测和创建必要的列组。

# 捕获列组使用情况

启用工作量监视

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> -- Switch on seed column usage for 300 seconds
SQL> EXEC dbms_stats.seed_col_usage(null, null, 300)
PL/SQL procedure successfully completed.
```

# 捕获列组使用情况：运行工作量

```
SQL> EXPLAIN PLAN FOR
  2 SELECT *
  3 FROM   customers
  4 WHERE  cust_city='Los Angeles'
  5 AND    cust_state_province='CA'
  6 AND    country_id=52790;
Explained.
```

## PLAN\_TABLE\_OUTPUT

```
Plan hash value: 2008213504
```

Id   Operation	Name	Rows
0   SELECT STATEMENT		1
1   TABLE ACCESS FULL	CUSTOMERS	1

```
8 rows selected.
```

```
SQL> EXPLAIN PLAN FOR
  2 SELECT country_id, cust_state_province, count(cust_city)
  3 FROM   customers
  4 GROUP BY country_id, cust_state_province;
```

第一个查询返回的实际行数为 932。优化程序低估了该值，因为它假设每个谓词都将减少行数。

第二个查询返回的实际行数为 145。优化程序高估了该值，因为它假设国家/地区和省/市/自治区之间没有关系。

Id   Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0   SELECT STATEMENT		1949	31184	408 (1)	00:00:01
1   HASH GROUP BY		1949	31184	408 (1)	00:00:01
2   TABLE ACCESS STORAGE FULL	CUSTOMERS	55500	867K	406 (1)	00:00:01

# 复查列组使用情况

```
SQL> SELECT dbms_stats.report_col_usage ('SH', 'CUSTOMERS')
  FROM dual;
```

COLUMN USAGE REPORT FOR SH.CUSTOMERS

- 1. COUNTRY\_ID : EQ
- 2. CUST\_CITY : EQ
- 3. CUST\_STATE\_PROVINCE : EQ
- 4. (CUST\_CITY, CUST\_STATE\_PROVINCE, COUNTRY\_ID) : FILTER
- 5. (CUST\_STATE\_PROVINCE, COUNTRY\_ID) : GROUP\_BY

EQ 表示该列在查询 1 中用在等式谓词中。

在查询 2 中,  
GROUP\_BY 列用在  
GROUP BY 表达式中。

FILTER 表示列在一起用作过滤器谓词而非用作联接等。它来自于查询 1。



海量数据  
VASTDATA

海量数据学院  
VASTDATA EDUCATION

# 创建在工作量监视期间检测到的列组

```
SQL> SELECT dbms_stats.create_extended_stats
      ('SH', 'CUSTOMERS')
  FROM dual;
```

```
#####
EXTENSIONS FOR SH.CUSTOMERS
.....<<<<<<
1. (CUST_CITY, CUST_STATE_PROVINCE, COUNTRY_ID) :
   SYS_STUMZ$C3AIHLPBROI#SKA58H_N      created
2. (CUST_STATE_PROVINCE, COUNTRY_ID) :
   SYS_STU#S#WF25Z#QAHIE#MOFFMM_      created
#####
```

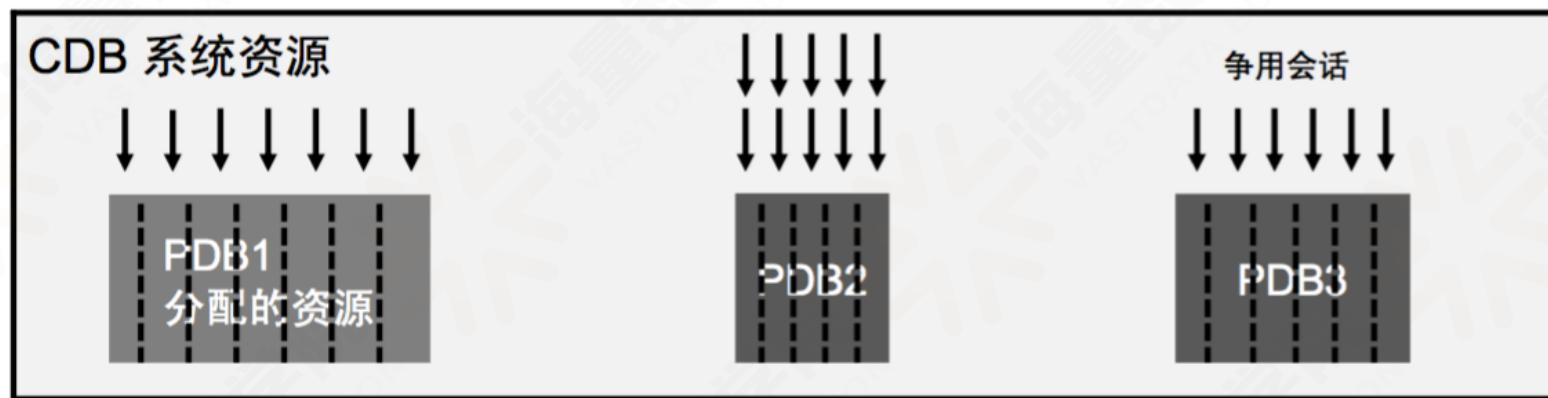
# 自动动态采样

- 从 Oracle Database 12c 开始，优化程序自动确定动态采样是否有用，以及将哪个动态采样级别用于所有 SQL 语句。
- OPTIMIZER\_DYNAMIC\_SAMPLING 初始化参数设置为值 11 时，将启用自动动态采样。
- 将范围扩大到非并行语句。
- 扩展功能以包括联接和分组方式。
- 使动态采样查询的结果永久保留在高速缓存中，以消除重复采样的开销。
- 附加统计信息来源（例如，区映射）可以弥补易失表上的过时统计信息。

# 资源管理器和可插入数据库

在 CDB 中，资源管理器可以管理：

- PDB 之间的资源
- 每个 PDB 中的资源



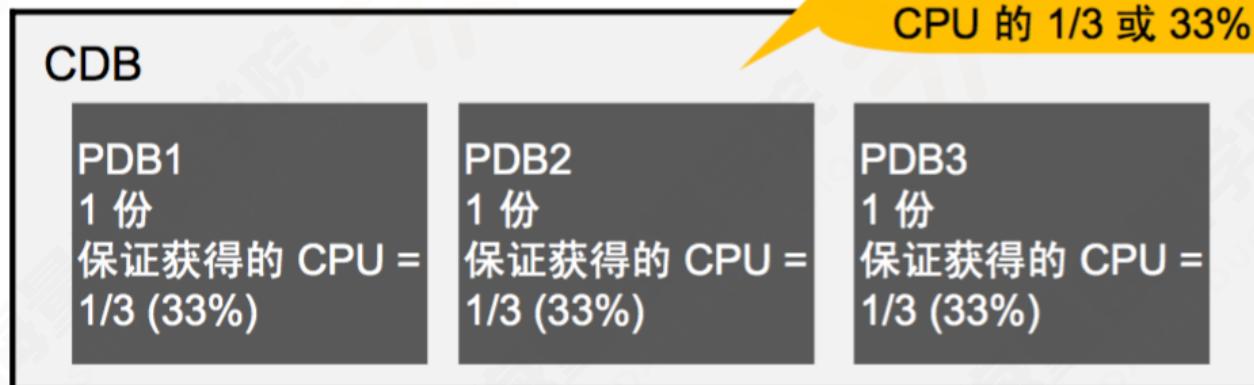
# 在 PDB 之间协调资源

- PDB 争用资源：CPU、Exadata I/O、并行服务器
  - 系统份额用于为每个 PDB 分配资源。
  - 限制用于限定每个 PDB 的资源利用率。
- 插入新的 PDB 时，CDB DBA 可以指定默认分配或显式分配。



# CDB 资源计划基础：份额

每个 PDB 都有一个份额。  
总共 3 份，每个 PDB 保证获得  
CPU 的 1/3 或 33%。



- 使用“份额”指定资源。
  - 添加或删除 PDB 后无需重新计算。
  - PDB1 保证获得 CPU 的 33%。
  - PDB1 最多获得 CPU 的 100%。
  - PDB1 实际使用 CPU 的 20%。

# CDB 资源计划基础：份额

默认分配：一份

CDB

PDB1  
1 份  
保证获得的 CPU =  
 $1/4$  (25%)

PDB2  
1 份  
保证获得的 CPU =  
 $1/4$  (25%)

PDB3  
1 份  
保证获得的 CPU =  
 $1/4$  (25%)

新的 PDB 获得默认  
分配：1 份。

PDB4  
1 份  
保证获得的 CPU =  
 $1/4$  (25%)

CDB

PDB1  
1 份  
保证获得的 CPU =  
 $1/5$  (20%)

PDB2  
1 份  
保证获得的 CPU =  
 $1/5$  (20%)

PDB3  
1 份  
保证获得的 CPU =  
 $1/5$  (20%)

PDB4  
2 份  
保证获得的 CPU =  
 $2/5$  (40%)

如果该 PDB 比较重要，可以显式为其  
分配更多份额。

# CDB 资源计划基础：限制

- 可以为每个 PDB 定义两个限制：
  - CPU、Exadata I/O 和并行服务器的利用率限制
  - 要覆盖利用率限制的并行服务器限制
- 默认值：100%
- 可以更改默认值。

PDB1

```
utilization_limit = 30
```

```
Parallel_server_limit = 50
```

# CDB 资源计划：完整示例

指定 CPU、  
Exadata I/O 和  
并行服务器的分配

限制 CPU 和  
Exadata I/O 的  
使用率

限制并行服务  
器的使用率

PDB/指令名称	份额	利用率限制	并行服务器限制
(默认分配)	(1)	(100%)	(100%)
(自动任务分配)	(-1)	(90%)	(100%)
PDB1	1	30%	50%
PDB2	1	30%	80%
PDB3	1	30%	30%
PDB4	2	100%	100%

PDB1

- 保证获得 CPU 和 Exadata 磁盘带宽的 1/5 (20%)
- 最多获得 CPU 和 Exadata 磁盘带宽的 30%
- 最多获得并行服务器的 50%

# 创建 CDB 资源计划

1. 创建暂挂区。
2. 创建 CDB 资源计划。

```
SQL> EXEC DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN( -
    plan      => 'daytime_plan', -
    comment  => 'CDB resource plan for mycdb')
```

3. 创建 PDB 的指令。

```
SQL> EXEC DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE( -
    plan                  => 'daytime_plan', -
    pluggable_database   => 'salespdb', -
    shares                => 3, -
    utilization_limit    => NULL, -
    parallel_server_limit => NULL)
```

4. (可选) 更新默认 PDB 指令。
5. (可选) 更新默认自动任务指令。
6. 验证并提交暂挂区。

# 设置默认指令

- 设置默认 PDB 指令：

```
SQL> EXEC DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE( -  
      plan                      => 'daytime_plan', -  
      new_shares                 => 1, -  
      new_utilization_limit     => 50, -  
      new_parallel_server_limit => 50)
```

- 设置自动任务 CDB 资源计划指令：

```
SQL> EXEC DBMS_RESOURCE_MANAGER.UPDATE_CDB_AUTOTASK_DIRECTIVE( -  
      plan                      => 'daytime_plan', -  
      new_shares                 => 1, -  
      new_utilization_limit     => 75, -  
      new_parallel_server_limit => 75)
```

# 查看 CDB 资源计划指令

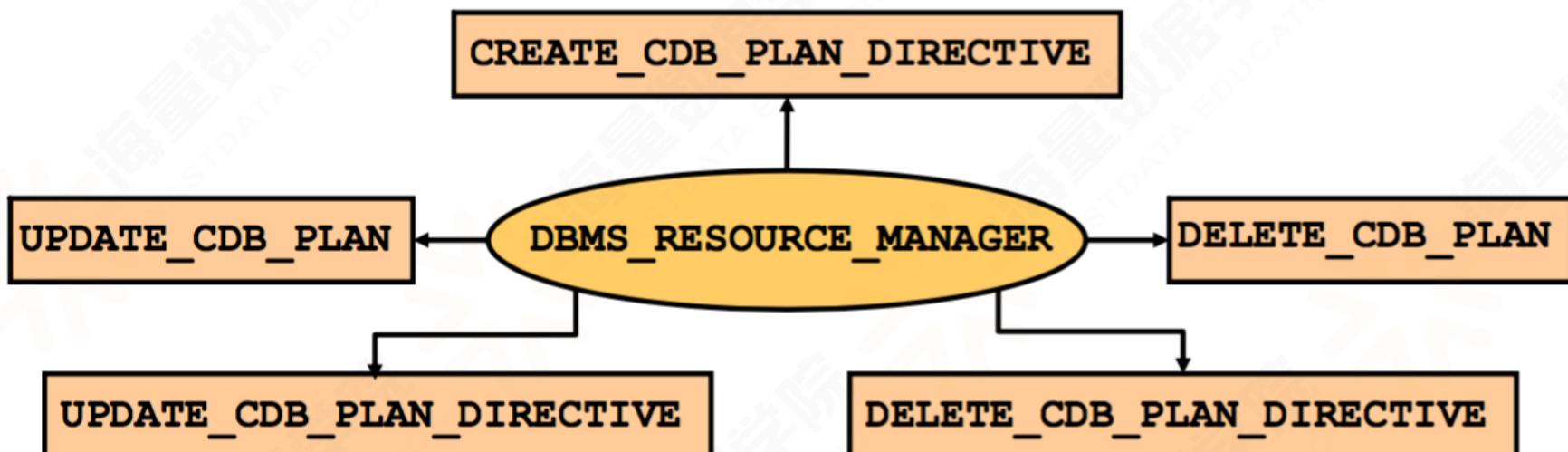
```
SQL> SELECT plan, pluggable_database, shares,
      utilization_limit, parallel_server_limit
    FROM dba_cdb_rsrc_plan_directives
  ORDER BY plan;
```

Plan	Pluggable Database	Shares	Parallel	
			Utilization Limit	Server Limit
DAYTIME_PLAN	ORA\$DEFAULT_PDB_DIRECTIVE	1	50	50
DAYTIME_PLAN	ORA\$AUTOTASK	1	75	75
DAYTIME_PLAN	SALESPDB	3		
DAYTIME_PLAN	HRPDB	1	70	70
DEFAULT_CDB_PLAN	ORA\$DEFAULT_PDB_DIRECTIVE	1	100	100
DEFAULT_CDB_PLAN	ORA\$AUTOTASK		90	100
DEFAULT_MAINTENANCE_PLAN	ORA\$DEFAULT_PDB_DIRECTIVE	1	100	100
DEFAULT_MAINTENANCE_PLAN	ORA\$AUTOTASK		90	100
ORA\$INTERNAL_CDB_PLAN	ORA\$DEFAULT_PDB_DIRECTIVE			
ORA\$INTERNAL_CDB_PLAN	ORA\$AUTOTASK			

# 维护 CDB 资源计划

```
SQL> CONNECT sys AS SYSDBA
```

```
SQL> EXEC DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```



```
SQL> EXEC DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

```
SQL> EXEC DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

# 管理 PDB 中的资源

- 在非 CDB 数据库中，使用资源计划管理数据库中的工作量。
- 在 PDB 中，也使用资源计划（又称为 PDB 资源计划）管理工作量。
- 功能相似，但以下差异除外：

非 CDB 数据库	PDB 数据库
多级资源计划	仅单级资源计划
最多 32 个使用者组	最多 8 个使用者组
子计划	无子计划



# 管理 PDB 资源计划

- 连接到 PDB 以管理 PDB 计划。
- 使用的过程与在非 CDB 环境中管理资源计划的过程完全一样。
- 对于 CREATE\_PLAN\_DIRECTIVE 过程：
  - 新的 SHARE 参数
  - 将 MAX\_UTILIZATION\_LIMIT 和 PARALLEL\_TARGET\_PERCENTAGE 参数替换为 UTILIZATION\_LIMIT 和 PARALLEL\_SERVER\_LIMIT
- 可以使用 V\$RSRC\_PLAN 查看 CDB 和 PDB 资源计划。



# 结合使用

- CDB 和 PDB 资源计划如何协同工作?
  1. 根据 CDB 资源计划将资源分配给 PDB
  2. 根据 PDB 资源计划将资源分配给使用者组

CDB 计划

PDB	份额	利用率限制
PDB1	1	50%
PDB2	1	50%
PDB3	2	100%

PDB3 计划

使用者组	份额	利用率限制
OLTP	3	100%
报告	1	50%
其他	1	50%

- 这对 PDB3 报告意味着什么?
  - 保证获得  $50\% (2/4) \times 20\% (1/5) =$  资源的 10%
  - 最多获得  $100\% \times 50\% =$  资源的 50%

# 注意事项

- 设置 PDB 资源计划是可选操作。如果未指定，PDB 中的所有会话将得到同等对待。
- 将非 CDB 插入带有计划的 CDB：
  - 如果满足以下条件，则计划在 PDB 上的运行方式完全一样：
    - 使用者组  $\leq 8$
    - 无子计划
    - 所有分配都在级别 1
  - 计划将进行转换。
    - 原始计划以 LEGACY 状态存储在字典中。



# 失控查询和资源管理器

- 用于触发使用者组切换的新参数:
  - SWITCH\_IO\_LOGICAL
  - SWITCH\_ELAPSED\_TIME
- 名为 LOG\_ONLY 的新元使用者组
- 添加到 V\$SQL\_MONITOR 的新列:
  - RM\_LAST\_ACTION
  - RM\_LAST\_ACTION\_REASON
  - RM\_LAST\_ACTION\_TIME
  - RM\_CONSUMER\_GROUP
- 始终会填充 V\$RSRCMGRMETRIC 和 V\$RSRCMGRMETRIC\_HISTORY



# 控制 IM 列存储重新填充资源消耗

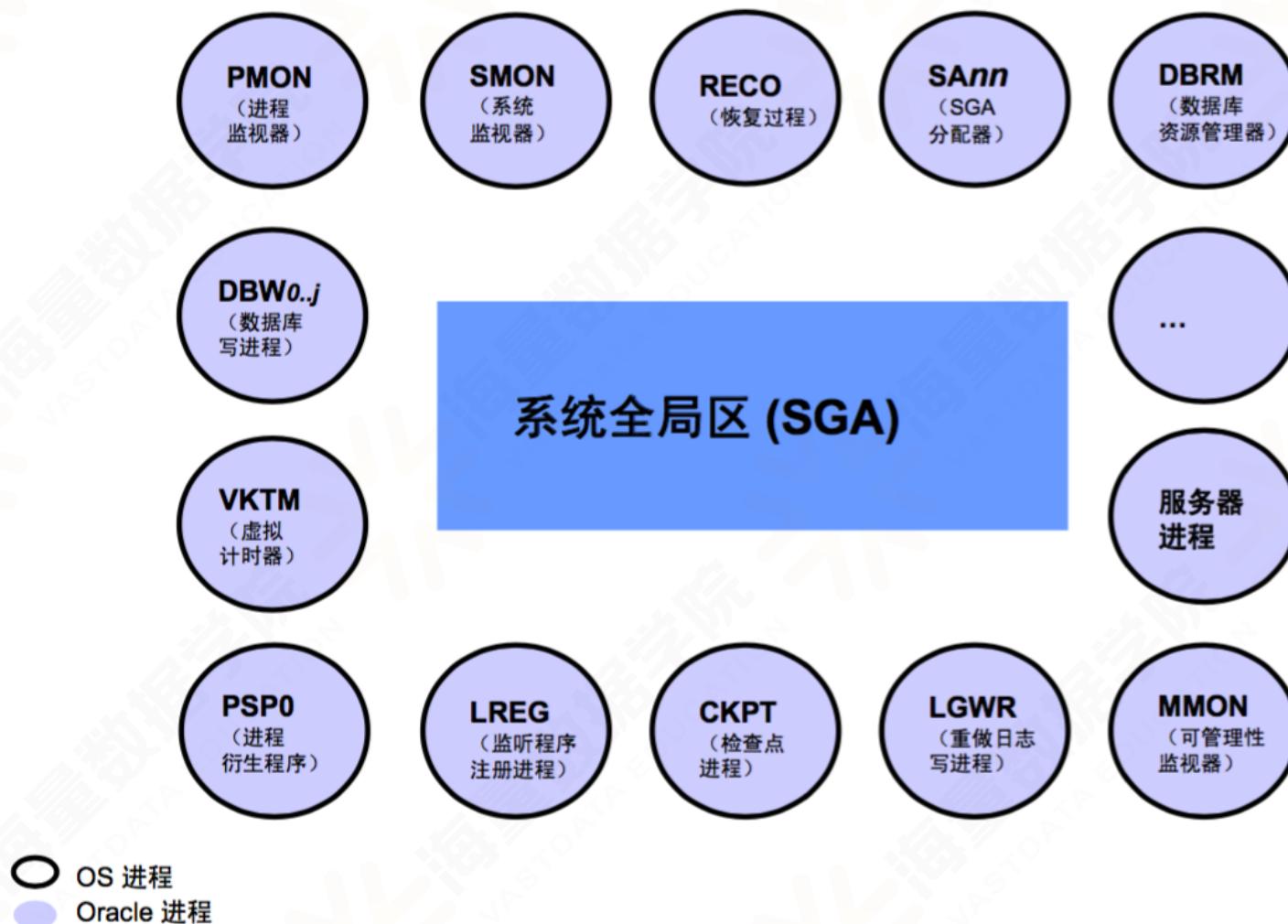
- IM 列存储填充会占用大量 CPU。
- 存在不同类型的 IM 列存储填充：

说明	默认使用者组
内存中段（例如：优先级不是 NONE 的段）的填充	ora\$autotask
按需填充（例如：当用户访问优先级为 NONE 的内存中段时）	用户的使用者组

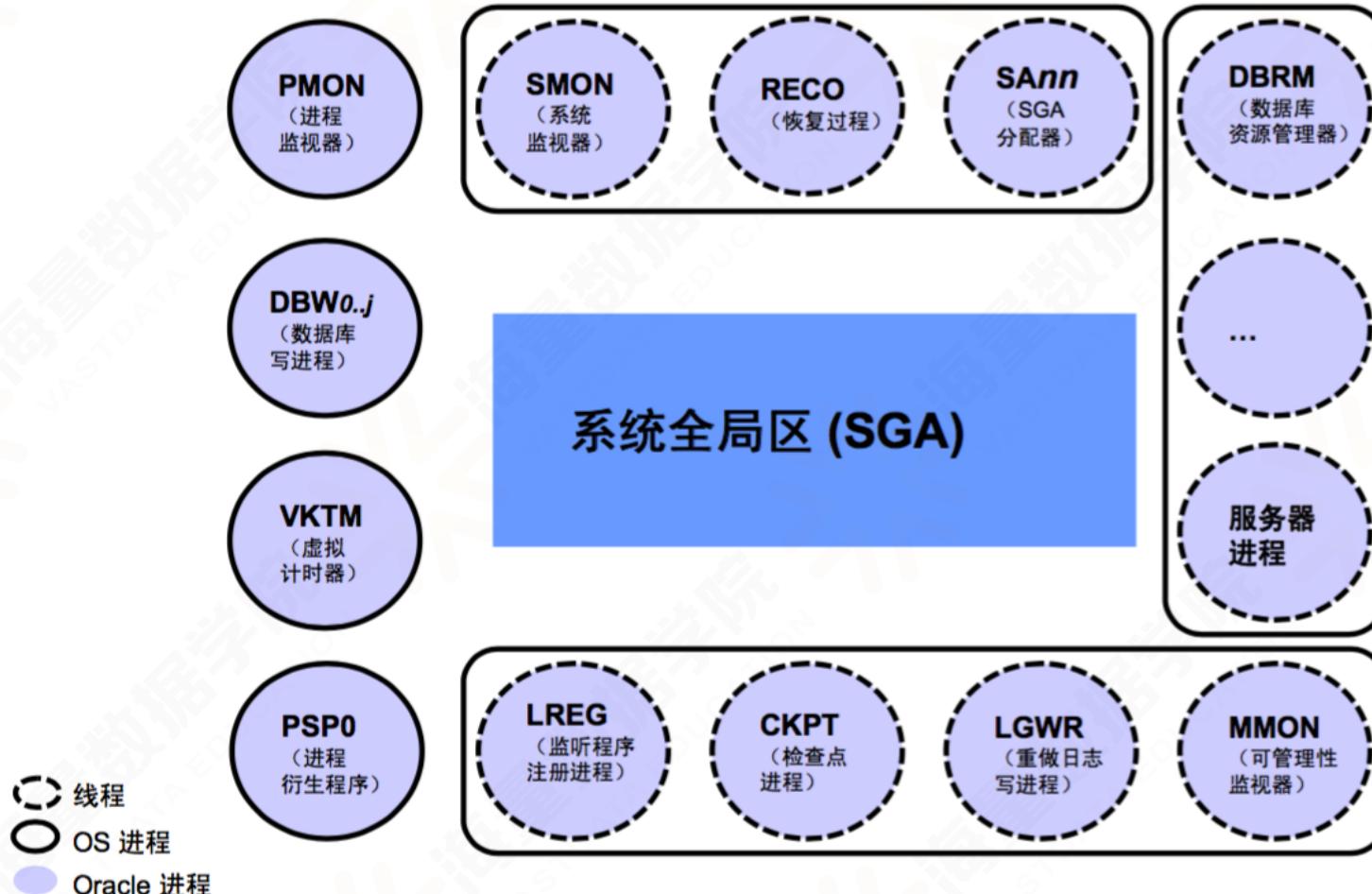
→ 通过更改优先级管理填充操作的 CPU 占用率：

- 启用现成的资源计划之一 (DEFAULT\_PLAN)
- 创建您自己的资源计划
- 将带有“INMEMORY”值的 ORACLE\_FUNCTION 属性映射到其他使用者组

# 默认 UNIX/Linux 体系结构



# 多进程多线程 UNIX/Linux 体系结构



# 多进程多线程体系结构：优点和设置

- CPU 和内存使用量减少
- 系统可靠性更高
- 并行操作性能更高

```
SQL> CONNECT sys AS SYSDBA
Enter password:
Connected.
SQL> ALTER SYSTEM SET threaded_execution=true SCOPE=SPFILE;
System altered.
SQL> SHUTDOWN IMMEDIATE
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> STARTUP
...
Database opened.
SQL>
```

# 多进程多线程体系结构：注意事项

- 线程仍然将 PGA 用于专用内存。
- 线程仍然将 SGA 内存用于进程间通信。
- 监听程序不会衍生线程。
- 需要用于 SYSDBA 验证的口令文件。
- 作为最佳实践，设置以下参数：

```
DEDICATED_THROUGH_BROKER_LISTENER = on
```



# 多进程多线程体系结构：监视

```
SQL> select spid, stid, pname from v$process order by spid;
```

SPID	STID	PNAME
------	------	-------

31562	31562	PMON
31566	31566	PSPO
31570	31570	VKTM
31576	31576	SCMN
31576	31580	GEN0
31576	31627	SMON
31576	31633	LREG
31576	31624	LG01

...		
31590	31608	DIAO
31590	31590	SCMN
31590	31642	D000
...		
31598	31602	OFSD
31598	31598	SCMN
31612	31612	DBW0

47 rows selected.

```
SQL>
```

```
$ ps -ef | grep orcl
oracle    598 19126  0 16:30 pts/0  00:00:00 grep orcl
oracle    31562     1  0 14:04 ?        00:00:01 ora_pmon_orcl
oracle    31566     1  0 14:04 ?        00:00:01 ora_psp0_orcl
oracle    31570     1  1 14:04 ?        00:01:52 ora_vktm_orcl
oracle    31576     1  0 14:04 ?        00:00:05 ora_u004_orcl
oracle    31590     1  0 14:04 ?        00:00:35 ora_u005_orcl
oracle    31598     1  0 14:04 ?        00:00:00 ora_u006_orcl
oracle    31612     1  0 14:04 ?        00:00:00 ora_dbw0_orcl
$
```

```
$ ps -eLo "pid tid comm args" | grep ora_
31562 31562 ora_pmon_orcl   ora_pmon_orcl
31566 31566 ora_psp0_orcl   ora_psp0_orcl
31570 31570 ora_vktm_orcl   ora_vktm_orcl
31576 31576 ora_scmn_orcl   ora_u004_orcl
31576 31579 oracle          ora_u004_orcl
31576 31580 ora_gen0_orcl   ora_u004_orcl
31576 31583 ora_mman_orcl   ora_u004_orcl
$
```

# 数据库智能闪存高速缓存增强功能

- 闪存高速缓存中有多个闪存驱动器
  - 闪存高速缓存设备动态启用/禁用
- 更改为内存中并行查询 (parallel query, PQ) 算法
  - 启用了自动 DOP 时
  - 缓冲区高速缓存分成 OLTP 和 DW 部分
  - 单独的高速缓存替换算法用于 DW 部分



# 启用和禁用闪存设备

- 在实例启动时指定闪存设备。

```
db_flash_cache_file = /dev/raw/sda, /dev/raw/sdb  
db_flash_cache_size = 32G, 64G
```

- 禁用闪存设备。

```
db_flash_cache_size = 0, 64G
```

- 启用闪存设备。

```
db_flash_cache_size = 32G, 64G
```

# 内存中 PQ 算法：优点

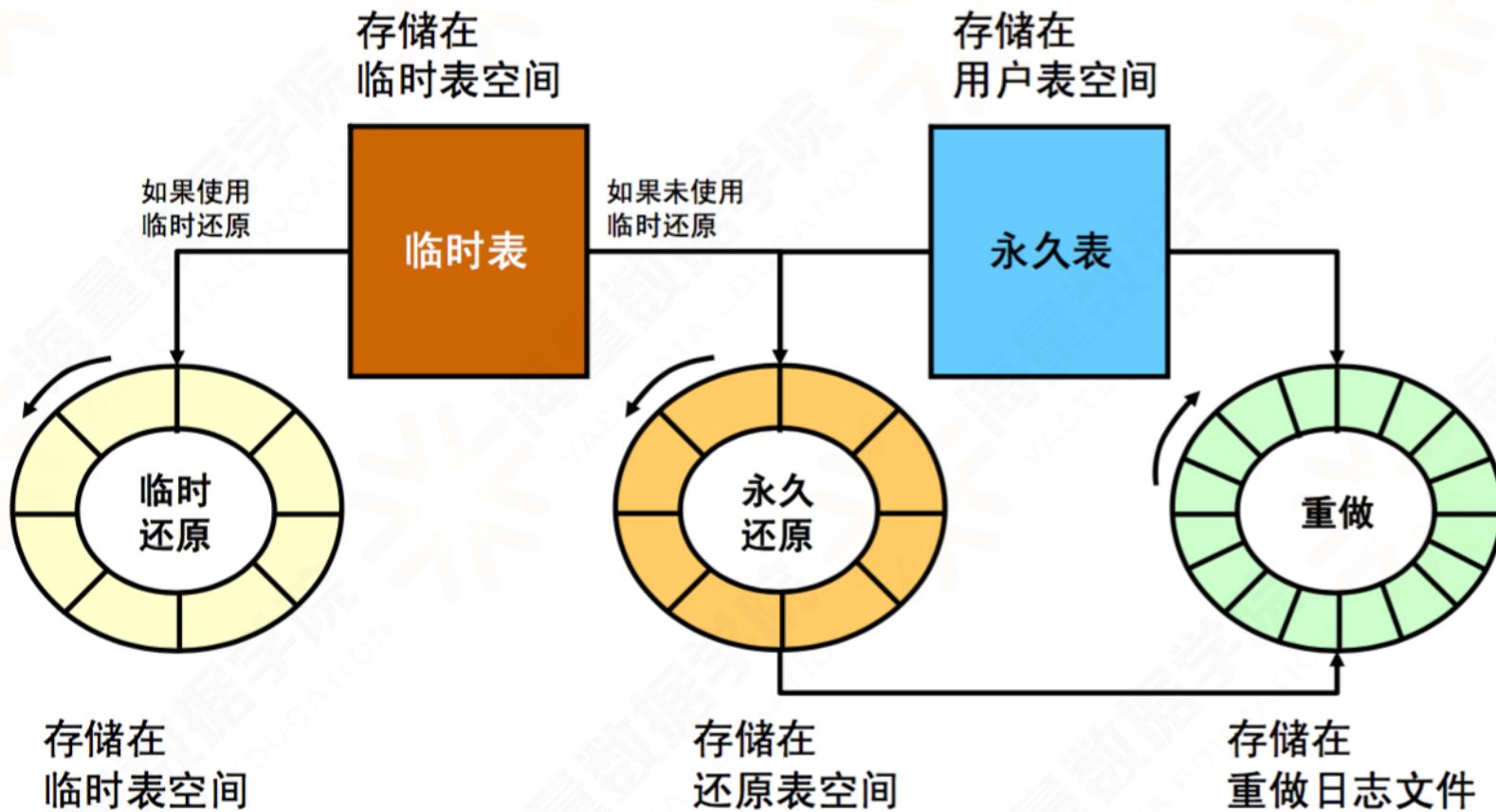
- 允许并行查询利用智能闪存高速缓存
- 允许并行读取智能闪存高速缓存和磁盘
- 不会使缓冲区高速缓存或闪存高速缓存发生崩溃
- 在 I/O 与 CPU 资源之间寻求最佳平衡

# 智能闪存高速缓存：新统计信息

用于衡量对象替换算法使用情况的几种新统计信息包括：

- 数据仓库扫描的块数
- 数据仓库扫描的块数 – 磁盘
- 数据仓库扫描的块数 – 闪存
- 数据仓库扫描的块数 – 内存
- 数据仓库扫描的对象数

# 临时还原：概览



# 临时还原：优点和设置

- 临时还原可减少还原表空间中存储的还原数据量。
- 临时还原可减小重做日志的大小。
- 临时还原支持在具有 Oracle Active Data Guard 选件的物理备用数据库中对临时表执行 DML 操作。

```
SQL> ALTER SESSION SET TEMP_UNDO_ENABLED = TRUE;
```

```
SQL> ALTER SYSTEM SET TEMP_UNDO_ENABLED = TRUE;
```

- 会话首次使用临时对象时会选择临时还原模式。

# 临时还原监视

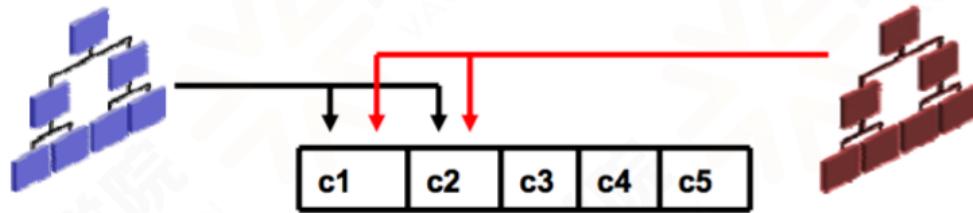
```
SQL> SELECT to_char(BEGIN_TIME, 'dd/mm/yy hh24:mi'),  
      TXNCOUNT, MAXCONCURRENCY, UNDOBLKCNT, USCOUNT,  
      NOSPACEERRCNT  
  FROM    V$TEMPUNDOSTAT;  
  
TO_CHAR(BEGIN_ TXNCOUNT MAXCONCURRENCY UNDOBLKCNT USCOUNT NOSPACEERRCNT  
----- ----- ----- ----- -----  
...  
19/02/14 22:19      0          0          0          0          0  
19/02/14 22:09      0          0          0          0          0  
...  
19/02/14 13:09      0          0          0          0          0  
19/02/14 12:59      3          1          24         1          0  
  
576 rows selected.  
  
SQL>
```

# 限制程序全局区的大小

- PGA 内存使用量可能会因为以下原因超过 `PGA_AGGREGATE_TARGET` 设置的值：
  - `PGA_AGGREGATE_TARGET` 是一个目标，而不是一个限制
  - `PGA_AGGREGATE_TARGET` 仅控制可优化内存的分配
- 使用 `PGA_AGGREGATE_LIMIT` 初始化参数指定 PGA 内存使用量的硬限制
- 当达到 `PGA_AGGREGATE_LIMIT` 时
  - 使用最多不可优化内存的会话将中止其调用。
  - 如果 PGA 总内存使用量仍然超过限制，则使用最多不可优化内存的会话将被终止。

# 为什么同一组列有多个索引

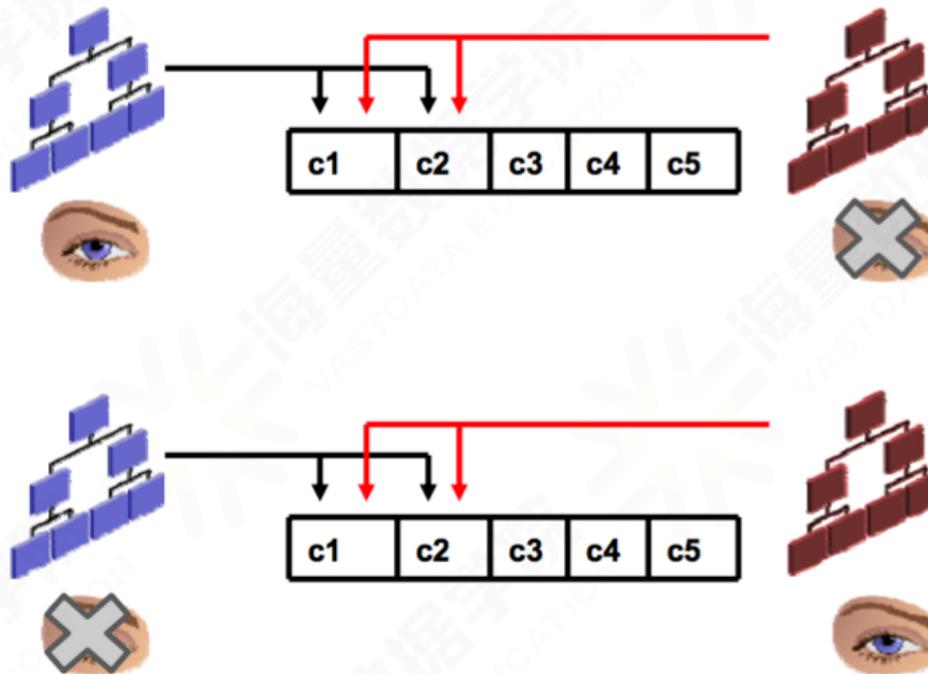
- 执行应用程序移植而不删除现有索引，而使用其他属性重新创建索引。
- 多个不同索引：
  - 唯一索引和非唯一索引



- 不同分区：本地分区和全局分区

# 对同一组列创建多个索引

一次只有一个索引可见。



如果 `OPTIMIZER_USE_INVISIBLE_INDEXES=TRUE`，  
优化程序会使用不可见索引。

# SQL\*Plus 中的不可见列和隐藏列

- 不可见列是用户指定的隐藏列。

```
SQL> CREATE TABLE mytab (col1 VARCHAR2(10),  
                           col2 NUMBER INVISIBLE);
```

```
SQL> DESC mytab
```

Name	Null?	Type
COL1		VARCHAR2 (10)

不可见列未显示

- 可在以后设为可见
- 隐藏列不会影响访问表的现有应用程序。
- 只有通过在查询和其他操作中显式引用隐藏列的名称，才能访问隐藏列。

```
SQL> INSERT INTO mytab (col1, col2) values ('A',1);
```



# SET COLINVISIBLE 和 DESCRIBE 命令

SET COLINVISIBLE 命令用于在 DESCRIBE 命令中显示不可见的列。

```
SQL> SET COLINVISIBLE ON
```

```
SQL> DESC mytab
```

Name	Null?	Type
COL1		VARCHAR2 (10)
COL2 (INVISIBLE)		NUMBER

# 联机重新定义：包含 VPD 的表

重新定义包含虚拟专用数据库 (VPD) 策略的表，而不更改表中任何列的属性。

1. 使用 DBMS\_RLS.ADD\_POLICY 过程，在 HR.EMP 表中应用 VPD 策略

```
SQL> EXEC DBMS_RLS.ADD_POLICY ( object_schema => 'hr', -  
object_name => 't1', policy_name => 't1_pol', -  
function_schema => 'sys', policy_function=> 'auth_sal', -  
statement_types => 'select, insert, update, delete')
```

2. 开始重新定义过程：

- a. 验证该表是否是联机重新定义的潜在候选对象。
- b. 创建临时表。
- c. 开始重新定义。

```
SQL> EXEC DBMS_REDEFINITION.START_REDEF_TABLE ( uname => 'hr', -  
orig_table => 't1', int_table => 'int_t1', -  
options_flag => DBMS_REDEFINITION.CONS_USE_PK, -  
copy_vpd_opt => DBMS_REDEFINITION.CONS_VPD_AUTO)
```

3. 完成重新定义过程。

# 联机重新定义: **dml\_lock\_timeout**

FINISH\_REDEF\_TABLE 过程中的 **dml\_lock\_timeout** 参数指定该过程等待待处理 DML 提交的时长。

1. 验证该表是否可以联机重新定义。

```
SQL> EXEC DBMS_REDEFINITION.CAN_REDEF_TABLE (...);
```

2. 创建临时表。
3. 开始重新定义。

```
SQL> EXEC DBMS_REDEFINITION.START_REDEF_TABLE (...);
```

4. 复制从属对象。

```
SQL> EXEC DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS (...);
```

5. 完成重新定义过程。

```
SQL> EXEC DBMS_REDEFINITION.FINISH_REDEF_TABLE (uname => 'hr', -  
        orig_table      => 't1', int_table => 'int_t1'-  
        dml_lock_timeout => 100)
```

# 高级行压缩：新功能名称和语法

11g

## COMPRESS

方法	CREATE 和 ALTER TABLE	典型应用
基本	COMPRESS [BASIC]	DSS
OLTP	COMPRESS FOR OLTP	OLTP、 DSS

```
SQL> CREATE TABLE tab1  
COMPRESS [BASIC];
```

```
SQL> CREATE TABLE tab1  
COMPRESS FOR OLTP;
```

12c

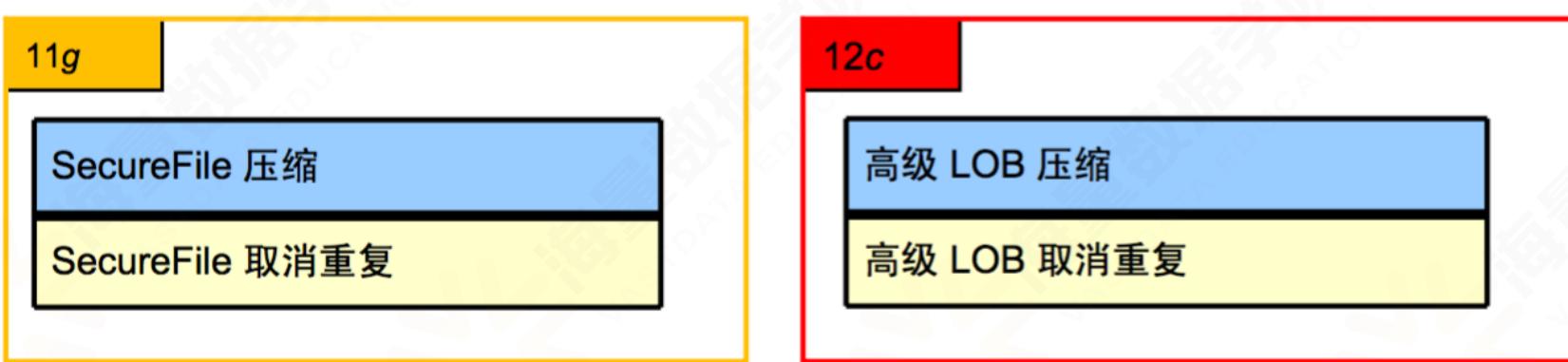
## ROW STORE COMPRESS

方法	CREATE 和 ALTER TABLE	典型应用
基本	ROW STORE COMPRESS	DSS
高级	ROW STORE COMPRESS <b>ADVANCED</b>	OLTP、 DSS

```
SQL> CREATE TABLE tab1 ROW  
STORE COMPRESS [BASIC];
```

```
SQL> CREATE TABLE tab1 ROW  
STORE COMPRESS ADVANCED;
```

# LOB 压缩：新名称



# 使用压缩指导

- 压缩指导有助于确定最佳压缩率。

```
BEGIN
    DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
        'USERS', 'HR', 'EMP', NULL, DBMS_COMPRESSION.COMP_ADVANCED,
        blkcnt_cmp, blkcnt_ncmp, row_cmp, row_ncmp, cmp_ratio,
        comptype_str,1000,1);
    DBMS_OUTPUT.PUT_LINE('Block count compressed = ' || blkcnt_cmp);
    DBMS_OUTPUT.PUT_LINE('Block count uncompressed = ' || blkcnt_ncmp);
    DBMS_OUTPUT.PUT_LINE('Row count per block compressed = ' || row_cmp);
    DBMS_OUTPUT.PUT_LINE('Row count per block uncompressed=' || row_ncmp);
    DBMS_OUTPUT.PUT_LINE('Compression type = ' || comptype_str);
    DBMS_OUTPUT.PUT_LINE('Comp ratio= '|| blkcnt_ncmp/bblkcnt_cmp||'to 1');
END;
```

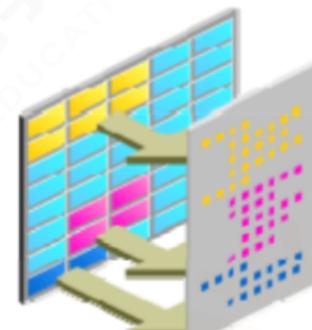
- DBA\_TABLES 视图中 COMPRESS\_FOR 列中的新压缩类型



# 增强的联机 DDL 功能

可以使用新的 **ONLINE** 关键字来允许在以下 DDL 操作期间执行 DML 语句：

- DROP INDEX
- DROP CONSTRAINT
- ALTER INDEX UNUSABLE
- SET COLUMN UNUSED



# DROP INDEX / CONSTRAINT



- **ONLINE** 表示在删除索引时允许对表或分区执行 DML 操作。
- 分区索引和非分区索引都支持联机 DROP INDEX。

```
SQL> DROP INDEX schema.index ONLINE FORCE;
```

- 联机 DROP CONSTRAINT 可让您从数据库中删除完整性约束条件，但有一些限制：
  - 无法删除包含 CASCADE 的约束条件
  - 无法删除引用约束条件

```
SQL> ALTER TABLE hr.employees  
DROP CONSTRAINT emp_email_uk ONLINE;
```

# 索引 UNUSABLE

- 指定 UNUSABLE 子句，以将索引、索引分区或索引子分区标记为 UNUSABLE。
- 指定 **ONLINE** 子句，表示在将索引标记为 UNUSABLE 时允许对表或分区执行 DML 操作。

```
SQL> ALTER INDEX hr.i_emp_ix UNUSABLE ONLINE;
```

```
SQL> SELECT status FROM user_indexes  
      WHERE table_name='EMP';
```

INDEX_NAME	STATUS
I_EMP_IX	UNUSABLE

# SET UNUSED 列

- SET UNUSED 子句可以是释放数据库空间的第一步，方法是删除不再需要的列。
- **ONLINE** 关键字表示在将列标记为 UNUSED 时允许对表执行 DML 操作。

```
SQL> CREATE TABLE emp (ename VARCHAR2(20), id NUMBER);
SQL> INSERT INTO emp VALUES('Tim', 4);
SQL> SELECT * FROM emp;
SQL> ALTER TABLE emp SET UNUSED (ename) ONLINE;

SQL> DESC emp;
Name          Null?    Type
-----        -----
ID           NUMBER
```



扫描二维码关注“海量数据学院”  
助您职业生涯强力进阶  
微信号：vastdataedu

联系方式：13811106886  
QQ：1883926

荟萃业界名师 培养数据英才



海量数据学院  
VASTDATA EDUCATION