How to write an efficient SQL

TINGKUN YANG

YUNHE ENMO (BEIJING) TECHNOLOGY CO.,LTD

云和恩墨 成就所托

Introduction

- 杨廷琨 TINGKUN YANG
  - Oracle ACE Director
  - ACOUG Vice President
  - One of Authors of <Oracle Database Performance Tuning>, <Oracle DBA Notes>, <Oracle DBA Notes 3> & < Oracle Performance Optimization and Diagnosis Case Selection>
- There are more than 2500 technical articles in personal BLOG http://yangtingkun.net
- CTO of Enmotech which company is the leader of Database Services industry in CHINA, and include 5 ACE Directors and 4 ACEs and more than 20 Oracle OCM

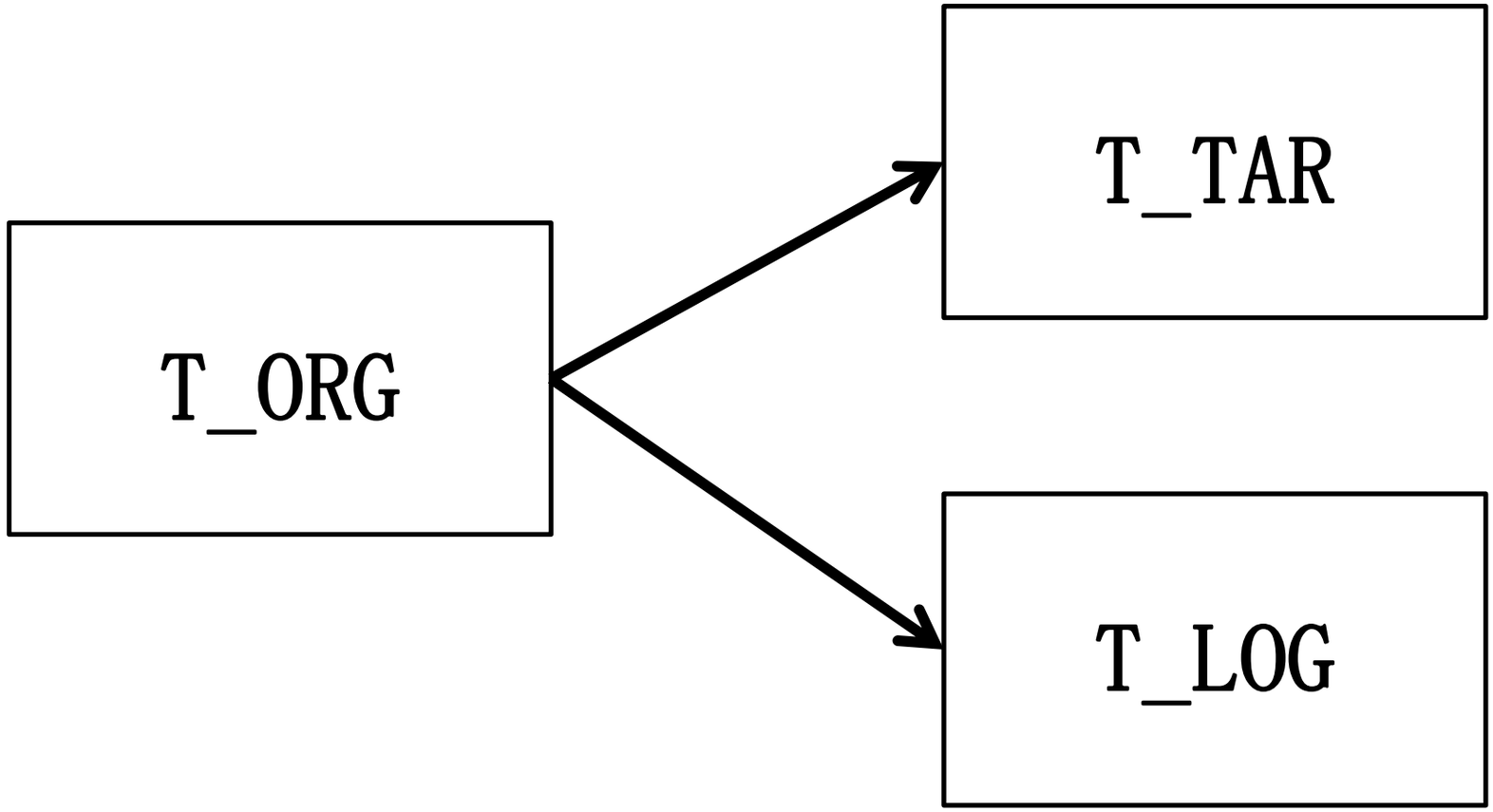
Why do we need efficient SQL

- SQL is currently the second largest programming language, 50% of developers are using SQL
- 80% of database problems are caused by SQL statements
- 80% of the SQL problems come from 20% of SQL statements
- Performance failures caused by a single SQL in high-concurrency systems are very common

1. Use new feature reasonably
2. Data processing as a whole
3. Design SQL execution plan
4. Filter unnecessary data from beginning

Use new feature reasonably

Target： We need to read the data from table T_ORG, and insert into table T_TAR, at the same time, we must insert the same data into another table T_LOG.



Use new feature reasonably

Methods:

- Create Trigger

```
CREATE OR REPLACE TRIGGER T_INSERT_TAR
   AFTER INSERT ON T_TAR
   FOR EACH ROW
BEGIN
   INSERT INTO T_LOG (ID, NAME)
   VALUES (:NEW.ID, :NEW.NAME);
END;
/
```

Use new feature reasonably

Methods:

- Create Trigger
  - Too heavy
  - Differences between implementation and demand
  - Increase follow-up maintenance costs
  - Trigger is less efficient

Use new feature reasonably

Methods:

- Create Trigger
- Double Insert

```
INSERT INTO T_TAR SELECT * FROM T_ORG;
INSERT INTO T_LOG SELECT * FROM T_ORG;
```

Use new feature reasonably

Methods:

- Create Trigger
- Double Insert
  - Consistency
    - Lock
    - Serial transaction
    - Temporary table
    - As of query
  - Atomicity
    - Exception handling

Use new feature reasonably

Methods:

- Create Trigger
- Double Insert
- Open Cursor

```
SQL> BEGIN
  2      FOR I IN (SELECT * FROM T_ORG) LOOP
  3        INSERT INTO T_TAR VALUES (I.ID, I.NAME);
  4        INSERT INTO T_LOG VALUES (I.ID, I.NAME);
  5      END LOOP;
  6  END;
  7  /

PL/SQL procedure successfully completed.
```

Use new feature reasonably

Methods:

- Create Trigger
- Double Insert
- Open Cursor
- Low efficiency

Use new feature reasonably

Methods:

- Create Trigger
- Double Insert
- Open Cursor
- Bulk Into Variable

Use new feature reasonably

```
SQL> DECLARE
  2       TYPE T_ID IS TABLE OF T_ORG.ID%TYPE;
  3       TYPE T_NAME IS TABLE OF T_ORG.NAME%TYPE;
  4       V_ID T_ID;
  5       V_NAME T_NAME;
  6     BEGIN
  7       SELECT ID, NAME BULK COLLECT INTO V_ID, V_NAME
  8         FROM T_ORG;
  9       FORALL I IN 1..V_ID.COUNT
 10         INSERT INTO T_TAR VALUES(V_ID(I), V_NAME(I));
 11       FORALL I IN 1..V_ID.COUNT
 12         INSERT INTO T_LOG VALUES(V_ID(I), V_NAME(I));
 13    END;
 14    /

PL/SQL procedure successfully completed.
```

Use new feature reasonably

Methods:

- Create Trigger
- Double Insert
- Open Cursor
- Bulk Into Variable
  - High complexity
  - Less efficient

Use new feature reasonably

Methods:

- Create Trigger
- Double Insert
- Open Cursor
- Bulk Into Variable
- Insert All

```
SQL> INSERT ALL INTO T_TAR (ID, NAME)
  2      INTO T_LOG (ID, NAME)
  3      SELECT ID, NAME FROM T_ORG;
```

Use new feature reasonably

“New” feature：

● INSERT ALL/ANY	Insert multiple tables or insert with condition
● MERGE	Combining UPDATE and INSERT statements
● WITH	Reduce reading of duplicate result sets
● ANALYTIC FUNCTIONS	Row-level calculations avoid self-join
● TOP-N	Pagination query
● PIVOT/UNPIVOT	Rank conversion query
● MODEL	Report array processing
● PATTERN	Process complex patterns contained in the data

1. Use new feature reasonably
2. Data processing as a whole
3. Design SQL execution plan
4. Filter unnecessary data from beginning

Data processing as a whole

Determine whether the tablespace is automatically expanded:

```
SQL> select distinct tablespace_name, autoextensible
  2       from DBA_DATA_FILES
  3      where autoextensible = 'YES'
  4      union
  5      select distinct tablespace_name, autoextensible
  6       from DBA_DATA_FILES
  7      where autoextensible = 'NO'
  8        and tablespace_name not in
  9          (select distinct tablespace_name
 10             from DBA_DATA_FILES
 11            where autoextensible = 'YES');

TABLESPACE_NAME                AUT
------------------------------ ---
SYSAUX                         YES
SYSTEM                         YES
TEST                           NO
UNDOTBS1                       YES
USERS                          YES
```

Data processing as a whole

```
SQL> select distinct tablespace_name, autoextensible
  2    from DBA_DATA_FILES
  3   where autoextensible = 'YES'
  4   union
  5   select distinct tablespace_name, autoextensible
  6    from DBA_DATA_FILES
  7   where autoextensible = 'NO'
  8     and tablespace_name not in
  9         (select distinct tablespace_name
 10          from DBA_DATA_FILES
 11         where autoextensible = 'YES');
```

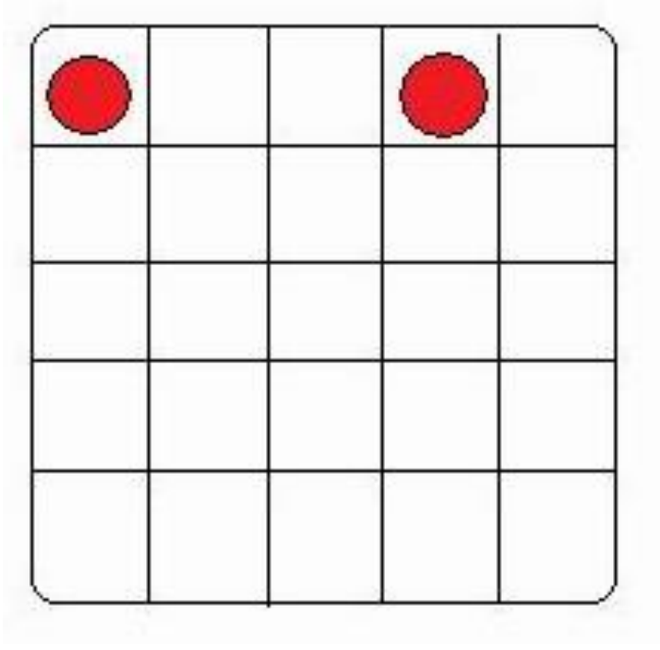
- benefits: clear thinking
- Disadvantages: Inefficient and redundant
  - Scan view DBA_DATA_FILES three times
  - Include three DISTINCT operation
  - Include UNION data set operation
  - Include NOT IN subquery

Data processing as a whole

```
SQL> SELECT TABLESPACE_NAME, MAX(AUTOEXTENSIBLE)
  2  FROM DBA_DATA_FILES
  3  GROUP BY TABLESPACE_NAME;

TABLESPACE_NAME                MAX
------------------------------ ---
SYSAUX                         YES
UNDOTBS1                       YES
USERS                          YES
TEST                           NO
SYSTEM                         YES
```

For strings, 'YES' is bigger than 'NO'.

Data processing as a whole



Question: In a 5x5 chessboard, each row, column, and slash (the slashes not only include diagonal lines) can hold up to two balls. In order to place as many balls as possible, how many ways do you have in total?

Demand: Implementation with one SQL statement.

Output format: From the first row to the fifth row of the checkerboard, each row is sequentially output from the first column to the fifth column. 0 means not to put the ball and 1 means to put. For example : 10010000000000000000000.

Data processing as a whole

```
SQL> with c as
  2    (select rownum - 1 c from dual connect by rownum <= 2),
  3  line_seed as
  4    (select c1.c || c2.c || c3.c || c4.c || c5.c line,
  5    c1.c c1, c2.c c2, c3.c c3, c4.c c4, c5.c c5
  6    from c c1, c c2, c c3, c c4, c c5
  7    where c1.c + c2.c + c3.c + c4.c + c5.c = 2
  8    order by 1 desc)
  9  select * from line_seed;

LINE                C1         C2         C3         C4         C5
---------- ---------- ---------- ---------- ---------- ----------
11000                1          1          0          0          0
10100                1          0          1          0          0
10010                1          0          0          1          0
10001                1          0          0          0          1
01100                0          1          1          0          0
01010                0          1          0          1          0
01001                0          1          0          0          1
00110                0          0          1          1          0
00101                0          0          1          0          1
00011                0          0          0          1          1

10 rows selected.
```

Data processing as a whole

```
with c as
(select rownum - 1 c from dual connect by rownum <= 2),
line_seed as
(select c1.c || c2.c || c3.c || c4.c || c5.c line, c1.c c1, c2.c c2, c3.c c3, c4.c c4, c5.c c5
from c c1, c c2, c c3, c c4, c c5
where c1.c + c2.c + c3.c + c4.c + c5.c = 2
order by 1 desc)
select rownum, line1.line || line2.line || line3.line || line4.line || line5.line result
from line_seed line1, line_seed line2, line_seed line3, line_seed line4, line_seed line5
where line1.c1 + line2.c1 + line3.c1 + line4.c1 + line5.c1 = 2
  and line1.c2 + line2.c2 + line3.c2 + line4.c2 + line5.c2 = 2
and line1.c3 + line2.c3 + line3.c3 + line4.c3 + line5.c3 = 2
and line1.c4 + line2.c4 + line3.c4 + line4.c4 + line5.c4 = 2
and line1.c5 + line2.c5 + line3.c5 + line4.c5 + line5.c5 = 2
and line3.c1 + line4.c2 + line5.c3 <= 2
and line2.c1 + line3.c2 + line4.c3 + line5.c4 <= 2
and line1.c1 + line2.c2 + line3.c3 + line4.c4 + line5.c5 <= 2
and line1.c2 + line2.c3 + line3.c4 + line4.c5 <= 2
and line1.c3 + line2.c4 + line3.c5 <= 2
and line1.c3 + line2.c2 + line3.c1 <= 2
and line1.c4 + line2.c3 + line3.c2 + line4.c1 <= 2
and line1.c5 + line2.c4 + line3.c3 + line4.c2 + line5.c1 <= 2
and line2.c5 + line3.c4 + line4.c3 + line5.c2 <= 2
and line3.c5 + line4.c4 + line5.c3 <= 2;
```

Data processing as a whole

```
SQL> with c as
  2  (select rownum - 1 c from dual connect by rownum <= 2),
  3  lines as
  4  (select to_number(c1.c || c2.c || c3.c || c4.c || c5.c) line
  5  from c c1, c c2, c c3, c c4, c c5
  6  where c1.c + c2.c + c3.c + c4.c + c5.c = 2
  7  order by 1 desc)
  8  select ltrim(to_char(line1.line, '09999')) || chr(10)
  9    || ltrim(to_char(line2.line, '09999')) || chr(10)
 10    || ltrim(to_char(line3.line, '09999')) || chr(10)
 11    || ltrim(to_char(line4.line, '09999')) || chr(10)
 12    || ltrim(to_char(line5.line, '00009')) result
 13  from lines line1, lines line2, lines line3, lines line4, lines line5
 14  where line1.line + line2.line + line3.line + line4.line + line5.line = 22222
 15  and ltrim(to_char(line1.line + 10*line2.line + 100*line3.line + 1000*line4.line +
10000*line5.line), '012') is null
 16  and ltrim(to_char(10000*line1.line + 1000*line2.line + 100*line3.line + 10*line4.line +
line5.line), '012') is null
 17  and rownum = 1;

RESULT
--------------------------------
11000
10010
00011
01100
00101
```

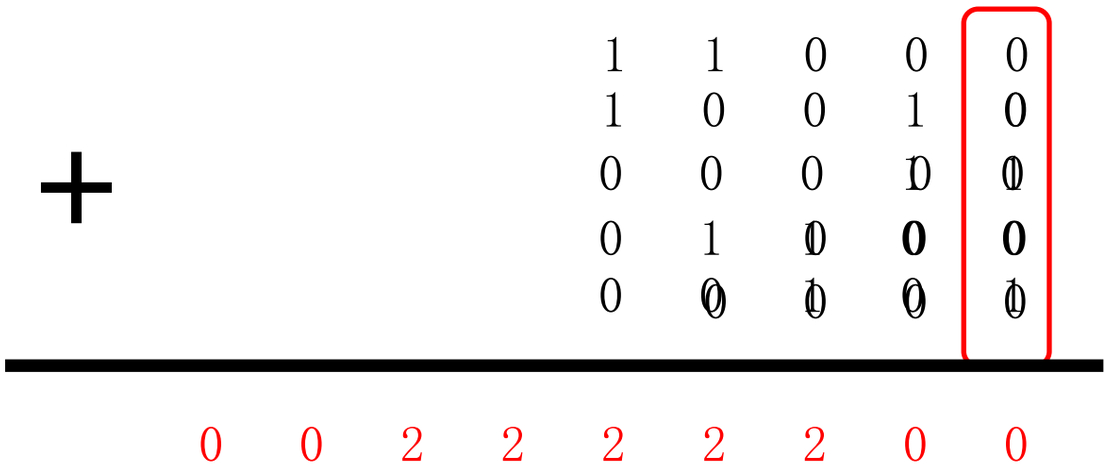
Data processing as a whole

```
 where line1.line + line2.line + line3.line + line4.line + line5.line = 22222
 and ltrim(to_char(line1.line + 10*line2.line + 100*line3.line + 1000*line4.line +
10000*line5.line), '012') is null
 and ltrim(to_char(10000*line1.line + 1000*line2.line + 100*line3.line + 10*line4.line +
line5.line), '012') is null

RESULT
-----------------------------------
```

	1	1	0	0	0
	1	0	0	1	0
+	0	0	0	1	1
	0	1	1	0	0
	0	0	1	0	1
	2	2	2	2	2

Data processing as a whole

```
 where line1.line + line2.line + line3.line + line4.line + line5.line = 22222
 and ltrim(to_char(line1.line + 10*line2.line + 100*line3.line + 1000*line4.line +
10000*line5.line), '012') is null
 and ltrim(to_char(10000*line1.line + 1000*line2.line + 100*line3.line + 10*line4.line +
line5.line), '012') is null

RESULT
-----------------------------------
```



Data processing as a whole

```
 where line1.line + line2.line + line3.line + line4.line + line5.line = 22222
 and ltrim(to_char(line1.line + 10*line2.line + 100*line3.line + 1000*line4.line +
10000*line5.line), '012') is null
 and ltrim(to_char(10000*line1.line + 1000*line2.line + 100*line3.line + 10*line4.line +
line5.line), '012') is null

RESULT
-----------------------------------
```

	1	1	0	0	0	0	0	0
		1	0	0	1	0	0	0
+			0	0	0	1	1	0	0
				0	1	1	0	0
					0	0	1	0	1
	1	2	0	0	2	2	2	0	1

Data processing as a whole

- Tom said:
  - You should do it in a single SQL statement if at all possible
  - If you cannot do it in a single SQL Statement, then do it in PL/SQL
  - If you cannot do it in PL/SQL, try a Java Stored Procedure.

- We said:
  - Do not use multiple SQL if you can use single.
  - Do not scan multiple times if you can scan once.
  - Processing the data as a whole not to process row by row.

1. Use new feature reasonably
2. Data processing as a whole
3. Design SQL execution plan
4. Filter unnecessary data from beginning

Design SQL execution plan

A customer's SQL statement is slow to execute and consumes a lot of logical reads

```
SQL> EXPLAIN PLAN FOR
  2  SELECT VCM.POL_ID, TC.MOBILE, TC.FIRST_NAME PH_NAME,
  3     PKG_UTILS.F_GET_DESC('T_TITLE', DECODE(TC.GENDER,'M', '1', 'F', '2'), '211'),
  4     PKG_UTILS.F_GET_DESC('V_PRO_LIFE_3',
  5              (SELECT T.PROD_ID FROM T_CON_PROD T WHERE T.POL_ID = VCM.POL_ID AND T.MASTER_ID IS NULL), '211'),
  6     SUM(TPA.FEE_AMOUNT), VCM.POLICY_CODE, PKG_UTILS.F_GET_DESC('T_LIA_STAT',VCM.LIA_STATE, '211'),
  7     PKG_UTILS.F_GET_DESC('T_SAL_CHAN', VCM.CHANN_TYPE, '211'), VCM.CHANN_TYPE,TC.CUSTOMER_ID, TCO.ORGAN_CODE
  8  FROM V_CON_MAS VCM, T_CON_MAS TCM, T_AGENT TA, T_CUSTOMER TC,
  9     T_POH VPH,T_COM_ORG TCO, T_P_A TPA
 10  WHERE VCM.POL_ID = VPH.POL_ID
 11  AND VCM.SERVICE_AGENT = TA.AGENT_ID
 12  AND VCM.POL_ID = TCM.POL_ID
 13  AND VPH.PARTY_ID = TC.CUSTOMER_ID
 14  AND VCM.ORGAN_ID = TCO.ORGAN_ID
 15  AND VCM.POL_ID = TPA.POL_ID
 16  AND VCM.LIABILITY_STATE = 1
 17  AND TPA.FEE_TYPE IN (41, 569)
 18  AND TPA.FEE_STATUS <> 2
 19  AND (EXISTS (
 20     SELECT 1 FROM T_PO_CH T, T_CH TC WHERE T.MAS_C_ID = TC.CH_ID  AND TC.CH_ID = TPA.CH_ID AND T.SERV_ID = 3 ))
 21  AND TPA.DUE_TIME >= (TRUNC(:B1 ) + 7)
 22  AND TPA.DUE_TIME < (TRUNC(:B1 ) + 8)
 23  AND REGEXP_LIKE(TRIM(TC.MOBILE), '^\d{11}$')
 24  AND NOT EXISTS (
 25     SELECT 1 FROM T_DATA_EXT SDE, T_DATA TSD
 26     WHERE SDE.DATA_ID = TSD.DATA_ID AND SDE.RELAT_VALUE_1 = VCM.POL_ID AND TSD.SMS_ID = 12 AND SDE.RELAT_VALUE_2
= TO_CHAR(:B1 , 'yyyy-MM-dd'))
 27  GROUP BY VCM.POL_ID, TC.MOBILE, TC.FIRST_NAME, TC.GENDER, VCM.POLICY_CODE, VCM.LIABILITY_STATE, VCM.CHANNEL_TYPE
, TC.CUSTOMER_ID, TCO.ORGAN_CODE;
```

Design SQL execution plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		356	53400	901K (1)
* 1	INDEX RANGE SCAN	IDX_CON_PROD__PRD_ID	1	12	1 (0)
2	HASH GROUP BY		356	53400	901K (1)
* 3	FILTER				
* 4	FILTER				
5	NESTED LOOPS		356	53400	84579 (2)
* 6	HASH JOIN		356	51264	84578 (2)
7	TABLE ACCESS FULL	T_C_O	143	2145	4 (0)
8	NESTED LOOPS SEMI		356	45924	84573 (2)
9	NESTED LOOPS		370	44030	84481 (2)
10	NESTED LOOPS		6067	539K	82962 (2)
11	NESTED LOOPS		6036	465K	81452 (2)
* 12	HASH JOIN		6037	430K	81451 (2)
* 13	TABLE ACCESS BY INDEX ROWID	T_P_A	4812	145K	596 (1)
* 14	INDEX RANGE SCAN	IDX_P_A__DUE	14477		10 (0)
15	VIEW	V_CON_MAS	877K	35M	80844 (2)
16	UNION-ALL				
* 17	TABLE ACCESS FULL	T_CON_MAS_LOG	877K	36M	80844 (2)
* 18	FILTER				
* 19	TABLE ACCESS FULL	T_CON_MAS	155K	6357K	21715 (1)
* 20	INDEX UNIQUE SCAN	PK_T_AGENT	1	6	1 (0)
21	TABLE ACCESS BY INDEX ROWID	T_POH	1	12	1 (0)
* 22	INDEX RANGE SCAN	IDX_POH_PO_ID	1		1 (0)
* 23	TABLE ACCESS BY INDEX ROWID	T_CUSTOMER	1	28	1 (0)
* 24	INDEX UNIQUE SCAN	PK_T_CUSTOMER	1		1 (0)
* 25	TABLE ACCESS BY INDEX ROWID	T_PO_CH	244K	2389K	1 (0)
* 26	INDEX RANGE SCAN	IDX_PO_CH__MAS_CH_ID	1		1 (0)
* 27	INDEX UNIQUE SCAN	PK_T_CON_MAS	1	6	1 (0)
28	NESTED LOOPS				
29	NESTED LOOPS		1	34	4591 (1)
30	TABLE ACCESS BY INDEX ROWID	T_DATA_EXT	1	25	4590 (1)
* 31	INDEX SKIP SCAN	IDX_DATA_EXT__RELAT_	1		4589 (1)
* 32	INDEX UNIQUE SCAN	PK_T_DATA	1		1 (0)
* 33	TABLE ACCESS BY INDEX ROWID	T_DATA	1	9	1 (0)

Design SQL execution plan

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

Table created.

SQL> create index ind_obj_id on t_objects(object_id);

Index created.

SQL> create table t_tab as select * from dba_tables;

Table created.

SQL> create table t_ind as select * from dba_indexes;

Table created.

SQL> create index ind_tab_name on t_tab(table_name);

Index created.

SQL> create index ind_ind_name on t_ind(index_name);

Index created.

SQL> create view v_seg as
  2  select owner, table_name, tablespace_name, blocks from t_tab where temporary = 'N'
  3  union all
  4  select owner, index_name, tablespace_name, num_rows from t_ind where status = 'N/A';

View created.
```

Design SQL execution plan

```
SQL> select obj.owner, obj.object_name, created, v.blocks
  2  from t_objects obj, v_seg v
  3  where obj.object_id = 12345
  4  and obj.object_name = v.table_name;

no rows selected
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	194	43
* 1	HASH JOIN		1	194	43
2	TABLE ACCESS BY INDEX ROWID	T_OBJECTS	1	115	2
* 3	INDEX RANGE SCAN	IND_OBJ_ID	1		1
4	VIEW	V_SEG	3340	257K	40
5	UNION-ALL				
* 6	TABLE ACCESS FULL	T_TAB	1184	95904	15
* 7	TABLE ACCESS FULL	T_IND	2156	178K	25

Design SQL execution plan

```
SQL> select /*+ index(t_tab ind_tab_name) */ obj.owner, obj.object_name, created, v.blocks
  2  from t_objects obj, v_seg v
  3  where obj.object_id = 12345
  4  and obj.object_name = v.table_name;

no rows selected
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	194	43
* 1	HASH JOIN		1	194	43
2	TABLE ACCESS BY INDEX ROWID	T_OBJECTS	1	115	2
* 3	INDEX RANGE SCAN	IND_OBJ_ID	1		1
4	VIEW	V_SEG	3340	257K	40
5	UNION-ALL				
* 6	TABLE ACCESS FULL	T_TAB	1184	95904	15
* 7	TABLE ACCESS FULL	T_IND	2156	178K	25

Design SQL execution plan

```
SQL> select /*+ index(v.t_tab ind_tab_name) */ obj.owner, obj.object_name, created, v.blocks
  2  from t_objects obj, v_seg v
  3  where obj.object_id = 12345
  4  and obj.object_name = v.table_name;

no rows selected
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	194	1210
* 1	HASH JOIN		1	194	1210
2	TABLE ACCESS BY INDEX ROWID	T_OBJECTS	1	115	2
* 3	INDEX RANGE SCAN	IND_OBJ_ID	1		1
4	VIEW	V_SEG	3340	257K	1207
5	UNION-ALL				
* 6	TABLE ACCESS BY INDEX ROWID	T_TAB	1184	95904	1182
7	INDEX FULL SCAN	IND_TAB_NAME	2367		11
* 8	TABLE ACCESS FULL	T_IND	2156	178K	25

Design SQL execution plan

```
SQL> select obj.owner, obj.object_name, created, v.blocks
  2  from t_objects obj,
  3  (select owner, table_name, tablespace_name, blocks from t_tab where temporary = 'N'
  4  union all
  5  select owner, index_name, tablespace_name, num_rows from t_ind where status = 'N/A') v
  6  where obj.object_id = 12345
  7  and obj.object_name = v.table_name;

no rows selected
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	194	43
* 1	HASH JOIN		1	194	43
2	TABLE ACCESS BY INDEX ROWID	T_OBJECTS	1	115	2
* 3	INDEX RANGE SCAN	IND_OBJ_ID	1		1
4	VIEW		3340	257K	40
5	UNION-ALL				
* 6	TABLE ACCESS FULL	T_TAB	1184	95904	15
* 7	TABLE ACCESS FULL	T_IND	2156	178K	25

Design SQL execution plan

```
SQL> select /*+ index(v.t_tab ind_tab_name) */ obj.owner, obj.object_name, created, v.blocks
  2  from t_objects obj,
  3  (select owner, table_name, tablespace_name, blocks from t_tab where temporary = 'N'
  4  union all
  5  select owner, index_name, tablespace_name, num_rows from t_ind where status = 'N/A') v
  6  where obj.object_id = 12345
  7  and obj.object_name = v.table_name;

no rows selected
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	194	1210
* 1	HASH JOIN		1	194	1210
2	TABLE ACCESS BY INDEX ROWID	T_OBJECTS	1	115	2
* 3	INDEX RANGE SCAN	IND_OBJ_ID	1		1
4	VIEW		3340	257K	1207
5	UNION-ALL				
* 6	TABLE ACCESS BY INDEX ROWID	T_TAB	1184	95904	1182
7	INDEX FULL SCAN	IND_TAB_NAME	2367		11
* 8	TABLE ACCESS FULL	T_IND	2156	178K	25

Design SQL execution plan

```
SQL> select /*+ index(t_tab ind_tab_name) */ obj.owner, obj.object_name, created, blocks
  2  from t_objects obj,
  3  (select owner, table_name, tablespace_name, blocks from t_tab where temporary = 'N'
  4  union all
  5  select owner, index_name, tablespace_name, num_rows from t_ind where status = 'N/A')
  6  where obj.object_id = 12345
  7  and obj.object_name = table_name;

no rows selected
```

```
-------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |     1 |   194 |    43 |
|*  1 |  HASH JOIN                    |             |     1 |   194 |    43 |
|   2 |   TABLE ACCESS BY INDEX ROWID| T_OBJECTS   |     1 |   115 |     2 |
|*  3 |    INDEX RANGE SCAN           | IND_OBJ_ID  |     1 |       |     1 |
|   4 |   VIEW                        |             |  3340 |   257K|    40 |
|   5 |    UNION-ALL                  |             |       |       |       |
|*  6 |     TABLE ACCESS FULL         | T_TAB       |  1184 | 95904 |    15 |
|*  7 |     TABLE ACCESS FULL         | T_IND       |  2156 |   178K|    25 |
-------------------------------------------------------------------------------------
```

Design SQL execution plan

```
SQL> SELECT ID, OBJECT_ALIAS, DEPTH
  2  FROM V$SQL_PLAN
  3  WHERE SQL_ID IN
  4  (SELECT SQL_ID FROM V$SQL
  5  WHERE SQL_TEXT LIKE 'select /*+ index(t_tab ind_tab_name) */%')
  6  ORDER BY SQL_ID, ID;

        ID OBJECT_ALIAS                          DEPTH
---------- ------------------------------ ----------
         0                                         0
         1                                         1
         2 OBJ@SEL$1                               2
         3 OBJ@SEL$1                               3
         4 from$_subquery$_002@SEL$1               2
         5                                         3
         6 T_TAB@SEL$2                             4
         7 T_IND@SEL$3                             4

-------------------------------------------------------------------------------
| Id  | Operation                      | Name        | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |             |     1 |   194 |    43 |
|*  1 |  HASH JOIN                     |             |     1 |   194 |    43 |
|   2 |   TABLE ACCESS BY INDEX ROWID| T_OBJECTS   |     1 |   115 |     2 |
|*  3 |    INDEX RANGE SCAN            | IND_OBJ_ID  |     1 |       |     1 |
|   4 |   VIEW                         |             |  3340 |   257K|    40 |
|   5 |    UNION-ALL                   |             |       |       |       |
|*  6 |     TABLE ACCESS FULL          | T_TAB       |  1184 | 95904 |    15 |
|*  7 |     TABLE ACCESS FULL          | T_IND       |  2156 |   178K|    25 |
-------------------------------------------------------------------------------
```

Design SQL execution plan

```
SQL> select /*+ index("from$_subquery$_002".t_tab ind_tab_name) */ obj.owner, obj.object_name,
created, blocks
  2  from t_objects obj,
  3  (select owner, table_name, tablespace_name, blocks from t_tab where temporary = 'N'
  4  union all
  5  select owner, index_name, tablespace_name, num_rows from t_ind where status = 'N/A')
  6  where obj.object_id = 12345
  7  and obj.object_name = table_name;

no rows selected
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation                      | Name          | Rows  | Bytes | Cost  |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |               |     1 |   194 |  1210 |
|*  1 |  HASH JOIN                     |               |     1 |   194 |  1210 |
|   2 |   TABLE ACCESS BY INDEX ROWID  | T_OBJECTS     |     1 |   115 |     2 |
|*  3 |    INDEX RANGE SCAN            | IND_OBJ_ID    |     1 |       |     1 |
|   4 |   VIEW                         |               |  3340 |  257K |  1207 |
|   5 |    UNION-ALL                   |               |       |       |       |
|*  6 |     TABLE ACCESS BY INDEX ROWID| T_TAB         |  1184 | 95904 |  1182 |
|   7 |      INDEX FULL SCAN           | IND_TAB_NAME  |  2367 |       |    11 |
|*  8 |     TABLE ACCESS FULL          | T_IND         |  2156 |  178K |    25 |
-----------------------------------------------------------------------------------------
```

Design SQL execution plan

```
SQL> select /*+ index(@v_1 t_tab ind_tab_name) */ obj.owner, obj.object_name, created, blocks
  2  from t_objects obj,
  3  (select /*+ qb_name(v_1) */ owner, table_name, tablespace_name, blocks from t_tab where
temporary = 'N'
  4  union all
  5  select owner, index_name, tablespace_name, num_rows from t_ind where status = 'N/A')
  6  where obj.object_id = 12345
  7  and obj.object_name = table_name;

no rows selected
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes | Cost  |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |     1 |   123 |  1210 |
|*  1 |  HASH JOIN                          |               |     1 |   123 |  1210 |
|   2 |   TABLE ACCESS BY INDEX ROWID       | T_OBJECTS     |     1 |    44 |     2 |
|*  3 |    INDEX RANGE SCAN                 | IND_OBJ_ID    |     1 |       |     1 |
|   4 |   VIEW                              |               |  3340 |  257K |  1207 |
|   5 |    UNION-ALL                        |               |       |       |       |
|*  6 |     TABLE ACCESS BY INDEX ROWID     | T_TAB         |  1184 | 28416 |  1182 |
|   7 |      INDEX FULL SCAN                | IND_TAB_NAME  |  2367 |       |    11 |
|*  8 |     TABLE ACCESS FULL               | T_IND         |  2156 | 66836 |    25 |
-----------------------------------------------------------------------------------------
```

Design SQL execution plan

```
SQL> select /*+ push_pred(v) */ obj.owner, obj.object_name, created, v.blocks
  2  from t_objects obj, v_seg v
  3  where obj.object_id = 12345
  4  and obj.object_name = v.table_name;

no rows selected
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		1	78	6 (0)
1	NESTED LOOPS		1	78	6 (0)
2	TABLE ACCESS BY INDEX ROWID BATCHED	T_OBJECTS	1	44	2 (0)
* 3	INDEX RANGE SCAN	IND_OBJ_ID	1		1 (0)
4	VIEW	V_SEG	1	34	4 (0)
5	UNION ALL PUSHED PREDICATE				
* 6	TABLE ACCESS BY INDEX ROWID BATCHED	T_TAB	1	24	2 (0)
* 7	INDEX RANGE SCAN	IND_TAB_NAME	1		1 (0)
* 8	TABLE ACCESS BY INDEX ROWID BATCHED	T_IND	1	31	2 (0)
* 9	INDEX RANGE SCAN	IND_IND_NAME	1		1 (0)

Design SQL execution plan

```
SQL> alter session set "_optimizer_push_pred_cost_based"=false;

Session altered.
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		286	42900	659K (1)
* 1	INDEX RANGE SCAN	IDX_CON_PROD__PRD_ID	1	12	1 (0)
2	HASH GROUP BY		286	42900	659K (1)
.					
.					
.					
* 12	TABLE ACCESS BY INDEX ROWID	T_P_A	4812	145K	596 (1)
* 13	INDEX RANGE SCAN	IDX_P_A__DUE	14477		10 (0)
* 14	VIEW	V_CON_MAS	1	42	1 (0)
15	UNION-ALL PARTITION				
* 16	TABLE ACCESS BY INDEX ROWID	T_CON_MAS_LOG	1	44	1 (0)
* 17	INDEX RANGE SCAN	IDX_CON_MAS_L__LAST_CM	1		1 (0)
* 18	FILTER				
* 19	TABLE ACCESS BY INDEX ROWID	T_CON_MAS	1	42	1 (0)
* 20	INDEX UNIQUE SCAN	PK_T_CON_MAS	1		1 (0)
21	NESTED LOOPS				
22	NESTED LOOPS		1	34	4591 (1)
23	TABLE ACCESS BY INDEX ROWID	T_DATA_EXT	1	25	4590 (1)
.					
.					
.					
* 34	INDEX UNIQUE SCAN	PK_T_CON_MAS	1	6	1 (0)
* 35	INDEX UNIQUE SCAN	PK_T_COM_ORG	1		1 (0)
36	TABLE ACCESS BY INDEX ROWID	T_COM_ORG	1	15	1 (0)

Design SQL execution plan

The importance of use the HINT:

- Control SQL execution
- Understand the difference in the efficiency of the execution plan
- A powerful tool for analyzing SQL problems
- Bypass bugs or performance issues
- Force SQL to execute according to design ideas

1. Use new feature reasonably
2. Data processing as a whole
3. Design SQL execution plan
4. Filter unnecessary data from beginning

Filter unnecessary data from beginning

How to find all prime numbers within 100 through one SQL:

```
SQL> WITH T
  2  AS
  3  (SELECT ROWNUM RN FROM DUAL CONNECT BY LEVEL < 100)
  4  SELECT RN FROM T
  5  WHERE RN > 1
  6  MINUS
  7  SELECT A.RN * B.RN FROM T A, T B
  8  WHERE A.RN <= B.RN
  9  AND A.RN > 1
 10  AND B.RN > 1;

        RN
----------
         2
         3
         5
.
.
.
        97

25 rows selected.

Elapsed: 00:00:00.09
```

Filter unnecessary data from beginning

```
SQL> WITH T
  2  AS
  3  (SELECT ROWNUM RN FROM DUAL CONNECT BY LEVEL < 10000)
  4  SELECT RN FROM T
  5  WHERE RN > 1
  6  MINUS
  7  SELECT A.RN * B.RN FROM T A, T B
  8  WHERE A.RN <= B.RN
  9  AND A.RN > 1
 10  AND B.RN > 1;

1229 rows selected.

Elapsed: 00:01:12.50

Statistics
----------------------------------------------------------
        543  recursive calls
        148  db block gets
     170002  consistent gets
      69148  physical reads
        884  redo size
      21808  bytes sent via SQL*Net to client
       1415  bytes received via SQL*Net from client
         83  SQL*Net roundtrips to/from client
          2  sorts (memory)
          1  sorts (disk)
       1229  rows processed
```

Filter unnecessary data from beginning

```
SQL> WITH T
  2  AS
  3  (SELECT ROWNUM RN FROM DUAL CONNECT BY LEVEL < 10000)
  4  SELECT RN FROM T
  5  WHERE RN > 1
  6  MINUS
  7  SELECT A.RN * B.RN FROM T A, T B
  8  WHERE A.RN <= B.RN
  9  AND A.RN > 1
 10  AND A.RN <= 100
 11  AND B.RN > 1
 12  AND B.RN <= 5000;

1229 rows selected.

Elapsed: 00:00:00.52

Statistics
----------------------------------------------------------
          9  recursive calls
         24  db block gets
       1727  consistent gets
         17  physical reads
        604  redo size
      21808  bytes sent via SQL*Net to client
       1415  bytes received via SQL*Net from client
         83  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)
       1229  rows processed
```

Filter unnecessary data from beginning

```
SQL> WITH T
  2  AS
  3  (SELECT ROWNUM * 2 + 1 RN FROM DUAL CONNECT BY LEVEL < 4999)
  4  SELECT 2 FROM DUAL
  5  UNION ALL
  6  (
  7   SELECT RN FROM T
  8   MINUS
  9   SELECT A.RN * B.RN FROM T A, T B
 10   WHERE A.RN <= B.RN
 11   AND A.RN <= 100
 12   AND B.RN <= 5000
 13  );

1229 rows selected.

Elapsed: 00:00:00.13

Statistics
----------------------------------------------------------
          9  recursive calls
         16  db block gets
        469  consistent gets
          9  physical reads
        604  redo size
      21807  bytes sent via SQL*Net to client
       1415  bytes received via SQL*Net from client
         83  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)
       1229  rows processed
```

Filter unnecessary data from beginning

```
SQL> WITH T
  2  AS
  3  (SELECT ROWNUM * 2 + 1 RN FROM DUAL CONNECT BY LEVEL < 4999)
  4  SELECT 2 FROM DUAL
  5  UNION ALL
  6  (
  7   SELECT RN FROM T
  8   MINUS
  9   SELECT A.RN * B.RN FROM T A, T B
 10   WHERE A.RN <= B.RN
 11   AND A.RN <= 99
 12   AND B.RN <= 3333
 13   AND A.RN * B.RN < 10000);

1229 rows selected.

Elapsed: 00:00:00.05

Statistics
----------------------------------------------------------
          2  recursive calls
         19  db block gets
        461  consistent gets
          9  physical reads
        940  redo size
      21807  bytes sent via SQL*Net to client
       1415  bytes received via SQL*Net from client
         83  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)
       1229  rows processed
```

Filter unnecessary data from beginning

The issue of how pirate is dividing the gold :

There were five pirates who looted 100 gold coins and wanted to divide the gold coins. The first step they will take randomly from the five notes written from 1 to 5, the number they get will determine their order.

The luckier get 1 can propose a distribution plan. If his plan is approved by more than half of the pirates, he will divide the gold according to his plan, otherwise the first person will be killed. Then the one who get 2 can propose a distribution plan and continue the above rules.

So the question is: If you get 1, what kind of distribution plan would you propose?

(We assume that every pirate is smart enough and pursues maximization of benefits.)

Filter unnecessary data from beginning

Distribution Principle for One: Exclusive

```
SQL> WITH A AS
  2  (SELECT 100 - ROWNUM + 1 N FROM DUAL CONNECT BY ROWNUM <= 101),
  3  MAX_ONE AS
  4  (SELECT MAX(N) MAX1 FROM A)
  5  SELECT * FROM MAX_ONE;

      MAX1
----------
       100
```

Filter unnecessary data from beginning

Distribution Principle for Two: Sadness

```
SQL> WITH A AS
  2  (SELECT 100 - ROWNUM + 1 N FROM DUAL CONNECT BY ROWNUM <= 101),
  3  MAX_ONE AS
  4  (SELECT MAX(N) MAX1 FROM A),
  5  MAX_TWO AS
  6  (SELECT /*+ LEADING (P2, P1) USE_NL(P1) */ P2.N MAX2, P1.N MAX1
  7  FROM A P1, A P2
  8  WHERE P1.N + P2.N = 100
  9  AND P1.N >= (SELECT MAX1 FROM MAX_ONE)
 10  AND ROWNUM = 1)
 11  SELECT * FROM MAX_TWO;

      MAX2       MAX1
---------- ----------
         0        100
```

Filter unnecessary data from beginning

Distribution Principle for Three: Lure

```
SQL> WITH A AS
  2  (SELECT 100 - ROWNUM + 1 N FROM DUAL CONNECT BY ROWNUM <= 101),
  3  MAX_ONE AS
  4  (SELECT MAX(N) MAX1 FROM A),
  5  MAX_TWO AS
  6  (SELECT /*+ LEADING (P2, P1) USE_NL(P1) */ P2.N MAX2, P1.N MAX1
  7  FROM A P1, A P2
  8  WHERE P1.N + P2.N = 100
  9  AND P1.N >= (SELECT MAX1 FROM MAX_ONE)
 10  AND ROWNUM = 1),
 11  MAX_THREE AS
 12  (SELECT /*+ LEADING(P3, P2, P1) USE_NL(P1) */ P3.N MAX3, P2.N MAX2, P1.N MAX1
 13  FROM A P1, A P2, A P3, MAX_TWO
 14  WHERE P1.N + P2.N + P3.N = 100
 15  AND SIGN(P2.N - MAX2) + SIGN(P1.N - MAX1) >=0
 16  AND ROWNUM = 1)
 17  SELECT * FROM MAX_THREE;

      MAX3       MAX2       MAX1
---------- ---------- ----------
        99          1          0
```

Filter unnecessary data from beginning

Distribution Principle for Four: Exclude

```
SQL> WITH A AS
  2  (SELECT 100 - ROWNUM + 1 N FROM DUAL CONNECT BY ROWNUM <= 101),
  3  MAX_ONE AS
  4  (SELECT MAX(N) MAX1 FROM A),
  5  MAX_TWO AS
  6  (SELECT /*+ LEADING (P2, P1) USE_NL(P1) */ P2.N MAX2, P1.N MAX1
  7  FROM A P1, A P2
  8  WHERE P1.N + P2.N = 100
  9  AND P1.N >= (SELECT MAX1 FROM MAX_ONE)
 10  AND ROWNUM = 1),
 11  MAX_THREE AS
 12  (SELECT /*+ LEADING(P3, P2, P1) USE_NL(P1) */ P3.N MAX3, P2.N MAX2, P1.N MAX1
 13  FROM A P1, A P2, A P3, MAX_TWO
 14  WHERE P1.N + P2.N + P3.N = 100
 15  AND SIGN(P2.N - MAX2) + SIGN(P1.N - MAX1) >=0
 16  AND ROWNUM = 1),
 17  MAX_FOUR AS
 18  (SELECT /*+ LEADING(P4, P3, P2, P1) USE_NL(P3) USE_NL(P2) USE_NL(P1) */ P4.N MAX4, P3.N MAX3, P2.N
MAX2, P1.N MAX1
 19  FROM A P1, A P2, A P3, A P4, MAX_THREE
 20  WHERE P1.N + P2.N + P3.N + P4.N = 100
 21  AND SIGN(P3.N - MAX3) + SIGN(P2.N - MAX2) + SIGN(P1.N - MAX1) > 0
 22  AND ROWNUM = 1)
 23  SELECT * FROM MAX_FOUR;

      MAX4       MAX3       MAX2       MAX1
---------- ---------- ---------- ----------
        97          0          2          1
```

Filter unnecessary data from beginning

```
SQL> WITH A AS
  2  (SELECT 100 - ROWNUM + 1 N FROM DUAL CONNECT BY ROWNUM <= 101),
  3  MAX_ONE AS
  4  (SELECT MAX(N) MAX1 FROM A),
  5  MAX_TWO AS
  6  (SELECT /*+ LEADING (P2, P1) USE_NL(P1) */ P2.N MAX2, P1.N MAX1
  7  FROM A P1, A P2
  8  WHERE P1.N + P2.N = 100
  9  AND P1.N >= (SELECT MAX1 FROM MAX_ONE)
 10  AND ROWNUM = 1),
 11  MAX_THREE AS
 12  (SELECT /*+ LEADING(P3, P2, P1) USE_NL(P1) */ P3.N MAX3, P2.N MAX2, P1.N MAX1
 13  FROM A P1, A P2, A P3, MAX_TWO
 14  WHERE P1.N + P2.N + P3.N = 100
 15  AND SIGN(P2.N - MAX2) + SIGN(P1.N - MAX1) >=0
 16  AND ROWNUM = 1),
 17  MAX_FOUR AS
 18  (SELECT /*+ LEADING(P4, P3, P2, P1) USE_NL(P3) USE_NL(P2) USE_NL(P1) */ P4.N MAX4, P3.N MAX3, P2.N MAX2, P1.N MAX1
 19  FROM A P1, A P2, A P3, A P4, MAX_THREE
 20  WHERE P1.N + P2.N + P3.N + P4.N = 100
 21  AND SIGN(P3.N - MAX3) + SIGN(P2.N - MAX2) + SIGN(P1.N - MAX1) > 0
 22  AND ROWNUM = 1),
 23  FIVE AS
 24  (SELECT /*+ LEADING(P5, P4, P3, P2, P1) USE_NL(P4) USE_NL(P3) USE_NL(P2) USE_NL(P1) */ P5.N N5, P4.N N4, P3.N N3,
P2.N N2, P1.N N1
 25  FROM A P1, A P2, A P3, A P4, A P5, MAX_FOUR
 26  WHERE P1.N + P2.N + P3.N + P4.N + P5.N = 100
 27  AND SIGN(P4.N - MAX4) + SIGN(P3.N - MAX3) + SIGN(P2.N - MAX2) + SIGN(P1.N - MAX1) >= 0
 28  AND ROWNUM = 1)
 29  SELECT * FROM FIVE;
        N5         N4         N3         N2         N1
---------- ---------- ---------- ---------- ----------
        97          0          1          0          2

Elapsed: 00:02:04.19
```

Filter unnecessary data from beginning

```
SQL> WITH A AS
  2  (SELECT 100 - ROWNUM + 1 N, ROWNUM - 1 NA FROM DUAL CONNECT BY ROWNUM <= 101),
  3  MAX_ONE AS
  4  (SELECT MAX(N) MAX1 FROM A),
  5  MAX_TWO AS
  6  (SELECT /*+ LEADING(MAX_ONE, P2, P1) USE_NL(P2) USE_NL(P1) */ DECODE(P1.NA, MAX1, P2.N - 1, P2.N) MAX2, P1.NA MAX1
  7  FROM A P1, A P2, MAX_ONE
  8  WHERE P1.NA + P2.N = 100
  9  AND P1.NA >= MAX1
 10  AND ROWNUM = 1),
 11  MAX_THREE AS
 12  (SELECT /*+ LEADING(MAX_TWO, P3, P2, P1) USE_NL(P3) USE_NL(P2) USE_NL(P1) */ P3.N MAX3, P2.NA MAX2, P1.NA MAX1
 13  FROM A P1, A P2, A P3, MAX_TWO
 14  WHERE P1.NA + P2.NA + P3.N = 100
 15  AND P3.N + P2.NA <= 100
 16  AND CASE WHEN P2.NA > MAX2 THEN 1 ELSE -1 END + CASE WHEN P1.NA > MAX1 THEN 1 ELSE -1 END >= 0
 17  AND ROWNUM = 1),
 18  MAX_FOUR AS
 19  (SELECT /*+ LEADING(MAX_THREE, P4, P3, P2, P1) USE_NL(P4) USE_NL(P3) USE_NL(P2) USE_NL(P1) */ P4.N MAX4, P3.NA MAX3, P2.NA MAX2, P1.NA MAX1
 20  FROM A P1, A P2, A P3, A P4, MAX_THREE
 21  WHERE P1.NA + P2.NA + P3.NA + P4.N = 100
 22  AND P4.N + P3.NA <= 100
 23  AND P4.N + P3.NA + P2.NA <= 100
 24  AND CASE WHEN P3.NA > MAX3 THEN 1 ELSE -1 END + CASE WHEN P2.NA > MAX2 THEN 1 ELSE -1 END + CASE WHEN P1.NA > MAX1 THEN 1 ELSE -1 END > 0
 25  AND ROWNUM = 1),
 26  MAX_FIVE AS
 27  (SELECT /*+ LEADING(MAX_FOUR, P5, P4, P3, P2, P1) USE_NL(P5) USE_NL(P4) USE_NL(P3) USE_NL(P2) USE_NL(P1) */ P5.N N5, P4.NA N4, P3.NA N3, P2.NA N2, P1.NA N1
 28  FROM A P1, A P2, A P3, A P4, A P5, MAX_FOUR
 29  WHERE P1.NA + P2.NA + P3.NA + P4.NA + P5.N = 100
 30  AND P5.N + P4.NA <= 100
 31  AND P5.N + P4.NA + P3.NA <= 100
 32  AND P5.N + P4.NA + P3.NA + P2.NA <= 100
 33  AND CASE WHEN P4.NA > MAX4 THEN 1 ELSE -1 END + CASE WHEN P3.NA > MAX3 THEN 1 ELSE -1 END + CASE WHEN P2.NA > MAX2 THEN 1 ELSE -1 END + CASE WHEN P1.NA > MAX1 THEN 1 ELSE -1 END >= 0
 34  AND ROWNUM = 1),
 35  FIVE AS
 36  (SELECT /*+ LEADING(MAX_FOUR, MAX_FIVE, P5, P4, P3, P2, P1) USE_NL(MAX_FIVE) USE_NL(P5) USE_NL(P4) USE_NL(P3) USE_NL(P2) USE_NL(P1) */ P5.N N5, P4.NA N4, P3.NA N3, P2.NA N2, P1.NA N1
 37  FROM A P1, A P2, A P3, A P4, A P5, MAX_FOUR, MAX_FIVE
 38  WHERE P1.NA + P2.NA + P3.NA + P4.NA + P5.N = 100
 39  AND MAX_FIVE.N5 = P5.N
 40  AND P5.N + P4.NA <= 100
 41  AND P5.N + P4.NA + P3.NA <= 100
 42  AND P5.N + P4.NA + P3.NA + P2.NA <= 100
 43  AND CASE WHEN P4.NA > MAX_FOUR.MAX4 THEN 1 ELSE -1 END
 44     + CASE WHEN P3.NA > MAX_FOUR.MAX3 THEN 1 ELSE -1 END
 45     + CASE WHEN P2.NA > MAX_FOUR.MAX2 THEN 1 ELSE -1 END
 46     + CASE WHEN P1.NA > MAX_FOUR.MAX1 THEN 1 ELSE -1 END >= 0)
 47  SELECT * FROM FIVE;

        N5         N4         N3         N2         N1
---------- ---------- ---------- ---------- ----------
        97          0          1          0          2
        97          0          1          2          0

Elapsed: 00:00:00.02
```

Filter unnecessary data from beginning

```
WITH A AS
.
.
.
FROM A P1, A P2, A P3, MAX_TWO
WHERE P1.NA + P2.NA + P3.N = 100
AND P3.N + P2.NA <= 100
.
.
.
WHERE P1.NA + P2.NA + P3.NA + P4.N = 100
AND P4.N + P3.NA <= 100
AND P4.N + P3.NA + P2.NA <= 100
.
.
.
WHERE P1.NA + P2.NA + P3.NA + P4.NA + P5.N = 100
AND P5.N + P4.NA <= 100
AND P5.N + P4.NA + P3.NA <= 100
AND P5.N + P4.NA + P3.NA + P2.NA <= 100
.
.
.
```

Filter unnecessary data from beginning

- The principle of the minimum first step resultset
- If the conditions that can be put advance then do not put it behind
- Make full use of the ACCESS ability of Indexes
- Do not ignore the FILTER ability of Indexes

Summary

- Dealing with the problem as a whole, avoiding single row operation
- The principle of the minimum first step result set, reasonable choice of driving table
- Use new features like analysis functions to avoid repeated scans and reduce self-join
- Filter out unwanted data as much as possible at each step
- Write more SQL: Practice makes perfect
- Thinking more: The algorithm is key
- Perseverance: Optimization is endless

Enmotech

1 Introduction

Global supplier of end-to-end solutions of data assets

2 Solutions

Cloud Databases, Tuning, Operation and Maintenance, Recovery, consulting (Oracle & Mysql)

3 Products

zData/ MZ3/ZONE/
Yun mirror/DBPaaS

4 Customer&Cases

5000+ database platforms
500+ Enterprises and users in the industry

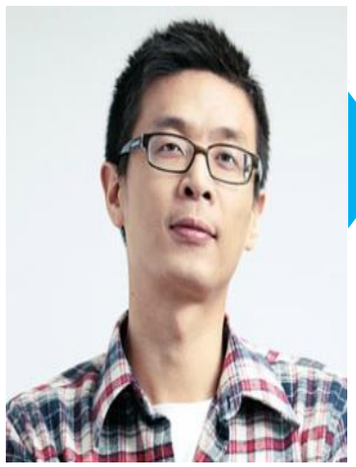
Core Team Member

Well established in Chinese market, Enmotech keeps providing customers with all-around enterprise-class solutions that center data and talent with our globally top-ranking technical personnel (the directors of Oracle ACE, Oracle ACE experts, Oracle OCM experts, DB2 experts, middleware experts, and Unix system experts) and unique service ideas.

Eygle盖国强

Oracle ACE Director

Guoqiang，Gai
Founder of ACOUG and Enmotech.
The First Oracle ACE & ACED in China



Kamus张乐奕

Oracle ACE Director

Co-founder of ACOUG and Enmotech, Extensive hands-on experience with Oracle Database RAC and HA solutions, expertise in fault diagnosis and performance tuning

YangTingkun 杨廷琨

Oracle ACE Director

Enmotech CTO , expertise in fault diagnosis and performance tuning, and known as Oracle Encyclopedia

Xiong Jun熊军

Oracle ACE Director

The core expert of ACOUG, be good at trouble shooting and data recovery. The author of the data recover software Oracle ODU

Core Team Member

Secooler侯圣文

Oracle ACE Director

President of Enmoedu, he has Rich experience in Oracle, MySQL, Hadoop training courses, good at combining theory with practice teaching, easy to understand. As the gold medal lecturer of OCM, he has trained more than 600 OCMs and 10,000 DBAs.

Joel Perez

Oracle ACE Director

Joel is an Expert DBA (Oracle ACE Director, OCM Cloud Admin. & OCM11g) with over 15 years of Real World Experience in Oracle Technology, specialized in design and implement solutions of: High Availability, Disaster Recovery, Upgrades, Replication, Tuning, Cloud and all area related to Oracle Databases.

Roger 李真旭

Oracle ACE

Core expert of Enmotech, Good at Oracle database failure recovery, performance optimization and troubleshooting

Li yinan 李轶楠

Oracle ACE

Good at Oracle database application requirements analysis, architecture design, data modeling, database disaster management, performance optimization.

Customer & Cases & Services

- 31 Provinces
- 5 main filed (Financial Services、Communications operations、Energy Transportation、Government / public utility、Manufacturing / Commerce)
- 17+ Industry
- 500+ Enterprises and users in the industry
- 5000+ database platforms

Singapore

Hong Kong

Taiwan

Websites

http://enmotech.com/

http://en.enmotech.com/

http://blog.enmotech.com/

云和恩墨
ENMOTECH

Thanks for listening

Contact us

Business Cooperation
marketing@enmotech.com
Business Consulting : (+86)
01059007017-7030

云和恩墨
ENMOTECH

数据驱动 成就未来

Make Your Data Dance