Oracle Database –

Memory and Caching Technologies

#oracledatabase #oraclecloud #autonomousdatabase #memory #caching #tables #cursors #resultcache #sql #plsql #19c #23ai #truecache #inmemory #bufferpools #tipps #performance #advisor #fastingest #fastlookup #memoptimize

Ulrike Schwinn
Oracle Data Management Expert

Videos, Blogs, GitHub from Oracle EMEA Experts and Specialists

YouTube: 23ai Playlist OraTech

GitHub: converged database

Blogs: blogs.oracle.com/coretec

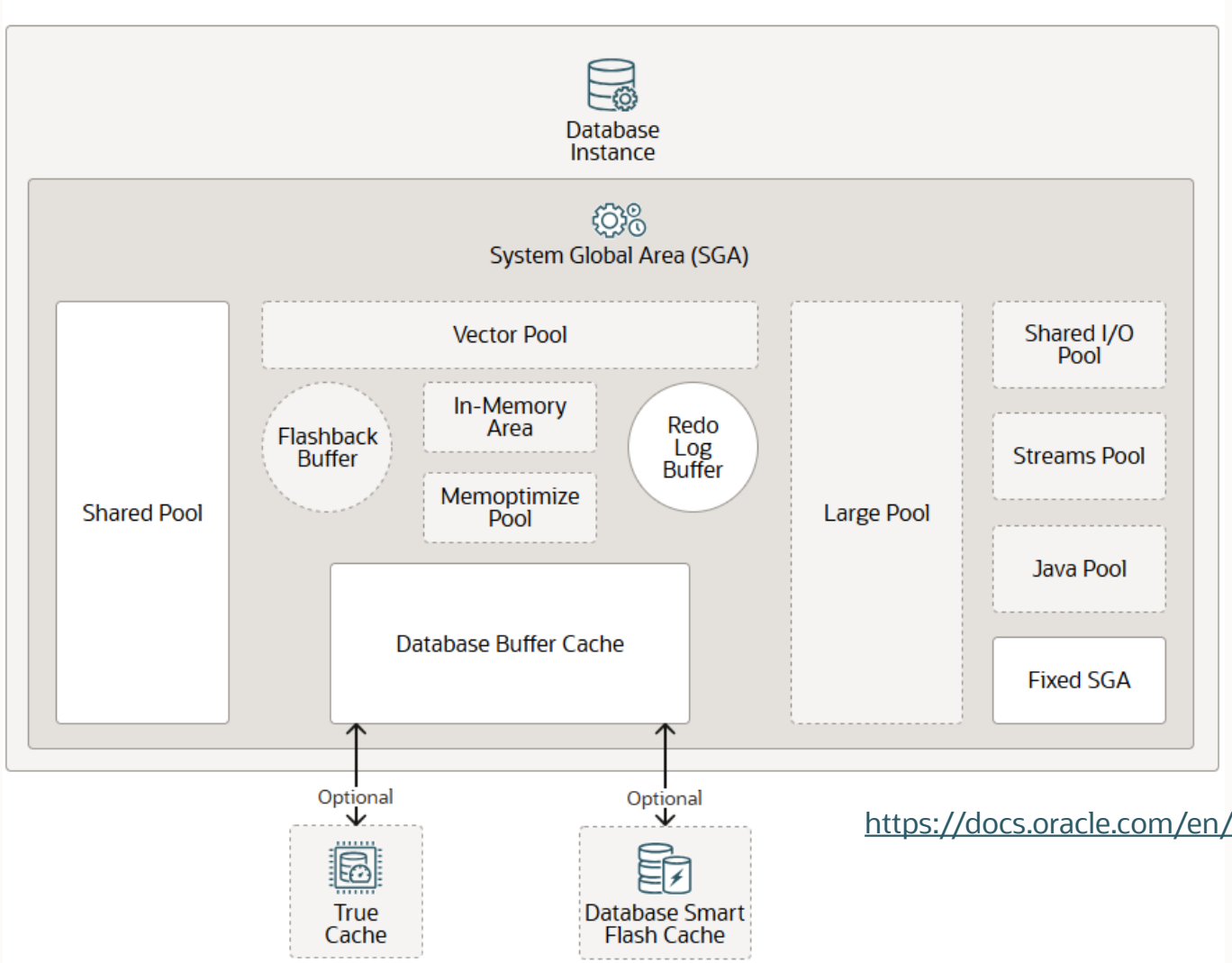
Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Some Performance Aspects

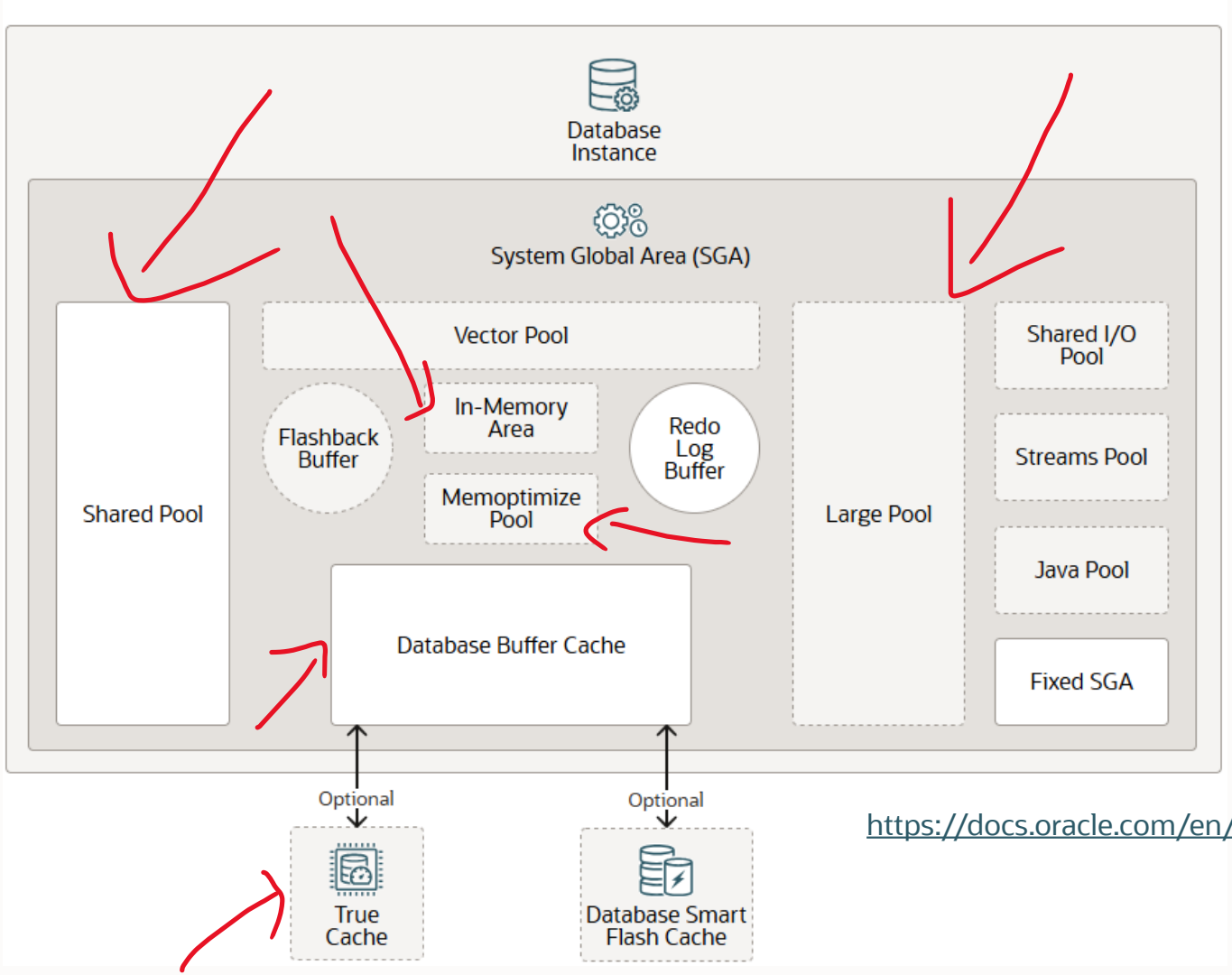
- When to check performance?
  - Loss in performance/Degradation/Complaints
  - New Versions/Feature Testing
  - Migration/Upgrade
- Focus: on statements/application code with „bad“ performance
- Know your **available access structure and design**
  - Indexes, IOT, MAVs, Views, Partitioning, data types, ...
- Make sure **statistics** are up-to-date
- Dont‘s
  - Row by Row Processing
- Do‘s
  - Bulk Loads, Direct Loads, Parallelization
  - Result Cache for compute intense queries
  - Object Caching/Warming up
  - Memory optimizing techniques such as compression, partitioning etc.
  - Use Advisory and monitoring framework and automations (if possible)

Oracle Database 23ai Technical Architecture – System Global Area



https://docs.oracle.com/en/database/oracle/oracle-database/23/dbiad/db_sga.html

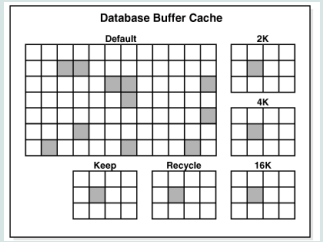
Oracle Database 23ai Technical Architecture – System Global Area



https://docs.oracle.com/en/database/oracle/oracle-database/23/dbiad/db_sga.html

Oracle Database Memory and Caching

Database Buffer



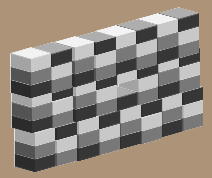
Result Cache

Server Result Cache

SQL Query Result Cache

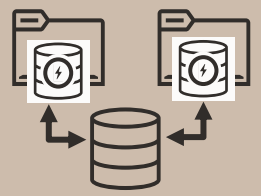
PL/SQL Functon Result Cache

Database Smart Flash Cache Buffer



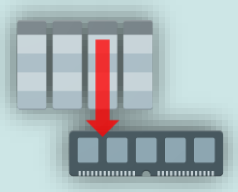
Flash Cache Level 2

23ai True Cache



High Speed Data Ingestion MemOptimized RowStore

Database In-Memory



Exadata

with HCC
Smart Scan
Storage Index
Smart Flash Cache

Oracle TimesTen In-Memory Database

memory-optimized relational database deployed in the middle-tier

Coherence Data Grid

in-memory distributed data grid solution for clustered applications

Database Buffer

SGA, LRU
Keep&Recycle Pools

Table data caching
PL/SQL& cursor caching

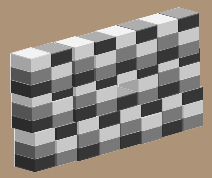
Result Cache

Result Cache Area
in Shared Pool
for compute intense
queries and PL/SQL
function results

High Speed
Data Ingestion
MemOptimized RowStore

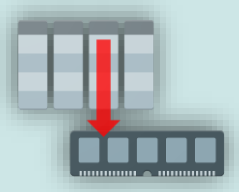
Oracle Database Memory and Caching

Database Smart
Flash Cache Buffer



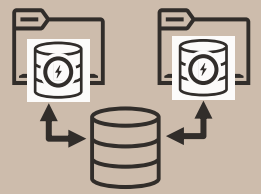
Flash Cache Level 2

Database
In-Memory



23ai

True Cache



List SGA Components

- v$sga_dynamic_components summarizes information based on all completed SGA resize operations.

```
COMPONENT                       CURRENT_SIZE
------------------------------ ------------
shared pool                      2.9595E+10
large pool                       4362076160
java pool                        1610612736
streams pool                      268435456
unified pga pool                          0
memoptimize buffer cache          536870912
DEFAULT buffer cache             9.9724E+10
KEEP buffer cache                         0
RECYCLE buffer cache                      0
DEFAULT 2K buffer cache                   0
…
In-Memory Area                            0
…
```

See also Documentation

- Use Memory Advisor with DB_CACHE_ADVICE set to ON
  - Enables statistics gathering used for predicting behavior with different cache sizes through the v$db_cache_advice performance view.

Usage of V$DB_CACHE_ADVICE

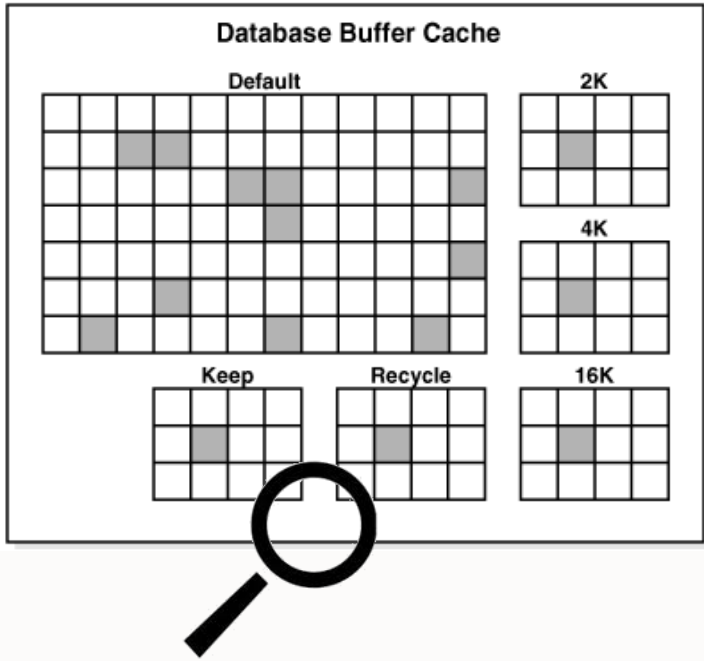
- Set DB_CACHE_ADVICE ON. If STATISTICS_LEVEL is TYPICAL or ALL, then default is already ON.
- Query V$DB_CACHE_ADVICE - here for default buffer pools.

```
SELECT size_for_estimate size, buffers_for_estimate buffers, estd_physical_read_factor estd_read,
estd_physical_reads estd_write
FROM V$DB_CACHE_ADVICE
WHERE name = 'DEFAULT' AND block_size =
(SELECT value FROM V$PARAMETER WHERE name = 'db_block_size');
```

SIZE	BUFFERS	ESTD_READ_FACTOR	ESTD_WRITE	
30	3,802	18.70	192,317,943	<== 10% of Current Size
60	7,604	12.83	131,949,536	
…				
212	26,614	1.74	17,850,847	
243	30,416	1.33	13,720,149	
273	34,218	1.13	11,583,180	
304	38,020	**1.00**	10,282,475	<== Current Size
334	41,822	.93	9,515,878	
364	45,624	.87	8,909,026	
395	49,426	.83	8,495,039	

See also Documentation

Buffer Cache Pools: KEEP and RECYCLE Pools



- Manually configure separate buffer pools that either **keep data** in the buffer cache or **make the buffers available for new data** immediately after using the data blocks.

- Pool configuration with `DB_KEEP_CACHE_SIZE, DB_RECYCLE_CACHE_SIZE`
- Assign schema object with ALTER/CREATE command

```
ALTER TABLE ... STORAGE (buffer_pool keep)
ALTER INDEX ... STORAGE (buffer_pool keep)
ALTER TABLE ... MODIFY LOB (lobcol) (STORAGE (buffer_pool keep))
```

- When to consider:
  - Objects in KEEP pool are HOT and should not be aged out.
  - Objects in RECYCLE pool should not consume unnecessary space in the cache.

See also Documentation

Are database objects cached?

- Check execution plans, statistics, or V$ views such as V$BH
- v$bh displays the status and number of pings for every buffer in the SGA.
- Example:

```
SELECT o.object_name, o.object_type, o.owner, COUNT(*) NUMBER_OF_BLOCKS
FROM dba_objects o, v$bh bh
WHERE o.data_object_id = bh.objd
AND o.owner in ('SH')
GROUP BY o.object_name, o.owner, o.object_type
ORDER BY COUNT(*) desc;

OBJECT_NAME                    OBJECT_TYPE                    OWNER NUMBER_OF_BLOCKS
------------------------------ ------------------------------ ----- ----------------
CUSTOMERS                      TABLE                          SH                 1673
SALES                          TABLE PARTIT                   SH                  233
SALES_PROMO_BIX                INDEX PARTITION                SH                   58
…
```

See also Documentation

Objects Caching/Pre-Loading

- Initiate object scans over tables, indexes , lob segments etc.

```
SELECT /*+ FULL(T1) */ sum(numeric_column), min(txt_column) FROM tabelle T1;
SELECT /*+ FULL(T2) */ dbms_lob.getlength(lob_column) FROM tabelle T2;
```

- Keep in mind: Short table scans are scans performed on tables under a certain size threshold. The definition of a small table is the maximum of 2% of the buffer cache or 20, whichever is bigger.
- Big tables (= segments) are not cached when read with a full scan.
  - By default, a big table (=segment) is typically a table (=segment) bigger than 2% of the buffer cache.
  - This behavior can be controlled by setting the "_small_table_threshold" parameter, that specifies the threshold in database blocks for deciding if a segment should be cached.
    - Modifiable with ALTER SESSION to get a bigger size

See also Documentation

Check hidden parameters

```
SELECT a.ksppinm "Parameter", b.KSPPSTDF "Default", b.ksppstvl "Session", c.ksppstvl "Instance",
       decode(bitand(a.ksppiflg/256,1),1,'TRUE','FALSE') IS_SESSION_MOD,
       decode(bitand(a.ksppiflg/65536,3),1,'IMMEDIATE',2,'DEFERRED',3,'IMMEDIATE','FALSE') IS_SYSTEM_MOD
FROM x$ksppi a,x$ksppcv b, x$ksppsv c
WHERE a.indx = b.indx AND a.indx = c.indx AND UPPER(ksppinm) like UPPER('%&name%');

Enter value for name: _small_table
old   5: AND UPPER(ksppinm) like UPPER('%&name%')
new   5: AND UPPER(ksppinm) like UPPER('%_small_table%')

Parameter                      Default  Session  Instance   IS_SESSION_MOD  IS_SYSTEM_MOD
------------------------------ -------- -------- ---------- --------------- ---------------
_small_table_threshold         TRUE     220314   220314     TRUE            DEFERRED
...
```

How To Query And Change The Oracle Hidden Parameters In Oracle 10g and Later (Doc ID 315631.1)

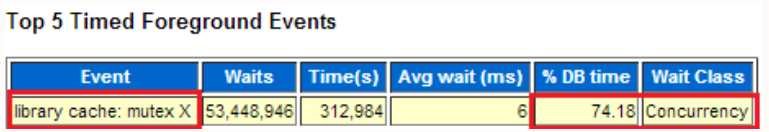
PL/SQL Object Caching

- To improve performance and avoid reparsing, prevent SQL or PL/SQL areas from aging out of the shared pool.
- dbms_shared_pool enables you to mark them for keeping or not-keeping , to purge, to markhot or unmarkhot
  - For frequently used triggers, procedures and tables, also for sequences, use KEEP

```
execute dbms_shared_pool.keep(name=> 'SH.TESTP1', flag => 'P');
execute dbms_shared_pool.keep(name=> '4158E358 2329752635', flag=> 'C');
```

  - For „Library cache: mutex X" as a top events, consider MARKHOT

```
execute dbms_shared_pool.markhot('SH.TESTP1', 'P');
```

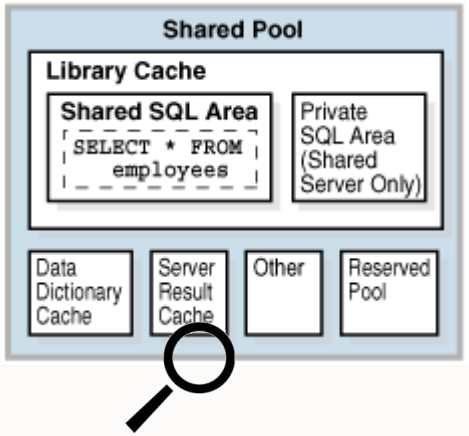


- Monitoring in v$db_object_cache

```
SQL> select name, kept, status from v$db_object_cache where name like 'TESTP1';
     NAME                           KEPT      STATUS
     ------------------------------ --------- ---------
     TESTP1                         YES       VALID
```

See also Documentation

SQL Query and PL/SQL Function Result Cache



Server Result Cache contains:

- **SQL query result cache** stores the results of queries and query fragments.
- **PL/SQL function result cache** stores function result sets.

Server Result Cache	On database startup, Oracle Database allocates memory to the server result cache in the shared pool.

SQL Query Result Cache	• Result cache hints at the application level
	`SELECT /*+ RESULT_CACHE */ ...`
	• Table annotations affect the entire query
	`CREATE TABLE sales (...) RESULT_CACHE (MODE DEFAULT)`

PL/SQL Function Result Cache	• Needs RESULT_CACHE clause

When to consider:

- Predictable queries/repeating queries
- Long-running queries with expensive calculations
- Functions that are invoked frequently but depend on information that changes infrequently

See also Documentation

Parameters

RESULT_CACHE_MAX_RESULT	5% % of RESULT_CACHE_MAX_SIZE any single result can use
RESULT_CACHE_MAX_SIZE	derived from memory components (**0 means disabled**)
RESULT_CACHE_MODE	MANUAL | MANUAL_TEMP | FORCE | FORCE_TEMP
RESULT_CACHE_REMOTE_EXPIRATION	minutes that a result using a remote object is allowed to remain valid.
RESULT_CACHE_MAX_TEMP_RESULT **21c**	5 % % of RESULT_CACHE_MAX_TEMP_SIZE
RESULT_CACHE_MAX_TEMP_SIZE **21c**	RESULT_CACHE_MAX_SIZE * 10 maximum amount of temporary tablespace that can be consumed by the result cache.
RESULT_CACHE_INTEGRITY **23ai**	ENFORCED | TRUSTED (*)
RESULT_CACHE_EXECUTION_THRESHOLD **21c**	2 maximum number of times a PL/SQL function can be executed before its result is stored in the result cache.
RESULT_CACHE_AUTO_BLOCKLIST **23ai**	ON | OFF to enable or disable adaptive result cache object exclusion

See also Documentation

Usage in Autonomous Database and Exadata

NAME	VALUE
result_cache_integrity	ENFORCED
result_cache_mode	FORCE
result_cache_max_size	5242880
result_cache_max_result	1
result_cache_remote_expiration	0
result_cache_execution_threshold	2
result_cache_max_temp_size	536870912
result_cache_max_temp_result	5
result_cache_auto_blocklist	ON

```
SQL> select name, value from v$parameter
     where name like 'result%';

NAME                                     VALUE
---------------------------------------- ----------
result_cache_integrity                   TRUSTED
result_cache_mode                        MANUAL
result_cache_max_size                    4194304
result_cache_max_result                  5
result_cache_remote_expiration           0
result_cache_execution_threshold         2
result_cache_max_temp_size               41943040
result_cache_max_temp_result             5
result_cache_auto_blocklist              ON
```

Result Cache: Query Hints

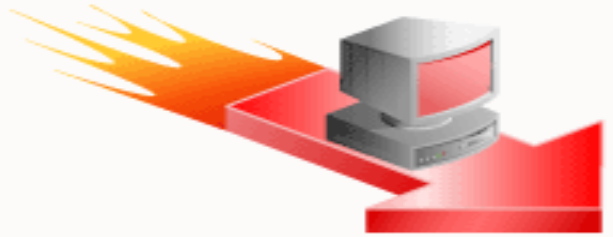
```
SELECT /*+ result_cache */ a.department_id "Department",
   a.num_emp/b.total_count "%_Employees",
   a.sal_sum/b.total_sal "%_Salary"
FROM
(SELECT /*+ result_cache */ department_id, COUNT(*) num_emp, SUM(salary) sal_sum
   FROM employees
   GROUP BY department_id) a,
(SELECT /*+ result_cache */ COUNT(*) total_count, SUM(salary) total_sal
   FROM employees) b
ORDER BY a.department_id;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Operation	Optimizer	Cost	Cardinality	Bytes	Partition Start	Partition Stop	Partition Id
SELECT STATEMENT	ALL_ROWS	7	11	715			
RESULT CACHE 7v838kf3k3zkg0yqpabtt131dt							
SORT(ORDER BY)		7	11	715			
NESTED LOOPS		7	11	715			
VIEW		3	1	26			
RESULT CACHE a6jgaqk8d0wzq48umutf26n89h							
SORT(AGGREGATE)			1	4			
TABLE ACCESS(FULL) HR.EMPLOYEES	ANALYZED	3	107	428			
VIEW		4	11	429			
RESULT CACHE a2ujsg3da29mhc8rhhmga0ygz7							
SORT(GROUP BY)		4	11	77			
TABLE ACCESS(FULL) HR.EMPLOYEES	ANALYZED	3	107	749			

See also Documentation

Enabling PL/SQL Result Cache for Functions



- To make a function result-cached, you must include the RESULT_CACHE clause in the function declaration and definition.

```
CREATE OR REPLACE FUNCTION get_date (p_id number, p_format varchar2) RETURN varchar2
RESULT_CACHE
IS
  v_date date;
BEGIN
  select hiredate into v_date
  from emp where empno = p_id;

  RETURN to_char(v_date, p_format);
END;
```

21c
- RESULT_CACHE_EXECUTION_THRESHOLD: max number of times a PL/SQL function can be executed before its result is stored in the result cache.

See also Documentation

Administration

Use dbms_result_cache.status | memory_report | flush | bypass

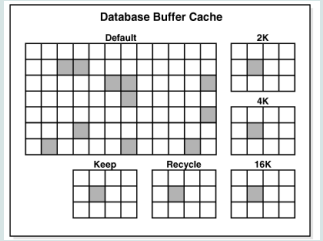
and v$views such as

```
SQL> select name, type, row_count,  status, invalidations, scan_count
     from v$result_cache_objects where scan_count!=0

NAME                           TYPE        ROW_COUNT STATUS     INVALIDATIONS SCAN_COUNT
------------------------------ ---------- ---------- ---------- ------------- ----------
"SCOTT"."GET_DATE"::8."GET_D Result              1 Published              0          2
ATE"#3a2b06cfa1322e42 #1

select /*+ result_cache */ c Result              1 Published              0          1
ount(*) from emp
```

See also Documentation

Oracle Database Memory and Caching

Database Buffer



Result Cache

Server Result Cache

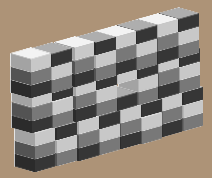
SQL Query Result Cache

PL/SQL Functon Result Cache

High Speed Data Ingestion MemOptimized RowStore

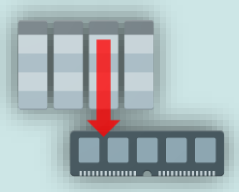
Large Pool to buffer INSERTs
Memoptimize Pool for fast lookups

Database Smart Flash Cache Buffer

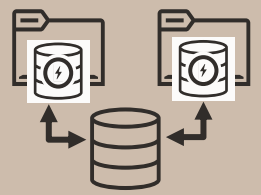


Flash Cache Level 2

Database In-Memory



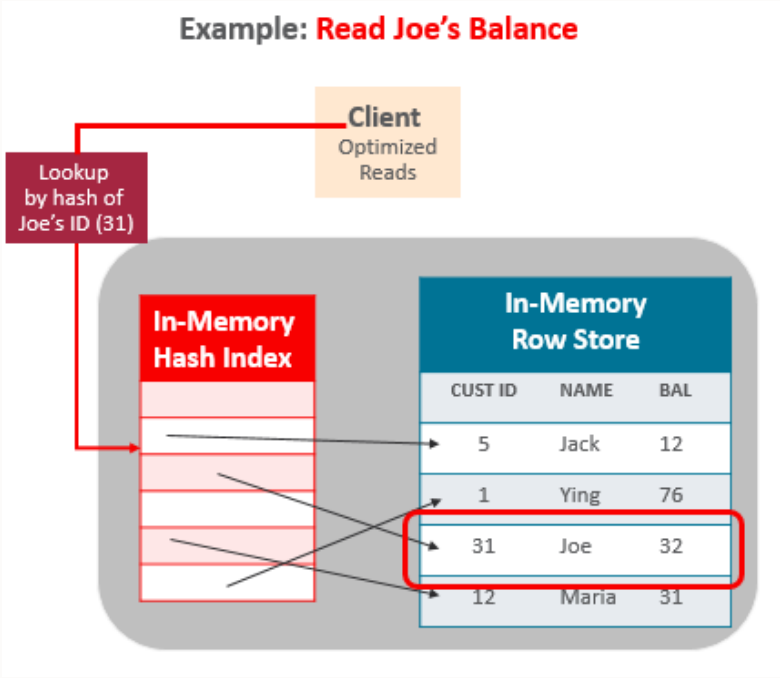
23ai True Cache



Data Streaming: Fast Lookup and Fast Ingest

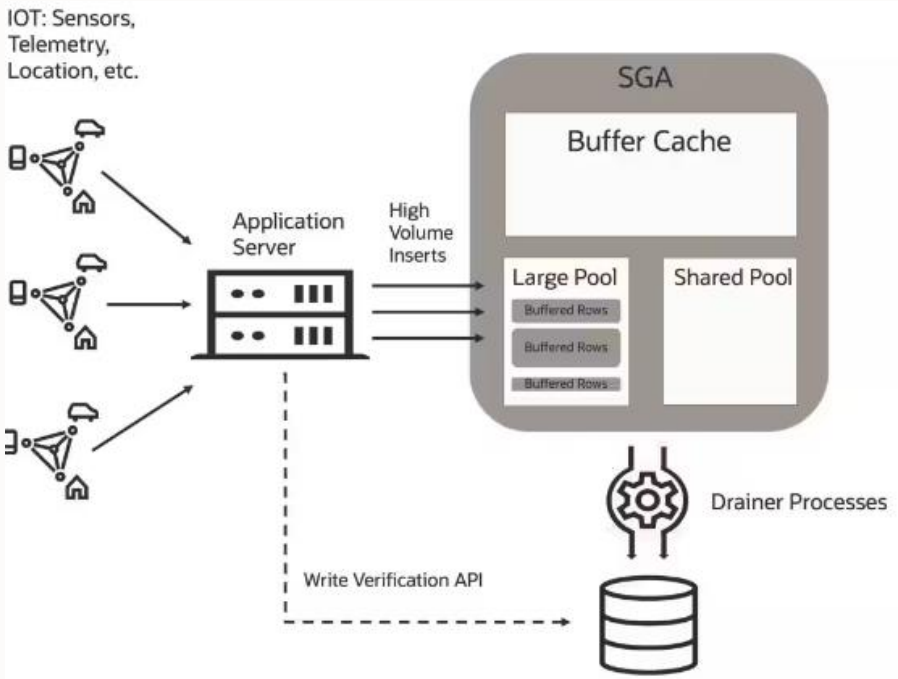
Fast Lookup

Scenarios: Data streaming applications like Internet of Things (IoT) **request/read** data for clients at a very high frequency.



Fast Ingest

Scenarios: IoT data from sensor, smart meter or even traffic camera might be collected and **written** to the database in high volumes for **later analysis**.



Included in Enterprise Edition from 19.12 onwards and in Database 23ai Free edition

Fast Lookup for high frequency queries - part 1

- Special **memoptimize pool** in SGA with buffer area and hash index
- Automatically managed
- For queries **with a single equality predicate on the primary key**

Workflow

1. Change MEMOPTIMIZE_POOL_SIZE in ROOT and restart
2. Add table attribute MEMOPTIMIZE FOR READ
3. Run DBMS_MEMOPTIMIZE.POPULATE
4. Query data!

```
-- in container ROOT set parameter and restart
SQL> alter system set MEMOPTIMIZE_POOL_SIZE = 500M SCOPE=SPFILE;
System altered.
-- in pdb
SQL> create table sh.sales_tab (sales_id   NUMBER(6) primary key,
                                prod_id     NUMBER(6) not null,
                                cust_id     NUMBER not null,
                                time_id     DATE not null,
                                quantity_sold NUMBER(3) not null,
                                amount_sold   NUMBER(10,2) not null);
Table created.
SQL> insert into sh.sales_tab
 (sales_id,prod_id, cust_id, time_id, quantity_sold, amount_sold)
  select rownum,prod_id,cust_id,time_id,quantity_sold,amount_sold
 from sh.sales;
918843 rows created.

-- enable it for the data stored in SH.SALES_TAB.
SQL> alter table sh.sales_tab MEMOPTIMIZE FOR READ;
Table altered.
SQL> execute dbms_stats.gather_table_stats('SH','SALES_TAB');
PL/SQL procedure successfully completed.
```

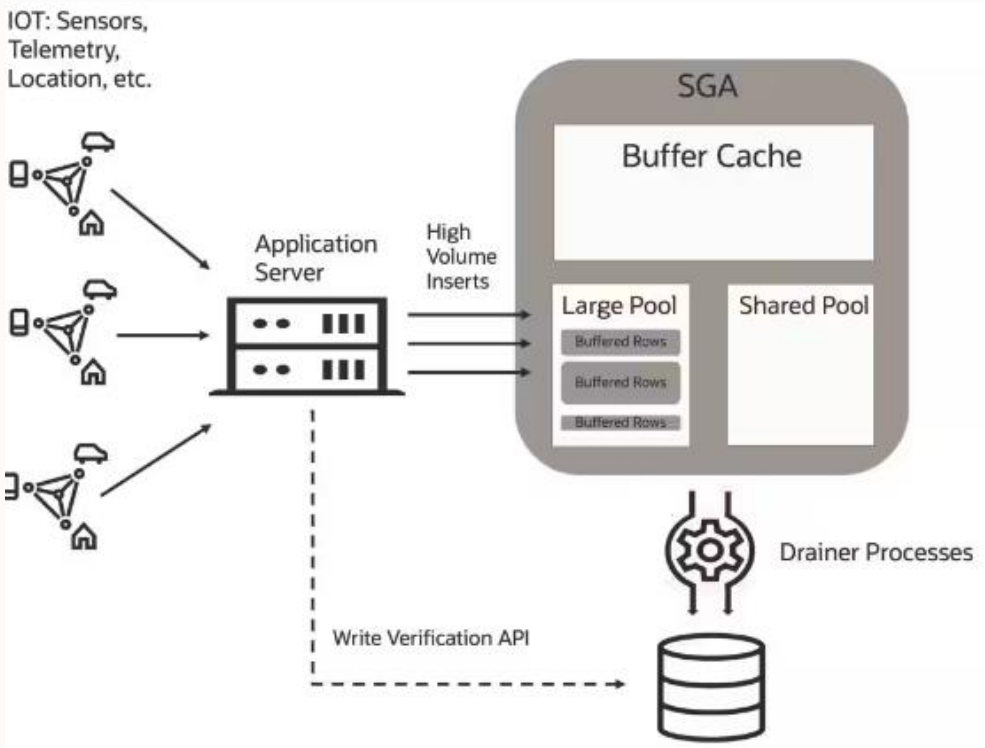
Fast Lookup for high frequency queries – part 2

- For queries **with a single equality predicate on the primary key**

```
--Populate the rowstore with DBMS_MEMOPTIMIZE.POPULATE
SQL> execute dbms_memoptimize.populate(schema_name=>'SH', table_name=>'SALES_TAB');
PL/SQL procedure successfully completed.

SQL> select * from sh.sales_tab where sales_id=5;
…
SQL> select * from dbms_xplan.display_cursor();
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------
SQL_ID  dfn2vxgz8vuyf, child number 1
-------------------------------------
select * from sh.sales_tab where sales_id=5
Plan hash value: 1081214910
------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |              |       |       |     3 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID READ OPTIM| SALES_TAB    |     1 |    27 |     3   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN READ OPTIM           | SYS_C0011070 |     1 |       |     2   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("SALES_ID"=5)
```

Fast Ingest for high-frequency data inserts

- Introduced in 19c
- Fast Ingest operations are very different from normal Oracle Database transaction processing.
- The normal transaction mechanisms are bypassed and the application needs to check the data was written.
  - COMMITs have no meaning.
  - There is no ability to rollback the inserts.
  - You cannot query data until it has been flushed to disk.
  - Index operations and constraint checking is done only when the data is written from the fast ingest area in the large pool to disk.
- Special packages like DBMS_MEMOPTIMIZE and DBMS_MEMOPTIMIZE_ADMIN support the technology.



Fast Ingest for high-frequency data inserts

- Uses the **large pool** to buffer the INSERTs before they are written to the hard disk

In **23ai**

- MEMOPTIMIZE_WRITE_AREA_SIZE enables explicit allocation for the memoptimize write area in large pool.
- MEMOPTIMIZE_WRITES parameter avoids having to use the MEMOPTIMIZE_WRITE hint.

```
-- in container ROOT: the size of the ingest buffer in large pool
SQL> alter system set
     memoptimize_write_area_size=300M scope=spfile;
System altered.

-- restart the database
-- connect to the pdb
-- create a table for fast ingest
SQL> create table sh.sales_write_tab
     (sales_id      NUMBER(6) primary key,
      prod_id       NUMBER(6) not null,
      cust_id       NUMBER not null,
      time_id       DATE not null,
      quantity_sold NUMBER(3) not null,
      amount_sold   NUMBER(10,2) not null)
      segment creation immediate
      memoptimize for write;
Table created.
```

Demo: Fast Ingest

- DBMS_MEMOPTIMIZE.WRITE_END flushes fast ingest data from large pool.

- 23ai Fast Ingest supports now also LOBs, Compression, Encryption, and Partitioning (interval, hash and sub-partition)

```
-- instruct the database to always use fast ingest for INSERTs
SQL> alter session set memoptimize_writes=on;
Session altered.

SQL> insert into sh.sales_write_tab
(sales_id,prod_id,cust_id,time_id,quantity_sold,amount_sold)
select rownum,prod_id,cust_id,time_id,quantity_sold,amount_sold
from sh.sales;
918843 rows created.

SQL> select count(*) from sh.sales_write_tab;
  COUNT(*)
----------
    113617

-- flush all the fast ingest data from large pool to disk

SQL> exec dbms_memoptimize.write_end;
PL/SQL procedure successfully completed.

SQL> select count(*) from sh.sales_write_tab;
      COUNT(*)
    ----------
        918843
```

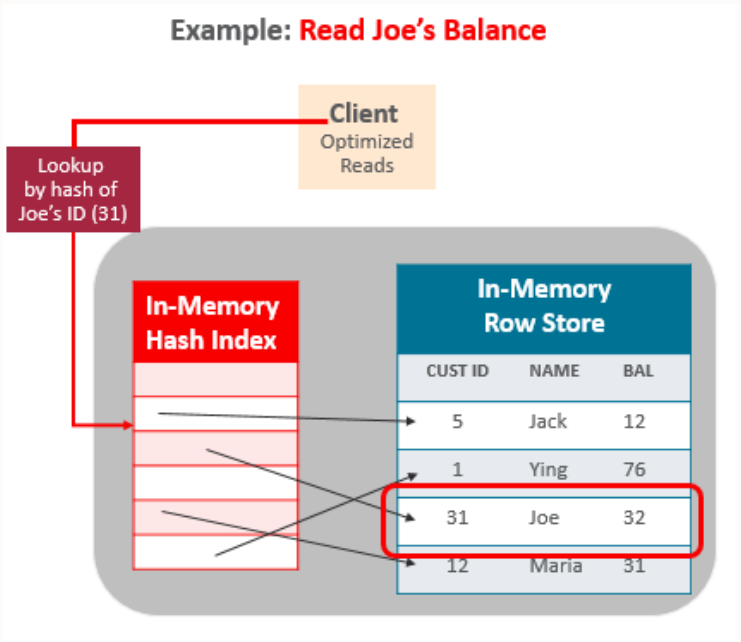
Fast Lookup and Fast Ingest in Oracle Database

Fast Lookup

Introduced in 18c

Uses special **memoptimize pool** in SGA with buffer area and hash index

For queries **with a single equality predicate on the primary key**

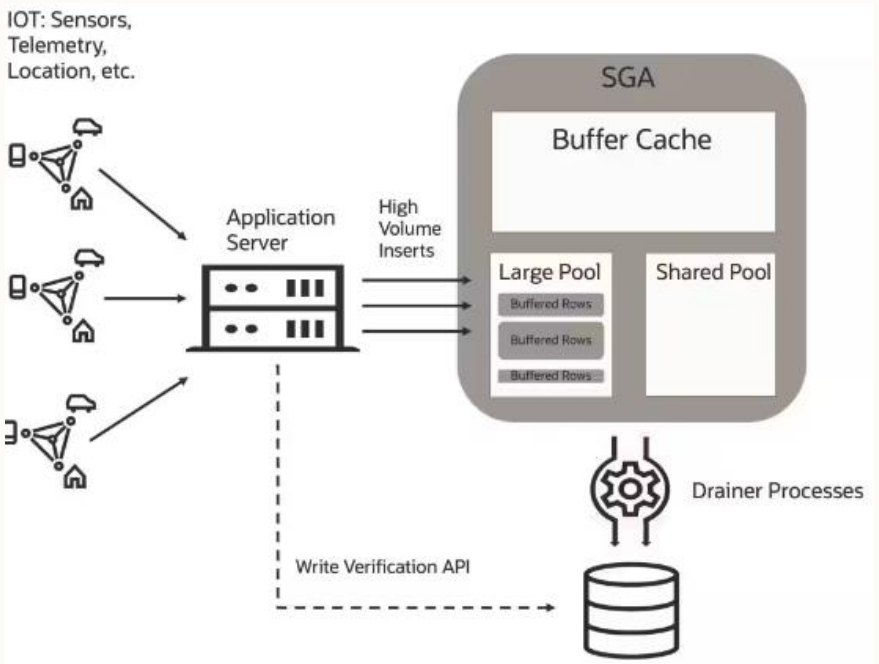


Fast Ingest

Introduced in 19c

Uses **large pool** to buffer the INSERTs before they are written to disk

In **23ai**, improved memory handling and new supported objects and technology



High Performance Data Streaming with Fast Lookup and Fast Ingest (YouTube)

Further Readings

Documentation

- Database Performance Tuning Guide 19c
- Database Performance Tuning Guide 23ai
- Database Concepts 23ai
- PL/SQL Packages and Types References 23ai
- Database Licensing Information User Manual

Blogs, Videos & White Paper

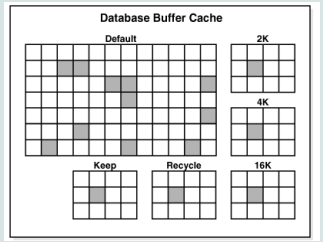
- Oracle Database 23ai Fast Ingest Enhancements
- New in Oracle Database 19c: Memoptimized Rowstore – Fast Ingest
- Best Practices For High Volume IoT workloads with Oracle Database 19c (White Paper)
- Fast Lookup with Memory Optimized Rowstore
- High Performance Data Streaming with Fast Lookup and Fast Ingest (YouTube)

- **All Links**

https://github.com/oracle-devrel/technology-engineering/tree/main/data-platform/core-converged-db/fast-ingest-lookup

Oracle Database Memory and Caching

Database Buffer



Result Cache

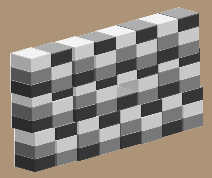
Server Result Cache

SQL Query Result Cache

PL/SQL Functon Result Cache

High Speed Data Ingestion MemOptimized RowStore

Database Smart Flash Cache Buffer

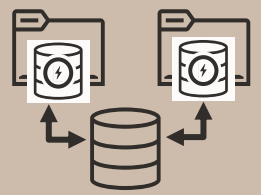


Flash Cache Level 2

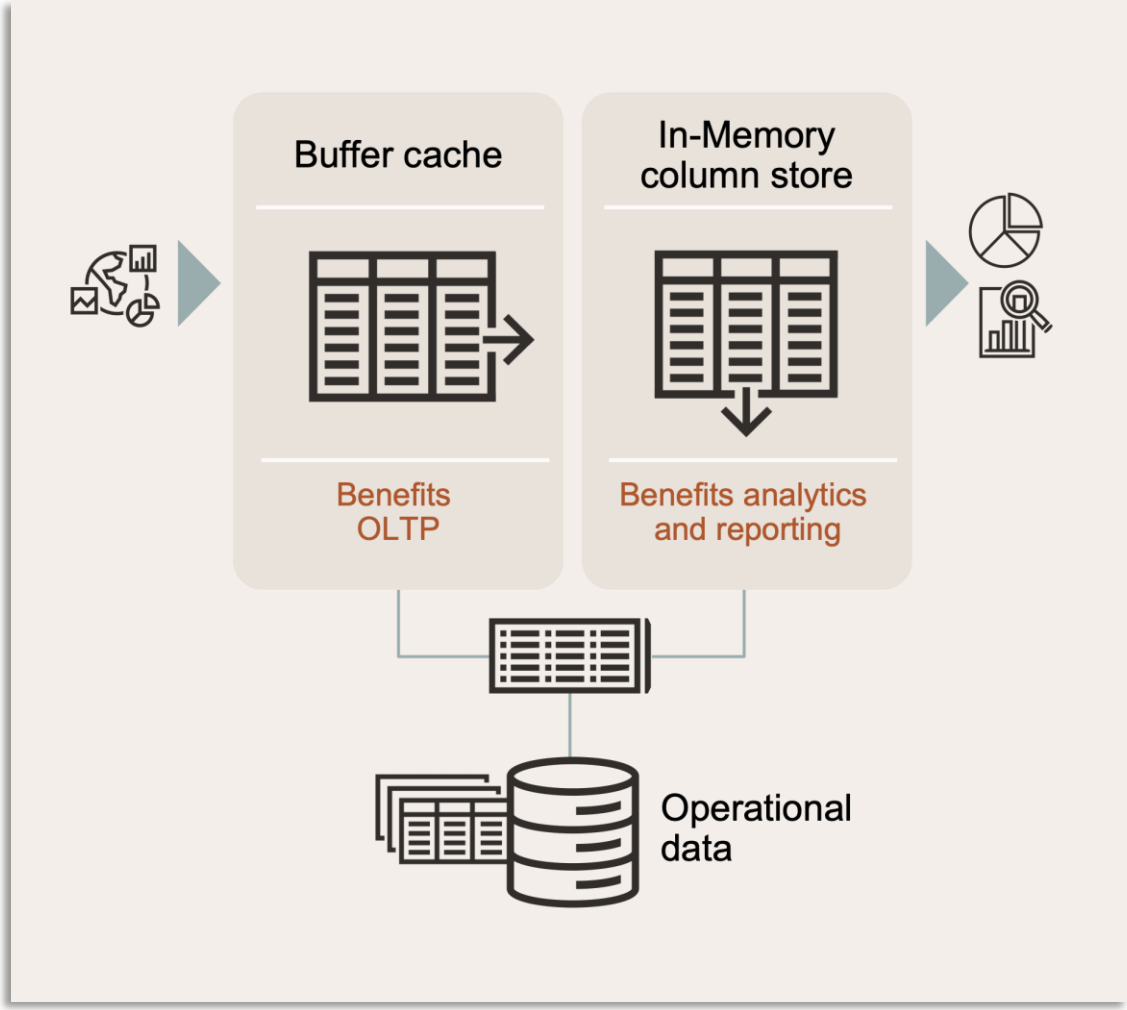
Database In-Memory

Row- and columnar formats for real-time analytics and mixed workloads

23ai True Cache



Database In-Memory: Enables Dual Formatted Data

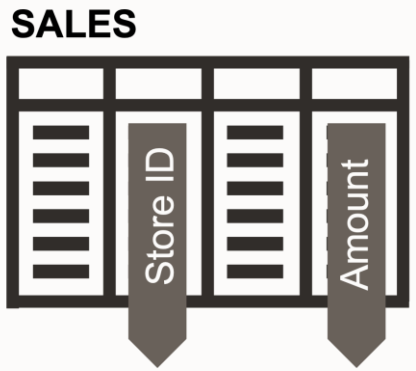


- BOTH row and column formats for same table
- Simultaneously active and transactionally consistent
- Analytics & reporting use new in-memory Column format
- OLTP uses proven row format
- No application changes required

Oracle Database In-Memory Resource Page

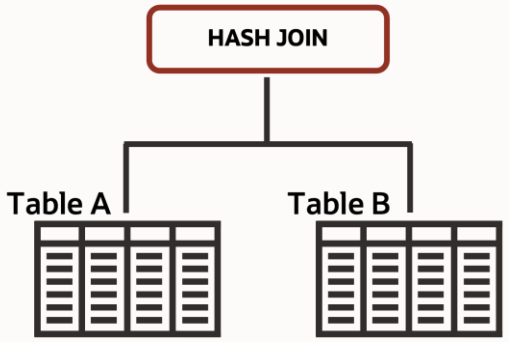
Improves All Aspects of Analytic Queries

Scans



- Speed of memory
- Scan and Filter only the needed Columns
- Vector Instructions

Joins

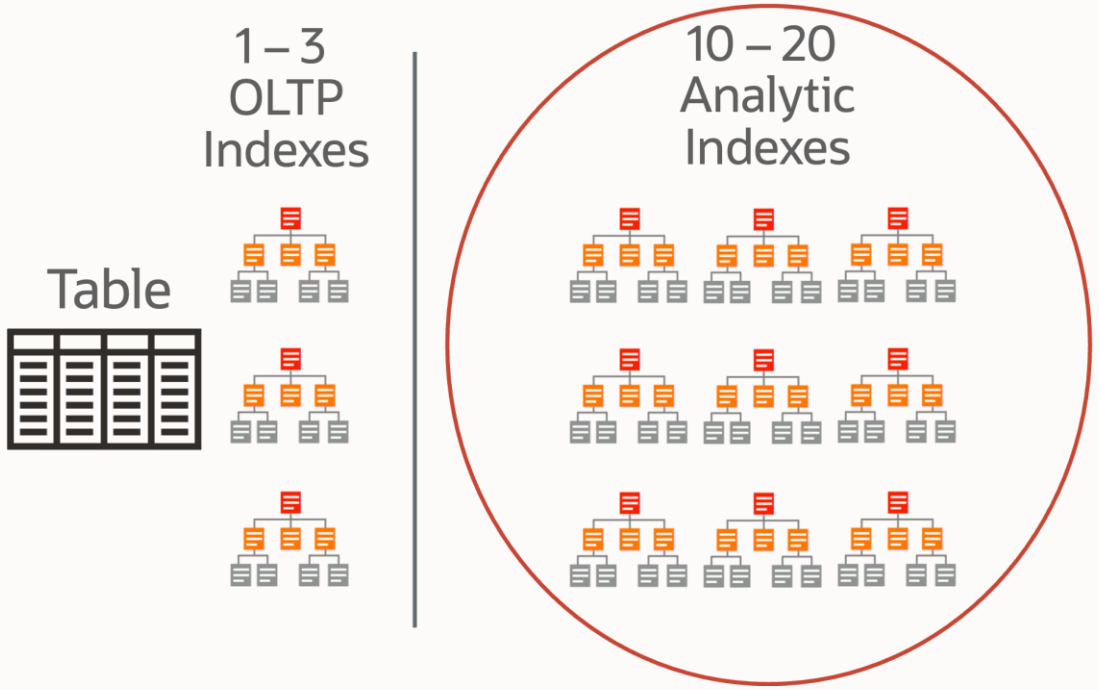


- Convert Star Joins into 10X Faster Column Scans
- Search large table for values that match small table

Reporting

- Create In-Memory Report Outline that is Populated during Fast Scan
- Runs Reports Instantly

Accelerates Mixed Workloads



- Inserting one row into a table requires updating 10-20 analytic indexes: Slow!
- Fast analytics only on indexed columns
- Analytic indexes increase database size

Column Store Replaces Analytic Indexes

- Fast analytics on any column
- Column Store not persistent so updates are: Fast
- No analytic indexes: Reduces database size

Database In-Memory: Simple to Implement and Simple to Test

1. Configure Memory Capacity
   `inmemory_size = XXX GB`

2. Configure tables or partitions to be in memory
   `alter table | partition … inmemory;`

3. Later drop analytic indexes to speed up OLTP

Note: In 19.8, with BASE_LEVEL value for the INMEMORY_FORCE, INMEMORY_SIZE parameter can be set up to a value of 16GB without having to license the Database In-Memory option.

```
SQL> -- In-Memory Column Store query
SQL>
SQL> select  max(lo_ordtotalprice) most_expensive_order From LINEORDER;

MOST_EXPENSIVE_ORDER
--------------------
            57346348

Elapsed: 00:00:00.01

-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |           |       |       |  5401 (100)|          |
|   1 |  SORT AGGREGATE             |           |     1 |     6 |            |          |
|   2 |   TABLE ACCESS INMEMORY FULL| LINEORDER |    59M|   343M|  5401   (16)| 00:00:01 |
-------------------------------------------------------------------------------------------

SQL> -- Buffer Cache query with the column store disabled via NO_INMEMORY hint
SQL>
SQL> select /*+ NO_INMEMORY */ max(lo_ordtotalprice) most_expensive_order From LINEORDER;

MOST_EXPENSIVE_ORDER
--------------------
            57346348

Elapsed: 00:00:08.38

-------------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |       |       |  123K(100)|          |
|   1 |  SORT AGGREGATE    |           |     1 |     6 |           |          |
|   2 |   TABLE ACCESS FULL| LINEORDER |    59M|   343M|   123K  (1)| 00:00:05 |
-------------------------------------------------------------------------------------
```

- See also "New In-Memory Eligibility Test"

Further Readings

Documentation

- Database In-Memory
- Database In-Memory Guide

Blogs and technical briefs

- DBIM Resources
- Oracle Database In-Memory
- When to Use Oracle Database In-Memory
- Oracle Database In-Memory Implementation Guidelines

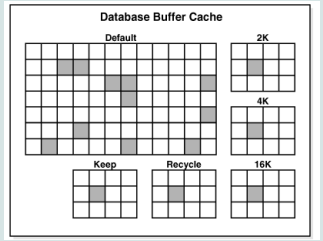
LiveLabs

- Boost Analytics Performance with Oracle Database In-Memory
- Database In-Memory Advanced Features

- **All Links**
  https://github.com/oracle-devrel/technology-engineering/tree/main/data-platform/core-converged-db/db-in-memory

Oracle Database Memory and Caching

Database Buffer



Result Cache

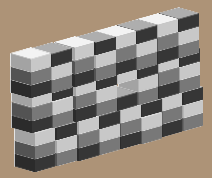
Server Result Cache

SQL Query Result Cache

PL/SQL Functon Result Cache

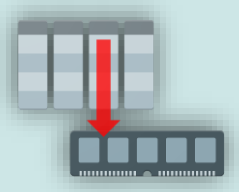
High Speed Data Ingestion MemOptimized RowStore

Database Smart Flash Cache Buffer



Flash Cache Level 2

Database In-Memory



23ai True Cache

In-memory, consistent, and automatically managed, read-only SQL cache

What is Oracle True Cache ?

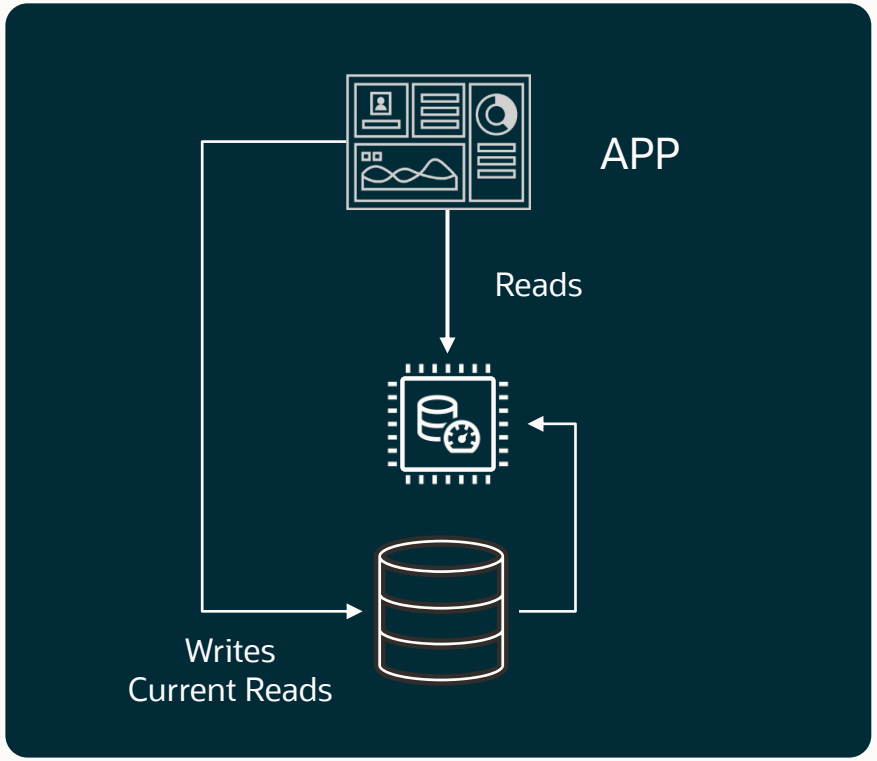
Oracle True Cache is a light-weight (nearly) disk-less Oracle database instance that is deployed as a cache

ANY SQL Query can be easily directed to the cache instead of the back-end database

- **Unique** transparent full-function data cache

Data requested by queries not found in True Cache is **automatically** retrieved from the back-end database

- Changes to data are **automatically** propagated to True Cache in **real-time**
- True Cache is **consistent** as of a point in time

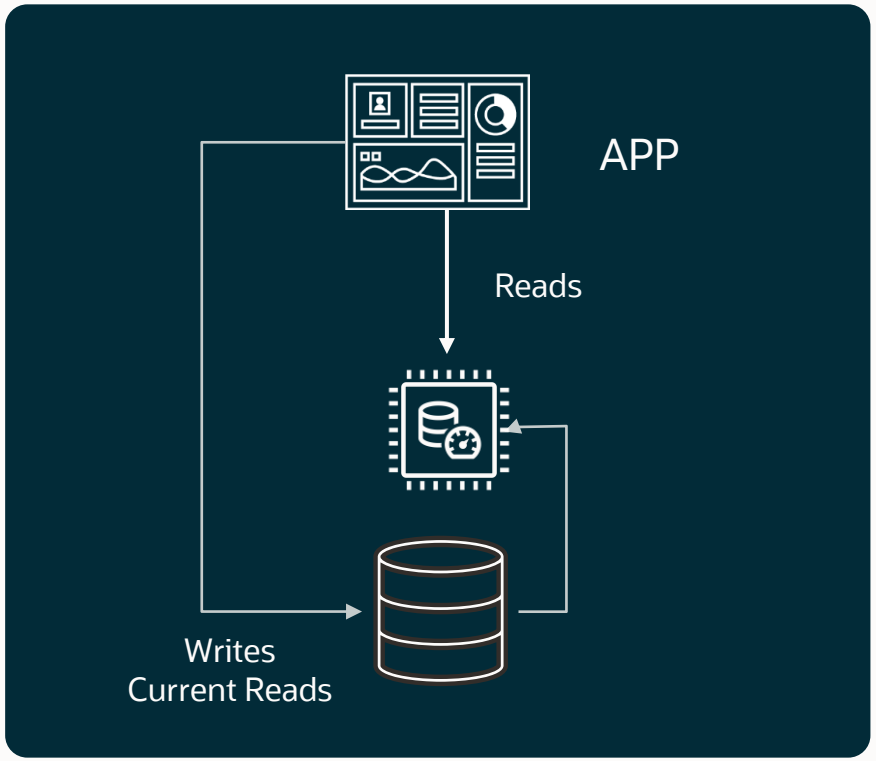


Further readings and links

How To Use True Cache

The application can query data from True Cache in the following two modes:

1. The application maintains two connections:
   - **A read-only connection to True Cache** and **a read-write connection to the database** and uses the read-only connection for offloading queries to True Cache
2. The application maintains one connection.
   - The JDBC driver maintains two connections internally and does the read-write split under application control
   - The application uses setReadOnly() calls to mark some sections of code as “read-only”
   - This is an existing JDBC API which MySQL already uses for similar purposes
   - JDBC driver will send read-only queries to True Cache and writes and DDLs to the primary



True Cache Use Cases

True Cache can be deployed as a:

40 Ora

Further Readings

Documentation

- True Cache on oracle.com
- True Cache FAQ
- Oracle True Cache User's Guide
- Java Support for True Cache
- Oracle True Cache Technical Architecture

Blogs

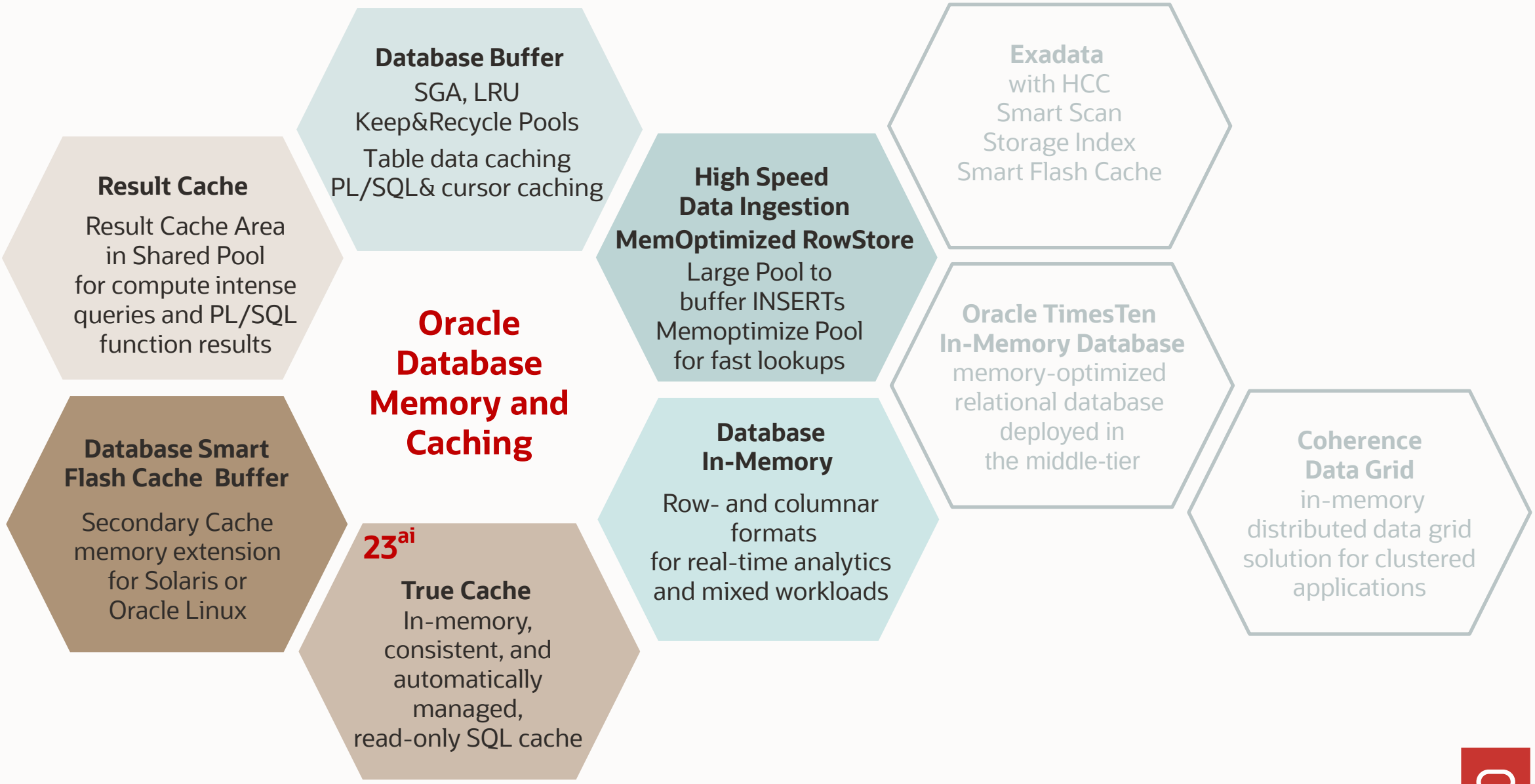
- Introducing Oracle True Cache: In-memory, consistent, and automatically managed SQL cache (Oracle Database 23ai)
- Getting started with True Cache in Oracle Database 23ai FREE (Blog)
- Oracle True Cache (YouTube)

LiveLabs

- Improve application performance with True Cache

All Links

https://github.com/oracle-devrel/technology-engineering/tree/main/data-platform/core-converged-db/true-cache


Database Buffer
SGA, LRU
Keep&Recycle Pools
Table data caching
PL/SQL& cursor caching
Result Cache
Result Cache Area
in Shared Pool
for compute intense
queries and PL/SQL
function results
Oracle
Database
Memory and
Caching
High Speed
Data Ingestion
MemOptimized RowStore
Large Pool to
buffer INSERTs
Memoptimize Pool
for fast lookups
Exadata
with HCC
Smart Scan
Storage Index
Smart Flash Cache
Oracle TimesTen
In-Memory Database
memory-optimized
relational database
deployed in
the middle-tier
Coherence
Data Grid
in-memory
distributed data grid
solution for clustered
applications
Database
In-Memory
Row- and columnar
formats
for real-time analytics
and mixed workloads
Database Smart
Flash Cache Buffer
Secondary Cache
memory extension
for Solaris or
Oracle Linux
23ai
True Cache
In-memory,
consistent, and
automatically
managed,
read-only SQL cache

Videos, Blogs, GitHub from Oracle EMEA Experts and Specialists

YouTube: 23ai Playlist OraTech

GitHub: converged database

Blogs: blogs.oracle.com/coretec

Thank you