Making Sense of Data Management on Intelligent Devices

The demand for embedded devices is growing rapidly, and there is a clear need for development of advanced software to deliver new features on limited hardware. Data management is a critical component in these new software systems. Embedded databases are used by portable media players to store information about music and video, GPS devices to store map data, and monitoring systems to log information. These and other leading-edge industries have learned the importance of managing data reliably with a relational embedded data management system.

Developers face unique challenges when designing and implementing software for custom embedded hardware. Embedded processor architectures, such as ARM, PowerPC, Atom™, each have unique characteristics. Footprint and performance are especially important, and access to source code for all software components is required for customization and portability.

To meet these requirements, embedded developers have often relied on custom solutions, using flat file formats to store data. However, increasing hardware capabilities make it possible to store more information on embedded devices than ever before. Fast read and write operations, protection from data loss and corruption, and multi-user access have become important requirements for embedded systems. Flat files are not able to fully address these issues.

What is an embedded database?

- An embedded database is a software library used by application developers to store data.
- The library adds database features to the application such as transaction logging, scalable index algorithms, and isolated concurrency.
- Unlike enterprise databases, an embedded database is distributed with the application and is not installed separately by the end-users.
- Embedded databases are especially well-suited for special-purpose devices and embedded systems with limited resources and a dedicated user interface.

Design Considerations for Embedded Data

Critical performance demands: Embedded devices operate under strict time constraints. Whether to satisfy impatient users or to keep up with a constant stream of incoming sensor data, performance is always important.

Fail-safe reliability: Embedded systems are subject to failure from unexpected power loss and other crash scenarios. If such a situation occurs during a write operation, data may be lost or even corrupted. Redundancy is necessary to ensure reliability.

Sharing data between concurrent tasks: Modern embedded systems are connected and intelligent, performing several tasks at once and often sharing data between those tasks. Locking primitives, such as mutexes, are cumbersome to use directly in complex scenarios.

Portability: The exact format of data in memory is determined by the processor architecture and the compiler. But platform-specific details, such as byte order, alignment, and structure padding, should not affect the format of data stored on persistent media, such as flash.

Embedded Relational Database

While each individual problem, in isolation, has a straightforward solution, it is difficult to address one requirement without compromising on the others. Just as saving data safely can limit throughput and the size of the data set, sharing access to the database complicates safe storage and also degrades performance. An embedded relational database management system (embedded RDBMS) provides a complete solution that carefully balances these requirements.

Embedded databases are used in a variety of applications, each with different requirements. To accommodate this, many options are available to control the behavior of the database:

- High performance read and write
- Main-memory and disk-based tables
- Single-user, multi-threaded, and client/server access models
- SQL queries and direct table cursors
- Integrated C/C++ APIs and ODBC

Relational Model

In a running application, data is organized in data structures, such as classes, that reference each other directly, sometimes in a hierarchy, but usually in a complex network. However, direct references are difficult to maintain when data is stored persistently, especially if it is shared with other tasks that approach the data in a different way. Even small changes to the application can easily break backward compatibility. Porting to a new processor or operating system, or even changing the compiler, can raise unexpected problems.

Instead, relational databases organize data in tables, where related tables share common fields. In this way, relationships are maintained naturally and can always be used in both directions. Data is easily accessed through SQL queries and standard interfaces such as ODBC. And because there is a clear boundary between the representation of data in a working application and the representation used when that data is stored, changing the application or supporting another platform is a straightforward process.

Consistent, Scalable Performance

Embedded devices need consistent, scalable performance across all operations, whether reading or writing to the database. Indexes are used to efficiently search the database and traverse the relationships between tables. B+ tree indexes are optimized to minimize disk I/O, and offer consistent performance regardless of the size of the table, even with limited random-access memory. For tables that can fit entirely in main memory, T-tree indexes ensure that processor instructions are minimized.

Shared Access with Multi-user Connections

An embedded database can be shared between several concurrent tasks, and can present each task with what seems to be exclusive access to the data for a short time. By automatically locking individual rows as they are read and modified, the database enables tasks to safely work in different parts of the database in tandem, only pausing or moving on to other work when they would interfere with each other.

Whether an application needs no shared access to a database, access from several threads, or from several processes, embedded databases can accommodate each scenario, and the same application code can be used in all cases.

Database Recovery

When a sudden power failure or crash occurs while writing to a file, data corruption and inconsistency can result. To prevent corruption, embedded databases first write each change to a separate log file before modifying the database file. Using the log, incomplete changes can be rolled back to restore the database to a known good state.

If the database software uses write-ahead logging, also known as undo/redo logging, changes can be written to the database file either before or after a transaction is committed. This significantly reduces write operations without compromising data integrity. In this way, high-throughput tasks that frequently update the database can coexist with tasks that modify a large portion of the database at once.

Conclusion

Flat file formats are not robust enough to handle all of the problems that embedded developers will face as storage media continues to grow in size. A relational embedded database is a powerful and important tool in any embedded developer's arsenal. And while many off-the-shelf solutions are available, it is important to select a product that can fully meet your application's needs.

Some databases provide only basic functionality, with limited support for concurrency and mediocre performance in serious applications. Others are bloated with features that are unnecessary on embedded systems, requiring complicated installation procedures and consuming more system resources than the application itself. Starting with a solution that is designed to meet the requirements of embedded systems and devices has a significant impact on the performance, maintainability, and extensibility of the application.

ITTIA DB SQL is a pure relational database library that provides embedded applications with a single solution to the most important challenges of data storage. ITTIA DB SQL is fully functional, supporting write-ahead logging, B+ and T-tree indexes, complex multi-user shared access, and more. A variety of platforms are supported, and source code is available for porting to new platforms, customizing the feature set to minimize the already low footprint, or just for the assurance of having total control. With a high-performance relational embedded database like ITTIA DB SQL, application developers can focus on the business logic that makes each product unique.