DataBlitz
Main Memory DataBase System
DataBlitz is a general purpose Main Memory Database System that enables:

- high-speed access to data
- concurrent access to shared data
- data integrity to be preserved in the presence of faults or failures

DataBlitz differs from typical commercial database systems in that:

- data is stored in main memory, not on disk
- data is accessed directly, not over a network
- there is no buffer manager
- lower level APIs are exposed to applications
• DataBlitz’s high performance (high throughput and short, predictable response times) makes it an ideal basis for a variety of purposes.

• The fast-emerging world of e-commerce thrives on speed and is a natural for DataBlitz.

• DataBlitz plays an enabling role in web servers as well as in web infrastructure (cache servers and other web accelerators).

• In financial trading, time is money. Even being a minute late may cause a trader to miss a market move. DataBlitz’s performance and strength in transaction processing make it an ideal basis for on-line stock exchanges as well as for program traders.

• Decision-support systems, currently based on off-line analysis of information in data warehouses, can be made much more effective by means of rapid analysis of real-time data based on DataBlitz.
Target Telecom Applications

- **Real-time billing:**
  - Bill for up-to-the-moment service and usage
  - Maintain billing history information
  - Keep customer and service summaries

- **Intelligent network applications**
  - 800 number translation
  - Intelligent 800 number service (via customer logic)
  - Call screening
  - Call routing (local number portability, unified messaging, etc.)

- **Switch-based functions**
  - Switching
  - Call routing
  - Call forwarding
  - Call waiting
High Performance Databases

- **Characteristics**
  - High throughput with hundreds to thousands of transactions per second
  - Real-time response on the order of a few milliseconds

- **Data must be in main memory for performance attainment**
  - Disk accesses have high latency (~20 milliseconds) and low throughput
  - Main-memory accesses have low latency (~100 nanoseconds) and high throughput and memory prices are falling reapidly
Existing Commercial Databases

Most commercial systems assume data is primarily disk-resident

- Every access incurs
  - Network latency
  - Disk latency
  - Buffer manager overhead
DataBlitz provides real-time concurrent access to shared data.

True 64-bit Architecture support!
Applications can be written to use one of four APIs

Lower level APIs offer simpler functionality and even higher performance
The Modular Architecture of DataBlitz
Database Management
- Keeps track of database files in a DataBlitz system
- Provides services to create, remove, resize, open, close and coordinate access to a database file
- Manages the virtual address space of processes and keeps track of start addresses for mapped database files
- Allocates/grows the underlying shared memory for a database file

Process Management
- Keeps track of active processes and resources held by them (e.g. mutexes)
- Performs cleanup for a failed process or thread

Communication Management
- Provides a framework for messaging (local and remote)
- Supports client/server remote procedure calls
Recovery/Fault Resilience

<table>
<thead>
<tr>
<th>Cleanup</th>
<th>Recovery</th>
<th>Archive</th>
<th>Codeword Protection</th>
<th>Memory Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Checkpoint</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Logging

- Recovery from system failures
  - Recovers database to transaction consistent state
  - Fuzzy checkpointing, multi-level and post-commit logging
- Recovery from process failures
  - Process failures detected and transactions aborted
- Recovery from media failures
  - Creation of and recovery from database archives
- Detecting data corruption (due to application errors)
  - Codeword (one-word checksum) for each database page
  - Any page written to disk must match codeword
- Preventing data corruption (due to application errors)
  - Uses mprotect call to cause page faults on bad writes
Traditional Recovery Algorithms

1. Each transaction writes the following to the log tail:
   - Undo (before image) logs before update
   - Redo (after image) logs after update

2. Write-ahead logging
   - Undo logs are flushed to disk before any affected page is written
   - Logs are flushed on transaction commit

3. Recovery in 3 phases
   - Analysis - find winner and loser transactions
   - Redo - redo effects of completed transactions
   - Undo - undo effects of incomplete transactions

4. Checkpointing
   - Flush dirty (updated) pages and note transaction information
   - Discard old part of the log
Recovery from System Failure

- Logs all updates on transaction commit/abort
- Writes periodic checkpoints of database state
- Performs recovery using checkpoint and logs

Diagram:
- Database
- Redo Log
- Undo Log
- Dirty Page Table
- Active Trans. Table
- Trans. Local Logs
- System Log Tail
- End of Stable Log
- Stable System Log
- In Main Memory
- On Disk
- End of Stable Log
- Database
- cur_ckpt
- Ckpt A
- Ckpt B
- Ckpt_dpt
- Active Trans Table (ATT) (undo logs)
- Repeat history
- Low Amount of disk I/O
  - Redo logs are written to disk
  - No undo logs to disk normally
- Ping-Pong Checkpointing Scheme
  - Write to alternate disk copies on successive checkpoints
    - resilient to crashes during checkpointing
  - Only relevant undo logs written to disk when checkpointing
    - undo logs of transactions that are active at the end of the checkpoint
- Fuzzy checkpoints
  - Updates execute concurrently with checkpoints
    - e.g. No page latching during checkpoints
  - Only dirty pages written to disk during checkpoints
- Single pass recovery algorithm
Extended Logging Support

- Logging support for high concurrency
  - Multi-level logging
    - Physical undo during updates, logical undo afterward
  - Post-commit actions
    - Actions that cannot be rolled back (e.g. freeing storage space)
- Support for user-defined operation logging
- Time stamp recovery
  - Recover the database to a transaction-consistent point in the past
- Turn redo logging off
  - Support atomic transactions without requiring them to be durable
When codeword protection is turned on for a database, a parity-based checksum (codeword) is maintained for each page, and updates performed through DataBlitz update the checksum.

The integrity of a page can be tested by this checksum by detecting errors, such as writes that did not use the DataBlitz interface, and random writes by means of bad application pointers.

The integrity of a page is always tested before it is written to disk, so the disk copy is never corrupted.

The user can also request "codeword audits" at other times.
• This allows pages to be protected using the UNIX system call `mprotect()`.
• Pages are unprotected when written, and reprotected on transaction commit.
• While the unprotect and reprotect are slow (by DataBlitz standards), memory protection can be very useful for debugging, for read-only databases, and for databases with relatively low update rates.
Concurrency Control

C++ Relational API

Collections API
- Heap
- File
- Hash
- Indices
- Ttrees

Storage Manager API
- Transactions
- Recovery/
- Fault Resilience
- Concurrency
- Control
- Storage
- Allocator
- Infrastructure Services

SQL
**Concurrency Control**

<table>
<thead>
<tr>
<th>Lock Table</th>
<th>Shared Mutexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock Manager</td>
<td>Exclusive Mutexes</td>
</tr>
</tbody>
</table>

- **Fast recoverable Mutexes**
  - Operating System semaphores are too expensive
  - Mutexes are implemented in user space for speed (e.g. spin locks)
  - Algorithms provided to detect a failed process holding a mutex
- **Lock manager**
  - Fine-grained locking (read/write/intention)
  - Supports lock upgrades/downgrades
  - Lock table used to obtain named locks with scalable latching
Storage Allocator

C++ Relational API

Collections API
- Heap
- File
- Hash
- Indices
- Ttrees

Storage Manager API
- Transactions
- Recovery/
- Fault Resilience
- Concurrency
- Control
- Storage
- Allocator
- Infrastructure Services

SQL
Chunks can have different recovery characteristics (zeroed memory, persistent memory).

Control data stored separately from data.
- Reduces probability of corruption due to stray application pointers.
- **BlzPtrs**
  - Database file plus offset
  - Enables database files to be mapped anywhere in the address space of a process
  - Enables database files to be resized
  - Size of database can exceed virtual address space of processes (e.g., 32-bit machines)

- **Multiple Storage Allocators**
  - Fixed-size allocator
  - Variable size allocator
    - Exact fit
    - Coalesce adjacent free blocks
Full Transaction Semantics

- Transaction commit, abort, and save point interfaces provide fault tolerance through internal locking and logging facilities which deliver (A)tomicity, (C)onsistency, (I)solation, and (D)urability properties.
- Use of transactions greatly eases the task of implementing systems that handle concurrent programs, and a variety of failures, while maintaining the integrity of vital data.
The heap file is an abstraction for handling a large number of fixed-length data items, and is implemented as a thin layer on top of the Storage Manager allocator.

In addition to insertion and deletion, the heap file supports locking and an unordered scan of items.

Item locks are obtained transparently when items are inserted, deleted, updated or scanned.
Extendible Hash Indices

C++ Relational API

Collections API
- Heap File
- Hash Indices
- Ttrees

Storage Manager API
- Transactions
- Recovery/
  Fault Resilience
- Concurrency Control
- Storage Allocator
- Infrastructure Services

SQL
DataBlitz offers both fixed and extended hash indices:

- Directory grows with number of elements in hash index
- Inserts/deletes/scans involving different buckets can execute concurrently
- Directory can grow while inserts/deletes/lookups continue
- Ideal for point look-ups on data
- Supports varying degrees of isolation (e.g., cursor stability, phantom)
Ttree Indices

C++ Relational API

Collections API
- Heap
- File
- Hash
- Indices
- Ttrees

Storage Manager API
- Transactions
- Recovery/
- Fault Resilience
- Concurrency
- Control
- Storage
- Allocator
- Infrastructure Services

SQL
• Ordered structure, like an AVL or height balanced tree
• Multiple keys per node
• Ideal for ordered scans over data
• Supports varying degrees of isolation (e.g. Cursor Stability, Phantom)
C++ Relational API

Collections API
- Heap
- File
- Hash
- Indices
- Ttrees

Storage Manager API
- Transactions
- Recovery/
- Fault Resilience
- Concurrency
- Control
- Storage
- Allocator
- Infrastructure Services

SQL
<table>
<thead>
<tr>
<th>Views</th>
<th>DML</th>
<th>DDL (SQL)</th>
</tr>
</thead>
</table>

- Supports Hash and Ttree indices on an arbitrary subset of attributes
- Predicate-based scans on tables are supported
- Materialized joins and one-to-many relationships
- Enforces referential integrity constraints
- Wide variety of field types
  - Variable length fields
  - Date, Time, Decimal, and other numeric types
  - Null values
- Views: simple select project joins on base tables
- Supports different isolation levels (e.g. Cursor Stability, Phantom)
- Schema/Data import/export
DataBlitz builds upon the Dharma/SQL engine to provide SQL, JDBC, and ODBC support.

The Dharma/SQL engine provides feature and performance parity with leading DBMS vendors.

The Dharma/SQL engine is production-proven with a large installed customer base.

Standards conformance:
- Complete SQL 92 entry level support
- ODBC 3.0 (core, level1 and most level 2 APIs)
- JDBC 1.2
SQL Interface Features

- **Query Optimization**
  - Statistics and cost-based optimization (e.g. query selectivity, table cardinality, histograms)
  - Join order/method selection for complex queries
  - Query rewrite algorithms for nested queries
  - Join hints for influencing query execution plans

- **Run-time optimizations**
  - Compiled/cached SQL for repeated queries
  - Multi-tuple fetches/appends
  - Tuple-level locking support
  - Execution plan data pipelining

- **Complete SQL-92 support plus**
  - Date/time/interval types
  - Variable length character strings
  - Nested queries
  - Transaction isolation levels
  - Recursive views
  - Schema manipulation statements

- **Java support**
  - Java stored procedures
  - Java triggers
  - JDBC 1.2
DataBlitz provides:
- the highest performance
- the most reliability
- the most flexibility
Data replication can help improve
- System availability (data still available even if site fails)
- System performance (replicated data can be accessed locally)

DataBlitz supports “update anywhere” replication model
- Data is replicated at the granularity of tables
- Any site can update/access replicas without consulting other sites
- Updates propagated to replicas asynchronously using logs
- Conflicts between updates resolved using timestamps

The DataBlitz replication manager guarantees that
- Every table update is eventually propagated to every table replica
- Table replicas converge to identical state when system is quiesced
Updates performed at site 1 via T1 propagate to site 3. Updates performed at site 2 via T2 propagate to site 1. Updates performed at site 3 via T3 propagate to site 2.

The DataBlitz replication protocol ensures that all sites converge to the same state.
Data Replication

• Handling conflicting updates
  • Update with largest timestamp wins
  • Timestamp and update information for loser updates output to a file

• Handling “normal” site failures
  • After site recovers, it resumes propagating updates and applying updates to replicated tables (shipped to it from other sites)

• Handling extended outages
  • Logs for certain updates to be applied at recovering site may have been truncated
  • After site recovers, it merges the most recent copy of the table from an operational site with its own copy of the table
**Hot spare** is the backup or secondary system
- Keeps in sync with the primary via logs
- **Higher availability** as the backup takes over when the primary fails
- Once the backup has taken over, it becomes the primary. The original primary would then become the secondary.
Additional DataBlitz Features

- Thread-safe
  - Concurrent thread access to DataBlitz

- True 64-bit architecture support
  - Database files greater than 4GB

- Configurability
  - Can turn features on and off (e.g., logging, locking)
  - Can fine tune various system parameters (e.g. number of chunks, maximum database files, active transaction table size)

- Bulk loading
  - Extremely fast data loading while locking and logging are disabled
- **Root server**
  - Initializes system structures (e.g. lock table)

- **Checkpoint server**
  - Checkpoints database files periodically

- **Flush server**
  - Writes logs to disk asynchronously

- **Cleanup server**
  - Detects failed processes and co-ordinates cleanup

- **Recovery server**
  - Recovers a database in response to data corruption

- **Mlock server**
  - Memory locks performance critical database files in memory
DataBlitz System Tools

- **Data Migration**
  - Tools for transferring schema information or table data between databases and files
  - Tools for transferring relational data in tabular form
- **RelDDL**
  - A stand-alone tool used for data definitions in standard SQL syntax
- **Administrative GUI**
  - Monitor system resource usage
    - Storage space (segments, bytes allocated in a chunk)
    - Servers, processes, threads
    - Transactions, locks
  - Perform archives/restore
  - Change configuration parameters
- **Archive/Restore**
  - C++ API and command line interpreter for database backups
- **Resource Monitoring/Checking**
  - C++ API and command line interpreter for checking usage of resources
DataBlitz
Main Memory Database System

Multiple Interface Layers

Fastest Main Memory Database System

High Availability

Concurrency

Flexibility

True 64-bit support