Hash join use memory optimization

ITOUG Tech Day – Database Stream
Milano/Roma – Gen/Feb 2019

About Donatello

I work as Oracle consultant in ICTeam

Oracle (from 8i to 18c)

Performance Assessment, optimizer Oracle, SQL, PL/SQL

ITOUG
ITALIAN ORACLE USER GROUP

dosettembrino

donatello.settembrino@icteam.it

icteam
LUTECH GROUP

ICTeam (Lutech group)

more than 130 IT specialists

Grassobbio (BG) - Italy

icteam
LUTECH GROUP

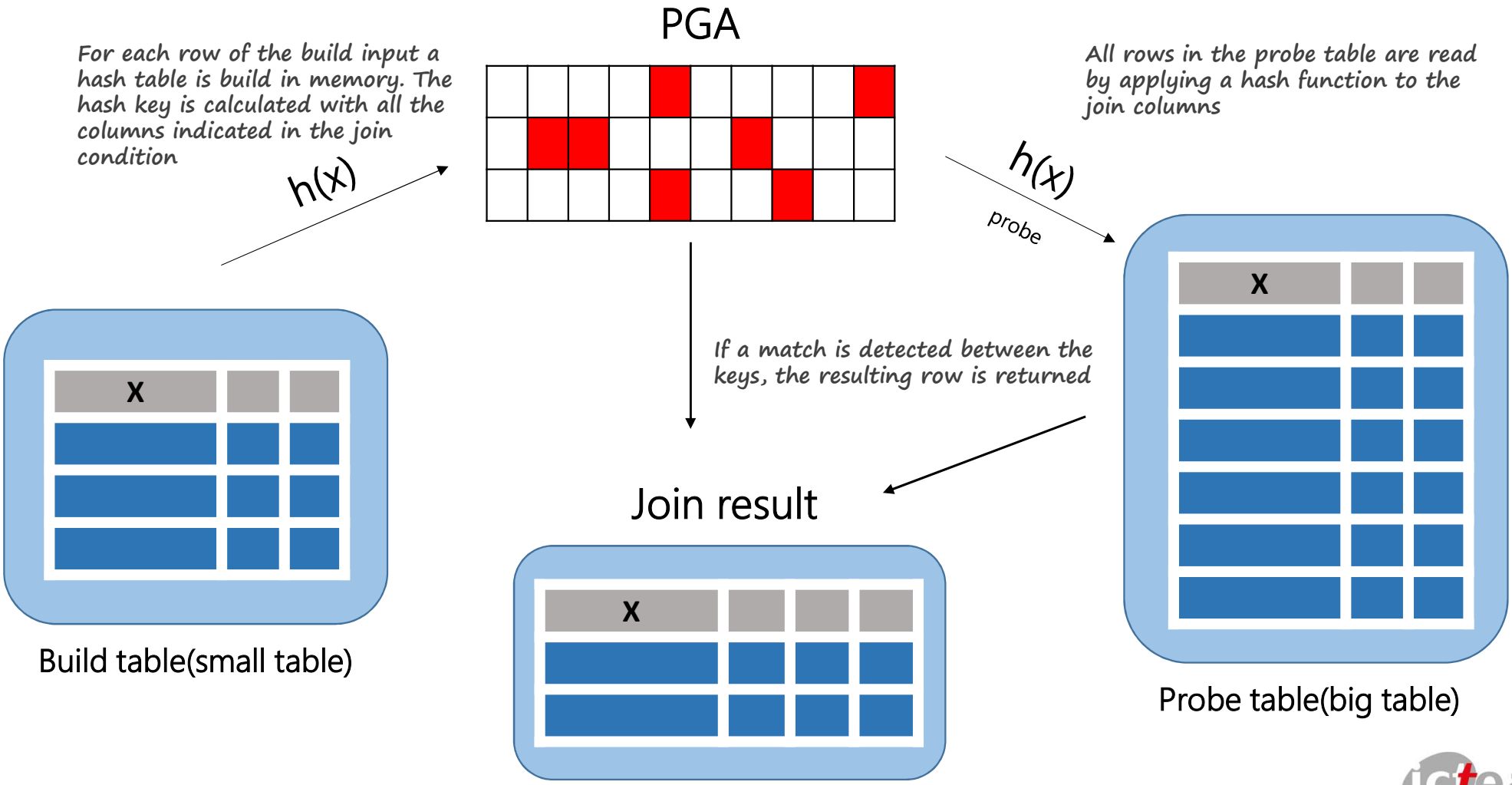
Agenda

- ✓ Hash Join operation
- ✓ Memory allocation
- ✓ Get details with the help of a sql trace
- ✓ How improve the performance
- ✓ Different Hash Join access type

Agenda

- ✓ **Hash Join operation**
- ✓ Memory allocation
- ✓ Get details with the help of a sql trace
- ✓ How improve the performance
- ✓ Different Hash Join access type

Hash Join Operation



Hash Join example

```
SELECT t1.object_name, t1.object_type, t2.data_object_id, t2.status
FROM t1
JOIN t2
ON t1.object_name = t2.object_name;
```

```
-------------------------------------
| Id  | Operation           | Name  |
-------------------------------------
|   0 | SELECT STATEMENT    |       |
|*  1 |  HASH JOIN          |       |
|   2 |   TABLE ACCESS FULL| T1    |
|   3 |   TABLE ACCESS FULL| T2    |
-------------------------------------
```

Agenda

- ✓ Hash Join operation
- ✓ **Memory allocation**
- ✓ Get details with the help of a sql trace
- ✓ How improve the performance
- ✓ Different Hash Join access type

PGA Management – policy of workarea size

workarea_size_policy → {auto, manual};

```
alter session set workarea_size_policy = auto;
```

```
alter session set workarea_size_policy = manual;
alter session set hash_area_size = 1048576; --> 1MB
```

Monitoring workarea

```
SELECT t1.object_name, t1.object_type, t2.data_object_id, t2.status
FROM t1
JOIN t2
ON t1.object_name = t2.object_name;

SELECT sql_id,
       workarea_address,
       policy,
       sid,
       operation_type,
       operation_id id,
       sql_exec_start ,
       number_passes nr_passes,
       round(actual_mem_used/1024/1024) workarea_size_mb,
       max_mem_used ,
       tempseg_size temp_size
FROM v$sql_workarea_active;
```

```
-------------------------------------------------------------------------------------------------------------------------------------
|SQL_ID       |WORKAREA_ADDRESS|POLICY|SID|OPERATION_TYPE|ID|SQL_EXEC_START       |NR_PASSES|WORKAREA_SIZE_MB|MAX_MEM_USED|TEMP_SIZE|
-------------------------------------------------------------------------------------------------------------------------------------
|0pj3uck85dakx|0000000124E12460|AUTO  |175|HASH-JOIN     |1 |20-nov-2018 12:58:57|0        |11              |11213824    |         |
-------------------------------------------------------------------------------------------------------------------------------------
```

The amount of memory is mainly driven by the amount data

```
SELECT t1.object_name, t1.object_type, t2.data_object_id, t2.status
FROM t1
JOIN t2
ON t1.object_name = t2.object_name;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |   164K|    10M|       | 18636   (1)| 00:03:44 |
|*  1 |  HASH JOIN         |      |   164K|    10M|  4304K| 18636   (1)| 00:03:44 |
|   2 |   TABLE ACCESS FULL| T1   |   100K|  3125K|       |  9106   (1)| 00:01:50 |
|   3 |   TABLE ACCESS FULL| T2   |   100K|  3222K|       |  9106   (1)| 00:01:50 |
-------------------------------------------------------------------------------------
```

```
SELECT t1.object_name, t1.object_type, t1.object_id, t2.data_object_id, t2.status
FROM t1
JOIN t2
ON t1.object_name = t2.object_name;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |   164K|    10M|       | 18659   (1)| 00:03:44 |
|*  1 |  HASH JOIN         |      |   164K|    10M|  4400K| 18659   (1)| 00:03:44 |
|   2 |   TABLE ACCESS FULL| T2   |   100K|  3222K|       |  9106   (1)| 00:01:50 |
|   3 |   TABLE ACCESS FULL| T1   |   100K|  3613K|       |  9106   (1)| 00:01:50 |
-------------------------------------------------------------------------------------
```

Select * is bad

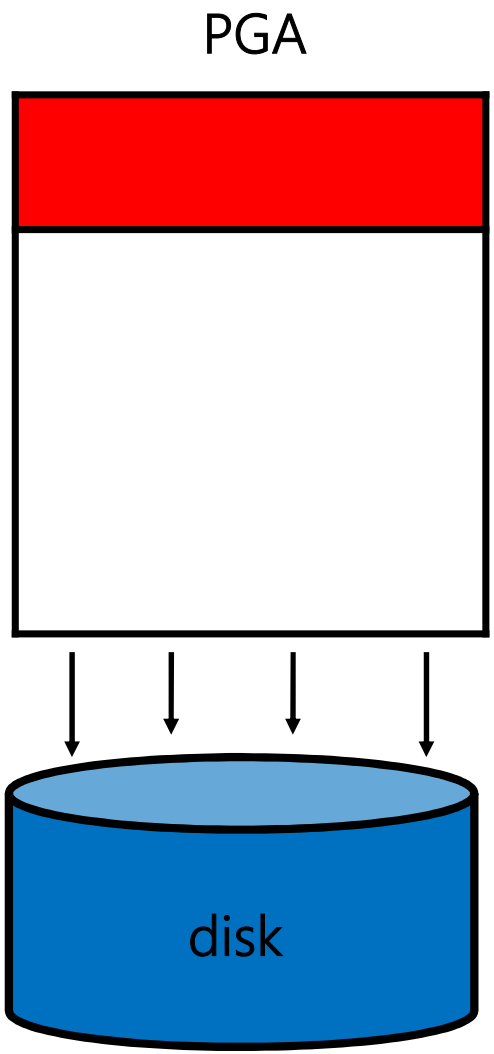
```
SELECT *
FROM t1
JOIN t2
ON t1.object_name = t2.object_name;
```

(join column + additional selected columns from the build table)

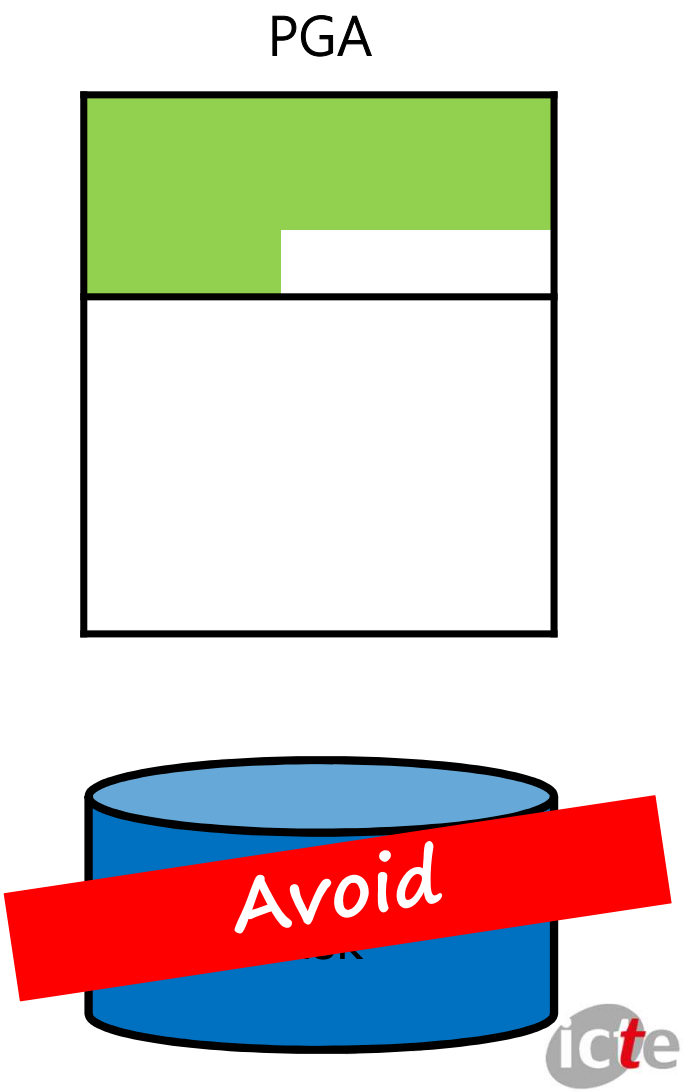
```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |   164K|   657M|       | 38178   (1)| 00:07:39 |
|*  1 |  HASH JOIN          |      |   164K|   657M|   201M| 38178   (1)| 00:07:39 |
|   2 |   TABLE ACCESS FULL| T1   |   100K|   200M|       |  9107   (1)| 00:01:50 |
|   3 |   TABLE ACCESS FULL| T2   |   100K|   200M|       |  9107   (1)| 00:01:50 |
-------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------------------------------------------------------
|SQL_ID       |WORKAREA_ADDRESS|POLICY|SID|OPERATION_TYPE|ID|SQL_EXEC_START      |NR_PASSES|WORKAREA_SIZE_MB|MAX_MEM_USED|TEMP_SIZE|
-----------------------------------------------------------------------------------------------------------------------------------------------
|05psmnxwxfv5j|0000000128C1B6C8|AUTO  |175|HASH-JOIN     |1 |20-nov-2018 16:49:54|1        |84              |85105664    |220200960|
-----------------------------------------------------------------------------------------------------------------------------------------------
```

Hash Join performance key

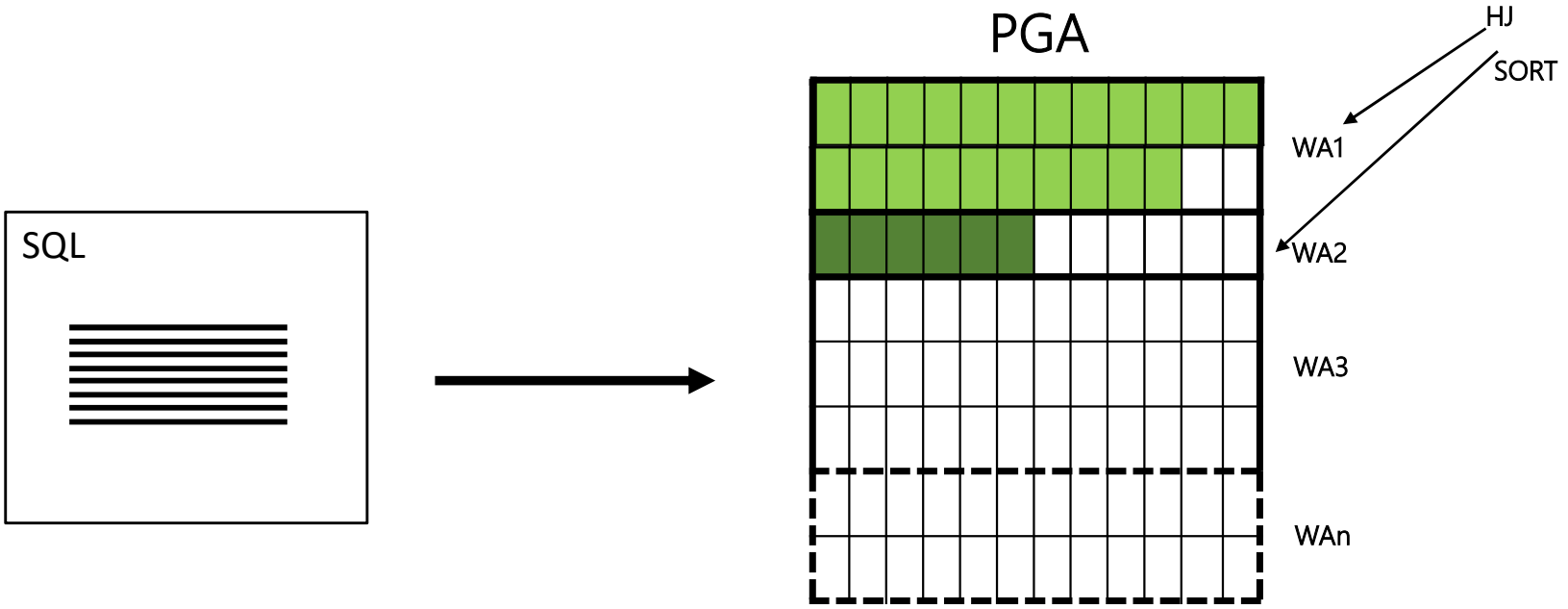


Avoiding the use of temporary space means to get more memory



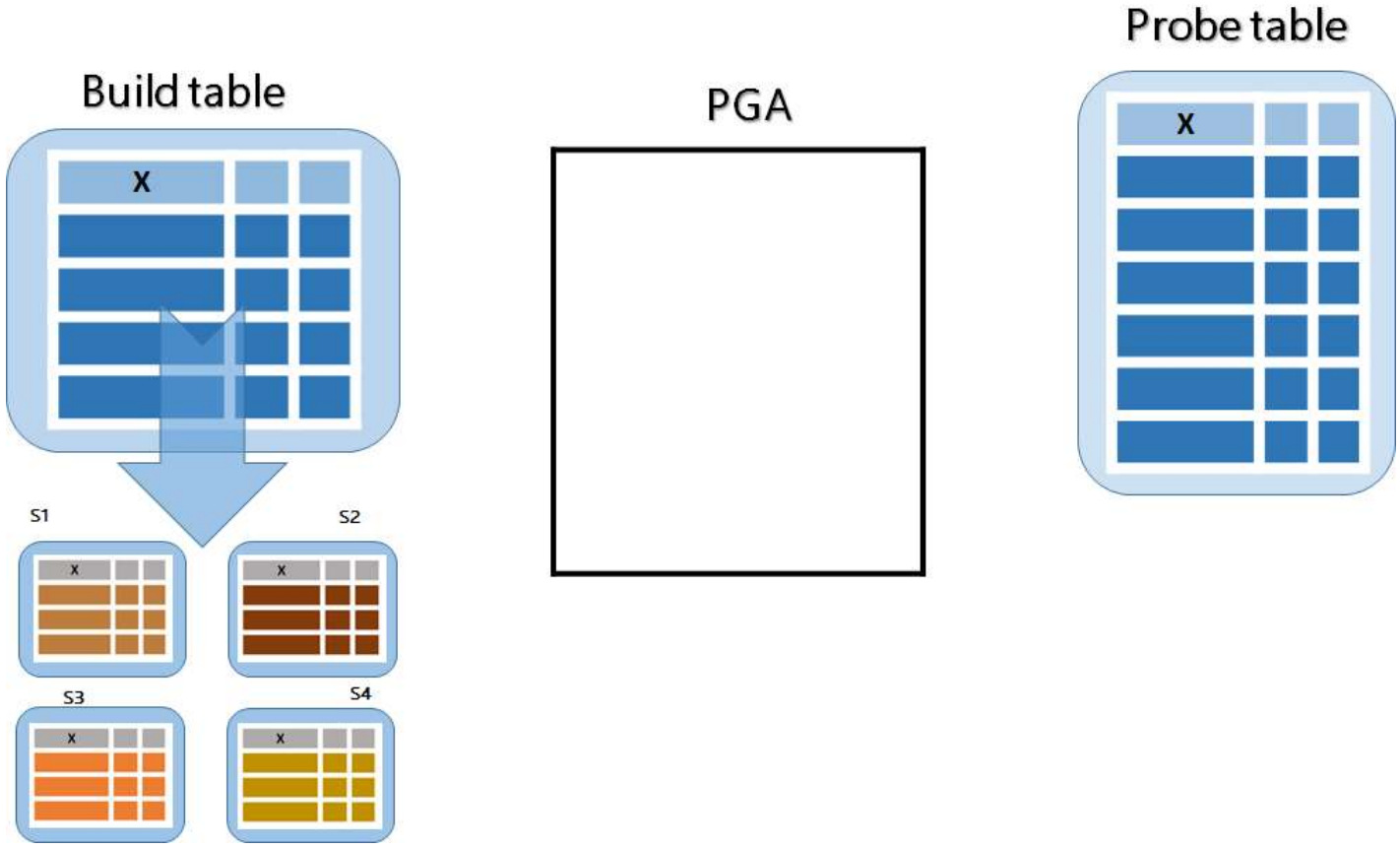
Workarea (OPTIMAL)

When the Inner table fits into memory …



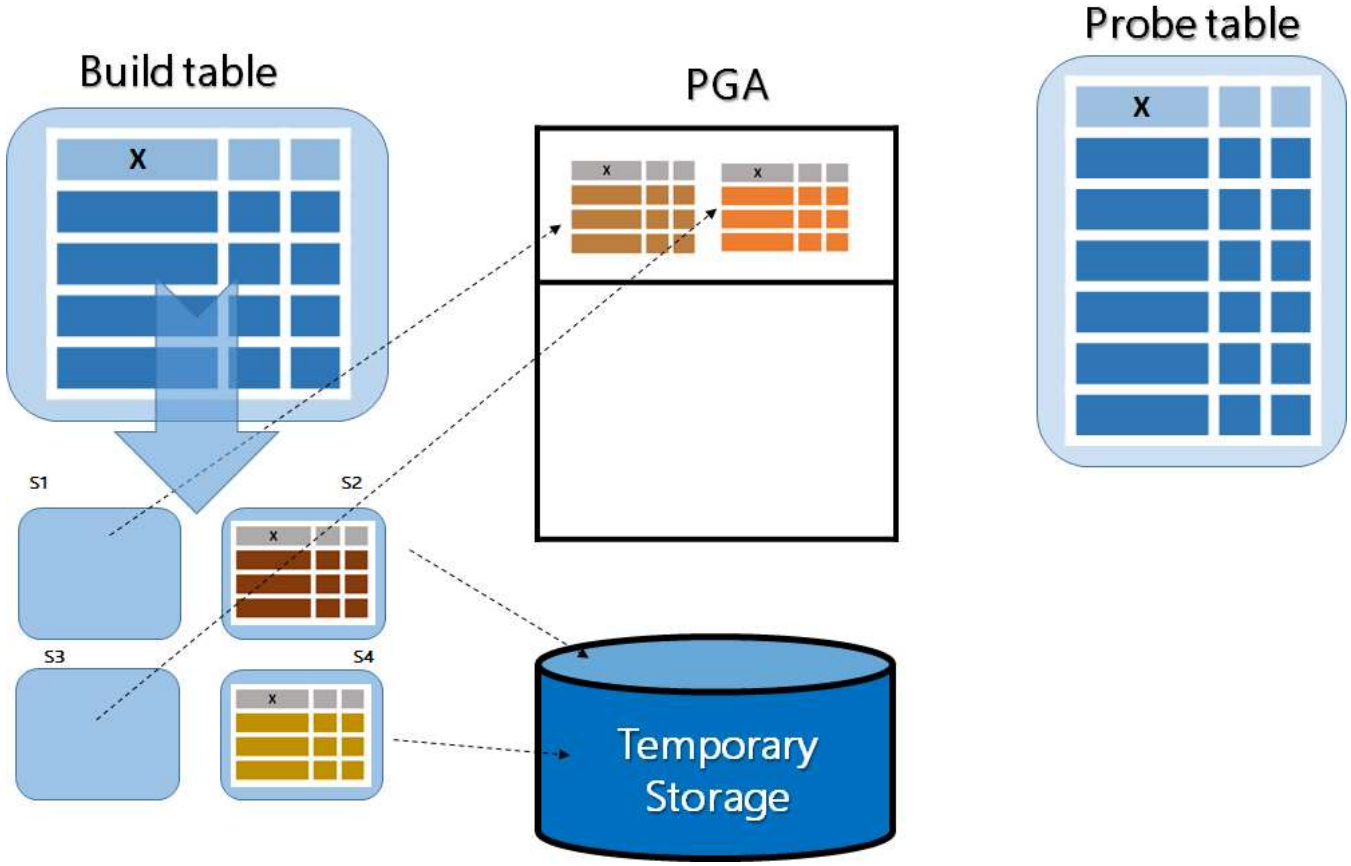
A workarea is allocated for a single operation not for a single session

Inner table NOT fit into memory– One pass



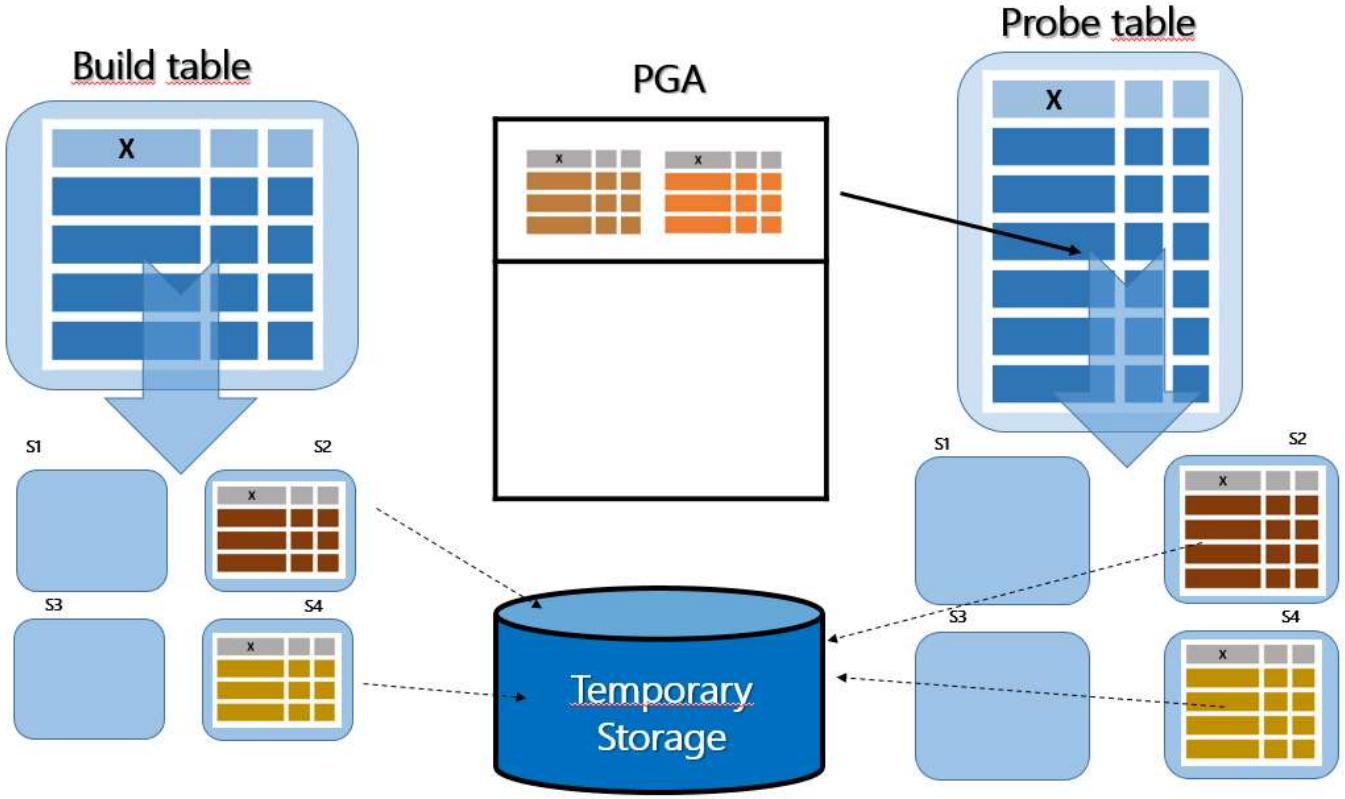
The build table is split into 2^n segments (before being used)

Inner table NOT fit into memory– One pass



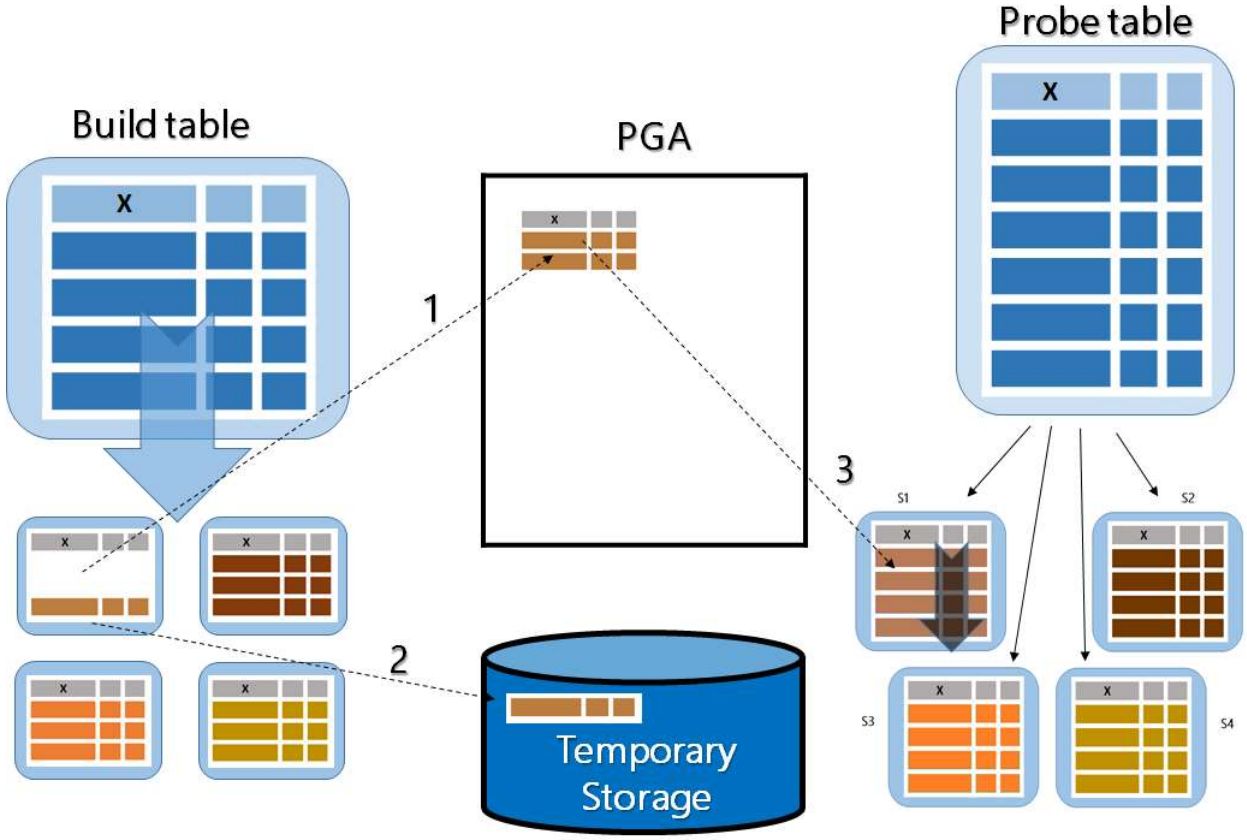
At this time there will be some segments of the build table in memory others segments on Temporary Storage

Inner table NOT fit into memory– One pass



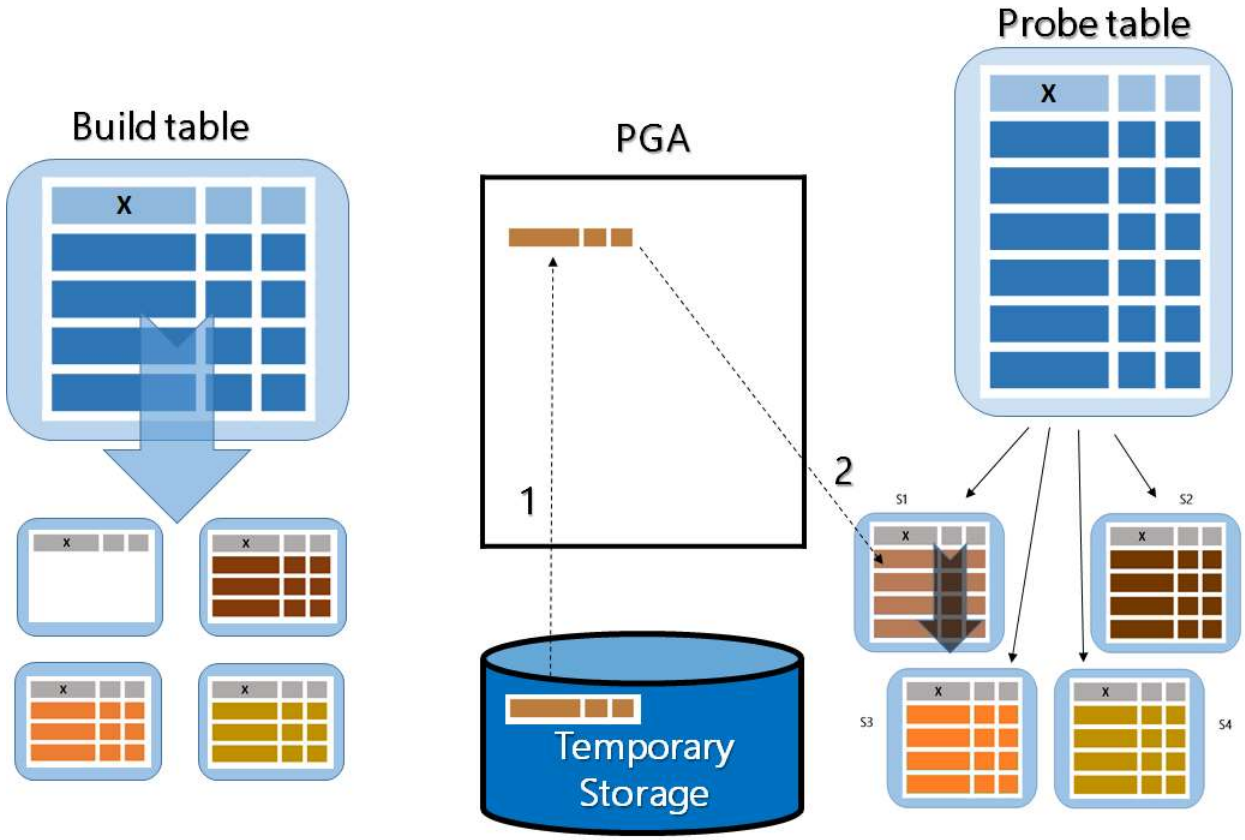
Then the probe table is read and partitioned in the same way as the build table so Oracle will simply check the correspondence between pairs of identical segments hash key (S1 (b) -> S1 (p), S2 (b) -> S2 (p) …)

Inner table NOT fit into memory – Multipass



When a build table segment does not fit completely into memory a part will be loaded into PGA (step nr. 1) the rest will go to disk (step nr. 2). The corresponding partition of the probe table will then be reread several times (multipass)

Inner table NOT fit into memory – Multipass



At this time the piece (cluster or slot) of the build table that is on temporary storage is loaded into memory and the corresponding segment (all the segment) of the probe table is reread for the second time (MULTIPASS)

Monitoring workarea executions

```
SQL> SELECT name, value
  2  FROM v$sysstat
  3  WHERE name IN ('workarea executions - optimal', 'workarea executions - onepass', 'workarea executions - multipass');

NAME                                                                  VALUE
---------------------------------------------------------------- ----------
workarea executions - optimal                                    336077370
workarea executions - onepass                                        18509
workarea executions - multipass                                         54


SQL> SELECT count(*)
  2  FROM t1
  3  JOIN t2
  4  ON t1.c1 = t2.c1;


SQL> SELECT name, value
  2  FROM v$sysstat
  3  WHERE name IN ('workarea executions - optimal', 'workarea executions - onepass', 'workarea executions - multipass');

NAME                                                                  VALUE
---------------------------------------------------------------- ----------
workarea executions - optimal                                    336077370
workarea executions - onepass                                        18510
workarea executions - multipass                                         54
```

Workarea oversizing


```
SQL> alter session set workarea_size_policy = manual;

Modificata sessione.

SQL> alter session set hash_area_siz
alter session set hash_area_size

ERRORE alla riga 1:
ORA-02017: Þ richiesto  intero

SQL> alter session set hash_area_size = 2147483647;

Modificata sessione.
```

Doc ID 453540.1

2 GB-1

Agenda

- ✓ Hash Join operation
- ✓ Memory allocation
- ✓ **Get details with the help of a sql trace**
- ✓ How improve the performance
- ✓ Different Hash Join access type

Get Hash Join details – trace with event 10104

```
alter session set tracefile_identifier = 'HJ_onepass_allrows';

alter session set events '10104 trace name context forever, level 10';

SELECT *
FROM t1
JOIN t2
ON t1.object_name = t2.object_name;

alter session set events '10104 trace name context off';
```

Trace details – Build phase

```
kxhfSetPhase: phase=BUILD
. . .
*** RowSrcId: 1 HASH JOIN STATISTICS (INITIALIZATION) ***
Join Type: INNER join
Original hash-area size: 16477926
Memory for slot table: 12189696
Calculated overhead for partitions and row/slot managers: 4288230
Hash-join fanout: 32
Number of partitions: 32
Number of slots: 48
Multiblock IO: 31
Block size(KB): 8
Cluster (slot) size(KB): 248
Minimum number of bytes per block: 8160
Bit vector memory allocation(KB): 4096
Per partition bit vector length(KB): 128
Maximum possible row length: 2387
Estimated build size (KB): 204
Estimated Build Row Length (includes overhead): 2143
# Immutable Flags:
  Not BUFFER(execution) output of the join for PQ
  Evaluate Left Input Row Vector
  Evaluate Right Input Row Vector
# Mutable Flags:
  IO sync
  Use PGA heap and cursor is pinned
kxhfAddChunk: add chunk 0 (sz=64) to slot table
kxhfAddChunk: chunk 0 (lbs=0x7f5cf3396828, slotTab=0x7f5cf33969f8) successfuly added
kxhfRemoveChunk: remove chunk 0 from slot table
Hash join retain heap migration ends now.
kxhfWrite: hash-join is spilling to disk
```

Nr. Of partition and slot of the Build Table

Hash Join spilling to Temporary Storage

Trace details – Partition statistics

```
*** RowSrcId: 1 HASH JOIN BUILD HASH TABLE (PHASE 1) ***
Total number of partitions: 8
Number of partitions left in memory: 8
Total number of rows in in-memory partitions: 1000
   (used as preliminary number of buckets in hash table)
Estimated max # of build rows that can fit in avail memory: 6944
### Partition Distribution ###
Partition:0    rows:120        clusters:1      slots:1      kept=1
Partition:1    rows:125        clusters:1      slots:1      kept=1
Partition:2    rows:137        clusters:1      slots:1      kept=1
Partition:3    rows:114        clusters:1      slots:1      kept=1
Partition:4    rows:128        clusters:1      slots:1      kept=1
Partition:5    rows:120        clusters:1      slots:1      kept=1
Partition:6    rows:121        clusters:1      slots:1      kept=1
Partition:7    rows:135        clusters:1      slots:1      kept=1
```

I have thousand rows in the build table, all in memory

I know their distribution in memory

Optimal HJ, all partitions have kept=1

Onepass HJ, at least one partition has kept=1

Multipass HJ, all partitions have kept=0

Trace details – Build phase

```
*** RowSrcId: 1 HASH JOIN BUILD HASH TABLE (PHASE 1) ***
Total number of partitions: 32
Number of partitions left in memory: 14
Total number of rows in in-memory partitions: 43511
   (used as preliminary number of buckets in hash table)
Estimated max # of build rows that can fit in avail memory: 100192
### Partition Distribution ###
Partition:0    rows:0          clusters:0      slots:0     kept=0
Partition:1    rows:0          clusters:0      slots:0     kept=0
Partition:2    rows:0          clusters:0      slots:0     kept=0
Partition:3    rows:0          clusters:0      slots:0     kept=0
Partition:4    rows:0          clusters:0      slots:0     kept=0
Partition:5    rows:0          clusters:0      slots:0     kept=0
Partition:6    rows:0          clusters:0      slots:0     kept=0
Partition:7    rows:0          clusters:0      slots:0     kept=0
Partition:8    rows:0          clusters:0      slots:0     kept=0
Partition:9    rows:0          clusters:0      slots:0     kept=0
Partition:10   rows:0          clusters:0      slots:0     kept=0
Partition:11   rows:0          clusters:0      slots:0     kept=0
Partition:12   rows:0          clusters:0      slots:0     kept=0
Partition:13   rows:0          clusters:0      slots:0     kept=0
Partition:14   rows:0          clusters:0      slots:0     kept=0
Partition:15   rows:0          clusters:0      slots:0     kept=0
Partition:16   rows:0          clusters:0      slots:0     kept=0
Partition:17   rows:0          clusters:0      slots:0     kept=0
Partition:18   rows:2986       clusters:26     slots:1     kept=1
Partition:19   rows:3176       clusters:27     slots:1     kept=1
Partition:20   rows:3134       clusters:27     slots:1     kept=1
Partition:21   rows:3069       clusters:26     slots:1     kept=1
Partition:22   rows:3178       clusters:27     slots:1     kept=1
Partition:23   rows:3065       clusters:26     slots:1     kept=1
Partition:24   rows:3214       clusters:27     slots:1     kept=1
Partition:25   rows:3165       clusters:27     slots:1     kept=1
Partition:26   rows:3084       clusters:26     slots:1     kept=1
Partition:27   rows:3029       clusters:26     slots:1     kept=1
Partition:28   rows:3140       clusters:27     slots:1     kept=1
Partition:29   rows:3309       clusters:28     slots:1     kept=1
Partition:30   rows:2940       clusters:25     slots:7     kept=1
Partition:31   rows:3022       clusters:26     slots:10    kept=1
```

The Build Hash Table has been divided into 32 partitions, 14 in memory and 18 to Temporary Storage

Then 18 partitions spilling to Temporary Storage (part nr. 0 .. Part nr.17 with kept=0)

14 partitions left in memory (part nr. 18 .. Part nr. 31 with kept=1)

Trace details – Probe phase

```
kxhfSetPhase: phase=PROBE_1
qerhjFetch: max build row length (mbl=2166)
*** RowSrcId: 1 END OF BUILD (PHASE 1) ***
  Revised row length: 2123
  Revised build size: 207270KB
kxhfResize(enter): resize to 393 slots (numAlloc=48, max=48)
kxhfResize(exit): resized to 393 slots (numAlloc=48, max=393)
  Slot table resized: old=48 wanted=393 got=393 unload=0
kxhfFlush(): pid=0 nRows=3383 build=1 topQ=0
kxhfWrite: Writing dba=730461 slot=23 part=0
kxhfFlush(): pid=1 nRows=3038 build=1 topQ=1
kxhfWrite: Writing dba=935455 slot=47 part=1
kxhfFlush(): pid=2 nRows=2942 build=1 topQ=2
kxhfWrite: Writing dba=760863 slot=42 part=2
kxhfFlush(): pid=3 nRows=2905 build=1 topQ=3
kxhfWrite: Writing dba=730430 slot=9 part=3
kxhfFlush(): pid=4 nRows=3425 build=1 topQ=4
kxhfWrite: Writing dba=790493 slot=27 part=4
kxhfFlush(): pid=5 nRows=3072 build=1 topQ=5
kxhfWrite: Writing dba=760925 slot=39 part=5
kxhfFlush(): pid=6 nRows=3259 build=1 topQ=6
kxhfWrite: Writing dba=957599 slot=0 part=6
kxhfFlush(): pid=7 nRows=3248 build=1 topQ=7
kxhfWrite: Writing dba=812191 slot=41 part=7
kxhfFlush(): pid=8 nRows=3252 build=1 topQ=8
kxhfWrite: Writing dba=812222 slot=30 part=8
kxhfFlush(): pid=9 nRows=3095 build=1 topQ=9
kxhfWrite: Writing dba=790400 slot=43 part=9
kxhfFlush(): pid=10 nRows=3040 build=1 topQ=10
kxhfWrite: Writing dba=730368 slot=19 part=10
kxhfFlush(): pid=11 nRows=2977 build=1 topQ=11
kxhfWrite: Writing dba=790431 slot=16 part=11
kxhfFlush(): pid=12 nRows=3242 build=1 topQ=12
kxhfWrite: Writing dba=812253 slot=45 part=12
kxhfFlush(): pid=13 nRows=3216 build=1 topQ=13
kxhfWrite: Writing dba=790462 slot=32 part=13
kxhfFlush(): pid=14 nRows=3032 build=1 topQ=14
kxhfWrite: Writing dba=730399 slot=11 part=14
kxhfFlush(): pid=15 nRows=3276 build=1 topQ=15
kxhfWrite: Writing dba=935517 slot=4 part=15
kxhfFlush(): pid=16 nRows=3088 build=1 topQ=16
kxhfWrite: Writing dba=775424 slot=38 part=16
kxhfFlush(): pid=17 nRows=2999 build=1 topQ=17
kxhfWrite: Writing dba=552704 slot=6 part=17
```

In the probe_1 phase the Oracle engine will ONLY WRITE the corresponding probe partitions that don't fit in memory during the build phase in the Temporary Storage

Trace details – and if the Build input not fit in memory (onepass)

```
kxhfSetPhase: phase=PROBE_2
...
Free existing hash-table
*** (continued) HASH JOIN BUILD HASH TABLE (PHASE 1) ***
Requested size of hash table: 1024
Actual size of hash table: 1024
Number of buckets: 8192
Match bit vector allocated: FALSE
*** RowSrcId: 1 HASH JOIN GET FLUSHED PARTITIONS (PHASE 2) ***
Getting a pair of flushed partitions.
 BUILD PARTITION: nrows:2942 size=(25 slots, 6200K)
 PROBE PARTITION: nrows:2942 size=(25 slots, 6200K)
kxhfAddChunk: add chunk 0 (sz=64) to slot table
```

In the probe_2 phase the Oracle engine get a pairs of flushed partition on Temporary Storage

2942 rows processed from the build and the probe table

```
*** RowSrcId: 1 HASH JOIN BUILD HASH TABLE (PHASE 2) ***
Number of blocks that may be used to build the hash hable 775
Number of rows left to be iterated over (start of function): 2942
Number of rows iterated over this function call: 2942
Number of rows left to be iterated over (end of function): 0
```

The are no rows to be submitted to iteration

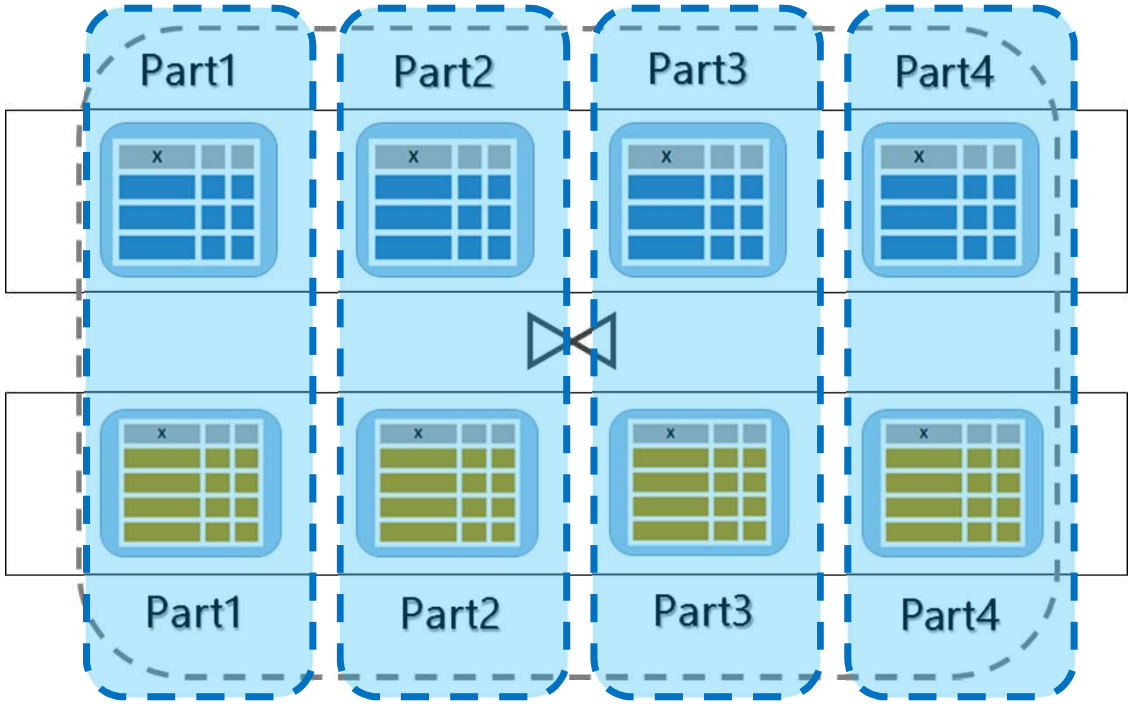
Agenda

- ✓ Hash Join operation
- ✓ Memory allocation
- ✓ Get details with the help of a sql trace
- ✓ **How improve the performance**
- ✓ Different Hash Join access type

How improve HJ performance – Partitioning

How can it help you from a performance point of view?

- **Pruning** (where col = value)
- **PWJ** [full or partial]



If SQL statement running in sequencial mode only one server process joins all rows between two table

PWJ (Partition Wise Join)

Is an optimization technique that allows to split join betweeen large tables into join of identical smaller segment pair

- ✓ < CPU
- ✓ < Memory usage
- ✓ < I/O
- ✓ < resource in general

Full PWJ (Partition Wise Join)

How do I recognize it?

PWJ

```
-----------------------------------------------------
| Id  | Operation           | Name    | Pstart| Pstop |
-----------------------------------------------------
|   0 | SELECT STATEMENT    |         |       |       |
|   1 |  PARTITION LIST ALL |         |     1 |     4 |
|*  2 |   HASH JOIN         |         |       |       |
|   3 |    TABLE ACCESS FULL| T_PART1 |     1 |     4 |
|   4 |    TABLE ACCESS FULL| T_PART2 |     1 |     4 |
-----------------------------------------------------
```

NO - PWJ

```
-----------------------------------------------------
| Id  | Operation           | Name    | Pstart| Pstop |
-----------------------------------------------------
|   0 | SELECT STATEMENT    |         |       |       |
|*  1 |  HASH JOIN          |         |       |       |
|   2 |   PARTITION LIST ALL|         |     1 |     4 |
|   3 |    TABLE ACCESS FULL| T_PART1 |     1 |     4 |
|   4 |   TABLE ACCESS FULL | T       |       |       |
-----------------------------------------------------
```

Conditions for get a Full PWJ

```
-----------------------------------------------------
| Id  | Operation           | Name    | Pstart| Pstop |
-----------------------------------------------------
|   0 | SELECT STATEMENT    |         |       |       |
|   1 |  PARTITION LIST ALL |         |     1 |     4 |
|*  2 |   HASH JOIN         |         |       |       |
|   3 |    TABLE ACCESS FULL| T_PART1 |     1 |     4 |
|   4 |    TABLE ACCESS FULL| T_PART2 |     1 |     4 |
-----------------------------------------------------
```

Full PWJ, *it's possibile with equi-partitioning table. No write/read build and probe table in segment pairs, partitioning has already done this work. So, there are not extra setup cost due to this. The join is performed on (identical) pairs of partitions*

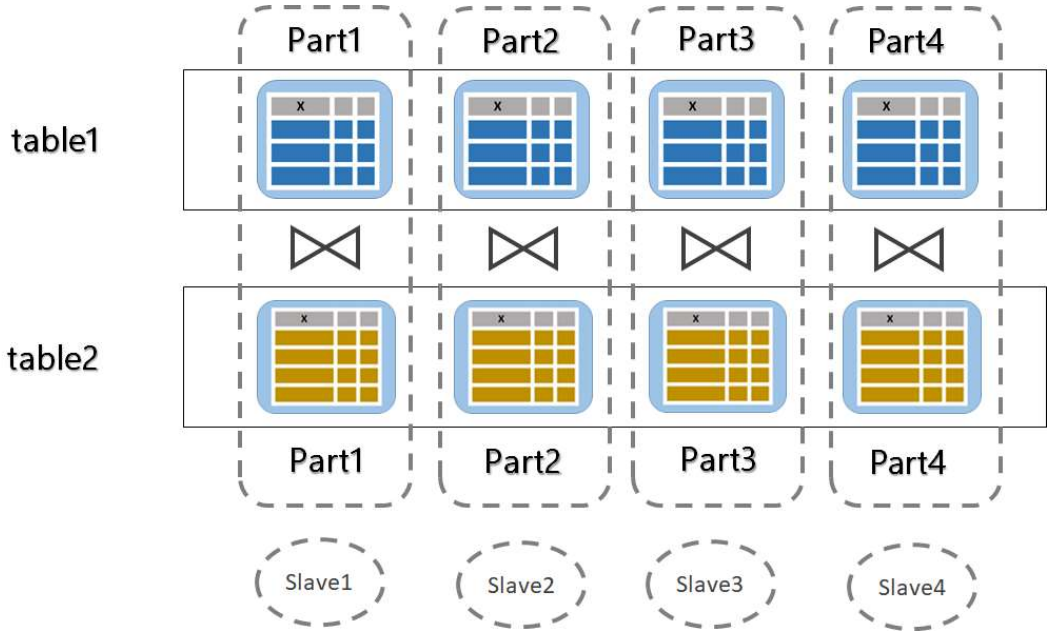
alter table t_part2 add partition p5 values (1000);

```
-------------------------------------------------------------
| Id  | Operation                  | Name    | Pstart| Pstop |
-------------------------------------------------------------
|   0 | SELECT STATEMENT           |         |       |       |
|*  1 |  HASH JOIN                 |         |       |       |
|   2 |   PART JOIN FILTER CREATE  | :BF0000 |       |       |
|   3 |    PARTITION LIST ALL      |         |     1 |     4 |
|   4 |     TABLE ACCESS FULL      | T_PART1 |     1 |     4 |
|   5 |   PARTITION LIST JOIN-FILTER|        |:BF0000|:BF0000|
|   6 |    TABLE ACCESS FULL       | T_PART2 |:BF0000|:BF0000|
-------------------------------------------------------------
```

If the partitioning schema between the two table is different (and other conditions are not met) Full PWJ ***CAN NOT*** *take place*

Full Partition Wise Join (PWJ) and parallelism

DOP = 4

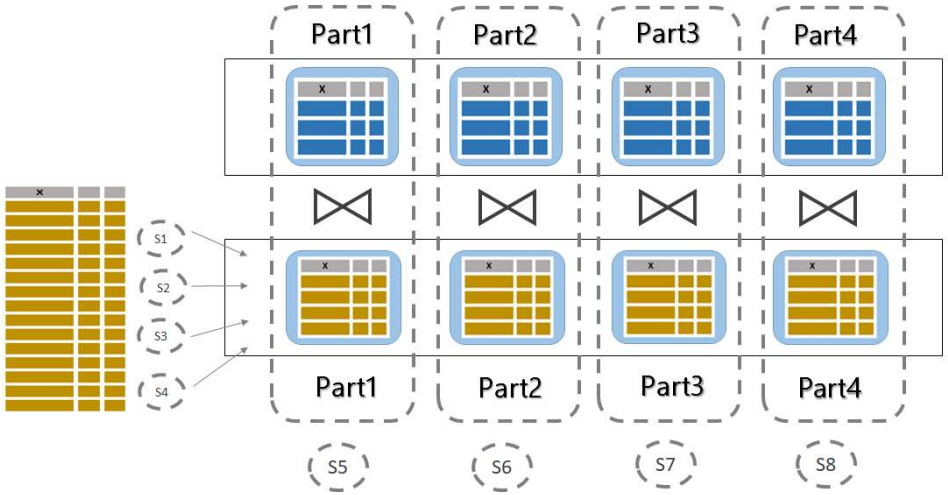



```
-----------------------------------------------------------------
| Id  | Operation                | Name     | Pstart| Pstop |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |       |       |
|   1 |  PX COORDINATOR          |          |       |       |
|   2 |   PX SEND QC (RANDOM)    | :TQ10000 |       |       |
|   3 |    PX PARTITION LIST ALL |          |     1 |     4 |
|*  4 |     HASH JOIN            |          |       |       |
|   5 |      TABLE ACCESS FULL   | T_PART1  |     1 |     4 |
|   6 |      TABLE ACCESS FULL   | T_PART2  |     1 |     4 |
-----------------------------------------------------------------
```

Partial Partition Wise Join and parallelism

When one of the two tables is not partitioned or is partitioned but on a different field from the join key (and SQL statement running in parallel)

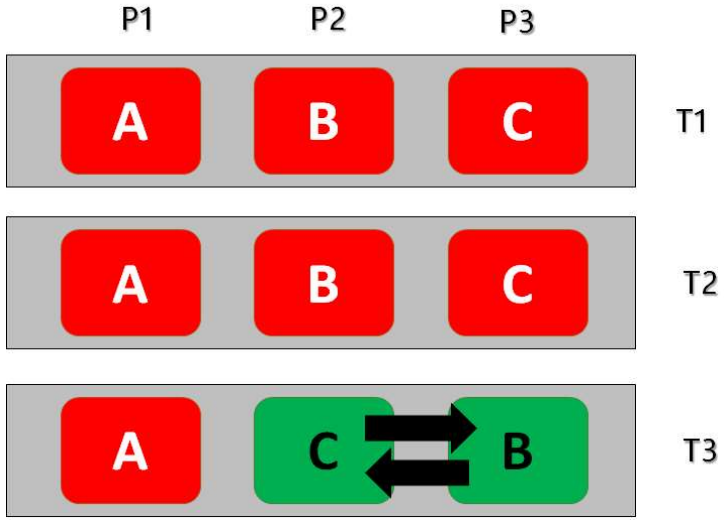
/*+ PARALLEL(4) */




```
-----------------------------------------------------------------------
| Id  | Operation                      | Name     | Pstart| Pstop |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT               |          |       |       |
|   1 |  PX COORDINATOR                |          |       |       |
|   2 |   PX SEND QC (RANDOM)          | :TQ10001 |       |       |
|*  3 |    HASH JOIN BUFFERED          |          |       |       |
|   4 |     PX PARTITION LIST ALL      |          |     1 |     4 |
|   5 |      TABLE ACCESS FULL         | T_PART1  |     1 |     4 |
|   6 |     PX RECEIVE                 |          |       |       |
|   7 |      PX SEND PARTITION (KEY)   | :TQ10000 |       |       |
|   8 |       PX BLOCK ITERATOR        |          |       |       |
|   9 |        TABLE ACCESS FULL       | T_PART2  |       |       |
-----------------------------------------------------------------------
```

Compared to the full PWJ the Oracle engine does an extra work to dynamically partition table2 as table1

Partition Wise Join by LIST pitfall



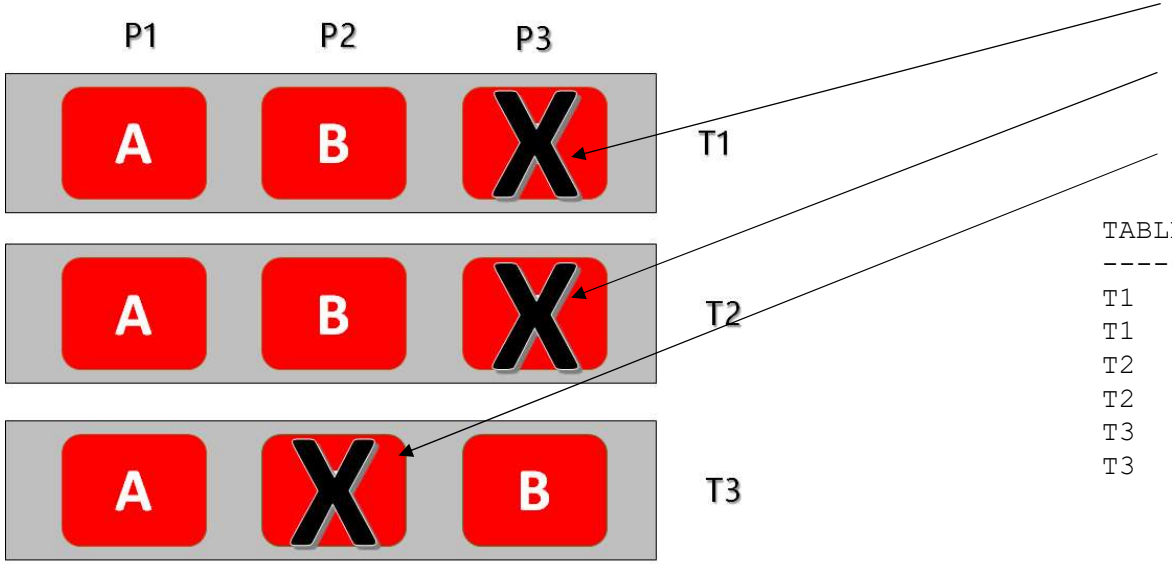
TABLE_NAME	PARTITION_NAME	PARTITION_POSITION
T1	A	1
T1	B	2
T1	C	3
T2	A	1
T2	B	2
T2	C	3
T3	A	1
T3	C	2
T3	B	3

even if the by-list partitioning schema is identical (same number of partitions, same values etc.)
but the partition order position is different, **PWJ can not take place**

Id	Operation	Name	Pstart	Pstop
0	SELECT STATEMENT			
1	PARTITION LIST ALL		1	3
* 2	HASH JOIN			
3	TABLE ACCESS FULL	T1	1	3
4	TABLE ACCESS FULL	T2	1	3

Id	Operation	Name	Pstart	Pstop
0	SELECT STATEMENT			
* 1	HASH JOIN			
2	PART JOIN FILTER CREATE	:BF0000		
3	PARTITION LIST ALL		1	3
4	TABLE ACCESS FULL	T1	1	3
5	PARTITION LIST JOIN-FILTER		:BF0000	:BF0000
6	TABLE ACCESS FULL	T3	:BF0000	:BF0000

Partition Wise Join by LIST pitfall



ALTER TABLE T1 DROP PARTITION C;

ALTER TABLE T2 DROP PARTITION C;

ALTER TABLE T3 DROP PARTITION C;

TABLE_NAME	PARTITION_NAME	PARTITION_POSITION
T1	A	1
T1	B	2
T2	A	1
T2	B	2
T3	A	1
T3	B	2

Id	Operation	Name	Pstart	Pstop
0	SELECT STATEMENT			
1	PARTITION LIST ALL		1	3
* 2	HASH JOIN			
3	TABLE ACCESS FULL	T1	1	3
4	TABLE ACCESS FULL	T2	1	3

Id	Operation	Name	Pstart	Pstop
0	SELECT STATEMENT			
1	PARTITION LIST ALL		1	3
* 2	HASH JOIN			
3	TABLE ACCESS FULL	T1	1	3
4	TABLE ACCESS FULL	T3	1	3

Agenda

- ✓ Hash Join operation
- ✓ Memory allocation
- ✓ Get details with the help of a sql trace
- ✓ How improve the performance
- ✓ **Different Hash Join access type**

Hash Join Access Type

with two table the Oracle don't have other possibility on the tree access

```
-------------------------------------
| Id  | Operation           | Name  |
-------------------------------------
|   0 | SELECT STATEMENT    |       |
|*  1 |  HASH JOIN          |       |
|   2 |   TABLE ACCESS FULL| T1    |
|   3 |   TABLE ACCESS FULL| T2    |
-------------------------------------
```

what happens in a HJ of multiple tables?

Hint

/*+ leading (table alias) */

Specify the order in wich the tables are accessed

/*+ use_hash (table alias) */

Instructs Oracle optimizer to join each table with another row source using hash join

/*+ swap_join_inputs (table alias) */

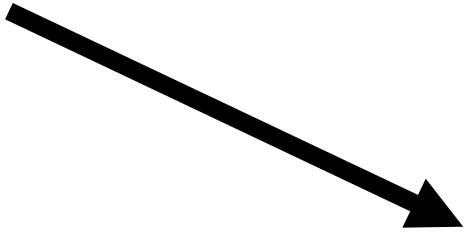
Allow the Oracle optimizer to swap table side when performing hash join (build table)

/*+ no_swap_join_inputs (table alias) */

Not allow the Oracle optimizer to swap table side when performing hash join (probe table)

Hash Join Access Type on multiple rowsource

*this is a **Right-Deep Join Tree***



```
SELECT  *
FROM tab1
JOIN tab2
ON tab1.c1 = tab2.c1
JOIN tab3
ON tab3.c1 = tab2.c1
JOIN tab4
ON tab4.c1 = tab3.c1;
```

```
Plan hash value: 2713846315

-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|*  1 |  HASH JOIN            |      |
|   2 |   TABLE ACCESS FULL   | TAB4 |
|*  3 |   HASH JOIN           |      |
|   4 |    TABLE ACCESS FULL  | TAB3 |
|*  5 |    HASH JOIN          |      |
|   6 |     TABLE ACCESS FULL | TAB2 |
|   7 |     TABLE ACCESS FULL | TAB1 |
-------------------------------------
```

The Oracle optimizer in this case choose this execution plan

Right-Deep Join Trees

```
SELECT  /*+ leading(tab4 tab3 tab2 tab1)
            use_hash(tab1 tab2 tab3)
            swap_join_inputs(tab1)
            swap_join_inputs(tab2)
            swap_join_inputs(tab3)
        */ *
FROM tab1
JOIN tab2
ON tab1.c1 = tab2.c1
JOIN tab3
ON tab3.c1 = tab2.c1
JOIN tab4
ON tab4.c1 = tab3.c1;
```

```
-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|*  1 |  HASH JOIN            |      |
|   2 |   TABLE ACCESS FULL   | TAB1 |
|*  3 |   HASH JOIN           |      |
|   4 |    TABLE ACCESS FULL  | TAB2 |
|*  5 |    HASH JOIN          |      |
|   6 |     TABLE ACCESS FULL| TAB3 |
|   7 |     TABLE ACCESS FULL| TAB4 |
-------------------------------------
```

Right-Deep Join Trees Hint explanation

```
SELECT  /*+ leading(tab4 tab3 tab2 tab1)
            use_hash(tab1 tab2 tab3)
            swap_join_inputs(tab1)
            swap_join_inputs(tab2)
            swap_join_inputs(tab3)
        */ *
FROM tab1
JOIN tab2
ON tab1.c1 = tab2.c1
JOIN tab3
ON tab3.c1 = tab2.c1
JOIN tab4
ON tab4.c1 = tab3.c1;
```

```
-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|*  1 |  HASH JOIN            |      |
|   2 |   TABLE ACCESS FULL   | TAB1 |
|*  3 |   HASH JOIN           |      |
|   4 |    TABLE ACCESS FULL  | TAB2 |
|*  5 |    HASH JOIN          |      |
|   6 |     TABLE ACCESS FULL| TAB3 |
|   7 |     TABLE ACCESS FULL| TAB4 |
-------------------------------------
```

swap_join_inputs(tab3)

tab3,tab4

swap_join_inputs(tab2)

tab2, (tab3, tab4)

swap_join_inputs(tab1)

(tab1, (tab2, (tab3, tab4)))

Right-Deep Join Trees workarea

```
-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|*  1 |  HASH JOIN            |      |
|   2 |   TABLE ACCESS FULL   | TAB1 |
|*  3 |   HASH JOIN           |      |
|   4 |    TABLE ACCESS FULL  | TAB2 |
|*  5 |    HASH JOIN          |      |
|   6 |     TABLE ACCESS FULL| TAB3 |
|   7 |     TABLE ACCESS FULL| TAB4 |
-------------------------------------
```

n = 3

How many workareas are allocated at the same time?

n = nr. join

Hash Join Access Type on multiple rowsource

Right-Deep Join Tree

is the only chance to solve a multi-table Hash Join?

Left-Deep Join Trees

```
SELECT  /*+ leading(tab1 tab2 tab3 tab4)
            use_hash(tab2 tab3 tab4)
            no_swap_join_inputs(tab2)
            no_swap_join_inputs(tab3)
            no_swap_join_inputs(tab4)
        */ *
FROM tab1
JOIN tab2
ON tab1.c1 = tab2.c1
JOIN tab3
ON tab3.c1 = tab2.c1
JOIN tab4
ON tab4.c1 = tab3.c1;
```

```
-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|*  1 |  HASH JOIN            |      |
|*  2 |   HASH JOIN           |      |
|*  3 |    HASH JOIN          |      |
|   4 |     TABLE ACCESS FULL| TAB1 |
|   5 |     TABLE ACCESS FULL| TAB2 |
|   6 |    TABLE ACCESS FULL | TAB3 |
|   7 |   TABLE ACCESS FULL  | TAB4 |
-------------------------------------
```

Left-Deep Join Trees Hint explanation

```
SELECT  /*+ leading(tab1 tab2 tab3 tab4)
            use_hash(tab2 tab3 tab4)
            no_swap_join_inputs(tab2)
            no_swap_join_inputs(tab3)
            no_swap_join_inputs(tab4)
        */ *
FROM tab1
JOIN tab2
ON tab1.c1 = tab2.c1
JOIN tab3
ON tab3.c1 = tab2.c1
JOIN tab4
ON tab4.c1 = tab3.c1;
```

```
-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|*  1 |  HASH JOIN            |      |
|*  2 |   HASH JOIN           |      |
|*  3 |    HASH JOIN          |      |
|   4 |     TABLE ACCESS FULL| TAB1 |
|   5 |     TABLE ACCESS FULL| TAB2 |
|   6 |    TABLE ACCESS FULL | TAB3 |
|   7 |   TABLE ACCESS FULL  | TAB4 |
-------------------------------------
```

no_swap_join_inputs(tab2)

tab1,tab2

no_swap_join_inputs(tab3)

(tab1, tab2), tab3

no_swap_join_inputs(tab4)

((tab1, tab2), tab3), tab4

Left-Deep Join Trees workarea

```
-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|*  1 |  HASH JOIN            |      |
|*  2 |   HASH JOIN           |      |
|*  3 |    HASH JOIN          |      |
|   4 |     TABLE ACCESS FULL| TAB1 |
|   5 |     TABLE ACCESS FULL| TAB2 |
|   6 |    TABLE ACCESS FULL | TAB3 |
|   7 |   TABLE ACCESS FULL  | TAB4 |
-------------------------------------
```

How many workareas are allocated at the same time?

no more than two workeareas

Zig-Zag Join Trees

```
select /*+ leading(tab2 tab3 tab1 tab4)
           use_hash(tab1 tab2 tab3 tab4)
           swap_join_inputs(tab1)
           no_swap_join_inputs(tab4)
       */ *
from tab1
join tab2
 on tab1.c1 = tab2.c1
join tab3
on tab3.c1 = tab2.c1
join tab4
on tab4.c1 = tab3.c1;
```

```
-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|*  1 |  HASH JOIN            |      |
|*  2 |   HASH JOIN           |      |
|   3 |    TABLE ACCESS FULL  | TAB1 |
|*  4 |    HASH JOIN          |      |
|   5 |     TABLE ACCESS FULL | TAB2 |
|   6 |     TABLE ACCESS FULL | TAB3 |
|   7 |   TABLE ACCESS FULL   | TAB4 |
-------------------------------------
```

Zig-zag Join Trees Hint explanation

```
SELECT  /*+ leading(tab2 tab3 tab1 tab4)
            use_hash(tab1 tab2 tab3 tab4)
            swap_join_inputs(tab1)
            no_swap_join_inputs(tab4)
        */ *
FROM tab1
JOIN tab2
ON tab1.c1 = tab2.c1
JOIN tab3
ON tab3.c1 = tab2.c1
JOIN tab4
ON tab4.c1 = tab3.c1;
```

```
-------------------------------------
| Id  | Operation            | Name |
-------------------------------------
|   0 | SELECT STATEMENT     |      |
|*  1 |  HASH JOIN           |      |
|*  2 |   HASH JOIN          |      |
|   3 |    TABLE ACCESS FULL | TAB1 |
|*  4 |    HASH JOIN         |      |
|   5 |     TABLE ACCESS FULL| TAB2 |
|   6 |     TABLE ACCESS FULL| TAB3 |
|   7 |   TABLE ACCESS FULL  | TAB4 |
-------------------------------------
```

(tab2, tab3), tab1

swap_join_inputs(tab1)

tab1, (tab2, tab3)

no_swap_join_inputs(tab4)

(tab1, (tab2, tab3)), tab4

Zig-zag Join Trees workarea

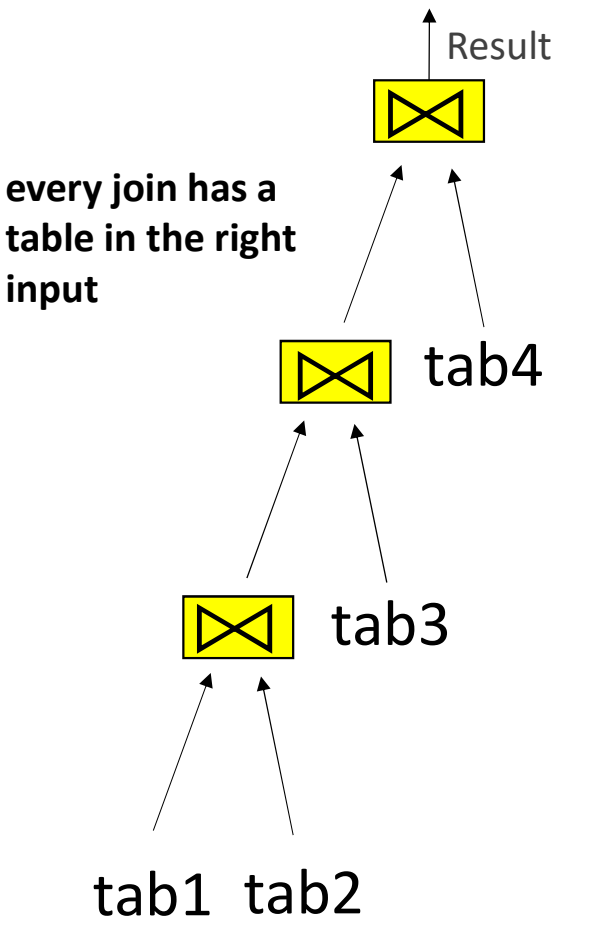
```
-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|*  1 |  HASH JOIN            |      |
|*  2 |   HASH JOIN           |      |
|   3 |    TABLE ACCESS FULL  | TAB1 |
|*  4 |    HASH JOIN          |      |
|   5 |     TABLE ACCESS FULL| TAB2 |
|   6 |     TABLE ACCESS FULL| TAB3 |
|   7 |   TABLE ACCESS FULL   | TAB4 |
-------------------------------------
```

How many workareas are allocated at the same time?

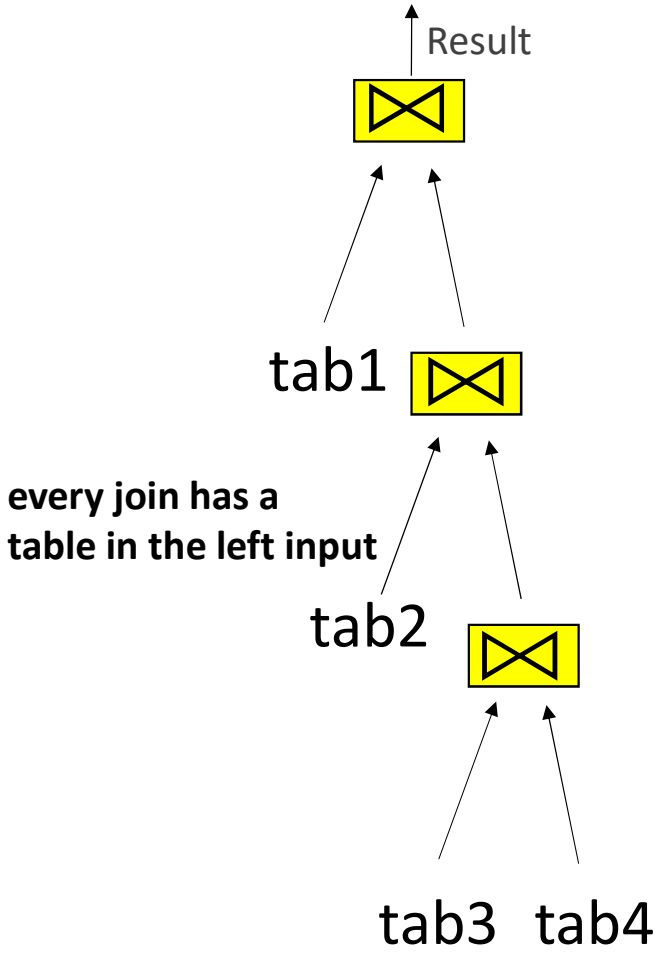
n = max when nr. join

Trees difference

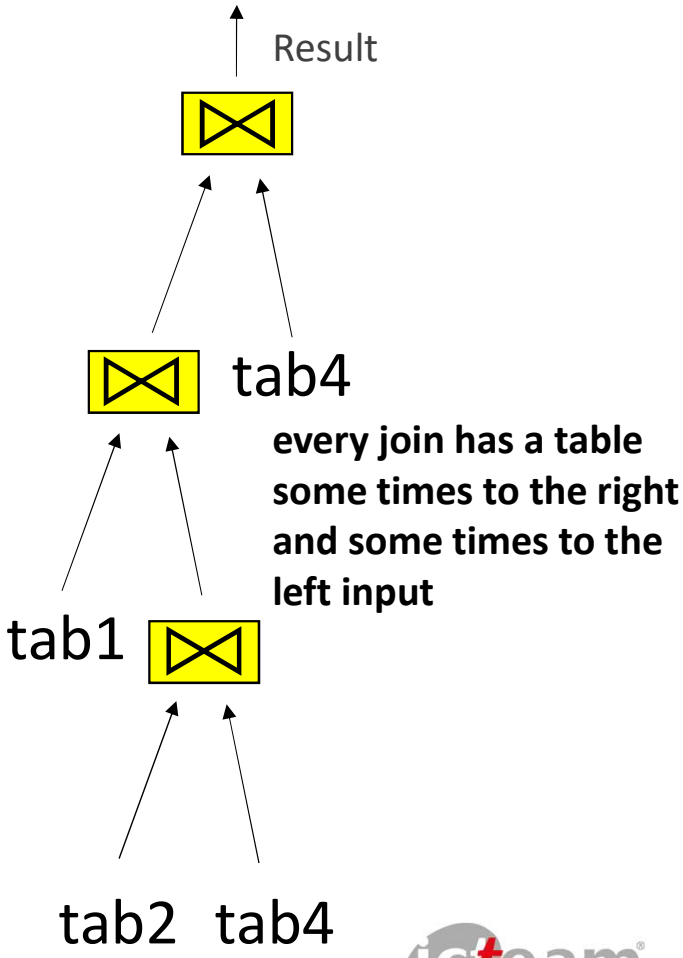
Left-deep join trees



Right-deep join trees



Zig-zag join trees



Memory optimization

Right-deep join trees

```
-------------------------------------
| Id  | Operation            | Name |
-------------------------------------
|   0 | SELECT STATEMENT     |      |
|*  1 |  HASH JOIN WA        |      |
|   2 |   TABLE ACCESS FULL  | TAB1 |
|*  3 |   HASH JOIN WB       |      |
|   4 |    TABLE ACCESS FULL | TAB2 |
|*  5 |    HASH JOIN WC      |      |
|   6 |     TABLE ACCESS FULL| TAB3 |
|   7 |     TABLE ACCESS FULL| TAB4 |
-------------------------------------
```

n (n=nr of join) workeareas allocated at the same time

Left-deep join trees

```
-------------------------------------
| Id  | Operation            | Name |
-------------------------------------
|   0 | SELECT STATEMENT     |      |
|*  1 |  HASH JOIN WA        |      |
|*  2 |   HASH JOIN WB       |      |
|*  3 |    HASH JOIN WA      |      |
|   4 |     TABLE ACCESS FULL| TAB1 |
|   5 |     TABLE ACCESS FULL| TAB2 |
|   6 |    TABLE ACCESS FULL | TAB3 |
|   7 |   TABLE ACCESS FULL  | TAB4 |
-------------------------------------
```

no more than two workeareas allocated at the same time

which is more wasteful in memory consumption?

Which is more wasteful in memory consumption?

Right-deep join trees

```
-------------------------------------
| Id  | Operation            | Name |
-------------------------------------
|   0 | SELECT STATEMENT     |      |
|*  1 |  HASH JOIN           |      |
|   2 |   TABLE ACCESS FULL  | DIM1 |
|*  3 |   HASH JOIN          |      |
|   4 |    TABLE ACCESS FULL | DIM2 |
|*  5 |    HASH JOIN         |      |
|   6 |     TABLE ACCESS FULL| DIM3 |
|   7 |     TABLE ACCESS FULL| FACT |
-------------------------------------
```

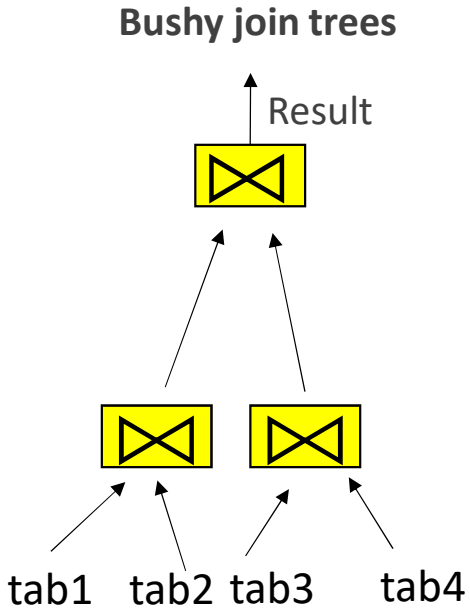
the fact table can be pushed as probe table and does not affect the memory consumption of workareas

Left-deep join trees

```
-------------------------------------
| Id  | Operation            | Name |
-------------------------------------
|   0 | SELECT STATEMENT     |      |
|*  1 |  HASH JOIN           |      |
|*  2 |   HASH JOIN          |      |
|*  3 |    HASH JOIN         |      |
|   4 |     TABLE ACCESS FULL| DIM1 |
|   5 |     TABLE ACCESS FULL| FACT |
|   6 |    TABLE ACCESS FULL | DIM2 |
|   7 |   TABLE ACCESS FULL  | DIM3 |
-------------------------------------
```

the size of the fact table influences the memory consumption of all workarea (unless you want to make a Cartesian product the dimension table)

Bushy Join Trees




```
-------------------------------------
| Id  | Operation             | Name |
-------------------------------------
|   0 | SELECT STATEMENT      |      |
|   1 |  HASH JOIN            |      |
|   2 |   VIEW                |      |
|   3 |    HASH JOIN          |      |
|   4 |     TABLE ACCESS FULL| TAB1 |
|   5 |     TABLE ACCESS FULL| TAB2 |
|   6 |   VIEW                |      |
|   7 |    HASH JOIN          |      |
|   8 |     TABLE ACCESS FULL| TAB3 |
|   9 |     TABLE ACCESS FULL| TAB4 |
-------------------------------------
```

If the childs of a join tree is a join node

Bushy Join Oracle 12c (12.2)

```
SELECT /*+ leading(block1 block2) */   *
FROM (SELECT /*+ no_merge
                 leading(tab1 tab2)
                 swap_join_inputs(tab1) */
             tab1.*
      FROM tab1
      JOIN tab2
      ON tab1.c1 = tab2.c1 ) block1
JOIN (SELECT /*+ no_merge
                 leading(tab3 tab4)
                 swap_join_inputs(tab3) */
             tab3.*
      FROM tab3
      JOIN tab4
      ON tab4.c1 = tab3.c1) block2
ON block1.c1 = block2.c1;
```

```
SELECT /*+ BUSHY_JOIN((TAB1 TAB2) (TAB3 TAB4)) */ *
FROM tab1
JOIN tab2
ON tab1.c1 = tab2.c1
JOIN tab3
ON tab3.c1 = tab1.c1
JOIN tab4
ON tab4.c1 = tab3.c1;
```

```
-------------------------------------
| Id  | Operation            | Name |
-------------------------------------
|   0 | SELECT STATEMENT     |      |
|   1 |  HASH JOIN           |      |
|   2 |   VIEW               |      |
|   3 |    HASH JOIN         |      |
|   4 |     TABLE ACCESS FULL| TAB1 |
|   5 |     TABLE ACCESS FULL| TAB2 |
|   6 |   VIEW               |      |
|   7 |    HASH JOIN         |      |
|   8 |     TABLE ACCESS FULL| TAB3 |
|   9 |     TABLE ACCESS FULL| TAB4 |
-------------------------------------
```

```
-----------------------------------------------
| Id  | Operation            | Name           |
-----------------------------------------------
|   0 | SELECT STATEMENT     |                |
|*  1 |  HASH JOIN           |                |
|   2 |   VIEW               | VW_BUSHY_3A2A9881 |
|*  3 |    HASH JOIN         |                |
|   4 |     TABLE ACCESS FULL| TAB1           |
|   5 |     TABLE ACCESS FULL| TAB2           |
|   6 |   VIEW               | VW_BUSHY_B039EE45 |
|*  7 |    HASH JOIN         |                |
|   8 |     TABLE ACCESS FULL| TAB3           |
|   9 |     TABLE ACCESS FULL| TAB4           |
-----------------------------------------------
```

Headquarters Via Azzano San Paolo , 139
24050 Grassobbio (BG)

Ufficio di Roma: Via Mar della Cina, 304 |
00144 Roma

Headquarters Lutech Via Milano, 150
20093 Cologno M.se (MI)

confidentiality notice

Le informazioni, i dati e le immagini contenuti in questo documento sono strettamente confidenziali e riservati, di esclusiva proprietà di Lutech. Sono disponibili esclusivamente per persone o società a cui è stato direttamente consegnato il documento e non sono divulgabili a terzi senza il consenso scritto dell'Ufficio Comunicazione di Lutech, che può essere contattato all'indirizzo comunicazione@lutech.it. I loghi di terze parti (es. partner, clienti, ecc.) sono da considerarsi indicativi.