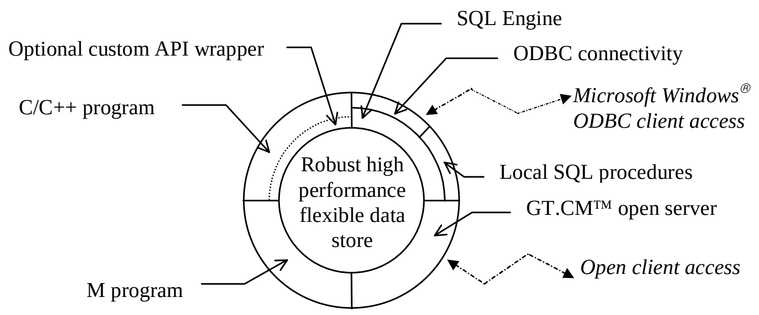
GT.M™ Database

GT.M includes a robust, high performance, multi-paradigm, open-architecture database. Relational, object-oriented and hierarchical conceptual models can be simultaneously applied to the same data which is accessible to concurrent host-based and client-server applications via C/C++, SQL and M.



The GT.M data store's underlying data is typeless (simply, an unstructured array of bytes). Relationships can be described equally correctly as:

- hierarchical
- multi-dimensional sparse arrays
- content-associative memory

Programmatically, the contents are accessed by name as persistent *global variables*. Relational, object-oriented and navigational database models all map well onto the GT.M database; thus the most suitable model can be chosen for each application or situation. Furthermore, multiple models can be applied at the same time to the same data, for mixed-mode (hybrid) concurrent access.

Consider a simple example:

- Products are assembled from parts or subsystems
- Subsystems are assembled from parts or other (smaller) subsystems
- Parts are purchased from vendors; each part has a different delivery time from each vendor which offers that part

In a relational database, this might be represented by the tables **Assemblies**, **Parts** and **Vendors**:

Assemblies

Item	Product
Go-cart	Yes
Mower	Yes
Drivetrain	No
Motor	No

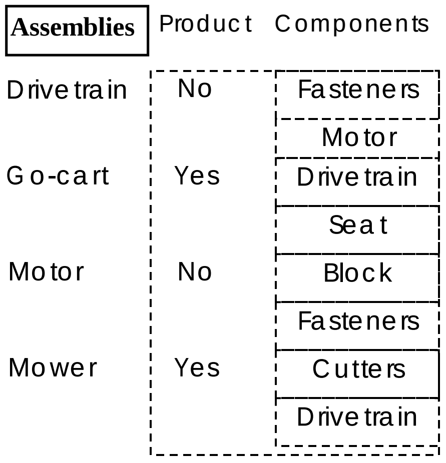
Parts

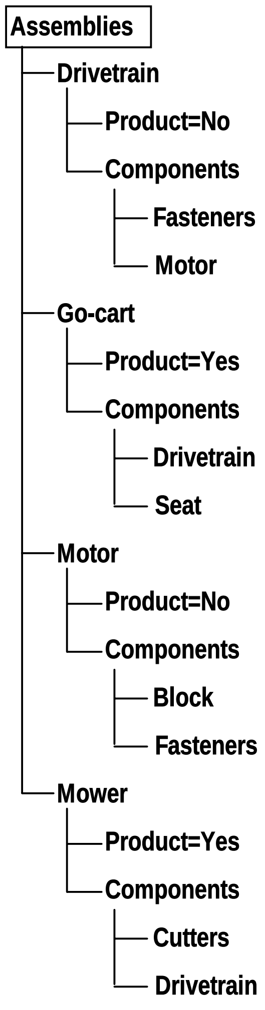
Item	Contains
Go-cart	Seat
Go-cart	Drivetrain
Mower	Drivetrain
Mower	Cutters
Drivetrain	Motor
Drivetrain	Fasteners
Motor	Fasteners
Motor	Block

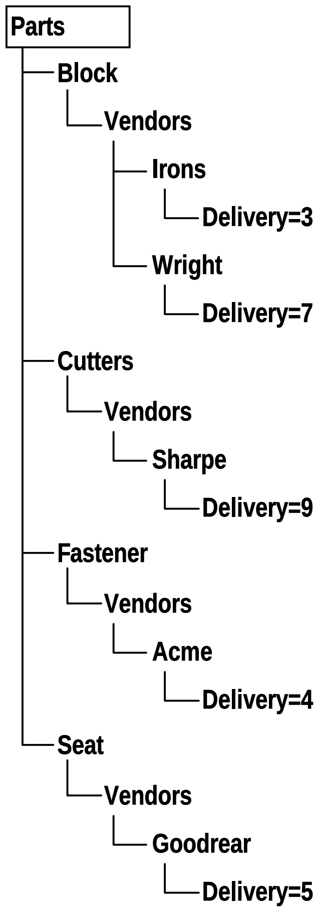
Vendors

Part	Vendor	Delivery
Seat	Goodrear	5
Cutters	Sharpe	9
Fasteners	Acme	4
Block	Wright	7
Block	Irons	3

An equivalent GT.M hierarchy is shown to the right. Access from C/C++ or M is available via two persistent global variables, **Assemblies** and **Parts**. The multi-dimensional sparse array view of the three dimensional array **Assemblies** is shown below (the four dimensional array **Parts** is not easily depicted on two dimensional paper).







The object oriented paradigm is fulfilled by considering **Assemblies** and **Parts** to be classes, with four instances of each. Since the data is typeless, binary information, such as stream of bytes to be fed to an interpreter or loader at run time, can also be stored.

Using SQL engine, the GT.M database can be mapped, accessed and manipulated and accessed using the relational tables shown. If access is desired from a PC client, relational access is available via ODBC. Thus, applications created with popular tools such as Visual Basic™ and PowerBuilder™ can access and update GT.M databases.

The global variables in a database file are shared by all processes accessing a database file with both automatic and programmatic interlocking to prevent race conditions.

Benefits of flexible modeling

The support of efficient access to the data means that the most appropriate model can be chosen on a case by case basis for each type of access to the database. For example:

- The most full featured report writers exist on the Windows/PC platform and support the relational model. Thus, it is the appropriate model for most client-server applications and for export.
- The hierarchical model allows a natural solution for a number of problems that degenerate into pathological cases in the relational model. For example, consider the query to report worst-case (maximum) delivery time for any component in the manufacturing of any product.[1] While straightforward in GT.M, this would require the proverbial "*n* way join from hell" in a traditional relational database.

Max_deliveries

Product	**Delivery**
Go-cart	**5**
Mower	**9**

- The object-oriented model can be used to implement "compute on demand" for data that is expensive to compute and not routinely accessed, or for data that must always be computed afresh.

Mixed mode hybrid access can be exploited to use the capabilities of one mode to overcome limitations of another. For example, the pathological SQL query for the worst case delivery time can be overcome in SQL engine by the insertion of executable code to exploit the hierarchical nature of the database and the fact that the database can store code as well as data. Thus, to a relational client, **Max_deliveries** can look just like another table, but one that need not actually exist in the database.

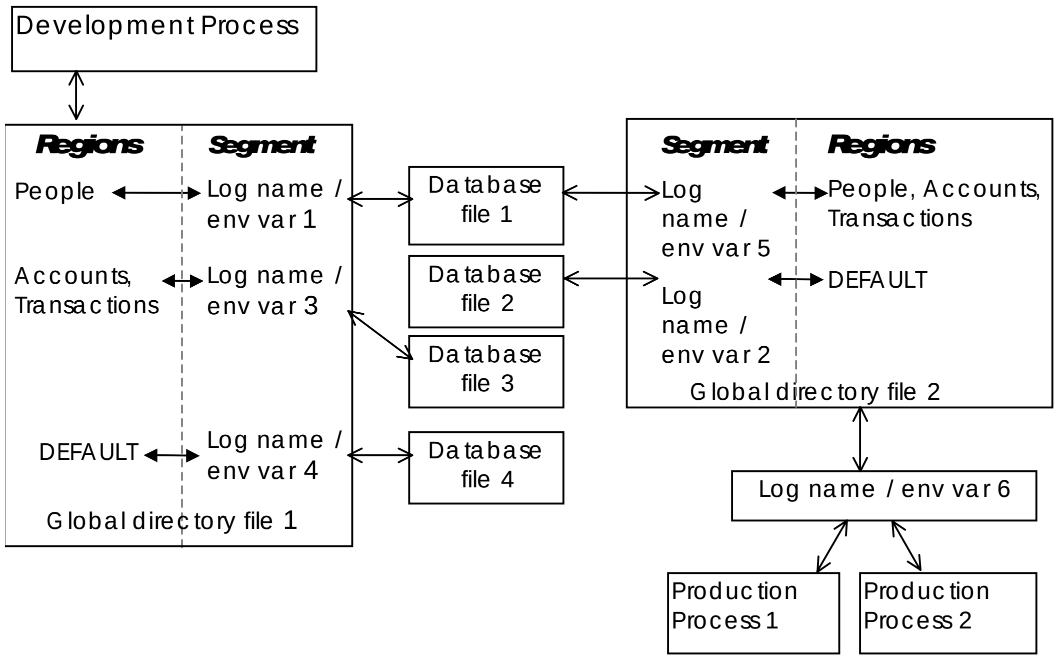
Other benefits

- **Compactness** Because of the database structure, and the use of key compression, GT.M databases are often a fraction of the size of equivalent databases relational databases on industry-standard SQL servers.
- **Capability** GT.M supports features such as application level transactions via "transaction start" and "transaction commit" markers.
- **Speed** GT.M databases are often much faster for transaction processing (mixed reading and writing activity) than typical relational databases. This advantage is often even more pronounced under heavy loads, because the compactness of GT.M databases reduces I/O activity, sometimes a greater bottleneck than the CPU.
- **Robustness** GT.M comes with features such as exception handling, journaling and recovery for implementing robust and resilient applications.

[1] Notice that the maximum delivery time for **Go-cart** comes from **5** for **Seat**, since **Delivery** for **Block** should be **3**, based on **Irons**.

Configuration Flexibility

GT.M provides flexibility in database configuration by allowing each process to distribute its logically monolithic global variable name space across an arbitrary number of database files with the aid of a *global directory* file, either in its working directory or specified via a logical name (OpenVMS) or environment variable (Unix). A global directory file maps a *region* (a range of global variable names) to a *segment* (a file name or a file specified indirectly via a logical name or environment variable).



In the example, two executing images share the same global directory file, and direct their People, Accounts and Transactions globals into the production database file. This mapping is defined by Global directory file 2, which is indirectly accessed through an environment variable / logical name (so that, in the event of operational need, the production processes can be redirected to a different global directory file). At the same time, a development process uses a different database file for its Accounts and Transactions variables. However, to facilitate testing, it is using the production database for People to which it has read-only access. Its global directory file merely resides in its current working directory, there being no need to direct it through a logical name / environment variable.

Executing processes can change global directories on the fly, or even force a specific database access to use a specific global directory other than its current global directory. Two global directories can map segments (either the same global variables or different global variables) to the same database file.

Database Summary

Format	Data is stored as high-performance B* trees in Sanchez' proprietary format.
Configuration	Global directories specify the relationship between global names or ranges of global names and database regions.
Logical Names	The global directory itself as well all regions specified by a global directory may be logical names / environment variables, allowing run-time reconfiguration of

	the database or a portion of the database without invading application programs.
Dynamic Global Directory Specification	Any database access can force the use of a specific global directory. The current global directory can be changed at any time.
Access Methods	The standard access method (Buffered Globals) provides database access using a caching system optimized for high performance database operation. For even more speed, the optional Mapped Memory access method allows entire database regions to be mapped into virtual address space. For extraordinary throughput on selected platforms, the database is mapped into virtual memory accessed with 64-bit pointers (VLM).
Application Transactions	"Transaction start" and "transaction commit" markers can implement application program level transactions.
Journaling and Recovery	Database updates may be journaled. Two types of journaling are available, one of which stores update information as well as the prior contents of database file disk blocks, and the other storing only update information. A utility restores a journaled database to a consistent state established immediately prior to a system failure. By including transaction marker commands, an application can extend consistency to logical transactions. Update information can also be converted into an ASCII form.
Security	GT.M supports the security mechanisms of the underlying operating system for controlling access.
Shutdown	Shutdown consists of running a utility to ensure that all GT.M processes are terminated, and all shared memory is released. Database regions automatically become inactive when no user references them.

Database Limits

String length	to 32,508 bytes
Numerics	18 digit precision in the range 10^{-43} to 10^{+46}
Variables	No GT.M limit on size or number; individual segments are subject to underlying computer system limits on file size
Key size	to 255 bytes
Node size	to 32K bytes for the sum of subscripts + data for a single M node
Block size	user specified by region from 512 to 65,024 bytes in multiples of 512