

# Maximizing Resources in Data Acquisition Systems with a Network Data Model

By: John Pai, Birdstep Technology

When deciding on a data management solution for a data acquisition device, there are several different solutions available depending on needs. Selecting the appropriate data model can significantly impact cost, quality, end-user experience and ultimately customer success.

Data acquisition devices have come a long way, but since the first devices were created, one thing has remained the same: they have limited resources. In such embedded devices high performance is paramount, and small data acquisition platforms need to handle the data throughput at high rates, manage data relationships and process the information with minimal processing power so that more is available for the actual acquisition application.

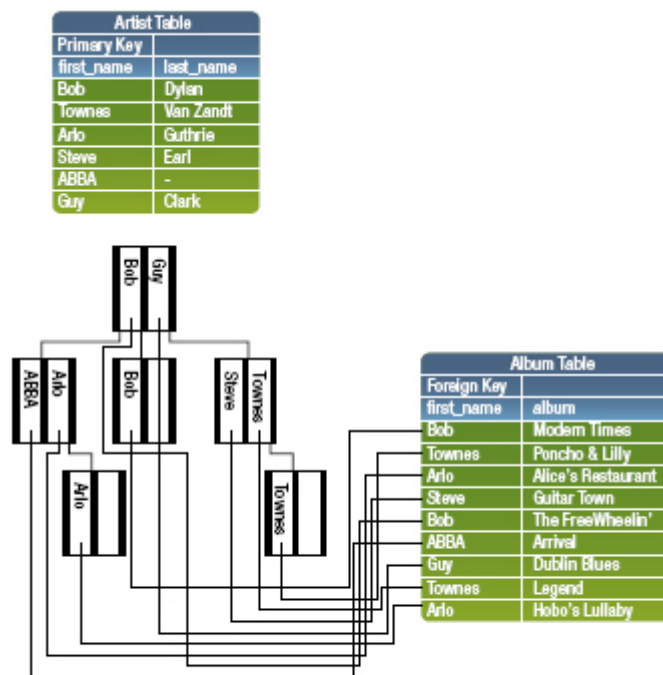
A common misconception when hearing the term “database” is the idea that the database is synonymous with enterprise-wise relational databases or inefficient and slow personal databases. Managing data in embedded devices is a different matter than managing data on PCs and servers. Enterprise databases are not designed for embedded devices, and their subsequent adaptations result in large memory footprints, often entirely too large to fit into the target device. Enterprise databases do not translate well into embedded devices, and attempting to adapt open source implementations presents its own special set of headaches. The embedded database is another type of database that is dramatically different from the well-known enterprise databases.

When it comes to choosing the right embedded database for data acquisition devices, there are three main types: flat-file, relational and network models. Which is the best one to use for a particular job will depend on factors such as type and amount of data to be processed, as well as how frequently it will be used.

## Flat-File and Relational Models

Essentially, the flat-file model is a set of strings in one or more files that can be parsed to get at the information they store. It is decent in storing simple lists and data values, and can get more complicated when trying to model more complex data. Modeling complex data creates a new set of headaches. One of the main issues with using flat files for even a semi-active database is the tendency to corruption. Generally, a flat-file database does not have a locking mechanism, which prevents data from being corrupted when multiple threads may simultaneously try to write to the database. Even when the device is designed for multiple threads, it is possible for two or more threads to cause a “race condition,” which could be prevented with a locking mechanism. A race condition may force the device to stall indefinitely, which is detrimental especially in an embedded device that normally does not reset easily. In a data acquisition device where reliability of data is also a priority, flat-file databases may not be the best choice.

The relational model stores data in tables composed of columns and rows. When data from more than one table is needed, a joint operation relates these different data using a duplicate column from each table (Figure 1). While the relational model is flexible, performance is limited by the need to create new tables to hold results from relational operations, and storing redundant columns. Even when designed efficiently, there are several sources of overhead. The overhead comes in duplication of data, in helping maintain database integrity, and a need for a foreign key to help maintain relationships in the relational database. The overhead results in excess in file size and extra I/O needed to perform basic database operation. Such overhead is especially expensive in resource-constrained devices.



**FIGURE 1**  
A relational model database with a relationship between an artist and an album.

Most developers are familiar with the relational database model, such as those from Oracle, Informix, Sybase, etc. Alternative data model architecture is more appropriate for resource-constrained devices such as data acquisition devices.

### The Network Model

The network model is conceived as a flexible way of representing objects and their relationships. The network model predates the relational model and can be viewed as a superset. This implies that anything expressed in the relational model can be expressed in the network model, even SQL support. The main advantage is the way the relationships are modeled.

A primary distinction to the relational data model is that the network model allows designers to describe relationships between records using “sets,” where pointers are used to relate objects directly and navigate between them (Figure 2). When compared to the relational model, the network model is faster, more reliable, uses disk space more efficiently and is better at expressing complex database designs.

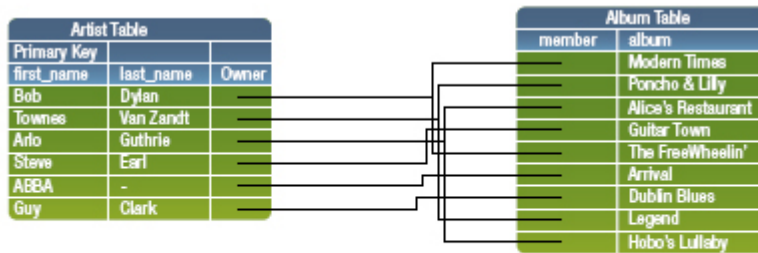


FIGURE 2

The same database as the one in Figure 1, but streamlined into a network model.

Consider an instance when traversing from one table to another using a relational link. After locating a key value in the first table, the database engine seeks out the value in an index file; this contains a reference to the second table. Searching the index file costs the system several disk accesses for each record accessed. In the network model, this process is accessed through sets. A set is a linked list representing a one-to-many relationship, which contains pointers to the next and previous member link of the set. Traversing from one member to another member and from member to owner requires only one disk access.

## Cost Implications

Consider reading records through the sets. When performing the same task with a database using a relational model, a B-Tree would need to be traversed first, prior to locating the actual record. It is also important to note, not only will there be disk access overhead, but also memory overhead. Any database cache will store the recently visited data, even B-Tree data. Since the B-Tree search ends up in cache, the cache must be sufficiently large to reduce the chances of additional disk access to update the data.

A write operation also involves heavy costs in a relational model. After a record is inserted into the table in a write operation, the database inspects the B-Tree to locate the record's index position. If there is no room available in the B-Tree, the tree needs to be reorganized to maintain efficiency. This reorganization process is unpredictable because it depends on the fullness of the tree and where in the tree the change must be made. The more nodes a tree contains, the greater the chance of a larger reorganization, which may be time-consuming and unpredictable. The reorganization process may also require the operating system to devote all computing resources into reorganizing in order to meet the time constraints. After the reorganization process is performed, the database can write a reference to the new record in the B-Tree. In a network model, adding a record is relatively simple, less taxing and predictable. The process involves adding a new record and setting pointers to owner, previous and next record. Then, setting the owner's last pointer to the new record. This process is fast, predictable and does not require reorganization of a B-Tree.

Another cost implication involves the features of a database that may help to reduce development time and minimize a measuring device's time-to-market. Some features can reduce development costs and exploit hardware such as circular tables and hybrid in-memory databases. Birdstep Technology also offers a database that may allow an embedded client to replicate the embedded database into a central aggregate point, a server database.

## An MP3 Player Benchmark

	Hardware			
	x86 Desktop (34,000 records)		ARM7 consumer device (1,776 records)	
	Relational	Network	Relational	Network
Records Inserted	81.62 sec	29.07 sec	193.88 sec	33.6 sec
Records Updated	103.28 sec	15.28 sec	Not Available	Not Available
Records Deleted	88.16 sec	17.03 sec	Not Available	Not Available
Records Selected	15.81 sec	.28 sec	1.25 sec	0.0012 sec

**TABLE 1**

Bench results for relational and network models on x86 and Arm7 devices.

So how much in the way of hardware resources can a developer save using the network model? In one example, a three-way relationship, artist->album->song, allowed a commercial MP3 player manufacturer to get some hard facts on resource savings. Developers carefully compared the relational and network models using both desktop hardware and consumer hardware. Table 1 illustrates a correlation between the lack of resources and a difference in savings. On both of these hardware solutions, the network model used 27 percent less disk space to store the same amount of records and relationships. All the storage savings can be attributed to replacing the artist->album and the album-> song foreign key index with pointers. Removing these data structures had a huge effect on the storage requirements. A B-Tree index typically requires 1.3 times the space of what it's indexing.

Birdstep Technology  
 Seattle, WA.  
 (206) 748-5300.  
 [www.raima.com].

© 2009 RTC Group, Inc., 905 Calle Amanecer, Suite 250, San Clemente, CA 92673