Most the major commercial database systems were designed in an era of small memories and uniprocessor machines. The only realistic design option was to store data on disk and bring disk pages into memory as needed for processing. This model served us well for many years but the world has changed. Today, a server with 32 processor cores and 1TB of memory costs as little as $50K. The majority of OLTP databases fit entirely in 1TB and even the largest OLTP databases can keep the active working set in memory. To take full advantage of this class of machines it is not sufficient to merely increase the size of the buffer pool. The architecture of the database system needs to be reconsidered from the ground up.

This issue aims to provide an overview of the current state of the art for database systems that have been designed to exploit large main memories and multi-core processors and are targeted for OLTP workloads. The issue covers seven different database systems, five commercial systems and two research prototypes. The commercial systems, roughly in order of launch date, are Oracle TimesTen, IBM solidDB, VoltDB, SAP HANA, and Microsoft's Hekaton. The two research systems are HyPer from the Technical University of Munich, and Calvin from Yale University. The issue also includes a paper on Bw-tree, a novel latch-free version of B-trees.

The design of any database system must solve a number of fundamental issues. What index structures to use? How to support high levels of concurrency? How to ensure transaction isolation and durability? What to do about long-running transactions? How to ensure high availability? It is fascinating to see the variety of solutions that the designers of the systems described here have come up.

The technique chosen for concurrency control, for example, has a major impact in system performance. Concurrency control adds overhead but also affects the level of concurrency that the system can support. VoltDB and HyPer partition the database and execute transactions serially within a partition. However, the system also needs to handle cross-partition transactions and the two systems use very different mechanisms for this. TimesTen, solidDB, Calvin, and Hekaton do not rely on partitioning but use different concurrency control algorithms. TimesTen uses a locking-based approach but uses lightweight latches instead of traditional locks. solidDB uses a similar approach but with a twist where readers avoid latching. Calvin has several alterative locking approaches which can be switched in and out for each other using Calvin's modular architecture, one of which (termed VLL) requires only lightweight semaphores in conjunction with a deterministic execution engine. Hekaton uses multiversioning to avoid interference between readers and writers and an optimistic concurrency control scheme

The system designers also opted for different index structures. For example, several systems support range indexes but implemented differently. TimesTen range indexes use T-trees, Hekaton uses Bw-trees while solidDB and HyPer both use tries but of very different design.

The papers in this issue provides a snapshot of the current state of the art and cover many innovate solutions. I would like to extend a heartfelt thanks to the authors for their efforts.

Happy reading! I trust that you will find the papers as interesting as I did.