In-Memory Databases

8

What the reader will learn:

- The origins of in-memory databases
- The advantages and disadvantages of in-memory databases
- Different implementations of in-memory databases
- The type of applications suited to in-memory databases
- The use of personal computers with in-memory databases

8.1 Introduction

Disk based database technology has influenced database design since the inception of electronic databases. One of the issues with disk based media is that the physical design of systems tries to speed up processing by reducing disk access, in other words disk input/output (I/O) is a limiting factor which needs to be optimised. In-memory databases have been described as a disruptive technology or disruptive tipping point because it provides a significant improvement in performance and use of system resources.

In-memory databases systems are database management systems where the data is stored entirely in main memory. There are several competing technologies that implement this. For example, Oracle's TimesTen system is effectively a relational system loaded into memory. Another other big player in the field is SAP with its HANA database which offers column-based storage. In contrast, Starcounter is a OLTP (On Line Transaction Processing) transaction database using its own proprietary object oriented data manipulation language based around NewSQL. A comparison between these technologies will be made later in the chapter.

8.2 Origins

In early computers, memory was always the most expensive hardware component. Until the mid-1970's magnetic core memory was the dominant memory technology. It was made of magnetised rings (cores) that could be magnetised in one of two

P. Lake, P. Crowther, *Concise Guide to Databases*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-1-4471-5601-7_8,

directions by four wires that passed through the centre of them forming a grid. Two wires controlled the polarity and the others were sensors. This allowed binary representation of data. It's one advantage was that it was non-volatile—when the power went off the contents of memory was not lost.

Core memory was however expensive and bulky. It also meant that programs were written in such a way as to optimise memory usage. One way of doing this was to use virtual memory where data was swapped in and out of memory and onto disk storage. This optimised memory usage but degraded overall performance.

From the mid-1970's memory started to get cheaper and faster. For example in 2001 the maximum capacity of memory was 256 megabytes. By 2012 that had risen to 16 gigabytes, a 64 fold increase. Cost on the other hand dropped from 0.2 US dollars a megabyte to just 0.009 US dollars per megabyte. But by far the biggest change was in speed where in 2002 response time using hard disk drives was 5 milliseconds, by 2012 using in-memory technology, speed had increased to 100 nanoseconds, a 50,000 fold increase.

The last hurdle to overcome was the D—the durability of ACID (atomicity, consistency, isolation and durability) issue of memory. ACID is discussed in more detail in Chap. 2. Early semi-conductor memory had been volatile so when power was lost, so was all data stored in memory. NVDIMM (Non-Volatile Dual In-line Memory Module) solved that problem; initially by battery backup (BBUDIMM) then by the use of super capacitors for power backup.

This increase in speed along with a falling cost and non-volatility lead to a greater interest in in-memory databases from the mid-2000's. Preimesberger (2013) coins the term 'data half-life' referring to data being more valuable in real time and diminishing in value over time. Therefore speed of processing giving real time insights which can be used immediately is important. In-memory databases therefore allow real-time online analytical processing (OLAP) analysis on data from online transaction processing (OLTP).

It should be noted that an influential paper by Garcia-Molina and Salem (1992) was already talking about the advantages of in-memory databases and the issues in implementing them. The issues raised in this paper seem to have influenced the direction of in-memory databases from that time.

8.3 Online Transaction Processing Versus Online Analytical Processing

One of the issues with management information systems is the various components are often at odds with each other. Business processing for example has different requirements to analytics. As a result it is common for there to be at least two separate systems. The first, usually based around a relational database was for operational systems with lots of update transactions. The second was based around archived data in a data warehouse. Transactions were added to a data warehouse, often with data from other sources. There were inserts, but no updates of records. The database part was usually associated with online transaction processing (OLTP) while the data warehouse was associated with by online analytical processing (OLAP).

In OLTP The data storage tends to be record or row based. Records are stored in blocks which can be cached in main memory. Sophisticated indexing then gives rapid access to individual records. The system slows if lots of records are required. By contrast, OLAP tends to use a column based approach and often works on data sets stored in a data warehouse. Optimisation, where attributes are converted to integers allows rapid processing, but the data is archival rather than operational. Chapter 4 discusses column based approaches and NoSQL in more detail. Plattner (2009) goes further and suggests data warehouses were a compromise with the flexibility and speed paid for by the additional resources needed for loading data, extracting data and controlling redundancy.

Interestingly OLTP was a prerequisite for developing OLAP, however OLAP provides the data necessary to understand the business and set strategic direction. Plattner argues that although data warehouses allow integration of data from many sources, integration of OLAP and OLTP into a single system has the capability of making both components more valuable for real time decision making.

8.4 Interim Solution—Create a RAM Disk

Placing an entire on disk database into memory in the form of a RAM disk where the database management software is treating memory as if it were a disk drive is at best an interim solution to creating an in-memory database. This will speed up reads and writes, but the systems still 'thinks' the data is going to a disk drive and as such will still operate caching and file input output services even though they are not required. This is because the optimisation strategy of a disk based database is diametrically opposed to an in-memory one. Disk based systems try and reduce the amount of time consuming I/O operations. This is where the primary resource overhead is located with data moved to numerous locations as it is used The trade-off is using memory for cache which in turn uses CPU cycles to maintain. There is also a lot of redundant data held in index structures which allow data to be directly retrieved from the indexes rather than doing a disk I/O. The time saved therefore, is down to eliminating the mechanical latency of the disk drive rather than any improvement in data access strategies.

Relational databases which are specifically designed to run in memory eliminate the multiple data transfers and strip out redundant caching and buffering. This reduces memory consumption and simplifies processing resulting in reduced CPU demands.

An experiment run by McObject, admittedly using their own in-memory database management systems, showed the dramatic differences in speed between a disk based, a RAM based and an in-memory database system:

'We tested an application performing the same tasks with three storage scenarios: using an on-disk DBMS with a hard drive; the same on-disk DBMS with a RAM-disk; and an IMDS (McObject's eXtremeDB). Moving the on-disk database to a RAM drive resulted in nearly 4× improvement in database reads, and more than 3× improvement in writes. But the IMDS (using main memory for storage)

outperformed the RAM-disk database by 4× for reads and 420x for writes.' (Steve Graves, co-founder and CEO of McObject available on line at ODBMS Industry Watch (2012) http://www.odbms.org/blog/2012/03/in-memory-database-systems-interview-with-steve-graves-mcobject/ last accessed 29/06/2013).

8.5 Interim Solution—Solid State Drive (SSD)

In many ways this is another type of RAM disk but the technology is different and there are reliability issues after a period of operation. Removable flash memory drives became available in 1995 and after an initially slow take up rate, these devices have become the main type of removal storage media for personal computing. They are fast, reliable and have a high capacity which means you can store an entire database on them. Their advantage over traditional disk drives is they have no moving parts so the issues of seek and latency times are removed. They are not without limitations as the following section will illustrate.

SSD's have a number of components: a processor, cache and NAND chips. NAND chips are named after NAND logic gates (the other type is NOR) and are random access memory chips (NOR chips are used for read only memory applications). In a flash drives the NAND chips are multi-layer chips (MLC) which give a greater capacity for the same area as a single layer NAND.

One disadvantage of SSDs, specifically the NAND chips is they do wear out. This is because they have individually erasable segments, each of which can be put through a limited number of erase cycles before becoming unreliable. This is usually around 3,000 to 5,000 cycles. To reduce the risk of uneven wear where some segments are used more heavily than others a process known as wear levelling is used. This involves arranging data so that erasures and re-writes are distributed evenly across the whole chip. It also means that everything wears out at the same time.

Georglev (2013) gives a summary of the issues involved in using SSDs compared with traditional HDD technology:

Fragmentation This is still an issues with SSD's despite access to all cells being equally fast. This is because if indexes are fragmented they require more reads than for one which is not. Fragmentation also reduces available disk capacity.

Access Speed There is still a difference between random and sequential reads and writes which also depend on block sizes. They are still faster than HDD's but performance diminishes with writes and particularly updates.

Speed vs. Capacity Fragmentation and wear (segments become unreliable) lower capacity, but speed is still greater than with a HDD. The biggest issue is when does the SSD become unusable. Wear levelling means once one segment becomes unreliable, the rest of the segments will soon become unreliable as well. The time it takes this to happen will depend on the number of update operations being performed—the more operations, the sooner the device will fail.

Caching data is held in cache as long as possible to reduce wear, in other words attempts are made to do as many updates as possible before writing to the NAND. The danger here is reliability may be compromised in the event of a power failure as cache tends to be volatile.

Data Most Suitable for SSDs Updates are the biggest issue for SSDs because when data is updated the entire block where the data resides has to be moved to a new block with the update, then the old block has to be erased. This causes both wear and gives a performance overhead. Given this situation SSD are best for static data where there many reads but few updates.

Data Recovery When a HDD fails there are many ways to extract most of the data depending on the nature of the failure. With an SSD once its dead, its dead—it is almost impossible to recover any data. It should also be noted that an SSD has a shelf life of about seven years. This means if it is not powered up for a significant period of time data loss should be expected.

The bottom line is that SSDs are useful as a backup and data transportation medium (as long as you take into account the data recovery issues). They are not an ideal replacement for disk based databases when lots of transactions involving updates are expected as their life cycle will be curtailed. If you are planning on using this technology you should read the technical specifications relating to the device and ask how many erase cycles can a segment sustain.

8.6 In-Memory Databases—Some Misconceptions

Populating the Database The first misconception is populating a very large in-memory database is a slower process than populating an equivalent database on disk. On disk systems use memory caches to speed up the process, but ultimately the cache becomes full and must be written to disk. This requires moving data from cache to I/O buffers. From there data is physically written to disk and this is relatively slow compared with other processes. As well as writing data to disk indexes mapping the data's storage location are developed. These grow bigger and more complex as more data is added to the system. Also as the size of the database grows the amount of data that can be held in cache memory becomes a smaller and smaller percentage of the total database further reducing performance. Finally as the database grows the physical size on the disk gets bigger resulting in higher seek times. Basically as the size of the database grows there is a continual degradation of performance.

By contrast the performance of an in-memory database remains roughly constant as more and more data is added because none of the on disk issues apply.

Single User, Single System In memory databases are not restricted to being single user. The database can be held in shared memory with the database management system handling concurrent access requests. They can also operate with remote servers

(nodes). In other words in-memory databases can be shared by multiple threads, processes and users.

An In-Memory Database Is the Same as an Embedded Database This is not a total misconception, some in-memory databases are the same as embedded databases. These are databases which are part of an application with the database itself being invisible to the end user. However, as seen with the previous point, in-memory databases can also employ the client server model with the use of shared memory.

8.7 In-Memory Relational Database—The Oracle TimesTen Approach

No one technology appears dominant in the in-memory database sphere. There are however several contrasting approaches. The first to be discussed is implementing what is essentially a relational database in-memory. Oracles TimesTen is an example of this. Simply put TimesTen is a relational database where all data resides in memory at runtime.

In a cached relational database management system a lot of processing power is used to manage memory buffers and control multiple data locations on disk and in memory. The in-memory approach only uses disks for persistence and recovery rather than primary storage. This gets over the D-durability issue of ACID. Changes from committed transactions are logged to disk and periodically a disk image of the database is updated (in Oracle's terms, a checkpoint). The timing of this checkpoint is configurable and many applications favour higher throughput over synchronous logging, in other words the period between checkpoints is maximised to improve performance. The trade-off is that there is a small risk that data could be lost although this is unlikely to be due to power failure if NVDIMM is used and more likely to be to a catastrophic hardware failure. Transactions are also logged asynchronously which means if the period between checkpoints is too long, there is a danger of running out of disk space. When a checkpoint is executed, the logs are cleared (For more information see Oracle 2006).

There are five system's components of TimesTen: Shared: libraries, memory resident data structures, system processes, administrative programs and checkpoint files and log files on disk (Fig. 8.1). The memory requirements for this are given by Oracle as PermSize + TempSize + LogBufMB + 20 MB overhead although discussion boards suggest an overhead of 64 MB is more realistic (see https://forums.oracle.com/thread/2547114 accessed 29/07/2013 for example). PermSize is the size of the permanent data and includes the tables and indexes. The permanent data partition is written to disk during checkpoint operations. TempSize is the size of the temporary data and includes locks, cursors, compiled commands, and other structures needed for command execution and query evaluation. LogBufMB is The size of the internal log buffer in MB and has a default of 64 MB (Oracle 2011).

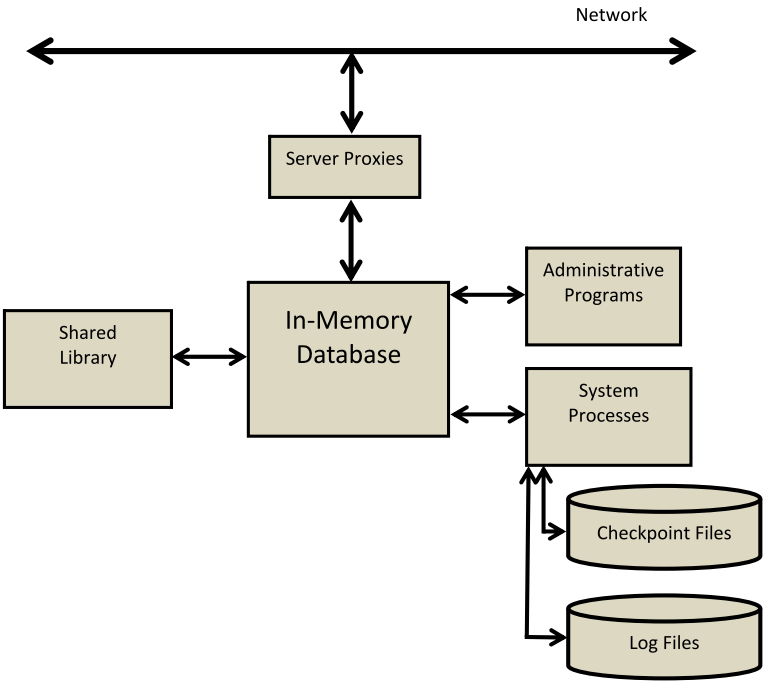

Fig. 8.1 Oracle TimesTen in-memory database

Although TimesTen is a relational database, this refers to the data structure. Traditionally a relational database management system (RDBMS) assumes the data it is managing is on disk at runtime. However with all data in memory the system optimises its management for that environment. One of the main differences is query optimisation.

In a disk based system data might be on disk or cached in main memory at any given moment. The balance is crucial in reducing disk input/output which from a resource point of view is expensive. However this problem goes away if all the data is in memory. The thing to be optimised now is processing and this can be influenced in a number of ways:

Indexing The primary function of an index in a disk based system is to locate a record on disk as quickly as possible based on an index key and record identifier. In an in-memory system index keys do not need to be stored. Indexes are implemented as record pointers which point to the corresponding record containing the key. In effect the record identifier is implemented as the record pointer. Because an index key is not stored in the index there is no duplication of key values in the index structure which reduces it size. Also pointers are all the same size so the need to manage variable length keys goes away making index implementation simpler.

Query Processing A common clause in SQL is ORDER BY. In a disk based system this is fast if the data in the table being targeted by the query is stored in the order requested. Often, in fact usually, this is not the case. The next best option is to use an index scan. This carries the overhead of random access to records which may result in a high input/output cost. The problem is exasperated if sorting is required on multiple columns. In an in-memory database, as seen in the indexing, the only entries in the index are the record identifiers which are implemented as record pointers. In other words index entries point directly to the memory address where data resides, so a random scan is no more expensive than a sequential scan, and of course there is no disk I/O. Buffer pool management is also unnecessary because you don't need to cache data in main memory because you are not doing any disk I/O.

Backup and Recovery as already mentioned the primary means of backing up a TimesTen database is by checkpoints where a snapshot is taken of the permanent data that includes the tables and indexes. These are used with the on disk log files to recreate the database in the event of a failure.

8.8 In-Memory Column Based Storage—The SAP HANA Approach

Like TimesTen, Hana incorporates a full database management system with a standard SQL interface and ACID. It is geared towards existing SAP users in that applications using its proprietary version of SQL (Open SQL) can run on the SAP HANA platform.

SAP HANA exploits parallel multicore processors, so if a native SQL command is received it is optimised to allow parallel execution by partitioning the data into sections. This scales with the number of available core processors. For 'Big Data', this can be partitioned across multiple hosts.

The libraries shown in Fig. 8.2 contain business applications, for example currency conversion and converting business calendars from different countries. These can be used for processing directly in main memory rather than the traditional way for utilising plain SQL.

HANA optimises memory usage by data compression and by adapting the data store for the task. For applications that are processed row by row, records are placed in sequence for the best performance. If the application is calculation intensive executing on a restricted number of columns, then these are aggregated into a column store. Finally, graphical objects are created by specifying minimal database schema information, such as an edge store name, a vertex store name, and a vertex identifier description (where a vertex is a description of a set of attributes describing a point on a graph). The object body is stored in sequence and the graph navigation based on the schema information is stored as another sequence to support unstructured and semi structured data storage and retrieval. For more information on graphical objects in SAP HANA see Rudolf et al. (2013).

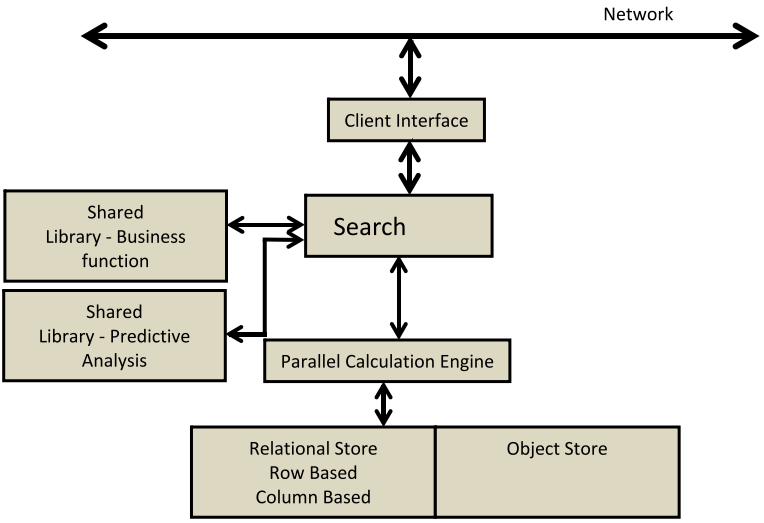

Fig. 8.2 SAP HANA In-memory database

Parallel Execution In SAP HANA sequential processing is avoided. Key to this is aggregation where values such as maximum, minimum, average and mode are required. Aggregation is performed by creating a number of threads that act in parallel. This is done by creating a aggregation function which creates threads that fetch a small partition of the input table which is then processed (for example to find the maximum). The process continues until the whole table is aggregated. Each thread has a hash table where it writes its aggregation results and when the threads are finished the buffered hash tables are merged.

Column Versus Row-Based Storage As mentioned you can specify whether to use column or row storage. Row based storage is recommended when:

- The table has a small number of rows
- Records are processed one at a time
- The complete record is needed
- Column values are mainly distinct, so compression rates would be low
- Aggregations and fast searching are not required

Column based storage is recommended when:

- Calculations are executed on a restricted number of columns
- Only a few columns are used as the basis of searches
- The table has a large number of columns
- There are a large number of records and columnar operations are required
- The majority of columns contain only a few distinct values (compared to the number of rows) which means a higher compression rate is possible.

Row based tables can be joined with column based tables although master data that is often joined with transaction data (the database versus data warehouse issue) is put in a column store.

The advantages and operation of column based databases are discussed in detail in Chap. 5 and will not be repeated here.

SAP claim increased benefits for using in-memory databases:

> "In financial applications, different kinds of totals and balances are typically persisted as materialized aggregates for the different ledgers: general ledger, accounts payable, accounts receivable, cash ledger, material ledger, and so on. With an in-memory column store, these materialized aggregates can be eliminated as all totals and balances can be computed on the fly with high performance from accounting document items." (SAP 2013b)

8.9 In-Memory On-line Transaction Processing—The Starcounter Approach

So far we have looked at two of the biggest players in the in-memory database field both of which have taken different philosophies to their implementation. Here we will look at one of the smaller players which have taken a very different approach to either of them.

Starcounter is an in-memory object oriented database management system which is focussed on OLTP (Online Transaction Processing). It is designed for "*highly transactional large-scale and real-time systems, systems supporting thousands or millions of simultaneous users, such as retail systems, adserve applications, online stores and finance applications*" (http://www.crunchbase.com/product/starcounter-in-memory-database accessed 29/06/2013).

A key feature of Starcounter is it integrates an application run time virtual machine with the database, a product they call VMDBMS. Effectively this allows the database data to reside in a single place in RAM and not get copied back and forth between the application and the database. This has been likened to effectively integrating the application with the database management system where you transfer code to the database image instead of data from the database image to the code. The downside of this is the database needs to understand the code of the application so is not universally language compatible.

Like Oracle's TimesTen product Starcounter only writes logs to disk. Reads only happen if and when there is a need to recover the database. Users can query the database using SQL, however Starcounter have created their own proprietary object-oriented data manipulation language called NewSQL (not to be confused with NoSQL discussed in Chap. 4). This is SQL like (in fact it is claimed to adhere to SQL 92 standard) and is embedded in java code, for example:

```
string query = "SELECT e FROM
Employee e WHERE e.FirstName = ?";
Employee emp = Db.SQL(query, "Warren").First
emp.PrintCV();
```

At the time of writing the biggest user of Starcounter is Gekas, a superstore in Sweden with 100,000 different product lines visited by 4.5 million customers each year (Reuters, May 15, 2013, http://www.reuters.com/article/2013/05/15/starcounter-idUSnBw155295a+100+BSW20130515 accessed 29/06/2013).

8.10 Applications Suited to In-Memory Databases

From the previous discussion it will come as no surprise that systems that need to do OLAP and OLTP simultaneously are those most suited to in-memory database deployment. As an example Savvis, an IT hosting and colocation service provider uses another of Oracles in-memory products, Exalytics. Here a broad range of users in the organisation need to view a variety of complex data sets to extract data for their specific requirements. This ranges from customer service staff who need to know what products a customer has purchased, service requests in progress and satisfaction survey feedback through to senior management who need financial data for strategic planning. The data all groups use has to be current from the operational database. The analytics are provided by a combination of visualisation and filtering of real-time data. The most effective way to provide this without building parallel systems is with an in-memory database. This particular application also allows various scenarios to be tested (so called what-if scenarios) using the operational data.

As a contrasting example Bigpoint GmbH, an on-line game developer uses SAP HANA to analyse the behaviour of players using its Battlestar Galactica online game. They use the in-memory database for analytics for targeted marketing of add-ons to the game and to sell virtual items to players in real time.

If we look at different types of applications, a study by King (2011) found in-memory database usage by organisations were as follows:

- Business Analytics 42 %
- Web-based transactions 38 %
- Reporting 35 %
- Finance (trading, market data, etc.) 32 %
- Billing and Provisioning 25 %
- Embedded/Mobile applications 18 %

Obviously most companies are implementing more than one as the survey asked respondents to 'tick all that apply'. Other applications included supply chain management, sensor data management, geo-spatial applications, network information management, product information management, online advertising and materials management. This is a wide range of applications with the common thread of needing a database capable of processing data in real-time and where a traditional disk based system was not fast enough.

In terms of growth, 77 % of organisations said their data requirements were expected to grow, with only 2 % expecting shrinkage. Likewise 68 % of surveyed organisations were expecting their use of in-memory databases to increase with only 9 % expecting a decrease. The remaining 23 % did not know. It is unfortunate that King did not get the organisations to say which sector they belonged to.

8.11 In Memory Databases and Personal Computers

The problem with in-memory databases for personal computers is not so much the fact it can't be done (it can) but rather should it be done? The applications which have been described so far have been for corporate systems which want to be able

to do analytics on real-time data with multiple simultaneous users. From this standpoint the main usage of a personal computer would be to access an in-memory database located on a server rather than having its own in-memory database.

The place where in-memory database technology is most likely to be found on personal computers is in the form of embedded databases. You will recall from the definition of an embedded database that these are databases you don't see, because they are hidden inside another application managing that applications data. They are typically single application databases that do not share their data with other applications. Some examples are as follows:

Games Systems The technical challenge with persistent games, particular massively multiplayer online (MMO) games is transaction management. Every move and interaction is a transaction. In single user games data corruption can be solved by restarting the session. Persistent games, on the other hand, must react by rolling back incomplete or failed transactions. These manage their view of the games world within a database. Early versions attempted this via a disk based solution, however disk resident databases have difficulties handling the transaction load required in MMO's. Games vendors also exploit the data streams being generated by MMO's as our earlier example of Bigpoint GmbH showed.

Route Delivery Management The database in this application is used to provide delivery drivers with the best route to the next delivery. Basically it is the database behind most satnav systems. As an example MJC2 (see http://www.mjc2.com/real-time-vehicle-scheduling.htm last accessed 29/07/2013) produces planning and scheduling software, part of which is real-time routing and scheduling for logistics. Their dynamic route planner software is able to schedule and reschedule very large distribution and transport operations in real-time, responding automatically to live data feeds from order databases, vehicle tracking systems, telematics and fleet management systems. The system requires an in-memory solution to allow managers to respond dynamically to a continuously changing operations including optimal responses to variable demand, last minute orders, cancellations and redirections.

Gas and Electricity Consumption Reading This is a sector specific application where the database is used to capture meter readings, often remotely. For example, Australian energy retailer, AGL has adopted an in-memory computing to help it cope with the vast amount of data coming from smart meters. Smart meters measure energy usage at regular intervals, for example every 30 minutes, and can tell how much electricity is consumed during on and off-peak periods. Instead of one simple reading every three months the meters now provide continuous data streams generating around 4400 readings per quarter instead of the traditional 1. As well as still generating customer bills, the data streams are analysed to separate out base meter data from other data for use in forecasting. The system used is SAP HANA.

Mobile Customer Resource Management (CRM) Marketing and sales need customer resource management applications without wanting to worry about the database driving it. This often relies on replication of a segment of the master

database. It may also be integrated with a satnav application similar to the route delivery management system described above. SAP have developed their CRM system to be deployed on SAP HANA to provide sales people with real-time customer information throughout the sales cycle on a mobile device.

8.12 Summary

Attaining the fastest speed possible has always been the goal of database applications. Unfortunately optimisation efforts in the past have been targeted at disk based systems which have led to duplicating data. One version of data has been placed in a database which is used for operational OLTP systems and the other is placed in data warehouses for applications associated with OLAP. This is because OLTP is I/O heavy while OLAP is computationally heavy and requires diametrically opposite optimisation routines. In this chapter we have seen that in-memory database applications have the capability of combing both requirements in a single system.

Common misconceptions of slow population of in-memory databases and them only being available for single user systems were discussed and dismissed. However there is no one standard approach to their implementation which ranged from Oracles' TimesTen which was an in memory relational system, SAP HANA which used column based storage through to new players exemplified by Starcounter which used an object oriented approach effectively integrating an application with the database management system.

8.13 Review Questions

The answers to these questions can be found in the text of this chapter.

- What is NVDIMM and why is it important in the context of in-memory databases?
- What are three misconceptions about in-memory databases?
- What are the performance increases an in-memory database can give compared to a on disk database?
- What approach to implementation of an in-memory database has Oracle taken with its TimesTen product?
- How do SAP HANA and Starcounter differ from Oracle TimesTen approach to in-memory database implementation?

8.14 Group Work Research Activities

These activities require you to research beyond the contents of the book and can be tackled individually or as a discussion group.

Activity 1 Wikipedia in its definition of 'embedded databases' seems to have fallen into the trap of considering in-memory databases as being the same thing. Go to http://en.wikipedia.org/wiki/Embedded_database where you will see a list headed 'Comparisons of database storage engines'. Examine the list and determine which are purely embedded (single user) and those which have client/server capabilities.

Activity 2 Look in your wallet or purse and see how many plastic cards you have. What are each of them for? (you may have bank cash cards, phone cards, loyalty cards, membership cards, student card, a driving licence among others). Which of these, when you use them, could be providing data for an OLTP or a OLAP. Make a list of 5 columns headed Card, OLTP, OLAP, Both and Neither and fill it in with an X in the appropriate part of the grid. For those with an X in OLTP, OLAP and both, work out what is happening to your data. For those labelled both—do you think there is a in-memory database behind the processing? (One pointer is do you get targeted advertising before you complete your transaction)

References

Garcia-Molina H, Salem K (1992) Main memory database systems: an overview. IEEE Trans Knowl Data Eng 4(6):509–516

Georglev F (2013) HDDs, SSDs and database considerations. https://www.simple-talk.com/sql/database-administration/hdds,-ssds-and-database-considerations/. Accessed 29/07/2013

King E (2011) The growth and expanding application of in-memory databases. Available at http://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&frm=1&source=web&cd=1&cad=rja&ved=0CDIQFjAA&url=http%3A%2F%2Fwww.loyola.edu%2F~%2Fmedia%2Fdepartment%2Flattanze%2Fdocuments%2FWP0611-107.ashx&ei=mjX2UcehMajI0AXkgIEI&usg=AFQjCNH18VJL5brvz-otsfSIeIhEj1ToTA&sig2=y6EKZgsDtgss7MRHyjqyXw. Last accessed 30/6/2013

Oracle Corp (2006) TimesTen in-memory database recommended programming practices release 6.0, p 28. http://download.oracle.com/otn_hosted_doc/timesten/603/TimesTen-Documentation/goodpractices.pdf. Accessed 24/07/2013

Oracle Corp (2011) Oracle TimesTen in-memory database operations guide release 11.2.1, Part 1 Managing TimesTen databases. http://download.oracle.com/otn_hosted_doc/timesten/1121/doc/timesten.1121/e13065/using.htm#BCGIDAJG. Accessed 29/07/2013

Plattner H (2009) A common database approach for OLTP and OLAP using an in-memory column database. In: Proceedings of SIGMOD 09 (special interest group on management of data), Providence, Rhode Island, USA. Available at www.sigmod09.org/images/sigmod1ktp-plattner.pdf. Last accessed 29/06/2013

Preimesberger C (2013) In-memory databases driving big data efficiency: 10 reasons why. e-week, available at http://www.eweek.com/database/slideshows/in-memory-databases-driving-big-data-efficiency-10-reasons-why/, last accessed 14/11/2013.

Rudolf M, Paradies M, Bornhövd C, Lehner W (2013) The graph story of the SAP HANA database. In: Proceedings of 15th GI-symposium database systems for business, technology and web, Madeburg, Germany. Available at http://www.btw-2013.de/proceedings/The%20Graph%20Story%20of%20the%20SAP%20HANA%20Database.pdf. Last accessed 29/07/2013

SAP (2013b) SAP white paper: SAP HANA® database for next-generation business applications and real-time analytics explore and analyze vast quantities of data from virtually any source at the speed of thought. Available at http://download.sap.com/download.epd?

context=E67AFF9FD4CFC6693FC443EB965A7B4FE599BA88ED6C9F622486D0E5BEB350EB7DAD751393D16A2D916A3C9EA834D1CB2A8138467760590E. Last accessed 29/06/2013

Further Reading

McObject (2013) In-memory database systems—questions and answers. Available at http://www.mcobject.com/in_memory_database. Last accessed 29/06/2013

Oracle Corp (2013) Oracle TimesTen in-memory database and Oracle in-memory database cache. Available at http://www.oracle.com/technetwork/products/timesten/overview/index.html. Last accessed 29/06/2013

SAP (2013a) Consolidate transactional and analytical workloads onto a single, real-time database—with SAP HANA. Available at http://www54.sap.com/pc/tech/in-memory-computing-hana/software/platform/database.html. Last accessed 29/06/2013