Oracle In-Memory: basics and beyond

Priit Piipuu

18.11.2025

Whoami: Priit Piipuu

Database performance engineer at FDJ United

Oracle Ace Pro

Member of Symposium 42

Blog: https://priitp.wordpress.com,

@ppiipuu.bsky.social

What it's all about

A gentle introduction to the in-memory analytics

Oracle In-Memory basics

What is under the hood?

A quick rundown of more interesting features of Oracle In-Memory

Presentation and scripts available in Github

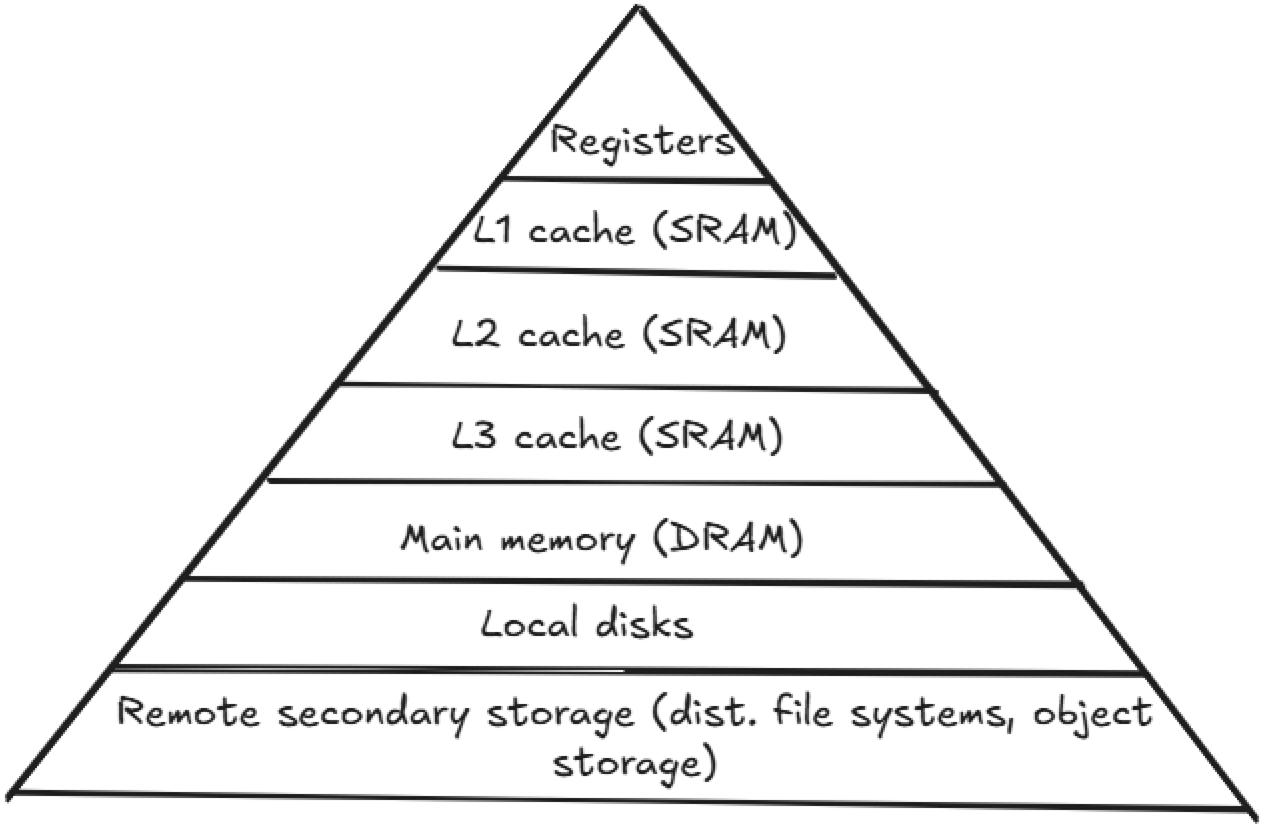
Safe harbour statement

This presentation describes things as they are in Oracle 26ai

It is tested on Oracle 19c and 21c

You should always read the latest version of In-Memory docs, even if you're on older version

What makes computation fast?


Registers
L1 cache (SRAM)
L2 cache (SRAM)
L3 cache (SRAM)
Main memory (DRAM)
Local disks
Remote secondary storage (dist. file systems, object storage)

RAM is the new disk – and how to measure its performance – Part 1 – Introduction

Tanel Poder

2015-08-09

[part 1 | part 2 | part 3]

RAM is the new disk, at least in the In-Memory computing world.

No, I am not talking about Flash here, but Random Access Memory – RAM as in SDRAM. I'm by far not the first one to say it. Jim Gray wrote this in 2006: *"Tape is dead, disk is tape, flash is disk, RAM locality is king"* (presentation).

Also, I'm not going to talk about how RAM is faster than disk (everybody knows that), but in fact how RAM is the *slow* component of an in-memory processing engine.

I will use Oracle's In-Memory column store and the *hardware performance counters* in modern CPUs for drilling down into the low-level hardware performance metrics about CPU efficiency and memory access.

But let's first get started by looking a few years into past into the old-school disk IO and index based SQL

Data-oriented design

Operates on arrays, this avoids call overhead and cache misses

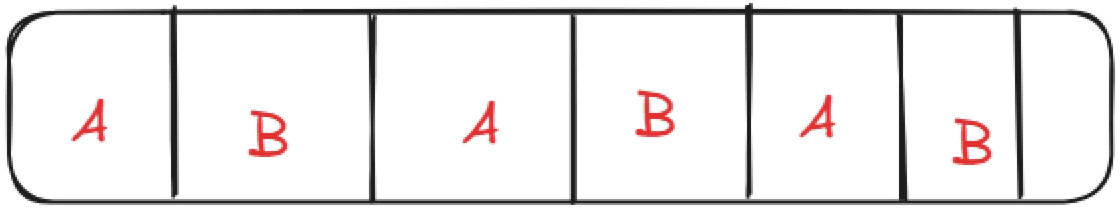
Prefers arrays to structures, gives better cache usage

Inlines subroutines, avoids deep call hierarchies

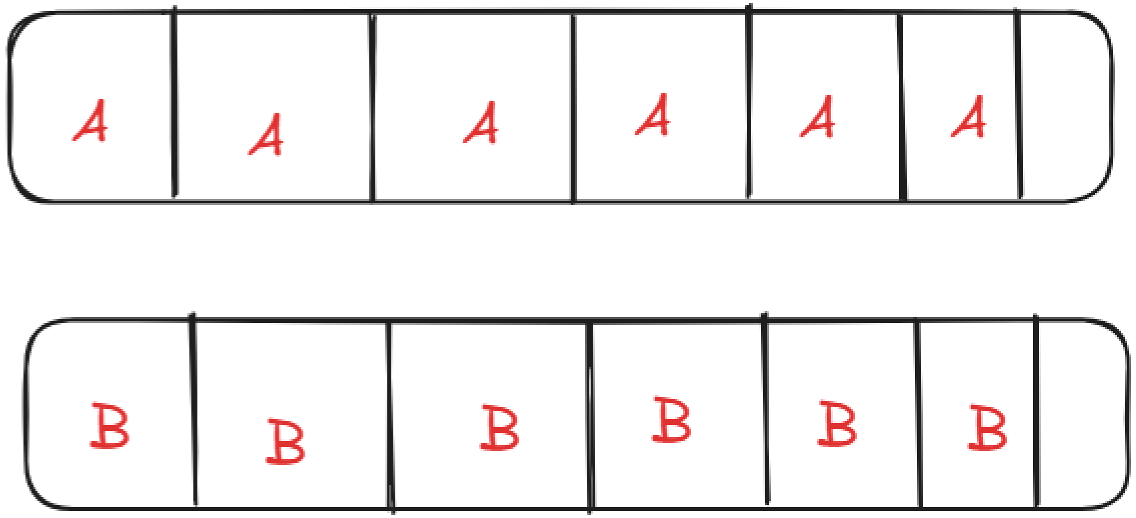
Tight control on memory allocation

Simplified example

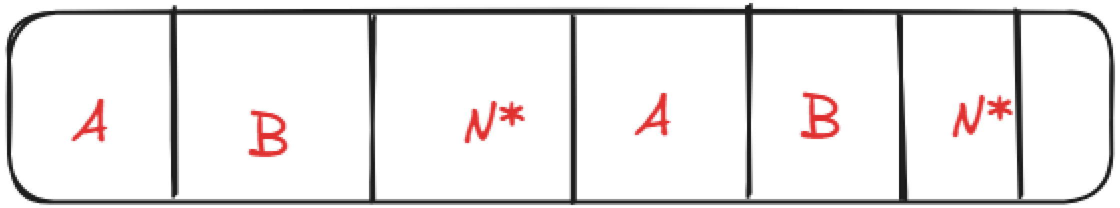
```go
var size uint64 = 100_000

type Foo struct {
        a uint64
        b uint64
}

foos := make([]Foo, size)
```


A
B
A
B
A
B

```go
type Bar struct {
        a []uint64
        b []uint64
}

 bars := Bar{}
 bars.a = make([]uint64, size)
 bars.b = make([]uint64, size)
```


A
A
A
A
A
A
B
B
B
B
B
B

```go
type Baz struct {
        a    uint64
        b    uint64
        next *Baz
}

bz := make([]Baz, size)
for i = 0; i < size; i++ {
   bz[i] = Baz{i, i, nil}
   if i < size - 1 {
      bz[i].next = &bz[i+1]
   }
}
```


A
B
N*
A
B
N*

```
[pripii@pripii-roadkill loctest]$ go run .
Foos:    10000      104366 ns/op
Bar:    21056        57861 ns/op
Baz:     5560       213953 ns/op
```

In case of Oracle In-Memory...

Event	Full table scan	In-Memory scan	Difference
branch-instructions	9700928898	716348559	13x
branch-misses	123790427	30305893	4x
cache-misses	33121710	4641178	7x
cycles	31763298910	3322369500	9.6x
instructions	65545341739	4178971681	15x

> Source: https://priitp.wordpress.com/2024/10/16/does-oracle-in-memory-use-simd-instructions-joelkallmanday/

Apache Arrow (I)

Framework and data interchange format

Language agnostic in-memory data structure specification

Metadata serialisation

Protocol for serialisation and generic data transport

Arrow columnar format

Data adjacency for sequential access

Constant time, zero copy random access

Plays well with SIMD and vectorization

Popular as data exchange protocol

- Even python-oracledb has support for the data frames

Implementations

C/C++

Pyarrow (wrapper around the C/C++ library)

Pola.rs (written in Rust, adds SQL support and other cool features)

Dremio: query engine and data warehouse built around Arrow.

pyarrow example

```
#!/usr/bin/env python3.12

from pyarrow import csv,string,decimal128

FNAME = '/u01/oracle/1brc/measurements.txt'
tbl = csv.read_csv(FNAME, csv.ReadOptions(column_names = ['city', 'temp']),
                    csv.ParseOptions(delimiter=';'),
                    csv.ConvertOptions(column_types={'city':string(),
                        'temp':decimal128(3, 2)}))

out = (tbl.group_by('city')
        .aggregate([('temp', 'max'), ('temp', 'min'), ('temp', 'mean')])
        .sort_by('city'))
```

Same query in Oracle SQL

```sql
SELECT/*+ parallel */
    city,
    MIN(temp),
    AVG(temp),
    MAX(temp)
FROM
    brc_ext
GROUP BY
    city
ORDER BY
    city;
```

Oracle In-Memory and competitors

	Arrow implementations	Oracle In-Memory
Data types	Has own type system	Subset of SQL types
Access	Constant time random access	Through SQL queries
Transactional	No	Yes
Compute functions	Yes	In-Memory expr
Automatic parallelization	Yes	Yes
Automatic memory management	No/DYI	Yes
Automatic In-Memory	No	Yes

Oracle In-Memory

Oracle In-Memory (I)

In-Memory column store (part of the SGA)

Query optimizations

Availability and automation

Integration with the Oracle features

Oracle In-Memory (II)

On CDB/instance level:

```
alter system set inmemory_size = 16G scope = spfile
```

On PDB level:

```
alter system set inmemory_size = 8G scope = spfile;
```

Automatic In-Memory sizing in 23ai

Can automatically shrink or grow In-Memory area.

`inmemory_size` becomes minimum size for In-Memory

`INMEMORY_LEVEL` should be set to `MEDIUM` or `HIGH`

ASMM manages In-Memory Area with other SGA components

Oracle In-memory Base Level

Oracle EE feature

IM column store size less than 16GB per CDB or instance

Compression level is set to `QUERY LOW`

No Automatic In-Memory

```
alter system set inmemory_force = base_level scope = spfile;
```

Populating the column store

During the population

- Database reads row format data from the disk
- Transforms into columnar format
- Stores it in the IM column store

Repopulation

Transforms *new* data into columnar format

Creates new IMCUs

`INMEMORY` attribute can be specified for

Tablespaces

Tables

Matrialized views

Set of columns

`INMEMORY` attribute

Objects that can't be populated:

- Indexes
- Index-oriented tables
- Hash clusters
- Objects owned by SYS
- Objects in `SYSTEM` or `SYSAUX` tablespaces

Ineligible data types

Data types that can't be populated:

- Out-of-line columns like varrays, nested table columns
- `LONG` or `LONG RAW` data types
- Extended data types

Partitioned tables

In-Memory can be specified either on table level or partition level

Partitions inherit table-level clause

Works with hybrid partitioning, but results may vary

External tables and external partitions

Some limitations:

- No subpartitions
- Column, distribute and priority clauses are not valid
- No join groups, In-Memory Optimized Arithmetic, In-Memory Expressions

In-Memory and LOBs

Out-of-line LOBs can't be populated, IM column store saves only the locator

Inline LOBS:

- IM column store allocates 4KB of continuous buffer space
- For OSON data upper limit is 32KB

In-Memory and queries

All referenced columns should be available in in-memory area

In Oracle 21c: columns can be read from disc during projection

```sql
alter table transactions_nopart inmemory
    memcompress for query high
    priority high
    distribute by partition
        for service inm_service
    duplicate all
;
```

```sql
alter table transactions_nopart inmemory
    memcompress for query high
    priority high
    distribute by partition
        for service inm_service
    duplicate all
;
```

Compress method for the table
Either DML
QUERY HIGH/LOW
CAPACITY HiGH/LOW

```
alter table transactions_nopart inmemory
    memcompress for query high
    priority high
    distribute by partition
        for service inm_service
    duplicate all
;
```

Priority of the population

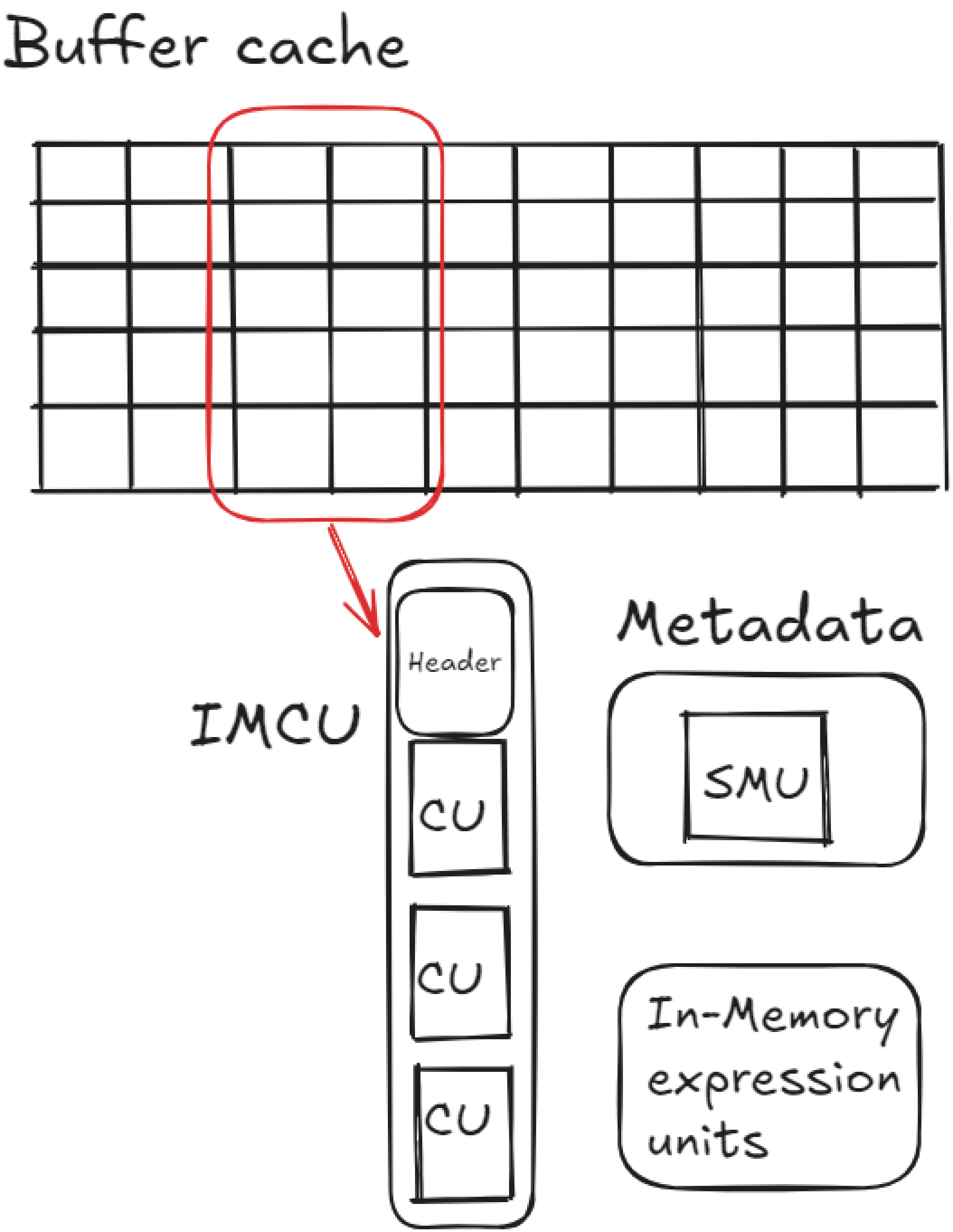
```
alter table transactions_nopart inmemory
    memcompress for query high
    priority high
    distribute by partition
        for service inm_service
    duplicate all
;
```

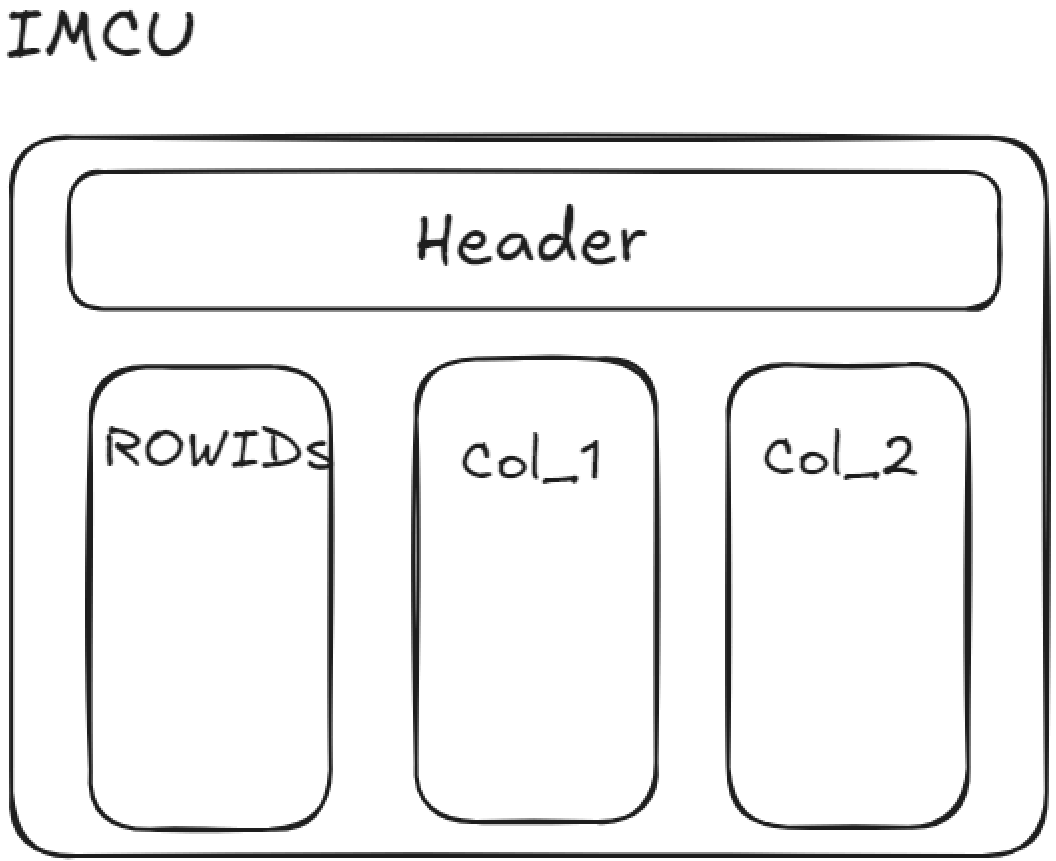
How to distribute data in Oracle RAC or Active DataGuard

```sql
alter table transactions_nopart inmemory
    memcompress for query high
    priority high
    distribute by partition
        for service inm_service
    duplicate all
;
```

Data duplication in engineered systems

Under the Hood


Buffer cache
Header
IMCU
CU
CU
CU
Metadata
SMU
In-Memory
expression
units


IMCU
Header
ROWIDs
Col_1
Col_2

Snapshot metadata units

Every IMCU has a separate SMU

SMU contains metadata for IMCU (Object and column numbers, mapping for rows)

SMU contains transaction journal

Transaction journal

Keeps IMCU transactionally consistent

In case of a change database adds rowid to the journal and marks it stale as of SCN

Stale rows are read from buffer cache

In-Memory expression units

Stores materialized In-Memory expressions and virtual columns

Logical extension of the parent IMCU

Maps to the same rowset as IMCU

Expression statistics store

- Maintained by the optimizer, stores statistics about expression evaluation
- Part of the data dictionary, used for IM expressions
- Exposed as DBA_EXPRESSION_STATISTICS view

IMCUs

Metadata pool

IM pool metadata in 23c

```
SYS@ORCLCDB 03-08-2024 09:13:11> select * from v$inmemory_area;

POOL                       ALLOC_BYTES USED_BYTES POPULATE_STATUS                CON_ID
-------------------------- ----------- ---------- -------------------------- ----------
1MB POOL                    6002049024 1918894080 DONE                                3
64KB POOL                   2562588672   96206848 DONE                                3
```

In-Memory store population and repopulation

Happens magically

Tasks are coordinated by In-Memory Coordination Process (IMCO)

Actual work is done by Space Management Worker Processes (Wnnn)

In-Memory store population

IMCO triggers population of all segments with priority higher than NONE

Segments with priority NONE are populated after they're scanned

Workers create IMCUs, SMUs, and IMEUs

In-Memory store repopulation (I)

Thresold-based, triggered when # of stale entries in IMCU reaches the threshold

Thresold is percentage of entries in transaction journal

Double buffering: new IMCU is created by combining old IMCUs with transaction journals

In-Memory store repopulation (II)

`INMEMORY_MAX_POPULATE_SERVERS` -> max number of workers
`INMEMORY_TRICKLE_REPOPULATE_PERCENT` -> max percent of time workers can do trickle repopulation

In-Memory dynamic scans (I)

Uses threads to scan the IMCUs

Uses idle CPU

Note: A lightweight thread used by IM dynamic scans is not the same as a regular thread in the multithreaded Oracle Database model.

In-Memory dynamic scans (II)

Enabled when a CPU resource plan is enabled and CPU utilization is low

`CPU_COUNT` must be `>= 24`

Query is candidate for dynamic scan if

- It access high number of IMCUs or columns
- Consumes all rows in the table
- Is CPU intensive

```sql
SELECT
    name,
    SUM(value)
FROM
    gv$sysstat
WHERE
    name LIKE '%(dynamic)%'
GROUP BY
    name;
```

ry Result

SQL | All Rows Fetched: 11 in 1,243 seconds

NAME	SUM(VALUE)
IM scan (dynamic) executing tasks	108872078
IM scan (dynamic) pending tasks	136778689
IM scan (dynamic) rows	643807255700 6
IM scan (dynamic) max degree	81019940
IM scan (dynamic) task reap time	231358687996 5
IM scan (dynamic) tasks processed by parent	310479
IM scan (dynamic) multi-threaded scans	8693796
IM scan (dynamic) tasks processed by thread	53940191
IM scan (dynamic) task execution time	80449423323
IM scan (dynamic) task submission time	424673185165 6
IM scan (dynamic) rs2 rowsets	8595729378

In-Memory joins (I)

IMCUs encoded with different dictionaries have to be decoded to be joinable

In-Memory join groups encode different tables with the same dictionary

Eliminates need for decompressing and hashing column values

External tables not supported!

In-Memory joins (II)

```
create inmemory join group cust_trans_jg ( customers ( id ),transactions ( customer_id) );
```

Common dictionary is build next time table is (re)populated

In-Memory joins (III)

```
INMTEST> column joingroup_name format A30
INMTEST> column column_name format A64
INMTEST> select joingroup_name, table_owner || '.' || table_name || '.' || column_name as column_name, flags, gd_address from user_joingroups;

JOINGROUP_NAME                 COLUMN_NAME                                                           FLAGS GD_ADDRESS
------------------------------ ---------------------------------------------------------------- ---------- ----------------
CUST_TRANS_JG                  INMTEST.CUSTOMERS.ID                                                      1 000000043FCFFF20
CUST_TRANS_JG                  INMTEST.TRANSACTIONS.CUSTOMER_ID                                          1 000000043FCFFF20
```

In-Memory joins (IV)

Usage is hidden really well

Easiest to spot in the SQL Monitoring Report

Vector aggregation

Uses arrays for joins and aggregation

Cost-based, used for `GROUP BY`

Does not support `GROUP BY ROLLUP`, `GROUPING SETS` and `CUBE`

Main benefit: allows vector joins and group by operations while scanning the fact table

KEY VECTOR and VECTOR GROUP BY

`INMEMORY_SIZE` must be set to non-zero value

Tables do not have to be populated to the IM store (!)

Transforms join between dimension and fact table into a filter

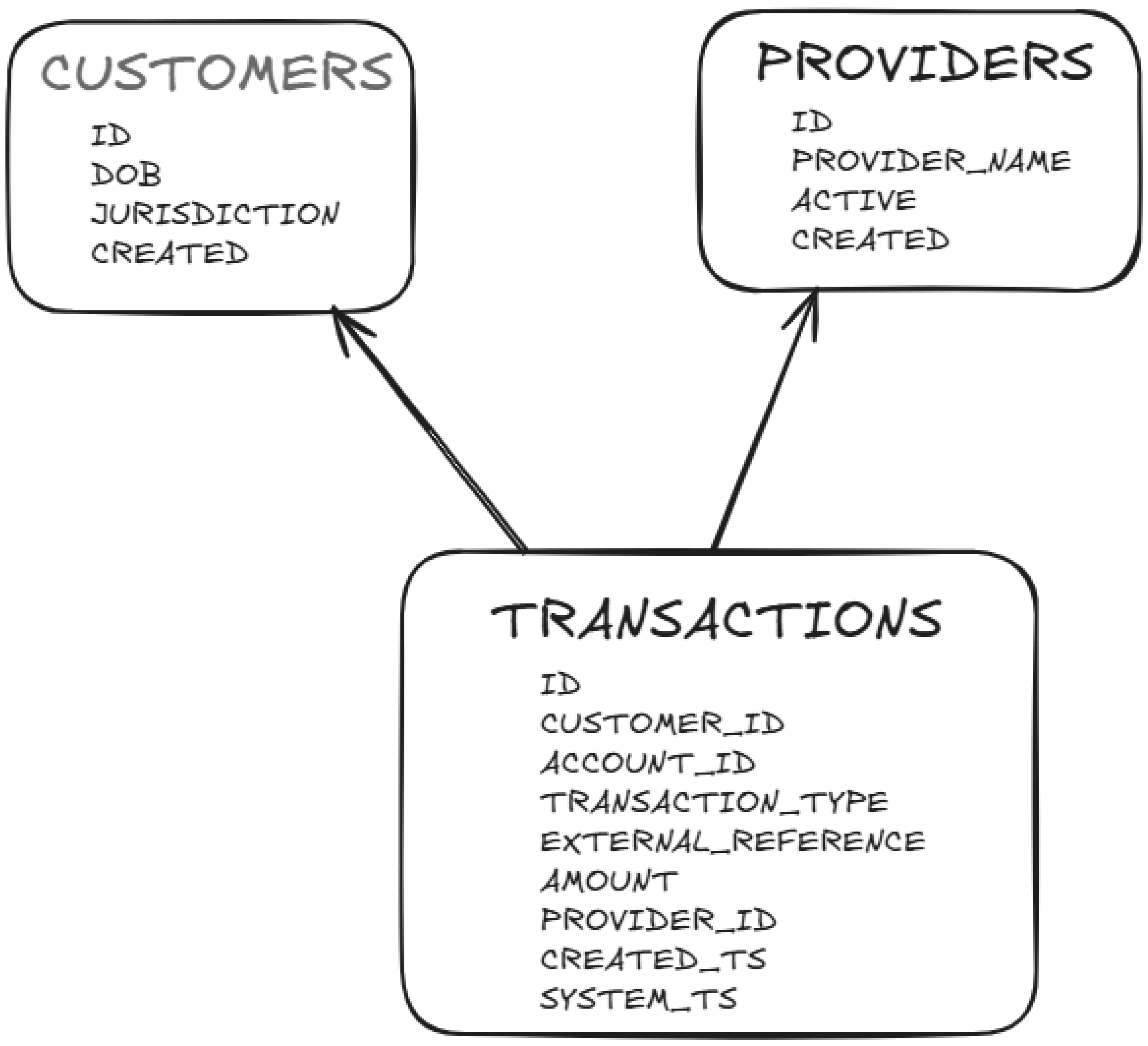
Key vectors are conceptually similar to the Bloom filters

KEY VECTOR and VECTOR GROUP BY: some conditions

Joins between large tables do not benefit from key vectors

Query joins the fact table with one or more dimensions

Multiple fact tables joined by same dimension also supported


CUSTOMERS
ID
DOB
JURISDICTION
CREATED
PROVIDERS
ID
PROVIDER_NAME
ACTIVE
CREATED
TRANSACTIONS
ID
CUSTOMER_ID
ACCOUNT_ID
TRANSACTION_TYPE
EXTERNAL_REFERENCE
AMOUNT
PROVIDER_ID
CREATED_TS
SYSTEM_TS

```sql
select c.jurisdiction,
       p.provider_name,
       sum(t.amount)
  from transactions t
  join customers c
on t.customer_id = c.id
  join providers p
on p.id = t.provider_id
 group by c.jurisdiction,
          p.provider_name;
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	Reads	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		48	00:00:03.54	216K	205K			
1	TEMP TABLE TRANSFORMATION		1		48	00:00:03.54	216K	205K			
2	LOAD AS SELECT (CURSOR DURATION MEMORY)	SYS_TEMP_0FD9D7D1D_2A534B	1		0	00:00:00.01	164	0	1024	1024	
3	HASH GROUP BY		1	3	3	00:00:00.01	163	0	1323K	1323K	
4	KEY VECTOR CREATE BUFFERED	:KV0000	1	3	3	00:00:00.01	163	0	1024	1024	
5	TABLE ACCESS FULL	CUSTOMERS	1	39999	39999	00:00:00.01	163	0			
6	LOAD AS SELECT (CURSOR DURATION MEMORY)	SYS_TEMP_0FD9D7D1E_2A534B	1		0	00:00:00.01	2	0	1024	1024	
7	HASH GROUP BY		1	16	16	00:00:00.01	2	0	1200K	1200K	
8	KEY VECTOR CREATE BUFFERED	:KV0001	1	16	16	00:00:00.01	2	0	1024	1024	
9	TABLE ACCESS FULL	PROVIDERS	1	16	16	00:00:00.01	2	0			
10	HASH GROUP BY		1	34	48	00:00:03.53	216K	205K	945K	945K	
* 11	HASH JOIN		1	34	48	00:00:03.53	216K	205K	1506K	1506K	1022K (0)
* 12	HASH JOIN		1	34	48	00:00:03.53	216K	205K	1744K	1744K	849K (0)
13	TABLE ACCESS FULL	SYS_TEMP_0FD9D7D1D_2A534B	1	3	3	00:00:00.01	0	0			
14	VIEW	VW_VT_18C3D19E	1	34	48	00:00:03.53	216K	205K			
15	VECTOR GROUP BY		1	34	48	00:00:03.53	216K	205K	65536	65536	21504 (0)
16	HASH GROUP BY			34					1647K	1647K	
17	KEY VECTOR USE	:KV0000	1	15M	15M	00:00:02.58	216K	205K	377M	6427K	
18	KEY VECTOR USE	:KV0001	1	15M	15M	00:00:01.88	216K	205K	308M	5835K	
19	PARTITION RANGE ALL		1	15M	15M	00:00:01.84	216K	205K			
20	TABLE ACCESS FULL	TRANSACTIONS	368	15M	15M	00:00:01.88	216K	205K			
21	TABLE ACCESS FULL	SYS_TEMP_0FD9D7D1E_2A534B	1	16	16	00:00:00.01	0	0			

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	Reads	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		48	00:00:03.54	216K	205K			
1	TEMP TABLE TRANSFORMATION		1		48	00:00:03.54	216K	205K			
2	LOAD AS SELECT (CURSOR DURATION MEMORY)	SYS_TEMP_0FD9D7D1D_2A534B	1		0	00:00:00.01	164	0	1024	1024	
3	HASH GROUP BY		1	3	3	00:00:00.01	163	0	1323K	1323K	
4	KEY VECTOR CREATE BUFFERED	:KV0000	1	3	3	00:00:00.01	163	0	1024	1024	
5	TABLE ACCESS FULL	CUSTOMERS	1	39999	39999	00:00:00.01	163	0			
6	LOAD AS SELECT (CURSOR DURATION MEMORY)	SYS_TEMP_0FD9D7D1E_2A534B	1		0	00:00:00.01	2	0	1024	1024	
7	HASH GROUP BY		1	16	16	00:00:00.01	2	0	1200K	1200K	
8	KEY VECTOR CREATE BUFFERED	:KV0001	1	16	16	00:00:00.01	2	0	1024	1024	
9	TABLE ACCESS FULL	PROVIDERS	1	16	16	00:00:00.01	2	0			
10	HASH GROUP BY		1	34	48	00:00:03.53	216K	205K	945K	945K	
* 11	HASH JOIN		1	34	48	00:00:03.53	216K	205K	1506K	1506K	1022K (0)
* 12	HASH JOIN		1	34	48	00:00:03.53	216K	205K	1744K	1744K	849K (0)
13	TABLE ACCESS FULL	SYS_TEMP_0FD9D7D1D_2A534B	1	3	3	00:00:00.01	0	0			
14	VIEW	VW_VT_18C3D19E	1	34	48	00:00:03.53	216K	205K			
15	VECTOR GROUP BY		1	34	48	00:00:03.53	216K	205K	65536	65536	21504 (0)
16	HASH GROUP BY			34					1647K	1647K	
17	KEY VECTOR USE	:KV0000	1	15M	15M	00:00:02.58	216K	205K	377M	6427K	
18	KEY VECTOR USE	:KV0001	1	15M	15M	00:00:01.88	216K	205K	308M	5835K	
19	PARTITION RANGE ALL		1	15M	15M	00:00:01.84	216K	205K			
20	TABLE ACCESS FULL	TRANSACTIONS	368	15M	15M	00:00:01.88	216K	205K			
21	TABLE ACCESS FULL	SYS_TEMP_0FD9D7D1E_2A534B	1	16	16	00:00:00.01	0	0			

```
<snip>
               3  key vector dgk batch parcels
               4  key vector dgk range parcels
           40580  key vector hash cells scanned
              22  key vector hash inserts
           40558  key vector hash probes
               2  key vector non cas merge operations
               1  key vector queries
        31984002  key vector rows processed by value
               2  key vectors created
               1  key vectors created (byte wide)
               1  key vectors created (nibble wide)
               2  key vectors created (simple layout)
<snip>
              96  vector group by accumspace cardinality
            3072  vector group by accumspace size
               1  vector group by used
<snip>
```

Compression and optimized arithmetic

Compression levels from `FOR DML` to `FOR CAPACITY HIGH`

`FOR QUERY LOW` seems to be using dictionary encoding only,

`FOR CAPACITY HIGH` uses Zstandard.

Optimized arithmetic

Parameter `INMEMORY_OPTIMIZED_ARITHMETIC`

Can be `DISABLE` (default) or `ENABLE`

When enabled, NUMBERs are encoded in CPU friendly format

Takes effect from `FOR QUERY` compression levels onwards

In-Memory expressions

Precompute and store computationally expensive expressions

Created automatically by the database

Database tracks most active expressions in the capture window

In-Memory virtual columns

Created by the user

Populated by the IM expressions infrastructure

IM expression capture

`DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS` identifies 20 hottest expressions in the time range

Possible time intervals:

- `CUMULATIVE` -- all expressions since the creation of the database
- `CURRENT` -- expressions from the past 24h
- `WINDOW` -- expressions from the last capture window

IM expression capture (II)

Captured expressions will become hidden `SYS_IME` columns

Old expressions will be markes with `NO INMEMORY` attribute

Table can have max 50 `SYS_IME` columns

To allow new columns, old columns must be dropped

IM expression capture (III)

Expressions are populated

- When `DBMS_INMEMORY_ADMIN.IME_POPULATE_EXPRESSIONS` is called
- When parent IMCUs are (re)populated

In-Memory optimized dates

Set the `INMEMORY_OPTIMIZED_DATE` to `ENABLE`

`DATE` fields will be populated with `MONTH` and `YEAR` IM expressions

Speeds up `EXTRACT` function

Available in Oracle 23ai

In-Memory and JSON

Useful for queries that scan large number of small JSON documents

Supports full-text search for `JSON` data type

Speeds up SQL/JSON path access

JSON data in IM column store is stored as OSON

In-Memory and JSON: limitations

Limitation: documents should be smaller than 32k

Parameter `max_string_size` should be set to `extended`

In-Memory and JSON: initialization parameters

`INMEMORY_EXPRESSIONS_USAGE` -> `STATIC_ONLY` or `ENABLE`

`INMEMORY_VIRTUAL_COLUMNS` -> `ENABLE`

Thank you!