ORACLE

When to Use Oracle Database In-Memory

Identifying Use Cases for Application Acceleration

March, 2022, Version 2.1

Purpose statement

This document provides an overview of Database In-Memory, enumerates high-level use cases, and explains the scenarios under which it provides a performance benefit. It is intended solely to give you some general guidelines so that you can determine whether your use case is a good match for this exciting new technology.

Disclaimer

Table of contents

Executive Overview

Oracle Database In-Memory is an unprecedented breakthrough in Oracle database performance, offering incredible performance gains for a wide range of workloads. Oracle Database In-Memory can provide orders of magnitude performance improvements for analytics workloads, as well as substantial improvements for mixed-workload Enterprise OLTP applications. This document briefly introduces Database In-Memory, enumerates high-level use cases, and explains the scenarios under which it provides a performance benefit. The purpose of this document is to give you some general guidelines so that you can determine whether your use case is a good match for this exciting new technology.

Introducing Database In-Memory

Database In-Memory features a highly optimized *In-Memory Column Store* (IM column store) maintained alongside the existing buffer cache as depicted below in Figure 1. The ***primary purpose*** of the IM column store is to accelerate column-oriented data accesses made by ***analytic operations***. It is similar in spirit to having a conventional index (for analytics) on every column in a table. However, it is much more lightweight than a conventional index, requiring no logging, or any writes to the database. Just as the performance benefit to an application from conventional indexes depends on the amount of time the application spends accessing data in the tables that are indexed, the benefit from the IM column store also depends on the amount of time the application spends on data access for analytic operations. It is therefore important to understand the basic characteristics of your application to determine the potential benefits from Database In-Memory.

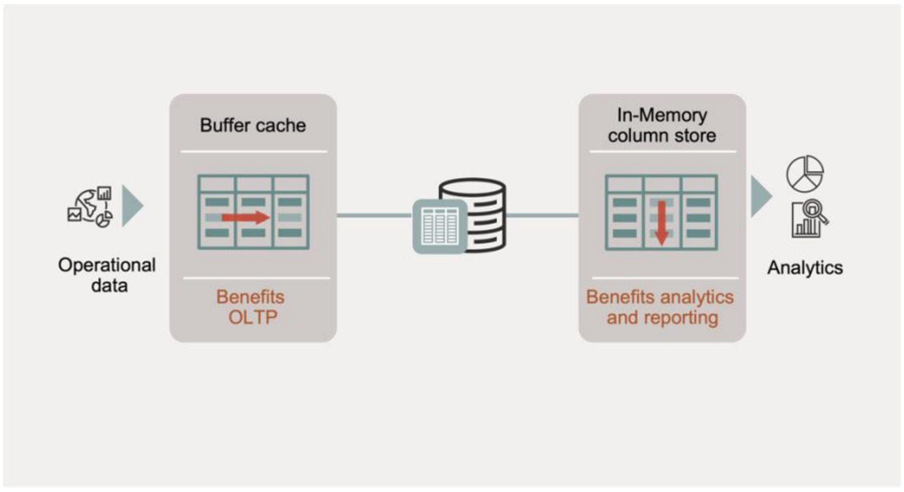

Figure 1: Dual format in-memory database with new in-memory column store

How Does Database In-Memory Improve Performance

The IM column store includes several optimizations for accelerated query processing. These are described in detail in the Database In-Memory Technical Brief so only a brief overview is provided here.

There are four basic architectural elements of the column store that enable orders of magnitude faster analytic query processing:

1. **Compressed columnar storage**: Storing data contiguously in compressed column units allows an analytic query to scan only data within the required columns, instead of having to skip past unneeded data in other columns as would be needed for a row major format. Columnar storage therefore allows a query to perform highly efficient sequential memory references while compression allows the query to optimize its use of the available system (processor to memory) bandwidth.

2. **Vector Processing**: In addition to being able to process data sequentially, column organized storage also enables the use of *vector processing*. Modern CPUs feature highly parallel instructions known as *vector instructions*. These instructions can process multiple values in one instruction – for instance, they allow multiple values to be compared with a given value (e.g. find sales with State = "California") in one instruction. Vector processing of compressed columnar data further multiplies the native scan speed obtained via columnar storage resulting in scan speeds exceeding tens of billions of rows per second, per CPU core.

3. **In-Memory Storage Indexes**: The IM column store for a given table is divided into units known as *In-Memory Compression Units* (IMCUs) that typically represent a large number of rows (typically around a half million rows). Each IMCU automatically records the min and max values for the data within each column in the IMCU, as well as other summary information regarding the data. This metadata serves as an *In-Memory Storage Index*: For instance, it allows an entire IMCU to be skipped during a scan when it is known from the scan predicates that no matching value will be found within the IMCU.

4. **In-Memory Optimized Joins and Reporting**: As a result of massive increases in scan speeds, the Bloom Filter operator (introduced earlier in Oracle Database 10g) can be commonly selected by the optimizer. With the Bloom Filter optimization, the scan of the outer (dimension) table generates a compact Bloom filter which can then be used to greatly reduce the amount of data processed by the join from the scan of the inner (fact) table. And with In-Memory, Bloom filter evaluation can run an order of magnitude faster thanks to vector processing. Similarly, an optimization known as *Vector Group By* can be used to reduce a complex aggregation query on a typical star schema into a series of filtered scans against the dimension and fact tables.

5. **Unique Features on Exadata**: Database In-Memory on Exadata provides the ability to automatically enable data to be encoded in the Smart Flash Cache of the Storage Servers using the same in-memory columnar formats as those used in the database tier. Exadata also enables the ability to duplicate objects in the Database In-Memory Column Store, a feature known as In-Memory Fault Tolerance. In the event of a node failure, duplicated objects will still be available in a surviving node's column store thus preserving analytic query performance. Further, when either the primary or the Active Data Guard standby database is on an Exadata Database machine, Database In-Memory can be used to further accelerate reporting queries on the Active Data Guard standby. Together, these three unique to Exadata Database In-Memory features, along with the ability to exploit Exadata's unique hardware features, represents the best platform for running Database In-Memory.

Apart from accelerating queries, Database In-Memory has the ability to speed up DML operations or writes to the database with the **ability to replace analytic Indexes**: Since the IM column store enables superfast analytics, it is possible to drop conventional indexes used only to accelerate analytic queries. Avoiding costly index maintenance allows update/insert/delete operations to be an order of magnitude faster. As stated earlier, the IM column store is a purely in-memory structure, and maintaining it has very low overhead.

The following table summarizes the key application design principles for maximizing the benefits of Database In-Memory. None of these principles are new, but they are even more important to follow when using Database In-Memory because of the incredible speedup it provides for analytic data access.

Table 1: General Guidelines for Maximizing the Benefits of Database In-Memory

Rule 1	Process Data in the Database, not in the Application	Instead of reading rows out of the database into the application in order to compute a metric such as a total or an average, it is far more efficient to push that computation down into the database. This is especially true with Database In-Memory, since the benefits of processing within the database are much higher.
Rule 2	Process Data in Sets, not Row by Row	This rule is generally applicable to any analytics workload since the costs of database entry and exit are amortized by the number of rows processed. Since the Database In-Memory Option can process billions of rows per second, it is important to give the database enough data to process on each invocation.
Rule 3	Use Representative Optimizer statistics	Plan differences can make a huge difference to the performance of a query especially when the in-memory access paths can provide orders of magnitude faster performance. To ensure that you have optimal plans, follow Oracle's recommended best practices for gathering a representative set of statistics.
Rule 4	When Possible, Use Parallel SQL	With Database In-Memory, IO bottlenecks are alleviated and CPU time dominates the overall execution profile. Parallelism is essential to maximize performance, using all available CPU cores for In-Memory processing. This is especially true in an Oracle Real Application Cluster environment, where Auto DOP is needed to fully utilize all available CPU cores across the cluster.

High-Level Use Cases for Database In-Memory

As depicted in Figure 2 below, Database In-Memory can be used both within Enterprise OLTP systems and within Data Warehouses for real time analytics.

Data Warehouse Systems

For Data Warehouses, Database In-Memory can significantly improve the performance of analytics and reporting on data that can be accommodated within the IM column store, such as on table partitions representing relatively near-term data.

- Tables within the Foundation layer and within the Access layer can leverage Database In-Memory. Due to the massive performance gains for queries on in-memory tables within the Foundation layer, it may be possible to eliminate indexes and other summary objects such as (pre-computed cubes) from the Access layer.
- Database In-Memory is particularly applicable to data marts. Pre-computed summaries and aggregates (such as Key Performance Indicators), can usually be stored easily within memory.

Note: Database In-Memory is not useful for the ETL or Staging layer where data tends to be written and read only once.

Enterprise OLTP Systems

Enterprise OLTP systems (those running packaged ERP, CRM, HCM applications such as Siebel, Peoplesoft, JD Edwards, etc.) typically include a mixture of both OLTP transactions and periodic analytic reporting. In these systems, Database In-Memory can be used for real-time reporting against the base OLTP data. As stated earlier, the IM column store can potentially replace analytic indexes in these systems with significant speedups for OLTP DML operations.

Removing analytic indexes results in many system-wide benefits, e.g. it reduces the total size of the database resulting in reduced storage requirements and faster backups. Analytic index removal also improves buffer cache hit rates, reduces overall redo and undo generation rates, and reduces the total I/O to the database.

Note: For specialized pure OLTP systems such as real-time trading or real-time telecommunications applications that do not have an Analytics component, there is no benefit from Database In-Memory. For such systems, the Oracle TimesTen In-Memory Database may be a better choice if the data can be stored in-memory.

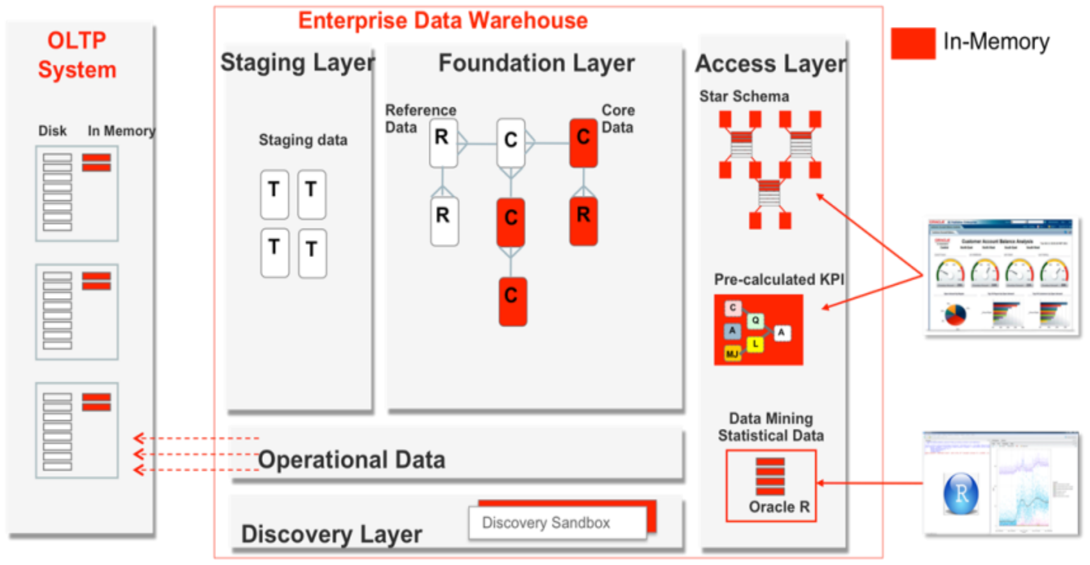

Figure 2: Use cases for Database In-Memory within the Enterprise

Understanding Your Application

Once there is a high-level match between Database In-Memory and your use case, it is important to understand where your application bottleneck is (i.e. where it spends the majority of its time) in order for you to be able to estimate overall benefits from Database In-Memory. The abstract pie chart shown in Figure 3 below depicts a typical application time profile.

Areas that Benefit from Database In-Memory

As described above, Database In-Memory provides optimizations for dramatically faster Analytic queries. Therefore the following workload time components potentially benefit from Database In-Memory (marked with a (*) in the pie chart):

1. *Data Access for Analytics and Reporting*: This is the core value proposition of Database In-Memory, to enable orders of magnitude faster analytic data access.
2. *Analytic Index Maintenance*: Database In-Memory often enables analytic indexes to be dropped, and eliminating the maintenance of these indexes improves overall application performance.

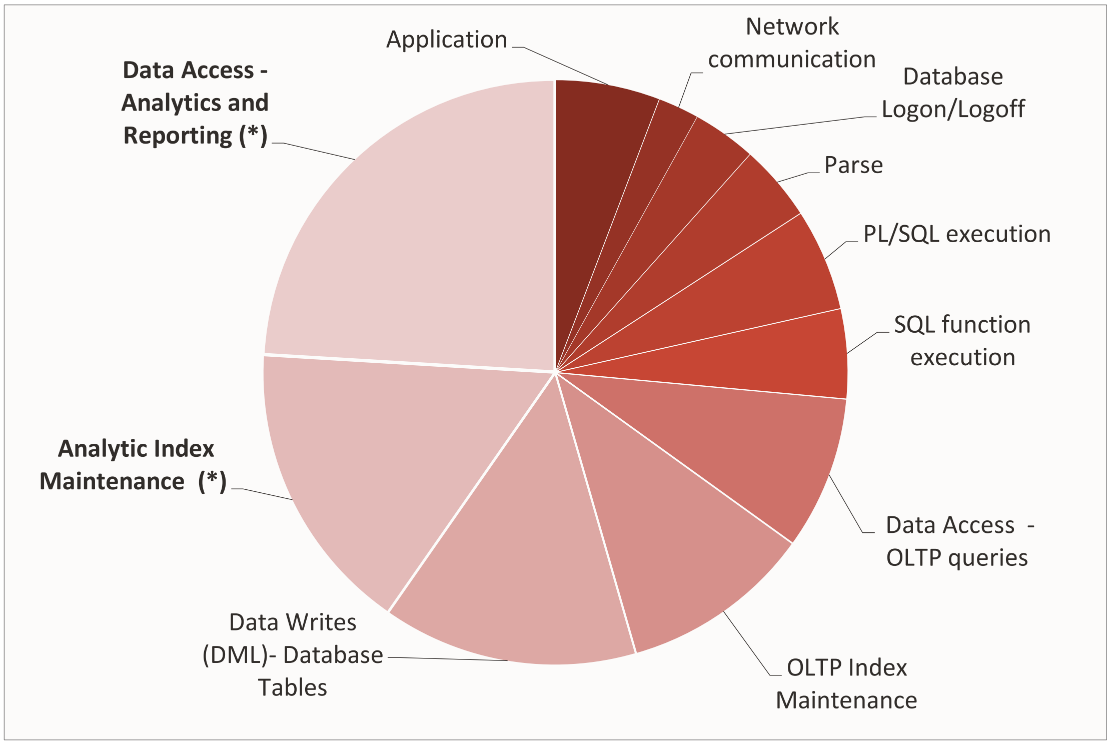

Figure 3: Abstract Time Profile for a Typical Application

Areas that do not benefit from Database In-Memory

As the pie chart shows, there are a number of other workload components that do not benefit since they are unrelated to analytic data access or indeed, to any aspect of SQL execution. The same Oracle Database best practices that have been in vogue prior to the existence of Database In-Memory still apply when minimizing the time spent in these areas.

1. *Application Time*: Time within the application is unaffected by optimizations within the database. Application–specific optimizations, including pushing more work into the database (as stated earlier in Table 1), are required to minimize this time.
2. *Network communication between client and database*: This is not affected by how fast the database runs. Standard techniques for reducing this time include using batched or array execution when possible to amortize the cost of database round-trips.
3. *Logon and Logoff*: Database connect / disconnect can be expensive, including the cost of authentication, process creation / teardown, etc. This time is completely unrelated to data processing. Standard techniques for minimizing this include keeping connections open and reusing connections when possible (e.g. connection pools).

4. *Parse Time*: While Database In-Memory can make the execution of SQL statements much faster, it has no impact on the time required to parse and optimize a SQL statement. Parse time should be minimized through common best practices by using bind variables, keeping cursors open, using session cached cursors, etc.

The remaining areas that do not benefit from Database In-Memory are more fundamental to the application, and it is less likely that they can be addressed without some application redesign:

5. *PL/SQL and SQL function execution*: If the application spends most of its time within PL/SQL procedures, functions, or within built-in or user-defined SQL functions, then the bottleneck is in computation, and not in analytic data access. While Database In-Memory does accelerate the evaluation of many query predicate expressions as well as many aggregation functions such as MIN(), MAX(), SUM(), etc., in general, a heavily compute-intensive application will benefit less from Database In-Memory.
6. *Data access by OLTP queries*: OLTP queries, characterized by highly selective lookups (e.g. by Primary Key) or simple primary-foreign key joins, will not benefit from Database In-Memory. If a workload is dominated by this type of data access, the Oracle TimesTen In-Memory Database may be a better match (if the data being accessed can fit in memory).
7. *OLTP index maintenance*: Likewise, the indexes that are present in order to purely accelerate OLTP queries (such as Primary / Foreign key indexes) or those required for referential integrity, are still necessary even if Database In-Memory is used. Thus the time spent in maintaining these indexes remains unchanged with Database In-Memory. Again, Oracle TimesTen may be a match for use cases dominated by this type of index access (if the data being accessed can be accommodated in memory).
8. *Writes to Database Tables*: The time spent on update/insert/delete DMLs against application tables will remain unchanged whether the tables are in memory or not. An application that is extremely write-intensive, and bottlenecked on DML, will be unlikely to benefit from Database In-Memory.

What Types of Queries Benefit from Database In-Memory

So far we have explained some of the high-level use cases for Database In-Memory, and shown an abstract breakdown of application execution time to show what areas can benefit from Database In-Memory. It is also necessary to understand what type of analytic queries benefit most from the IM column store since not all queries will benefit equally.

As a general rule of thumb: the greater the ratio of the total data accessed by a query to the data actually processed by the query, the greater the potential benefit from Database In-Memory.

In Table 2, below, we enumerate various query properties that impact how much a query can benefit from Database In-Memory. For each property, we show two queries that vary only in terms of the specified query property – showing one query for which Database In-Memory provides a smaller benefit, and one for which it provides a larger benefit. This is not intended to be a comprehensive list, rather to serve as a rough guide to help you understand what to look for when estimating the benefits from in-memory execution.

For this we assume a simple star schema representing sales by an online marketplace: A single SALES fact table and various dimension tables such as STORES, PRODUCTS, SHIPMENTS, CUSTOMERS, etc.

Table 2: Typical Query Properties and how they impact the benefit from Database In-Memory

Query Property	Description	Example Queries	
		Less Benefit	More Benefit
Number of columns selected	As more table columns are selected by a query, column-stitching costs start to dominate, reducing the benefit.	`SELECT * FROM Sales;`	`SELECT revenue FROM Sales;`
		Less benefit since the query selects all columns	*More benefit since the query selects only 1 column*

Number of values returned	The greater the number of values returned by a query, the smaller the IM benefit, because returning data back to the client will dominate the costs.	`SELECT revenue`
		`FROM Sales;`
			`SELECT SUM(Revenue)`
			`FROM Sales;`

		Less benefit since the query returns a value for each row in the table back to the application	*More benefit since the query returns only one value even though it scans all rows in the table*

Selectivity of column predicates	A more selective column predicate enables more filtering on the scan results, and reduces the amount of data that intermediate query plan nodes need to process. Selective predicates can also leverage in-memory storage indexes.	`SELECT MEDIAN(revenue)`
		`FROM Sales`
		`WHERE revenue > 2;`
			`SELECT MEDIAN(revenue) FROM`
			`Sales`
			`WHERE revenue < 2;`

		Less benefit since most rows will qualify (most sales are for items priced higher than $2) and a larger fraction of the query execution time will be spent in calculating the median.	*More benefit since fewer rows will qualify and a smaller fraction of the query execution time will be spent in calculating the median.*

Selectivity of Join Conditions	Similar to the above property, a more selective join condition will result in less data being processed by the join. The Bloom filter optimization, if it is used, will greatly improve performance.	`SELECT S1.id, S.revenue,`
		`P.id`
		`FROM Sales S,`
		`     Products P`
		`WHERE S.prod_id=P.id;`
			`SELECT S.id, S.revenue,`
			`P.id`
			`FROM Sales S,`
			`     Products P`
			`WHERE S.prod_id=P.id`
			`AND P.type='Footwear';`

		The above join will return a row for all sales records since each sale has a matching product. As a result the query will spend more of its time processing the join result.	*This join will only return rows that correspond to sales of footwear products, a subset of total sales. The above query can also potentially leverage the Bloom filter optimization that will build a compact bit vector of footwear product ids and apply it during the scan of the Sales table.*

Number of tables being joined	The greater the number of tables in a join, the larger the percentage of time spent in the join processing.	`SELECT <select list>`
		`FROM Sales, Products,`
		`Customers, Shipments,`
		`Stores, Suppliers,`
		`Warehouses`
		`WHERE <Join Condition>`
			`SELECT <select list>`
			`FROM Sales, Products,`
			`Customers`
			`WHERE <join condition>`

		The above query involves a join between 7 tables and will spend more time in join processing as a fraction of the total execution time and will benefit less with the tables in-memory.	*The above query involves a join between 3 tables and will spend less time in join processing as a fraction of total execution time, and will benefit more with the tables in-memory.*

Complexity of SQL functions	Queries involving computationally expensive SQL functions will benefit less from in-memory.	`SELECT I.id, sum(revenue)`
		`FROM Sales S, Items I`
		`WHERE S.item_id=I.id`
		`AND MyMatch(I.name,"LED`
		`TV")=1`
		`GROUP BY I.id;`
			`SELECT I.id, sum(revenue)`
			`FROM Sales S, Items I`
			`WHERE S.item_id=I.id`
			`AND I.name LIKE "%LED%TV"`
			`GROUP BY I.id;`

		The above query will spend more of in predicate evaluation using the user defined function MyMatch( ) and benefit less from Database In-Memory.	*The query will spend less time in predicate processing by applying the built-in string match LIKE operator to each item and benefit more from Database In-Memory.*

THE FACTS ABOUT ORACLE DATABASE IN-MEMORY

Powering the Real-Time Enterprise

Transparently Speed Up Analytics by Orders of Magnitude	Oracle Database In-Memory transparently extends industry-leading Oracle Database 12c with columnar in-memory technology. Users get immediate answers to business questions that previously took hours because highly optimized in-memory column formats and SIMD vector processing enable analytics to run at a rate of billions of rows per second per CPU core.
Unique Architecture Runs Analytics in Real-Time while Accelerating Mixed Workload OLTP	Column format is optimal for analytics while row format is optimal for OLTP. Oracle Database In-Memory uses both formats simultaneously to allow real-time analytics on both Data Warehouses and OLTP databases. Indexes previously required for analytics can be dropped, accelerating mixed-workload OLTP.
Compatible with All Existing Applications	Deploying Oracle Database In-Memory with any existing Oracle Database-compatible application is as easy as flipping a switch, no application changes are required. All of Oracle's extensive features, data types, and APIs continue to work transparently.
Industry-Leading Scale-Up	Oracle's highly mature scale-up technologies enable application transparent In-Memory scale-up on SMP computers with up to tens of terabytes of memory and thousands of CPU threads. Data is analyzed at the enormous rate of hundreds of billions of rows per second with outstanding efficiency and no feature limitations.
Industry-Leading Scale-Out	Oracle's highly mature scale-out technologies enable application transparent In-Memory scale-out across large clusters of computers with 100s of terabytes of memory and thousands of CPU threads. Data is analyzed at the enormous rate of trillions of rows per second with no feature limitations.
Industry-Leading High Availability and Security	Oracle's renowned Availability and Security technologies all work transparently with Oracle Database In-Memory ensuring extreme safety for mission critical applications. On Oracle Engineered Systems, In-Memory fault tolerance duplicates in-memory data across nodes enabling queries to instantly use an in-memory copy of data if a node fails.
Cost Effective for Even the Largest Database	Oracle Database In Memory does not mandate that all data must fit in memory. Frequently accessed data can be kept In-Memory while less active data is kept on much lower cost flash and disk.
Powering the Real-Time Enterprise	The ability to easily perform real-time data analysis together with real-time transaction processing on all existing applications enables organizations to transform into Real-Time Enterprises that quickly make data-driven decisions, respond instantly to customer demands, and continuously optimize all key processes.

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle