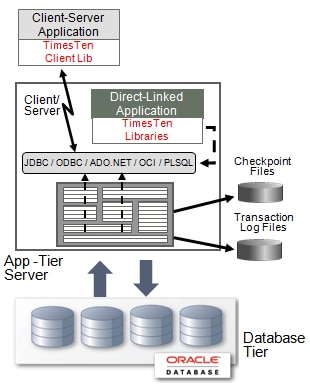
Oracle In-Memory Database Cache Overview

Oracle In-Memory Database Cache (IMDB Cache) is an Oracle Database product option ideal for **caching a performance-critical subset of an Oracle database in the application tier** for improved response time. Applications perform read/write operations on the cache tables using SQL and PL/SQL with automatic persistence, transactional consistency, and data synchronization with the Oracle database. ([Product Data Sheet](http://www.oracle.com/technetwork/database/database-technologies/timesten/overview/ds-imdb-cache-1470955.pdf))

For many enterprise applications, the majority of data in the corporate databases is historical and infrequently accessed. However, buried within this data are pockets of information that must be instantly accessible. For example, current active customers/users, open orders, recent transactions, product catalogs, etc.; caching these data in memory can yield significant improvement for application response time.



Oracle In-Memory Database Cache is built using [Oracle TimesTen In-Memory Database](http://www.oracle.com/technetwork/database/database-technologies/timesten/overview/ds-timesten-imdb-129255.pdf)(TimesTen) and is deployed in the application tier for multi-user and multi-threaded applications. Applications connect to the cache database and access the cached tables using standard SQL via JDBC, ODBC, ADO.NET, Oracle Call Interface (OCI), Pro\*C/C++, and Oracle PL/SQL programming interfaces. Cached tables operate like regular relational tables inside the TimesTen database and are persistent and recoverable.

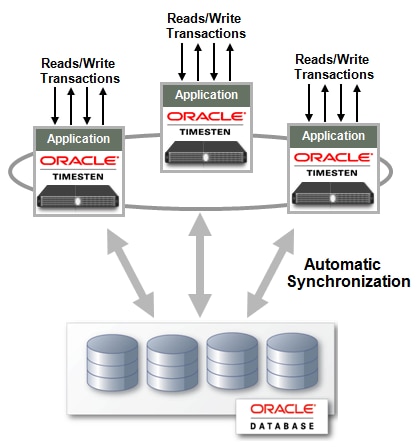
Applications using IMDB Cache may choose to configure a combination of **caching options:**

* Read-only caches - transactions are performed in the Oracle Database and the changes are refreshed to the TimesTen cache database.
* Read-write (or write-through) caches – transactions are performed in the TimesTen cache database and then propagated to the Oracle Database.
* On-demand and preloaded cached - data may be loaded on-demand or preloaded, and may be shared across the cache grid members, or reside only in a specific cache node.

**Data synchronization** with the Oracle Database is performed automatically.

* **Asynchronous write-through** cache leverages the speed of TimesTen by first committing the transactions locally in the cache database, and asynchronously sending the updates to the Oracle Database. Asynchronous write-through cache groups provide faster application response time and higher transaction throughput.
* **Synchronous write-through** cache will ensure that if the Oracle Database cannot accept the update(s), the transaction is rolled back from the cache database; with synchronous write-through, the application must wait for the commits to complete in both the Oracle Database and the TimesTen database.
* For **read-only caches**, incremental updates in the Oracle Database are asynchronously refreshed to the in-memory cache tables in the application-tier at user-specified intervals.

IMDB Cache is designed to continue running even after the Oracle Database server or network connection has been lost. Committed transactions to the cache database are tracked and persisted; and once the connection to the Oracle Database is restored, the transactions are propagated to the Oracle Database. Similarly, committed transactions on the source tables in the Oracle Database are tracked and refreshed to the TimesTen database once connection between the databases is re-established.



IMDB Cache **provides horizontal scalability** in performance and capacity through the in-memory cache grid, which consists of a collection of IMDB Caches for an application’s cached data. Cached data is distributed among the grid members and is available to the application with location transparency and transactional consistency. Online addition and removal of cache grid members can be performed without service interruption to the application.

Depending on data access patterns and performance requirements, an application may choose to allocate specific data partitions to some grid members for locality optimizations or make all data available to all grid members for location transparency. The cache grid software manages cache coherency and transactional consistency across the grid members.

Similar to the stand-alone TimesTen databases, IMDB Cache offers built-in mechanisms for transactional replication to provide high availability for the cache databases. Most enterprise applications cannot afford application down time; hence majority of the deployments add IMDB Cache replication for high availability and load balancing.

<http://www.oracle.com/technetwork/products/timesten/overview/timesten-imdb-cache-101293.html>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In-Memory Caching

The in-memory caching system is designed to increase application performance by holding frequently-requested data in memory, reducing the need for database queries to get that data.

The caching system is optimized for use in a clustered installation, where you set up and configure a separate external cache server. In a single-machine installation, the application will use a local cache in the application's server's process, rather than a cache server.

## Parts of the In-Memory Caching System

In a clustered installation, caching system components interoperate with the clustering system to provide fast response to client requests while also ensuring that cached data is available to all nodes in the cluster.

**Application server.** The application manages the relationship between user requests, the near cache, the cache server, and the database.

**Near cache.** Each application server has its own near cache for the data most recently requested from that cluster node. The near cache is the first place the application looks, followed by the cache server, then the database.

**Cache server.** The cache server is installed on a machine separate from application server nodes in the cluster. It's available to all nodes in the cluster (in fact, you can't create a cluster without declaring the address of a cache server).

**Local cache.** The local cache exists mainly for single-machine installations, where a cache server might not be present. Like the near cache, it lives with the application server. The local cache should only be used for single-machine installations or for data that should not be available to other nodes in a cluster. An application server's local cache does not participate in synchronization across the cluster.

**Clustering system.** The clustering system reports near cache changes across the application server nodes. As a result, although data is not fully replicated across nodes, all nodes are aware when the content of their near caches must be updated from the cache server or the database.

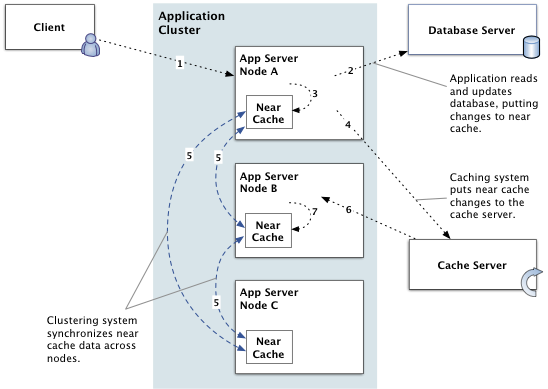
## How In-Memory Caching Works

For typical content retrievals, data is returned from the near cache (if the data has been requested recently from the current application server node), from the cache server (if the data has been recently requested from another node in the cluster), or from the database (if the data is not in a cache).

Data retrieved from the database is placed into a cache so that subsequent retrievals will be faster.

Here's an example of how changes are handled:

1. Client makes a change, such as an update to a user profile. Their change is made through node A of the cluster, probably via a load balancer.
2. The node A application server writes the change to the application database.
3. The node A app server puts the newly changed data into its near cache for fast retrieval later.
4. The node A app server puts the newly changed data to the cache server, where it will be found by other nodes in the cluster.
5. Node A tells the clustering system that the contents of its near cache have changed, passing along a list of the changed cache items. The clustering system collects change reports and regularly sends them in a batch to other nodes in the cluster. Near caches on the other nodes drop any entries corresponding to those in the change list.
6. When the node B app server receives a request for the data that was changed, and which it has removed from its near cache, it looks to the cache server.
7. Node B caches the fresh data in its own near cache.



<https://docs.jivesoftware.com/jive/6.0/community_admin/index.jsp?topic=/com.jivesoftware.help.sbs.online_6.0/admin/CachingOverview.html>