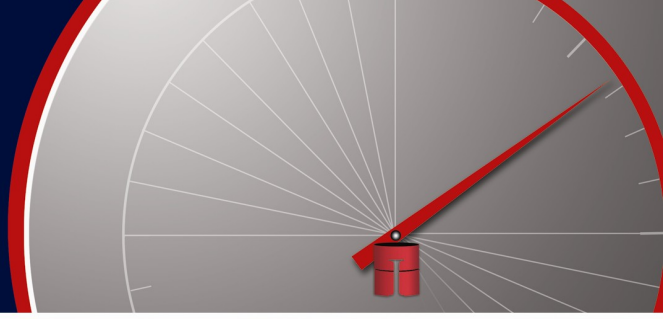
ITTIA
The Power to Manage Data



Data Management for Portable Media Players

Table of Contents

Introduction

Portable media players have evolved significantly in the decade that has passed since the first digital audio players first gained popularity in 1998. These devices could carry at most a few thousand songs, so they could utilize very basic algorithms to organize media, such as file directories. With the amount of storage capacity in portable media players ever growing, software developers must develop new techniques to manage music, video, and image files.

Information about each media file such as artist, album, date, location, and custom tags is used to organize media so that it is intuitive for the user to access. Known as metadata or tags, this information must first be scanned and aggregated to provide the responsiveness that consumers expect. Software developers now turn to relational embedded database management technologies to provide the scalability required by modern portable media players.

One of the most significant motivations for innovation, cost directly influences the success of a portable media player product in the consumer electronics sector. Devices are manufactured in very large quantities, so even a small savings on each device makes a huge competitive difference in the list price. For that reason, portable media players generally use inexpensive embedded processors with very little runtime memory (RAM), most of which is reserved for buffering decoded media. Facing this challenging landscape, developers know that very little memory is available to store and manage data. As the amount of metadata grows to include tens or hundreds of thousands of entries, they recognize a need for a powerful relational lightweight database management.

Today's consumers expect portable media players to be very responsive. With traditional algorithms, scanning a thousand media files whenever a media player device is turned on is barely noticeable. But the consumer would be unimpressed with the performance of such a device with ten thousand files or more. Instead, media players must scan each file only once, when it is first copied to the device, to avoid having a sluggish interface. Scanned data must be stored in a persistent location so that it is not forgotten when the player device is turned off.

A relational database stored on inexpensive NAND flash memory is a simple solution for the organization of media on a portable media player. A lightweight, embedded database management library uses proven algorithms to store and retrieve data, such as media file metadata, with minimum overhead and optimal performance. Design engineers capitalize on this advantage to reduce cost and deliver a produce with time-to-market dramatically reduced.

The New Role of Database

Database technology is not new. Most business applications and high-traffic websites rely on one or more back-end databases to store data safely and efficiently. The volume of data processed by these applications often requires dedicated server hardware and constant monitoring by a database administrator. For this reason, a traditional database management system is far too complex and requires too much overhead to be used on a portable media player.

However, the core database technology and algorithms are still valuable for portable media players. Software developers may decide to implement themselves only those database algorithms that apply directly to their application, but the resulting data management framework is not likely to follow standard conventions and terminology. In these cases, when more robust data management becomes necessary later in the product's life cycle, it becomes very difficult to maintain and enhance the product.

An embedded relational database provides a standard interface for reliable data management that is suitable for portable media player applications. The database is invisible to the consumer and requires no administration because it is linked directly into the application as an in-process software library. Applications that utilize a lightweight relational database are completed faster, store data more reliably, and are better able to adapt to changing requirements.

Design Considerations

Hardware Limitations

Portable media players are designed for a single task – playing music and video. Typically, these devices use dedicated digital signal processor (DSP) hardware to decode media, leaving the general-purpose CPU free to operate the user interface and send media to the DSP. Unlike a desktop application, where a rogue task can sometimes consume all CPU resources, Media player software is carefully designed to limit the amount of time spent on each task so that media playback is never interrupted.

ARM is a popular processor architecture for portable media players because of its robust power saving features. A variety of ARM core designs are used in portable media players. Some media players utilize a pair of variable clock-rate ARM7TDMI processors, which operate at no more than 80 MHz each and consume very little power. More robust media players based on the ARM11 core can operate at about 600 MHz and often do more than media playback. Portable media players typically have about 32 MB of RAM, most of which is reserved for skip protection.

Media files and metadata is stored on NAND flash memory or miniature hard drives. While these are both inexpensive and persistent, they require block-wise reads and writes: data stored on these devices must be copied to RAM to access individual bytes of data. This is not a problem for media files that are accessed sequentially because every byte in a block is used every time it is read. Metadata is accessed randomly, so data is often read into RAM far in advance of when it is actually needed. For this reason, it is necessary to cache frequently accessed blocks so that these extra reads are not wasted.

But because available memory is limited, a media player software must use a predetermined amount of RAM for general processing. The total RAM required for a software component, such as a lightweight relational database, is based on four measurements: static code, static data, stack, and heap. The exact amount of RAM required, or footprint, depends on which parts of the software are actively running at any given time.

Static code contains the machine instructions that control the processor. On portable media players, static code never changes, so it is hard-coded in read-only memory (ROM). With techniques such as dynamic loading, only code that is actively running must occupy space in RAM.

Static data is allocated when software is compiled and contributes a fixed amount to the RAM footprint, even when the software that uses it is not actively running. For this reason, code that is used infrequently should have a negligible static footprint.

The stack is used for temporary variables and to share data between functions, the basic building blocks of software. The size of the stack increases temporarily while a function is active and returns to its previous size when the function exits. Inactive code consumes zero stack footprint. In many cases, the maximum stack size can be calculated when the software is compiled.

The heap is general-purpose memory that is dynamically allocated by software at runtime. Heap memory can be used both for temporary storage by active software, or to save information for later use. Heap utilization is difficult to measure accurately, and is usually measured empirically.

Value of a Lightweight Relational Database

Database technology provides applications with a practical model for data storage that is powerful yet easy to use. Information is organized into tables with a predefined structure. By hiding the exact format used to store information, the database can cache information intelligently to optimize performance while providing a wide range of features, such as transactions, recovery, search, and shared access, that would be difficult to implement and maintain in a single

application otherwise.

Application code is frequently re-used to expand an existing product line or to jump-start development of a new product. Using a database gives an application a consistent architecture for persistent data storage, making it easy to add new features and migrate the application code to a new environment. Database is a good long-term investment because it lets applications scale through the entire life cycle of the application.

For example, many devices are initially designed to store data locally and only synchronize with other systems as a dedicated task. If such an application is database-driven, it can use row-level locking to interact with other systems and share data without interrupting normal usage. Sending and receiving data and running custom queries is greatly simplified by the database API.

Many devices, such as mobile inventory management devices, are used to manipulate a subset of data from a large back-end database server. An embedded relational database on the device allows this data to be saved reliably in a format that is easily shared with a back-end server.

Why Choose a Database

An out-of-the box database solution reduces total cost of ownership and provides a shorter time to market because the developers can focus on core application development. Software developers often underestimate the amount of effort required to build a reliable data management framework, resulting in missed deadlines or an unmarketable product.

Enterprise relational DBMS technology provides many of the features required for data management on portable media players, but the footprint requirements and cost of these monolithic systems make them impractical for embedded systems and devices. Enterprise databases are designed to support concurrent access from thousands of users and balance load across multiple servers, requiring sophisticated management and administration. These products also include many deprecated features required to support decades of legacy applications and are optimized for servers with gigabytes of RAM and large hard disk arrays.

Embedded relational databases are designed to run in resource-limited environments. The database can be embedded directly in the application, hiding the database from the end-user and greatly simplifying management and maintenance. With a small code footprint, well under a megabyte, an embedded relational database can easily fit on portable media player hardware and can even be customized to satisfy strict footprint requirements. Embedded lightweight databases use algorithms that are efficient for small-scale applications, but that easily scale to

store a large amount of data.

An ITTIA Solution—ITTIA DB

The ITTIA DB database library is a data management solution for devices with strictly limited resources, such as portable media players. ITTIA DB is a family of products for embedded developers that gives software developers the freedom to choose a solution that exactly meets their requirements, without being locked in to a data management technology that can only satisfy current needs.

Tailoring ITTIA DB for a Specific Device

ITTIA DB lets developers select the exact database features to meet the particular requirements of a device and application. Minimizing the database library's footprint contribution frees developers to implement more features in the application, or save cost with less expensive hardware.

By letting developers select only necessary features, ITTIA DB reduces code size and eliminates runtime overhead. Minimized library size leaves more room for application code on the device's ROM. When the program is loaded into RAM, not only is the RAM footprint reduced, but also less time is spent loading the program into memory. Further performance is gained when the application is run because unused database features are completely disabled, making the application more responsive and reducing power consumption.

The code footprint of ITTIA DB can be as low as 140 KB on some devices. With an achievable static data footprint of a few hundred bytes and peak heap allocation of 70 KB, ITTIA DB helps minimize hardware requirements and leaves room for more media player features.

Tables and Indexes

ITTIA DB organizes information into tables that can be quickly searched using B+tree indexes. Performance remains consistent even as the number of rows in a table increases, enabling applications to scale easily. B+tree indexes are optimized for block-wise reads and writes, which is essential for efficient storage to NAND flash and hard drives. Recently used blocks are temporarily cached in RAM to improve performance. And because ITTIA DB uses write-ahead logging, modified blocks can remain cached in memory even after a transaction is committed to significantly improve performance by reducing write operations.

Applications developed with ITTIA DB can utilize a low-level C or C++ API to access tables and indexes directly. This gives the application fine control over performance,

without the often-unnecessary overhead and complexity of SQL queries. When the flexibility of SQL is required, ITTIA DB delivers full support for runtime SQL both in the application itself and through development tools.

ITTIA DB databases can be created at runtime on the device, or prepared beforehand when the software is installed. Dynamic schema alteration, the ability to modify the structure of a database, can be disabled in ITTIA DB to reduce footprint.

Transactions and Recovery

ITTIA DB uses recovery logging to prevent data loss caused by unexpected power failure. Portable media players often copy meta data from media files into a database to greatly improve responsiveness and start-up time. In such an application, the database is rarely modified. Furthermore, in the unlikely event that a power failure occurs while writing changes, the database can be regenerated from the media files themselves. With ITTIA DB, developers can completely disable recovery logging, without giving up the benefits of transaction rollback.

Rollback helps application developers to gracefully cope with errors. If the software on a media player were to encounter a media file that is partially corrupt, it might have already added data about the file to several tables in the database. Rollback lets the application group changes together into an atomic transaction that either finishes completely or not at all. When an error occurs anywhere in the transaction, the application can roll back all changes with a single API call, regardless of where in the process the error occurred.

Simplifying the Software Development Process

ITTIA DB uses static typing, where type information is stored in the database schema as part of a table's description. Each column can contain only a specific type of data. This ensures that type mismatch errors are identified early, when there is the best chance to successfully fix the mistake. This is important when the database is shared between applications that are developed separately, as the database schema forms a contract by which all parties must abide.

The terminology used by ITTIA DB, both in the API and documentation, should be familiar to enterprise database developers. Some embedded database products use unusual terms for common concepts, which forces developers to learn a new way of talking about databases.

Flexible Deployment

Portable media players are often sold as a family of products with different price

points and features. Whether the media player application needs SQL or direct table access, single-user or multi-threaded concurrency, local database files or remote client/server connections, ITTIA DB provides the same API interface and stores data in a common portable file format. Design engineers capitalize on this flexibility to minimize hardware cost by including only those features that are required for each product.

Working with ITTIA Toward a Successful Deployment

ITTIA stands behind its database technology with guaranteed support through all phases of product development. ITTIA's experts provide indispensable advice during planning and design that gets development started in the right direction from the very beginning. ITTIA's training programs give developers an edge up with a thorough introduction to the capabilities of relational embedded database and best development practices. In application development, ITTIA supports customers with direct problem tracking and resolution. Support even extends to the deployment of the device and application, ensuring a smooth delivery and market adoption.

Conclusion

Almost every consumer now owns a portable media player, and the demand for more storage capacity in these and other embedded devices continues to drive down the cost of storage hardware. But new hardware capabilities present new problems to the software, which fortunately have already been solved in other contexts, such as back-end servers.

In this paper, we have explored how an lightweight relational embedded database like ITTIA DB fills the critical need in portable media players for scalable organization of meta data with technology that many developers are already familiar with. A focus on industry standards and interoperability gives embedded developers who use ITTIA DB an edge over others because they can leverage their own experience and the existing knowledge of others.

How to Reach Us

Home Page:
www.ittia.com

e-mail:
support@ittia.com

USA:
ITTIA
1611 116th Ave
Bellevue, WA 98004
425-462-0046